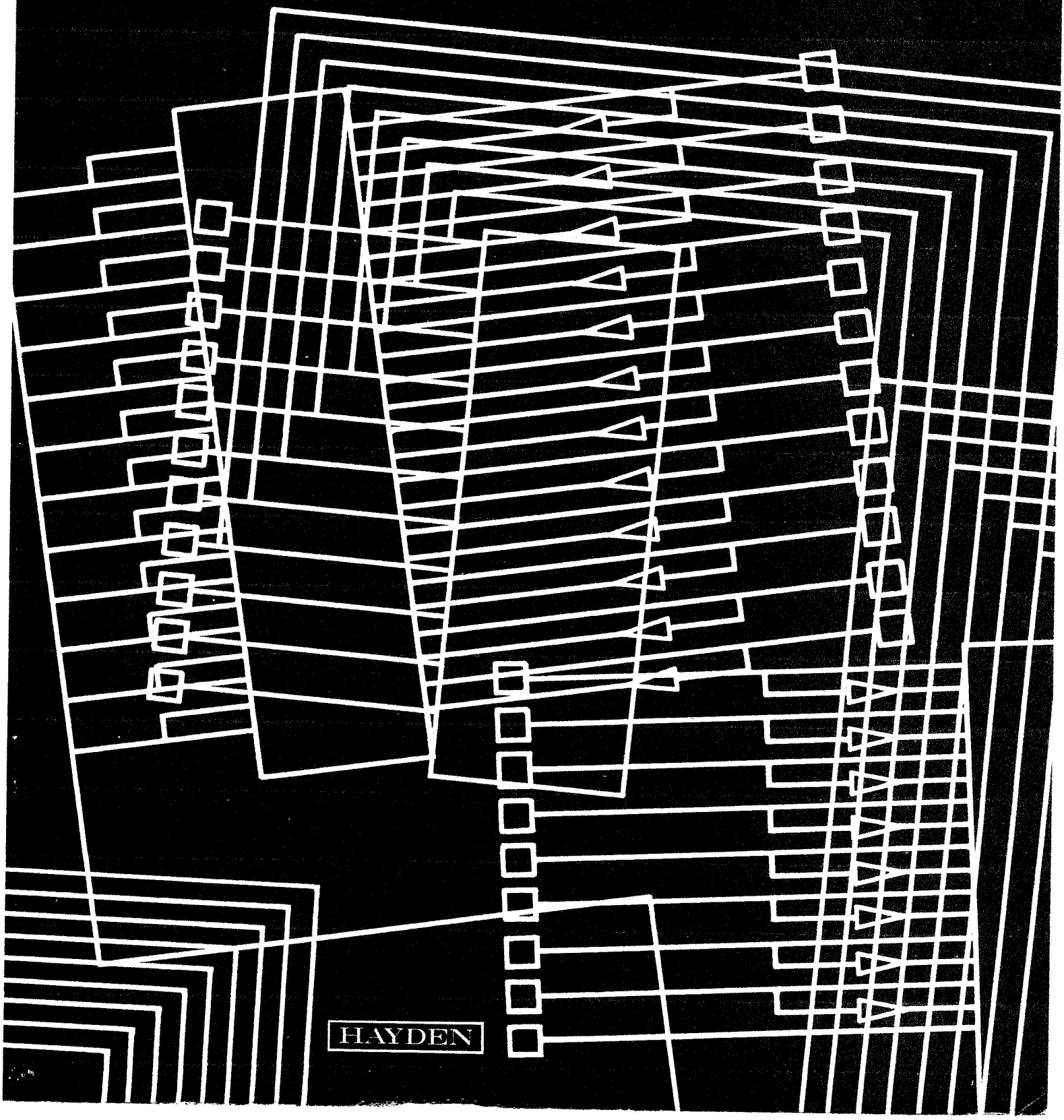
The S-100 BUS Handbook

DAVE BURSKY



HAYDEN

THE S-100 BUS HANDBOOK

THE S-100 BUS HANDBOOK

DAVE BURSKY

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

Dedication

I would like to thank the many friends that helped me to complete this book. Many thanks to Jeff Bierman and Robert Meehan for some of the photographs, and to Katherine Berger, Clare Bursky, and Hilary Mendelson for their typing and proofreading. I am also grateful for the encouragement of others too numerous to name.

Additionally, I would like to thank the many manufacturers that loaned various pieces of S-100 bus equipment to me for use in preparing the book. Some of the many companies include: AP Products, Advanced Micro Devices, Ball Brothers Research, Circuit Stik (now part of Bishop Graphics), Continental Specialties, E & L Instruments, EMM Semiconductor, Extensys Corp., Intel Corp., MITS (now part of Pertec Computer Corp.), Motorola, National Semiconductor, NEC Microcomputers, Oliver Audio Engineering, Seals Electronics, Shugart, Solid-State Music (now called SSM), Southwest Technical Products, Technical Design Labs (now called Xitan Corp.), Texas Instruments, Triple I Corp., Vector Electronics, and Vector Graphic Corp.

Library of Congress Cataloging in Publication Data

Bursky, Dave.
The S-100 bus handbook.

Includes index.
1. Microcomputers. 2. Microcomputers—Buses.
I. Title.
TK7888.3.B84 621.3819'58'3 79-26153
ISBN 0-8104-0897-X

1 2 3 4 5 6 7 8 9 PRINTING

80 81 82 83 84 85 86 87 88 YEAR

Preface

The material in this book has been put together from the many operating manuals and system descriptions published by the various manufacturers mentioned in this book. And, in many cases, the complete latest schematic revision of the particular board described has been included for handy reference. In total, this book is intended to serve as a quick, handy compendium of technical information about S-100 bus equipment. And, although some companies have changed names, been acquired by others, or closed their doors, many thousands of their boards are in the field, and this book will be one of the few places that data will be available.

The boards covered in this book span a wide range of functions, and although it would be impossible to cover every manufacturer's product in a single book, the equipment described here is representative of the many other products that are available. Companies such as MITS and IMSAI were the pioneers in the field of "personal" computing, which has now opened up to an extremely large assortment of low-cost completely assembled systems, some using the S-100 bus and others using proprietary bus structures.

However, another controversy covered in this book is that of just what is the S-100 bus. MITS introduced the original version of it on their Altair 8800 microcomputer system, revised it for the 8800b system and then developed an entirely new structure. The pin definitions picked by Imsai were very similar to those of MITS, but there were some differences—differences that make some of the boards that operate in one system incompatible on the other system. For future S-100 bus systems, many of the incompatibility problems should be eliminated as the "standard" for S-100 bus systems that has been developed by a committee of IEEE (Institute of Electrical and Electronic Engineers) becomes widely adopted. Included in Appendix D is a summary of the new standard, including a listing of all the pins and their definitions.

Not only will this book serve as a handy compendium of information about the boards, but it will serve as a simple guide to troubleshooting some of the basic simple system failures beyond "it doesn't work." Detailed troubleshooting information would require a book on each type of board, so the material included here will just help track down the problem to a defective board, system component, or program.

I would just like to add a word of thanks to all the microcomputer manufacturers that helped me put this material together and hope that for you, the user, it serves its intended purpose.

DAVE BURSKY

Contents

CHAPTER 1

Basic Introduction to Computers and Microprocessors

So you want to use a computer. But just buying one and reading the instruction and operating manuals will not get you very far, especially if you're not familiar with programming or electronics.

By itself, a computer is nothing more than a collection of electronic circuits arranged to process information that is fed into it. In addition to all the electronics, therefore, a computer requires some sort of language to communicate back and forth with the operator—you. Such a language consists of a stable of commands that can be combined in various ways to make the computer do almost anything—solve business problems, simulate speech, play music, play games, or even solve mathematical equations.

A Little Computer Prehistory

Ever since man started counting on his hands and toes, he has been trying to find easier and faster ways to do everything. The Chinese abacus, developed before 2000 B.C., was one of the first calculating machines (Fig. 1.1). It could add, subtract, multiply, and divide under the skillful control of an operator. And even today, if you were to go into the Chinese community of any sizable city you would probably find some shopkeepers still using the abacus to do their bookkeeping.

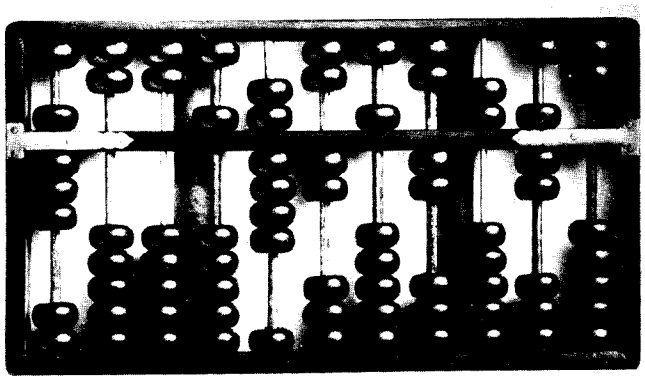


Fig. 1.1 The Chinese abacus, one of the first of man's calculating machines, is still in use today.

Early Western civilization, though, struggled along without such a handy device. Moreover, it had to make do with the numbering system devised by the Romans—what today we call Roman numerals:

$$I = 1, \quad V = 5, \quad X = 10,$$

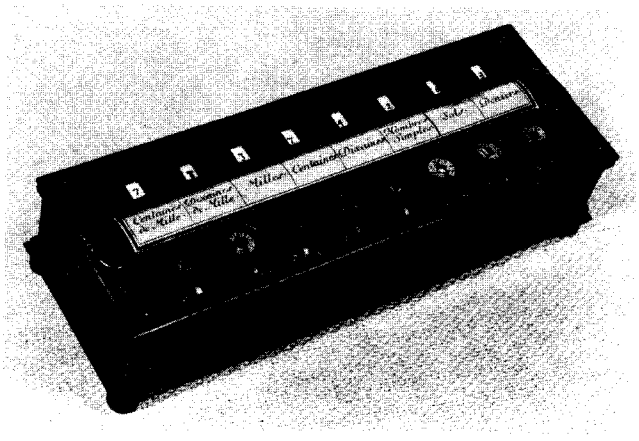
$$C = 100, \quad \text{and} \quad M = 1000$$

It wasn't until the thirteenth and fourteenth centuries that the decimal system of Arabic numerals we use today—0, 1, 2, 3, 4, 5, 6, 7, 8, and 9—came into widespread use. Except for the zero, the numbering system dates back to about the fifth century. However, the idea of a placeholder, the zero, was not developed until about the ninth or tenth century.

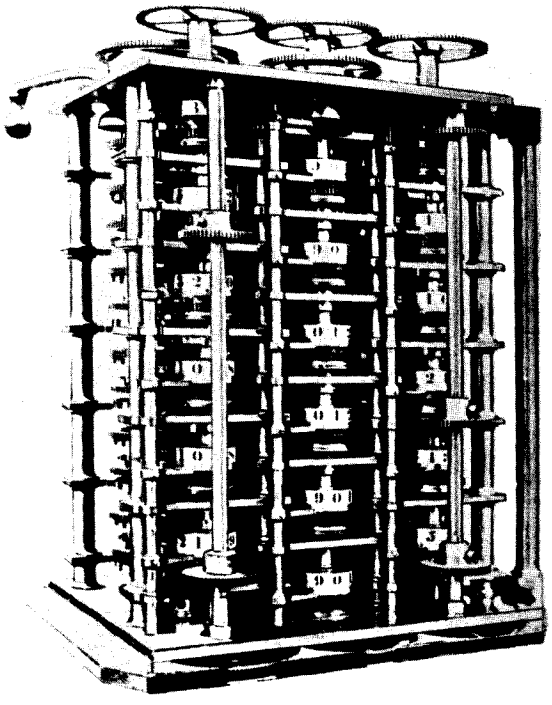
Today, the decimal numbering system is used throughout the world as a common mathematical tool. However, most computers don't operate with decimal numbers—they use one of the forms of binary number representations to perform their operations. We'll talk more about numbering systems and how they work in the next chapter.

It wasn't until the seventeenth century that modern computing devices began to take shape. Two of Europe's top philosopher-scientists—Blaise Pascal and Gottfried von Leibnitz—improved on some basic mathematical concepts in a way that made some crude calculators possible (crude, that is, by today's standards).

Pascal's contribution was to increase our understanding of the carry and borrow operations used in addition and subtraction. About the middle of the seventeenth century he developed an adding machine (Fig. 1.2a) that could perform all four basic functions—addition, subtraction, multiplication, and division—by means of notched wheels interconnected by gears. Each wheel had ten notches, and after every complete rotation of a lower wheel the next higher wheel would move ahead one notch. Multiplication consisted of nothing more than repeated additions, and division of nothing more than repeated subtractions. Most electric and gas meters used by the utilities today use the same principle to calculate power usage.



(A)



(B)

Fig. 1.2 The first crude calculator, "Pascal's Machine Arithmetique," was developed in the seventeenth century (a). During the nineteenth century, Babbage conceived a fully automatic calculating machine he called an analytical engine (b). It used punched cards to feed in the information.

Leibnitz improved on Pascal's machine by developing a way to do multiplication directly. The principle of his machine, called a stepped reckoner, was used in many electromechanical calculators until the 1960s. Of course, both Pascal's and Leibnitz's machines had to be manually operated; someone had to feed in the numbers and turn the crank for each operation.

About 200 years had to pass before Charles Babbage conceived of a fully automatic calculating machine—he called it the analytical engine (Fig. 1.2b). Unfortunately, his machine was too complex for the metal-working technology of the early 1800s, and it was never built. Babbage's concept of the analytical engine also fostered the idea of using punched cards to feed information (both instructions and data) into a machine for processing. Many computer installations still use Babbage's punched-card concept for entering programs and data.

While Babbage's idea remained sidelined, another mathematician, George Boole, developed a theory of logical algebra (what today engineers refer to as Boolean algebra) that has served as the basis for all modern computer theory. A lot of other developments had to take place, however, before modern computers could even be imagined.

Electricity was still the experimenter's parlor toy in the nineteenth century, and the theory of electricity had yet to be formulated. Once it had been, the great cataclysms of the twentieth century's two World Wars brought forth many advances in electrical machinery and computing devices. Not long after the end of World War II, several large computing machines were developed by researchers at Harvard and the University of Pennsylvania—the Mark I at Harvard, and ENIAC at the University of Pennsylvania.

By today's standards, ENIAC (only 30 years old) was a very primitive machine (Fig. 1.3). It used 18,000 vacuum tubes, weighed more than 30 tons, consumed 130,000 watts, and performed only 5000 operations per second. Modern programmable pocket calculators that can store instructions on magnetic cards have as much capability as did ENIAC.

While scientists struggled to keep ENIAC running, researchers at Bell Labs refined the principles of semiconducting materials. By the early 1950s semiconductors—transistors—started to replace vacuum tubes in many applications. They not only offered almost unlimited life compared to that of vacuum tubes, but were only a fraction the size.

Fig. 1.3 World War II spurred scientists at the University of Pennsylvania to develop the first digital computer—ENIAC. (*Courtesy* Sperry Univac)

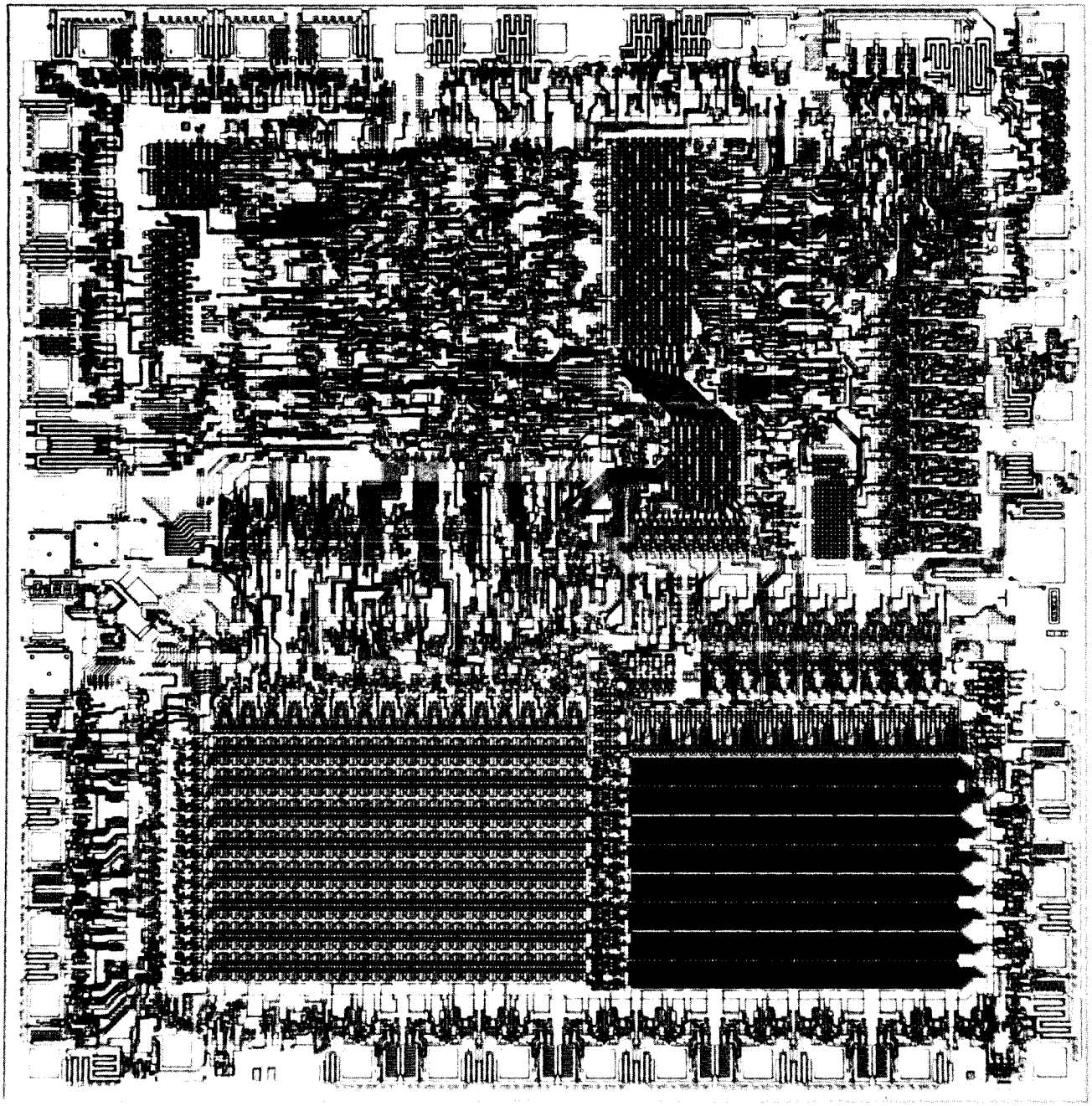


Fig. 1.4 Containing all the major logic sections that make up a computer, the 8048 microcomputer developed by Intel is built into a chip of silicon less than a quarter of an inch on a side. (*Courtesy* Intel)

Use of these semiconductor materials—germanium and silicon—to build miniaturized systems allowed designers to build machines that required a fraction of the power, were only a fraction the size, and only a small percentage of the weight of ENIAC. The transistor indeed marked the turning point of modern computer design.

But even since the invention of the transistor there have been major advances. In 1958, several companies managed to combine several transistors and some other components within a single tiny chip of silicon. These all-solid-state circuits, now referred to as monolithic integrated circuits, have reduced the number of actual components needed to build a computer system to a mere handful. As a matter of fact, today's technology has already made possible the complete computer on a single chip of silicon only 0.25 in. on a side (Fig. 1.4). The chip contains about 40,000 transistors and requires

data and instructions. Boolean algebra can thus be used to express many of the processes of computer systems since it is a form of logic reasoning involving two states: truth and untruth.

The ability of a computer to perform logic and math operations rapidly is its only strength. In logic, all statements must be either true or false; there are no intermediate conditions such as "maybe."

The binary numbering system provides us with just the right symbols to work with, and we will adopt them as a standard for the rest of the book. For a true logic statement, the binary value of 1 will be assigned, and for an untrue statement, a value of 0. Thus, a statement can equal 1 or 0 but it cannot have any other value (that is, it must be either true or false). Electronic circuits adapt easily to this system because only two voltages are needed and they can be as simple as ground (nothing, or 0 V) and some level V (which can be positive or negative). The true or false representation can be turned into an on and off equivalent for electronic circuits. Thus, when the circuit is turned on the statement could be true, and when the circuit is turned off the statement could be false. Depending on the type of logic circuits used, the positive level or ground could just as easily represent the true statement or the false statement.

Either statement standing by itself is of little logical interest. A statement has a truth value of 1 or 0, and that's all. However, when several statements are grouped together, other logical inferences can be developed. The three basic logic operators—AND, OR, and NOT—help connect statements so that conclusions can be drawn.

Using Logic Operators

The AND operator can be used to connect any number of logic statements, all of which must be true for the conclusion to be true. For instance, suppose a friend tells you that on Saturday you will find him at the park if the weather is good AND he has the day off from work. This remark can be written as a logic equation by using symbols: Let A represent the statement that the weather is good, B the statement that he has the day off from work, and C that you will find him at the park. Therefore, if A AND B then C.

Writing out an entire logic operation is often not necessary; a shorthand notation can be used. Sometimes the multiply dot · in the middle of a line is used to represent the AND function, and other times the dot is omitted and the individual letters are placed next to each other:

if A · B then C; if AB then C

Each of the two statements A and B can be either true or false. If true, we assign the statement a value of 1, if false, a value of 0. The statements can be tabulated to show all four possible combinations and the possible outcomes:

Statement A	Statement B	Outcome (A · B)
0	0	0
0	1	0
1	0	0
1	1	1

This type of listing is called a truth table since it shows every possible combination of the two statements.

The OR operator can also be used to connect any number of logic statements. However, unlike operations with the AND operator, only one statement of all those connected need be true for the outcome to be true. Let's use the same example we used for the AND operator but modify it slightly: A friend tells you that on Saturday you will find him at the park if the weather is good OR he has the day off from work.

We can write this statement in the form of a logic equation if we use the plus symbol + to represent the logic OR operation. Thus, if A + B then C. Whenever A or B is true, or they are both true, the outcome is true. Again, this can be shown in truth-table form very simply:

Statement A	Statement B	Outcome (A + B)
0	0	0
0	1	1
1	0	1
1	1	1

The only time your friend won't be at the park will be if the weather is not good AND he doesn't have the day off from work.

The last basic logic operator, the NOT function, can be used to invert or complement a logic statement. It is usually symbolized in shorthand by an overscore of the logic statement or statements you want to invert. Let's see how the logic statements used for the AND and OR operators can be rewritten.

The statement A refers to the fact that "the weather is nice"; therefore, $\overline{A}$ represents the statement that "the weather is not nice." Similarly, B represents the statement that "he has the day off from work," and $\overline{B}$ means that "he does not have the day off from work." Since A is represented by the binary 1, $\overline{A}$ would then be binary 0 ($A = 1, \overline{A} = 0$). Common pronunciations of $\overline{A}$ are "not A" and "A bar," although you may run across others. The truth table for the NOT function is very simple since it operates on one item at a time:

A	$\overline{A}$
0	1
1	0

Each of the logic operators has a physical equivalent that you might find easy to relate to. The simplest example is probably the common water faucet. If you take a look at your kitchen sink, you'll probably see something like the arrangement shown in Fig. 2.1. On top of the sink are two faucets (call them A and C) and a common spout. Under the sink you'll find two valves (call

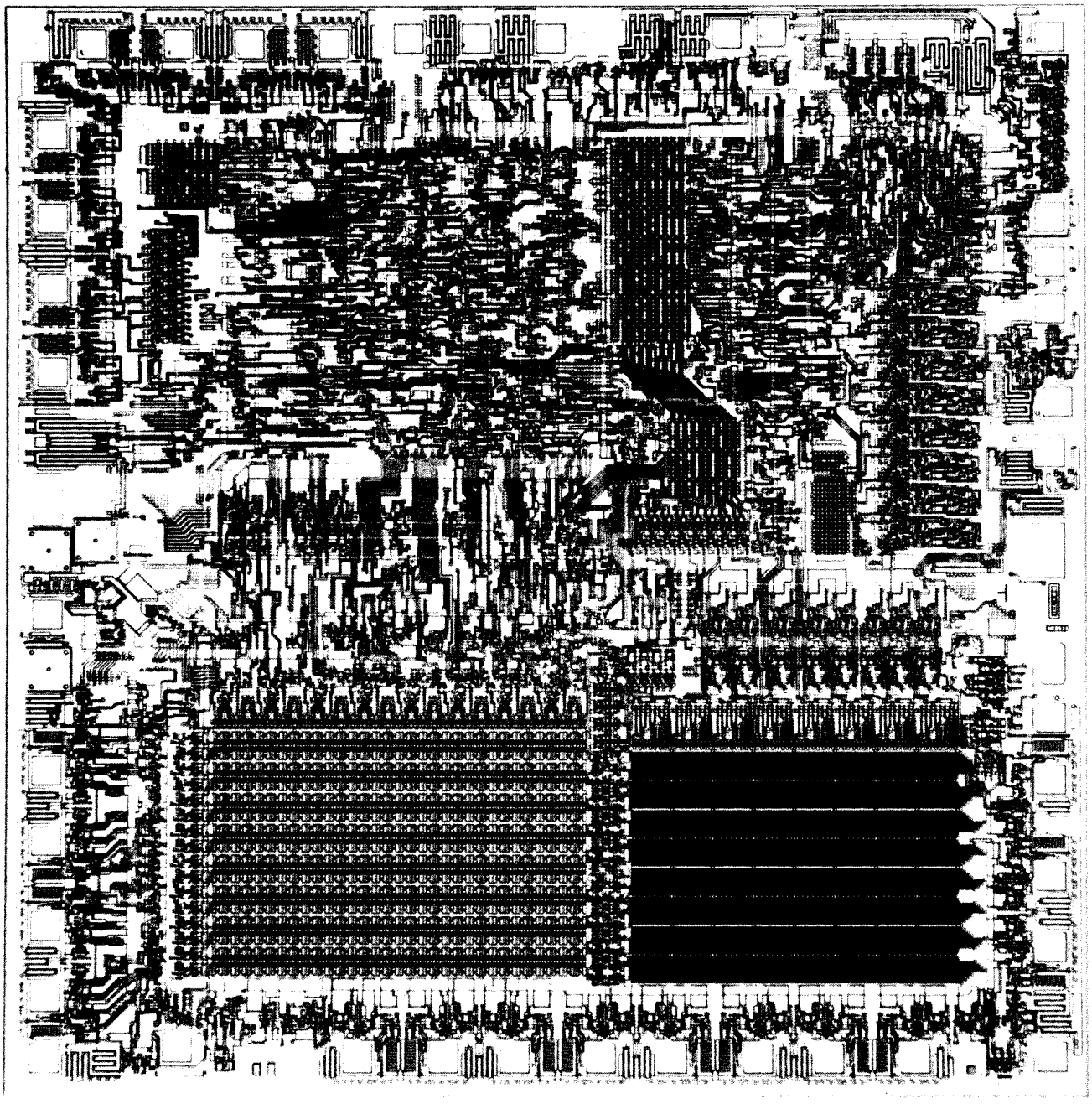


Fig. 1.4 Containing all the major logic sections that make up a computer, the 8048 microcomputer developed by Intel is built into a chip of silicon less than a quarter of an inch on a side. (*Courtesy* Intel)

Use of these semiconductor materials—germanium and silicon—to build miniaturized systems allowed designers to build machines that required a fraction of the power, were only a fraction the size, and only a small percentage of the weight of ENIAC. The transistor indeed marked the turning point of modern computer design.

But even since the invention of the transistor there have been major advances. In 1958, several companies managed to combine several transistors and some other components within a single tiny chip of silicon. These all-solid-state circuits, now referred to as monolithic integrated circuits, have reduced the number of actual components needed to build a computer system to a mere handful. As a matter of fact, today's technology has already made possible the complete computer on a single chip of silicon only 0.25 in. on a side (Fig. 1.4). The chip contains about 40,000 transistors and requires

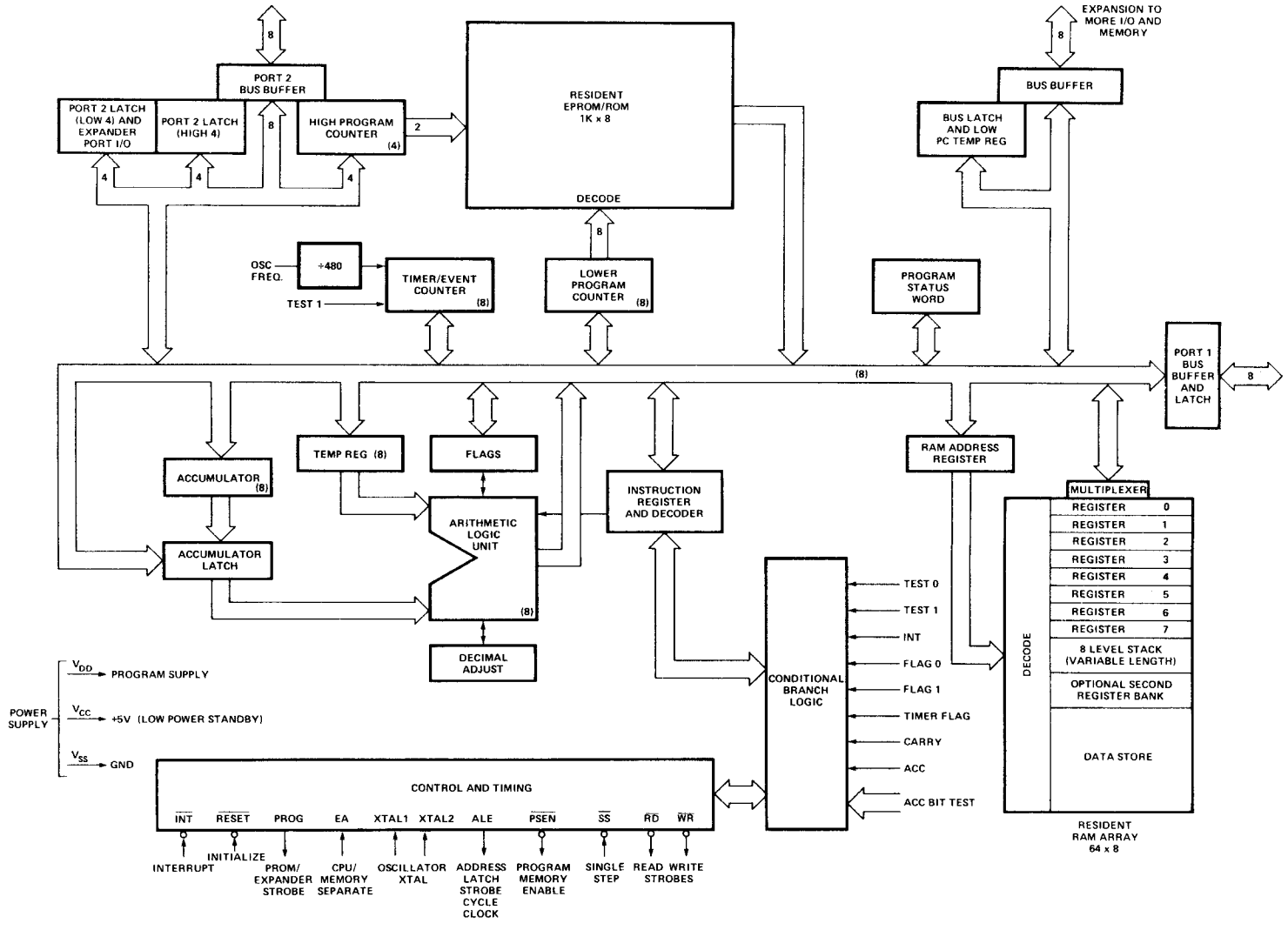


Fig. 1.4 (cont'd) Containing all the major logic sections that make up a computer, the 8048 microcomputer developed by Intel is built into a chip of silicon less than a quarter of an inch on a side. (*Courtesy* Intel)

less power than even one of the vacuum tubes used in ENIAC. And modern technology is striving to improve on this circuit—another two years will see up to 100,000 transistors on a single chip of silicon.

What Is a Computer?

Basically, any device can be called a computer that, once given instructions and information to process, proceeds to carry out those instructions. Human beings are forms of highly complex computers, too. Each of us can accept many types of data through our senses, and, based on our earlier experiences and learning (programming), can react to solve the problem posed. However, there is one major difference between mechanical and human computers: Human "computers" are capable of taking original action without instructions; mechanical computers can only do what they are instructed to do and cannot modify what they are doing without following a preordained procedure.

Modern computers can be split into two basic families: analog and digital. The analog computer works with signals that are continuous. By continuous, we mean signals that can take on an infinite number of values between two points, as shown in Fig. 1.5. Analog computers must know the value of these signals, or at least how a signal compares to the other signals also being used as information.

Conversely, digital computers work on discrete, discontinuous numerical values, such as those shown in Fig. 1.6. Currency offers a good example of such "discrete" values. The amount of money you pay for a candy bar is a digital quantity—say, between 0 and $1. Since only 100 discrete amounts (cents) exist between these limits, the change from one value to the next is discontinuous. But, since digital computers don't care

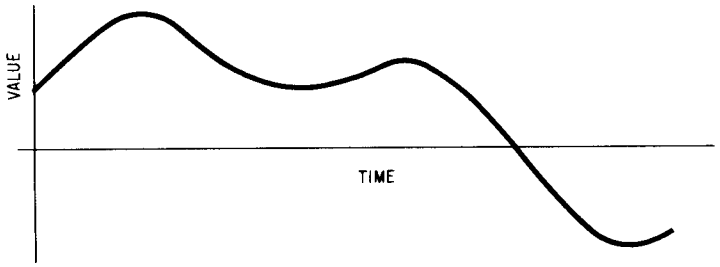


Fig 1.5 Analog signals can take on an infinite number of values between two points on the graph.

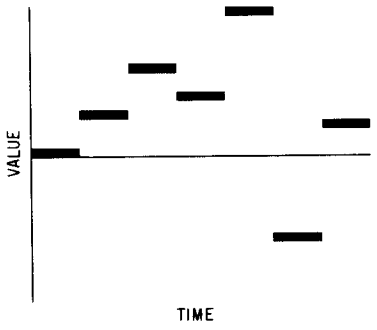


Fig 1.6 Digital signals are represented by discrete, discontinuous numerical values that can appear as steps on a graph.

whether one value is larger than another, a simple way to represent an amount had to be found. More about computer numbering systems and information representation will be presented in the next chapter.

Since this book deals with the use and circuit design of digital computer systems, the rest of the discussions pertain to digital computer systems only. (If you're interested in finding out more about analog computers, see the references in Appendix A.) Digital machines perform arithmetic operations and can make logic decisions based on instructions fed into the machine. Therefore, basic computer mathematics includes the operations of addition, subtraction, multiplication, and division, as well as logic operations such as AND, OR, NAND, NOR, Exclusive-OR (XOR), etc. (More about these functions later in the book.)

No two digital computers are identical, especially if you look inside. You can give them the same commands and the same data and, in all probability, get the same answers. But inside, each machine handles the information in a different way. The internal organization of a computer is often referred to as the machine's "architecture." All computers can, at least for the sake of analysis, be broken into five basic building blocks: an input section, a memory section, a control section, an arithmetic and logic section, and an output section. A typical interconnection diagram of these building blocks is shown in Fig. 1.7.

The input part of the computer is often some manually operated device similar to a typewriter, but it could just as easily be a magnetic tape reader or a

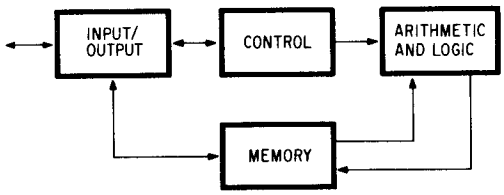


Fig. 1.7 Any computer system can be broken down into several basic building blocks.

punched tape reader or any of the many other types of input devices. Part of the input system's job is to translate the information prepared by the operator into a form the computer can digest.

Once information has been converted into digital signals, it is usually fed into the memory section of the computer, where it is stored until needed. The memory section also holds the instructions and often the basic operating procedures of the computer itself. Computer memory devices include such units as magnetic tapes, magnetic discs (mass storage devices), ferrite cores, and semiconductor circuits called flip-flops, RAMs, or ROMs.

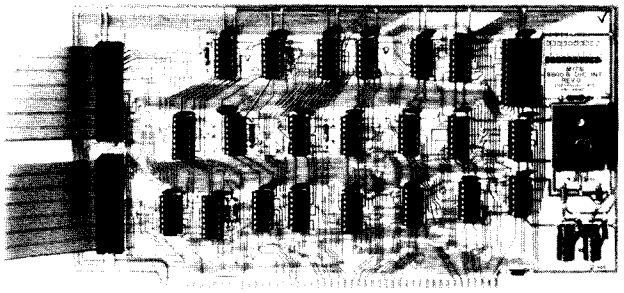
To coordinate all the operations of the computer, the control section selects information and instructions from a storage location in memory in the proper sequence and lets it flow to the proper section for processing. The control section is the decision-making element of any computer. Inside an actual machine, however, the control circuits are actually spread out through the entire machine and are not grouped together as shown in Fig. 1.7.

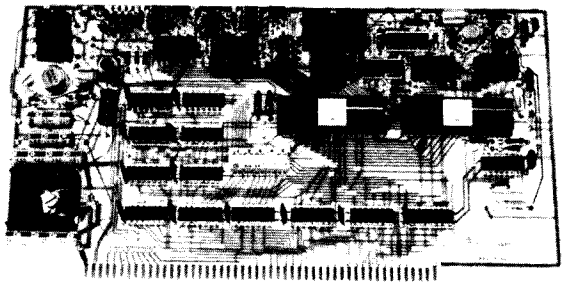
The actual processing is done by the arithmetic and logic section of the computer. In this section, digital information can be manipulated, analyzed, and rearranged under the direction of the control unit.

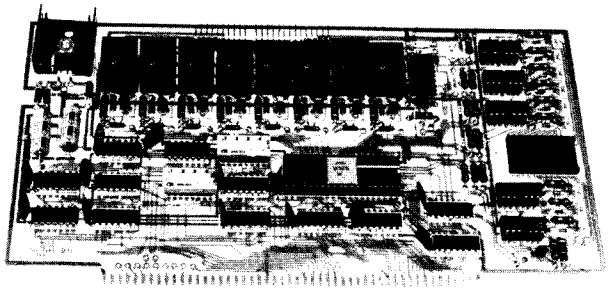
Once the information has been processed, it is often fed back into the memory before you see the answer. Under the direction of the control section, the answers are delivered to you by means of some output device—in some cases, the same machine that you entered the instructions into, and, in other cases, possibly a printer, a tv screen, or a magnetic-tape recorder.

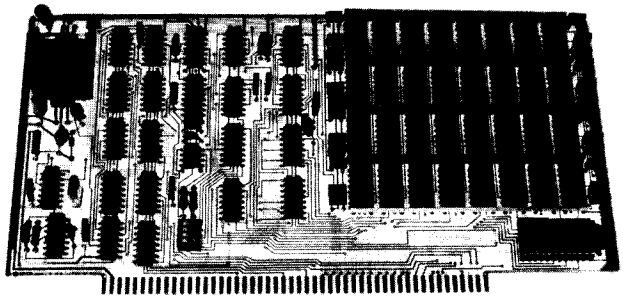
Of course, there are several parts of the computer that we've skipped over for the moment—the power supply, the front panel, and the cabinet. However, these sections are much like the "dressing" on a cake. They must be there, but all you have to know is that they're there and they do their work. Some operations of the front control panel will be discussed in a later chapter.

Often, the various sections of the computer are built on separate circuit cards and then the cards are connected together. The cards described in the following chapters are all identical in physical size—5 in. × 10 in.—and have an edge connector on them that permits up to 100 connections to the circuitry on the card (Fig. 1.8). In a computer, many of these cards are interconnected by a wiring scheme called a bus—a common









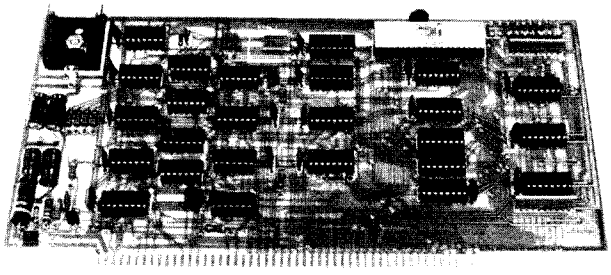


Fig. 1.8 Some typical S-100 bus compatible circuit cards. (*Courtesy* Pertec)

group of wires over which signals from all the boards can be transmitted.

The computers and cards discussed in this book all use the same bus structure, commonly referred to as the S-100 bus. This bus was originated by MITS, when the company introduced the Altair 8800—the first personal computer—back in 1975. Since then over 50 companies have adopted the same interconnect bus and offer a wide variety of computer cards that connect to the bus. The computers discussed in this book, the Altair 8800b from MITS and the Imsai 8080 from Imsai Manufacturing (Fig. 1.9), use the S-100 bus and can accept all of the bus-compatible cards made by other companies.

These computer systems are all based on electronic circuits that have been in existence since 1970, and some only in the last few years. The heart of these computer systems is the central processor unit (CPU), which contains the arithmetic and logic unit. Modern technology has been able to shrink the circuitry needed to build a CPU so that all the components needed fit on a single chip of silicon a mere quarter of an inch on a side—the microprocessor (Fig. 1.10). Of course, for it to work, power and special signals must be supplied and circuits to make the output signals stronger (buffers) must be used.

What Is a Microprocessor?

But a microprocessor is not a computer. It is just the processing section. Along with the basic microprocessor, many memory circuits, input/output circuits, and other specialized components are needed to make a full computer. And, just as with the large computers, all microprocessors are not the same. There are about 30 different types, made by about as many different companies, and each has its own architecture, instruction set, power supply requirements, and other peculiarities.

Both the Altair 8800b and the Imsai 8080 were designed to operate with one specific microprocessor as the central processor—the 8080A made by Intel Corp., and now available from about half a dozen vendors. However, since the creation of the 8080A in 1973, newer microprocessors, the Z-80 from Zilog and the 8085 from Intel, offer compatibility and improved performance.

Fig. 1.9 The heart of any computer system, the computer itself, typically comes in a large case with many front panel switches and indicators. The Imsai 8080 (left) and the Pertec Altair 8800b (right) were two of the first personal computer systems. (*Courtesy* Pertec and Imsai)

The microprocessor is just what its name implies—a miniature processor. Buried within the silicon chip are all the basic elements of a computer—the control section, the processing section, and some memory (usually referred to as temporary registers). Specifically, the 8080A has an internal structure as shown in Fig. 1.11. As you can see, there are quite a few subsections squeezed into that quarter-inch chip of silicon. To use the computer system, it's not necessary to know how the microprocessor works or what each of its internal subsections does. Also, for the most part, you must have a fairly good technical background in computers before you can even understand what the different sections do. For those of you with a reasonable background in computers, the next few paragraphs will try to summarize the characteristics of the 8080A. For those of you with no background, some additional reading from selections in Appendix A might be warranted if you want to understand the inner workings of computers.

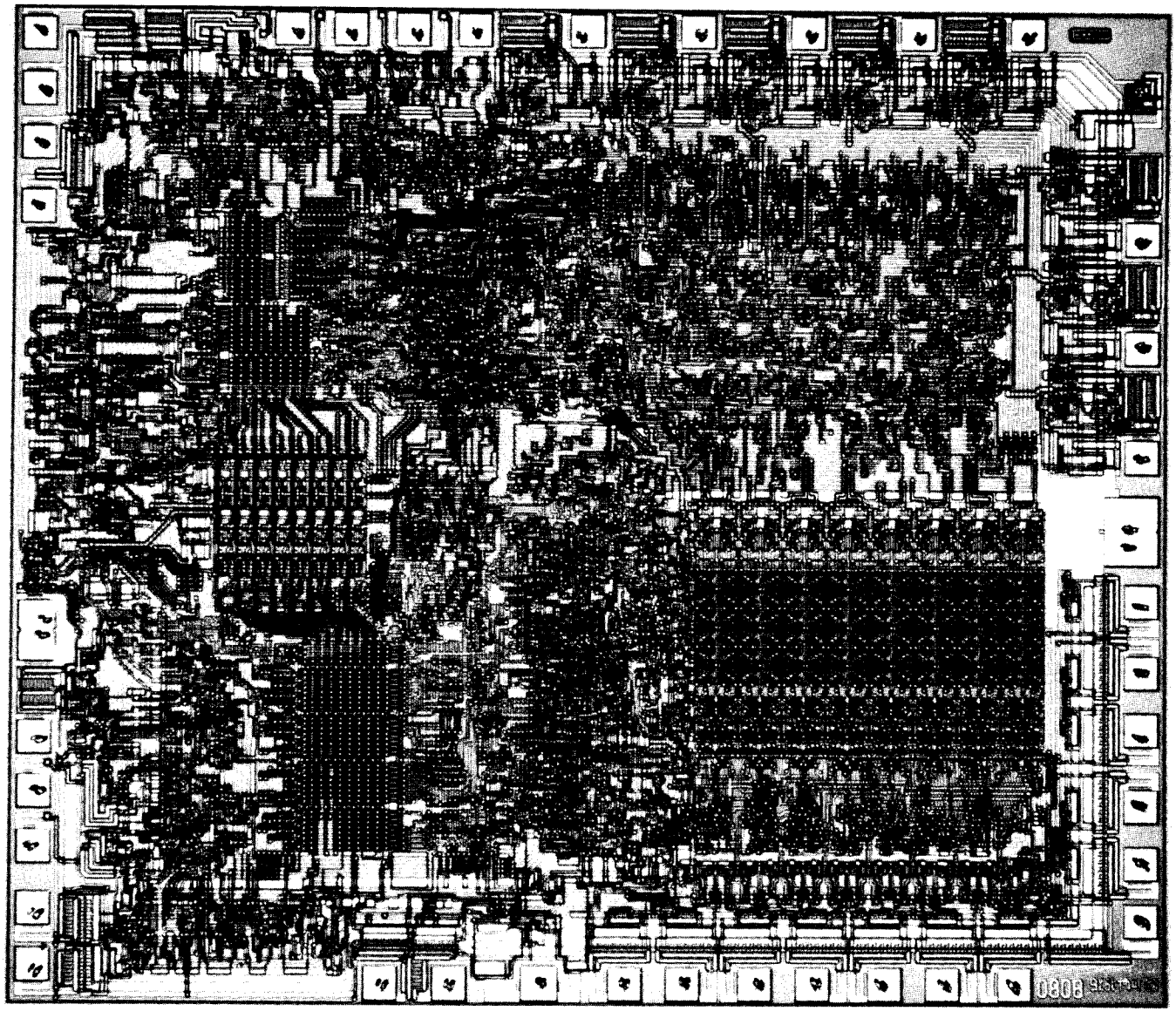
The 8080A microprocessor is a circuit designed to process information in digital form. It operates on digital information in groups of eight binary digits at a time (each binary digit is referred to as a bit, and a group of eight bits is referred to as a byte), and can perform many different types of mathematical and logic operations on the digital information. There are four basic internal sections of the 8080A—the register array and address logic, the arithmetic and logic unit, the instruction register and control section, and the bidirectional, three-state data bus interface.

The register section consists of an array of memory cells organized so that six 16-bit information blocks can be stored. Two of the 16-bit registers are assigned specific purposes—they serve as the program counter (PC) and the stack pointer (SP). The purpose of the program counter is to keep track of the location of the current program instruction in the computer's memory. The stack pointer maintains the location of a section of memory called a stack, which is used to hold memory addresses when a computer program calls a subroutine. (More about programming in a later chapter.)

Three more of the 16-bit registers are actually broken into six 8-bit registers that can be operated on individually or in pairs. The 8-bit registers are referred to as the B, C, D, E, H, and L registers and can be accessed in pairs as BC, DE, and HL. All of the registers discussed so far can be manipulated by instructions. One other register, called the temporary register, also stores up to 16 bits, but it cannot be controlled by instructions. This register, referred to as the W, Z register, is used only for the internal execution of instructions.

Bytes can be transferred between any of the registers inside the processor by the appropriate instruction. Double-byte transfers can also be performed between the register pairs and the SP and PC registers and the address logic. The address logic in turn feeds a 16-bit binary number to the memory array external to the processor, thus permitting the microprocessor to access any one of 65,536 memory locations.

The arithmetic and logic unit (ALU) within the 8080A performs the actual manipulating of the computer data. It performs the arithmetic, logic, and rotate operations dictated by the instructions. In addition to the circuits necessary to perform the operations, there are several registers used to hold intermediate information—an 8-bit register called the accumulator, another 8-bit register called the temporary accumulator, an 8-bit register called the temporary register, and a 5-bit register called the flag register that is used to hold indicators from operations that took place in the arithmetic and logic unit. The indicators are ZERO (shows when an operation leaves a zero result in the accumulator), CARRY (shows when an operation generates a carry from the most-significant bit position), SIGN (shows when an operation leaves a negative result in the accumulator), PARITY (shows whether the sum of all the bits left in the accumulator is odd or even), and AUXILIARY CARRY (shows when there is a carry



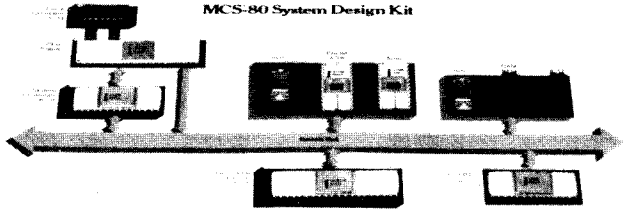


Fig. 1.10 Making the home computer system possible, the microprocessor offers the computing power of large computers in an extremely small package. Housed in a 40-pin DIP, the 200 mil square chip of silicon called the 8080 contains over 10,000 transistors. (*Courtesy* Intel)

generated from the fourth to the fifth bits in the accumulator).

The accumulator can be loaded from the ALU and the internal bus, and can transfer data to the temporary accumulator and the internal bus. A special instruction permits the contents of the accumulator and the AUXILIARY CARRY flag to be tested for decimal correction when the processor is handling decimal numbers in binary form. The DAA instruction permits the results to be corrected back to the decimal format.

The instruction register and control section of the microprocessor holds the current instruction and controls all the internal operations of the processor for the execution of that instruction. To properly synchronize all the internal operations, special timing signals called clocks are fed into the microprocessor.

Information flows in and out of the microprocessor over an 8-bit path called the data bus. Digital information can flow in either direction, depending on the operation the processor is performing. The bus is referred to as a three-state bus because in addition to the normal HIGH and LOW logic states possible, the bus can be

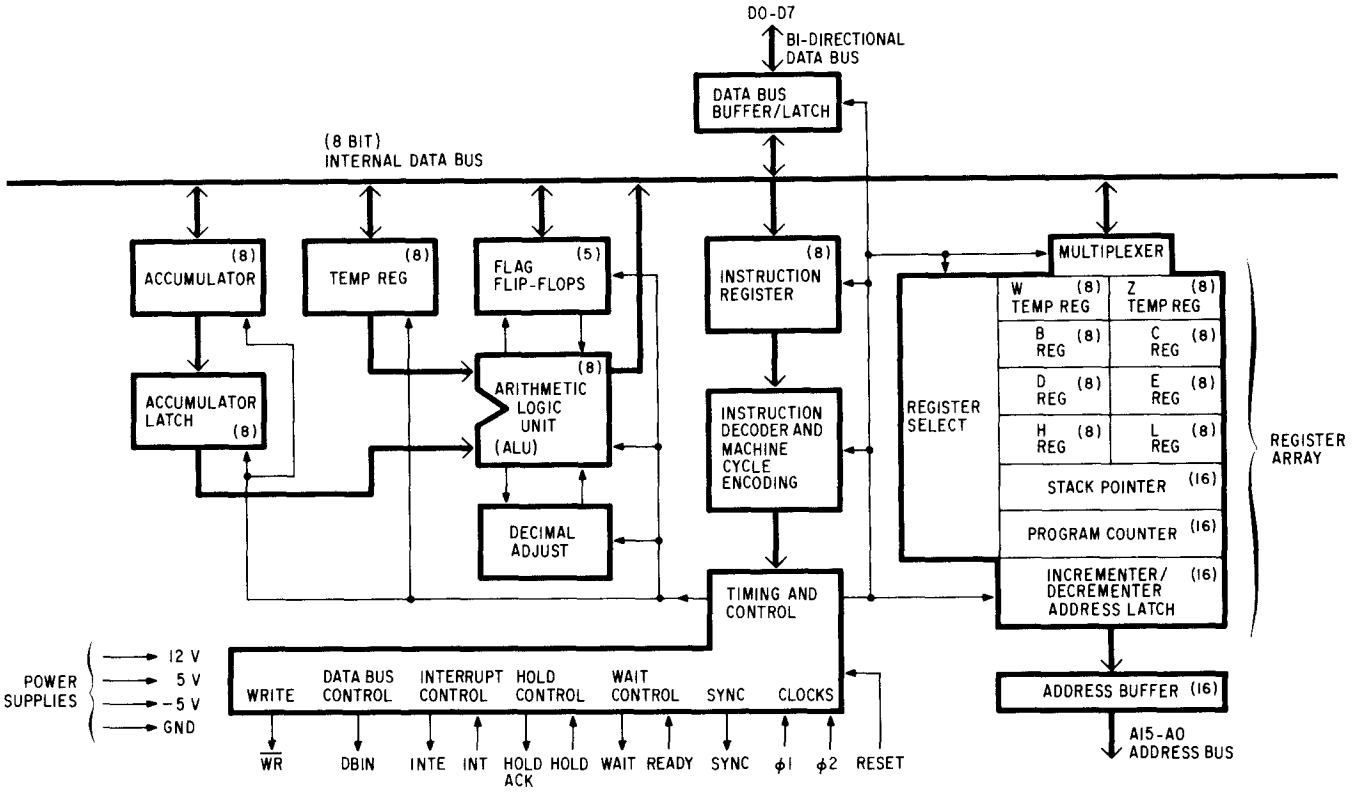


Fig. 1.11 The architecture of the 8080A microprocessor is similar to that of most computer central processors. An arithmetic and logic unit does all the processing and it is controlled by instructions and data fed into it.

made to go into a state where it effectively uses no power and has no effect on the other circuits it is connected to. This third state is often referred to as a high-impedance state. When the bus is not in use it is usually placed in this third state just to minimize power consumption.

The Microprocessor Operates in Cycles

Each time the microprocessor executes an instruction, it completes an *instruction cycle*. The duration of an instruction cycle includes the time required to pull an instruction from memory and execute it. During the pull, or fetch, part of the cycle, the selected instruction (one, two, or three bytes long) is extracted from the memory and deposited in the processor's instruction register. Then, during the execution phase of the cycle, the instruction is decoded and translated into specific actions by the control logic.

Every instruction cycle consists of one, two, three, four, or five machine cycles. The fetch portion of an instruction cycle requires one machine cycle every time a byte must be fetched from memory. The length of the execution portion of the cycle depends on the instruction being executed—some instructions may not require any machine cycles beyond those of the fetch operation, others may require additional cycles.

Each machine cycle further consists of three, four, or five states, where a state is the smallest unit of processing activity and is defined as the interval between two successive positive-going transitions of the phase-one clock signal. (The 8080A has all of its timing signals supplied by a two-phase clock generator that delivers two signals to the phase-one and phase-two clock input terminals, as shown in Fig. 1.12.)

Every instruction cycle has at least one memory-reference operation during which the instruction is fetched. An instruction cycle must always have a fetch, even if the execution of the instruction requires no further references to memory. The first machine cycle in every instruction cycle is thus a fetch operation. Beyond that, there are no restrictions; subsequent

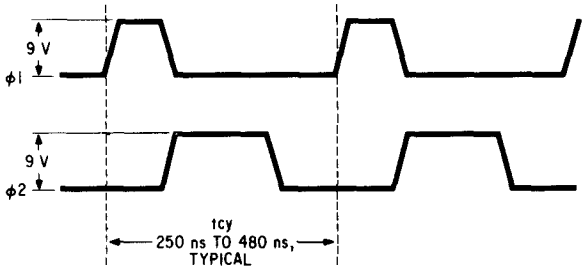


Fig. 1.12 Performing its instructions with timing derived from a two-phase clock signal, the 8080A operates at frequencies of up to 4 MHz.

machine cycles can perform any type of operation. While no one instruction cycle will contain more than five machine cycles, there are 10 types of machine cycles that can possibly occur during an instruction cycle:

1. Instruction fetch
2. Memory read
3. Memory write
4. Stack read
5. Stack write
6. Input
7. Output
8. Interrupt
9. Halt
10. Halt · Interrupt

The processor identifies the machine cycle in progress by transmitting an 8-bit status word during the first state of every machine cycle. Updated status information is presented on the 8080A's data lines during the SYNC interval in the timing sequence. These data are usually saved in a register and can be used to provide control signals for external circuitry.

Before delving into computer operations any further, let's backtrack and go through a quick review of computer mathematics and logic, as well as a simple summary of the basic electronics needed to understand the various components used in the computers.

CHAPTER 2

Binary Mathematics and Boolean Algebra

No matter which computer is used, or what instructions are given, the same number system performs all the mathematical operations and the same logic gates perform the Boolean operations. Computers use the binary numbering system to perform their operations since for electronic circuits the two states of the binary system—1 (HIGH) and 0 (LOW)—provide the simple equivalents to the ON and OFF states used in electronic switching systems. But no matter which numbering system is used, the operations performed—addition, subtraction, multiplication, and division—are all done in the same way; only the numbers change.

Computer Math: A Quick Review

Mathematics with the decimal numbering system has become so commonplace that no one really has to sit down and think about how a problem has to be done. If we wanted to solve some simple problems such as these:

$$\begin{aligned} 48 - 12 &= ? \\ 13 + 26 &= ? \\ 4 \times 4 &= ? \\ 18 \div 3 &= ? \end{aligned}$$

we wouldn't even slow down to give the answers; we would just rattle off 36, 39, 16, and 6.

But if we switch numbering systems to one we aren't familiar with, such as the binary numbering system used by all computers, some of us would be hard pressed to solve the same problems:

$$\begin{aligned} 110000 - 1100 &= ? \\ 1101 + 11010 &= ? \\ 100 \times 100 &= ? \\ 10010 \div 11 &= ? \end{aligned}$$

The answers are 100100, 100111, 10000, and 110. All the same mathematical rules apply to any numbering system, but the notation can confuse you. Let's try to clarify all numbering systems by first looking at the one we're most familiar with—the decimal system.

When we write down a number and then say it aloud, we begin to get the organizational picture of our numbering system. For instance, the number 3165 is pronounced three-thousand, one-hundred sixty-five. Now, if we break it down into the positional notation represented by each number spoken aloud, we get

$$\begin{aligned} 3 \times 1000 &= 3000 \\ 1 \times 100 &= 100 \\ 6 \times 10 &= 60 \\ 5 \times 1 &= 5 \\ &\overline{\quad\quad} \\ &3165 \end{aligned}$$

As written, the center column of the breakdown is nothing more than a listing of the powers of 10, similar to the breakdown shown in Table 2.1. Each column of a decimal number represents a power of 10, and the highest number that can appear in each column is a nine. The name of this numbering system stems from the Latin *decema* meaning 10, since there are a total of 10 symbols used to represent all numbers.

Table 2.1 Positive Powers of 10

10^0 =	1 = 1
10^1 =	10 = 10
10^2 =	100 = 10 × 10
10^3 =	1000 = 10 × 10 × 10
10^4 =	10,000 = 10 × 10 × 10 × 10
10^5 =	100,000 = 10 × 10 × 10 × 10 × 10
10^6 =	1,000,000 = 10 × 10 × 10 × 10 × 10 × 10

Since each column is represented by a power of 10, we can say that the columns are ordered in ascending or descending sequence, depending on which direction we read from. Normally, a number is read with the largest power of 10 first, so that the number follows a descending sequence. Another name for the highest order column is the "most-significant digit," and for the lowest order column the "least-significant digit." The total number of columns defines the number of digits.

We often use a decimal point in writing numbers that are a fraction of a whole number. To simplify the use of the decimal point, we use negative powers of 10 to represent numbers to its right, as shown in Table 2.2. Thus,

Table 2.2 Negative Powers of 10

$10^{-1} = 0.1$	$= 1/10$
$10^{-2} = 0.01$	$= 1/100$
$10^{-3} = 0.001$	$= 1/1000$
$10^{-4} = 0.0001$	$= 1/10{,}000$
$10^{-5} = 0.00001$	$= 1/100{,}000$
$10^{-6} = 0.000001$	$= 1/1{,}000{,}000$

a number such as 15.328 would be represented by

$$\begin{aligned} 1 \times 10 &= 10 &&= 10 \\ 5 \times 10^{0} &= 5 \times 1 &&= 5 \\ 3 \times 10^{-1} &= 3/10 &&= 0.3 \\ 2 \times 10^{-2} &= 2/(10 \times 10) &&= 0.02 \\ 8 \times 10^{-3} &= 8/(10 \times 10 \times 10) &&= 0.008 \\ & & & 15.328 \end{aligned}$$

The 10 different symbols used for the decimal system form the base, or radix, of the system (base 10). Any number, however, can be used as the base. Three of the other most common numbering systems in use are the octal, the hexadecimal, and, of course, the binary, where the respective bases are 8, 16, and 2.

Nondecimal Numbering Systems

In a numbering system based on eight symbols (the octal), we can count from 0 to 7 before running out of numbers. If we count higher, we must follow the same procedure used for decimal counting. After we fill up the first column, a placeholder (zero) must be inserted and a carry placed in the next higher column. The following octal count sequence, with the decimal equivalent shown just below it, illustrates this very clearly:

0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16

Of course, numbers written in octal do not have the same value as similar numbers written in decimal notation. In octal, any number filling a column represents a power of eight instead of 10. For instance, the number 1762_8 can be broken into its parts to convert it back into decimal notation.

$$\begin{aligned} 1 \times 8^{3} &= 1 \times 8 \times 8 \times 8 &&= 512 \\ 7 \times 8^{2} &= 7 \times 8 \times 8 &&= 448 \\ 6 \times 8^{1} &= 6 \times 8 &&= 48 \\ 2 \times 8^{0} &= 2 \times 1 &&= 2 \\ & & & 1010 \end{aligned}$$

A base that is larger than 10 can be used, but several new symbols will have to be added. Our alphabet fortunately provides a ready source of symbols. The letters A, B, C, D, E, and F, for instance, have been used to represent the numbers 10, 11, 12, 13, 14, and 15, respectively, in the hexadecimal numbering system. A comparison between the four most popular codes is given in Table 2.3.

Table 2.3 Comparison of Binary, Octal, Decimal, and Hexadecimal Codes

Binary	Octal	Decimal	Hexadecimal
0000	00	00	0
0001	01	01	1
0010	02	02	2
0011	03	03	3
0100	04	04	4
0101	05	05	5
0110	06	06	6
0111	07	07	7
1000	10	08	8
1001	11	09	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

Base 16 numbers also follow the same mathematical guidelines, as you can see from the following hexadecimal-to-decimal conversion of the number B13E:

$$\begin{aligned} B = 11 \times 16^{3} &= 11 \times 16 \times 16 \times 16 &&= 45{,}056 \\ 1 = 1 \times 16^{2} &= 1 \times 16 \times 16 &&= 256 \\ 3 = 3 \times 16^{1} &= 3 \times 16 &&= 48 \\ E = 14 \times 16^{0} &= 14 \times 1 &&= 14 \\ & & & 45{,}374 \end{aligned}$$

The binary number system is perhaps the simplest of systems to use and to understand. There are only two symbols in the binary system—0 and 1—but all the same rules again apply. A typical binary counting sequence, with the decimal equivalents shown on the line below, would be

0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Each *bi*nary dig*it*, or *bit*, represents a power of two, so that any number can be represented by adding bits together. As a typical example, let's break down the number 1100111 and convert it back into decimal notation:

$$\begin{aligned} 1 = 1 \times 2^{6} &= 1 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 &&= 64 \\ 1 = 1 \times 2^{5} &= 1 \times 2 \times 2 \times 2 \times 2 \times 2 &&= 32 \\ 0 = 0 \times 2^{4} &= 0 \times 2 \times 2 \times 2 \times 2 &&= 0 \\ 0 = 0 \times 2^{3} &= 0 \times 2 \times 2 \times 2 &&= 0 \\ 1 = 1 \times 2^{2} &= 1 \times 2 \times 2 &&= 4 \\ 1 = 1 \times 2^{1} &= 1 \times 2 &&= 2 \\ 1 = 1 \times 2^{0} &= 1 \times 1 &&= 1 \\ & & & 103 \end{aligned}$$

Converting a binary number into decimal notation is easy enough, and going the other way is also simple. The quickest conversion method is a simple process of dividing and checking the remainder. Let's look at an actual example, the conversion of 241 into binary:

Number		Divisor		Result		Remainder	Comments
241	÷	2	=	120	+	1	if remainder = 1, 2^0 is present
120	÷	2	=	60	+	0	if remainder = 0, 2^1 is not present
60	÷	2	=	30	+	0	if remainder = 0, 2^2 is not present
30	÷	2	=	15	+	0	if remainder = 0, 2^3 is not present
15	÷	2	=	7	+	1	if remainder = 1, 2^4 is present
7	÷	2	=	3	+	1	if remainder = 1, 2^5 is present
3	÷	2	=	1	+	1	if remainder = 1, 2^6 is present
1	÷	2	=	0	+	1	if remainder = 1, 2^7 is present

The binary number is thus 11110001, with the left-most digit representing the most-significant bit (MSB) and the right-most digit representing the least-significant bit (LSB).

Doing Math with Binary Numbers

In the binary system, the highest number that appears in each column is, of course, 1. Therefore, every time two 1s get added together, they generate a 0 and a carry. Let's look at a few simple examples:

```
  0     1     0     1
+ 0   + 0   + 1   + 1
---   ---   ---   ---
  0     1     1    10
                   └── Carry bit
```

The sum of 1 + 1 is 2, but 2 can also be represented as 2^1 and thus counts as a 1 in the next column. Let's now look at a more complicated example:

```
  1011    (decimal 11)
+  111    (decimal  7)
------
```

To do the addition, first combine the right-most bits to form a sum of 0 and generate a carry of 1 that gets placed in the next column. The second column then has a total of 3, which, of course, cannot exist in binary. A total of 2 must thus be carried into the third column and that leaves a remainder of 1, which stays in the second column. The 2 that was carried into the third column appears as a 1 and is summed, just as in the first column. The sum is 0, and another carry is generated. The last column follows the same procedure. Since the sum is again 0, another carry is generated and gets added to a placeholder 0. Diagramatically, the same problem can be shown as follows:

```
(1)←(1)←(1)←(1)←   Generated carries
     1    0    1    1
 0   0    1    1    1
-------------------
 1   0    0    1    0
```

Subtraction procedures are just the reverse; instead of a carry to the left, you must generate a borrow to the right. Here are a few simple examples that illustrate basic subtraction:

```
                        (1) ←— Borrow
  0     1     1     0
- 0   - 0   - 1   - 1
---   ---   ---   ---
  0     1     0     1
```

Let's take a closer look at the borrow with a more complex subtraction example:

```
  1001
-  110
------
```

For the right-most digits there is no borrow problem, and the difference is 1. The next digits, however, require a borrow from the left-most column, and the borrowing goes from left to right until the borrow reaches the second column. Since each column represents twice what is stored in the previous column, the borrowing of a 1 provides 2 for the column that needs the borrow. Thus, when the 1 is subtracted there is a remainder of 1. In the third column, 1 has already been borrowed from the 2 that was shifted right, so when 1 is subtracted the difference becomes 0. Diagrammatically, the subtraction looks like this:

```
     (1)←————————————  2nd borrow
     (10)   (10)←————  Borrows
  1   0      0   1
-     1      1   0
-------------------
  0   0      1   1
```

Let's compare this operation to a similar decimal subtraction:

```
 2    (15)   8    (12)←— Borrow plus whatever
 ß     5     ø     2    was in the column
 1     9     3     8
-------------------
 1     6     5     4
```

The binary numbering system is the easiest one for digital electronic circuits to use since only two electrical levels are needed: one to represent the binary 1 and another to represent the binary 0. Different computer systems use different electrical levels to represent 1 and 0, but whatever voltage levels are used, electronic circuits operate similarly.

The computing power of all computers is based on their ability to perform logic operations controlled by

data and instructions. Boolean algebra can thus be used to express many of the processes of computer systems since it is a form of logic reasoning involving two states: truth and untruth.

The ability of a computer to perform logic and math operations rapidly is its only strength. In logic, all statements must be either true or false; there are no intermediate conditions such as "maybe."

The binary numbering system provides us with just the right symbols to work with, and we will adopt them as a standard for the rest of the book. For a true logic statement, the binary value of 1 will be assigned, and for an untrue statement, a value of 0. Thus, a statement can equal 1 or 0 but it cannot have any other value (that is, it must be either true or false). Electronic circuits adapt easily to this system because only two voltages are needed and they can be as simple as ground (nothing, or 0 V) and some level V (which can be positive or negative). The true or false representation can be turned into an on and off equivalent for electronic circuits. Thus, when the circuit is turned on the statement could be true, and when the circuit is turned off the statement could be false. Depending on the type of logic circuits used, the positive level or ground could just as easily represent the true statement or the false statement.

Either statement standing by itself is of little logical interest. A statement has a truth value of 1 or 0, and that's all. However, when several statements are grouped together, other logical inferences can be developed. The three basic logic operators—AND, OR, and NOT—help connect statements so that conclusions can be drawn.

Using Logic Operators

The AND operator can be used to connect any number of logic statements, all of which must be true for the conclusion to be true. For instance, suppose a friend tells you that on Saturday you will find him at the park if the weather is good AND he has the day off from work. This remark can be written as a logic equation by using symbols: Let A represent the statement that the weather is good, B the statement that he has the day off from work, and C that you will find him at the park. Therefore, if A AND B then C.

Writing out an entire logic operation is often not necessary; a shorthand notation can be used. Sometimes the multiply dot · in the middle of a line is used to represent the AND function, and other times the dot is omitted and the individual letters are placed next to each other:

if A · B then C; if AB then C

Each of the two statements A and B can be either true or false. If true, we assign the statement a value of 1, if false, a value of 0. The statements can be tabulated to show all four possible combinations and the possible outcomes:

Statement A	Statement B	Outcome (A · B)
0	0	0
0	1	0
1	0	0
1	1	1

This type of listing is called a truth table since it shows every possible combination of the two statements.

The OR operator can also be used to connect any number of logic statements. However, unlike operations with the AND operator, only one statement of all those connected need be true for the outcome to be true. Let's use the same example we used for the AND operator but modify it slightly: A friend tells you that on Saturday you will find him at the park if the weather is good OR he has the day off from work.

We can write this statement in the form of a logic equation if we use the plus symbol + to represent the logic OR operation. Thus, if A + B then C. Whenever A or B is true, or they are both true, the outcome is true. Again, this can be shown in truth-table form very simply:

Statement A	Statement B	Outcome (A + B)
0	0	0
0	1	1
1	0	1
1	1	1

The only time your friend won't be at the park will be if the weather is not good AND he doesn't have the day off from work.

The last basic logic operator, the NOT function, can be used to invert or complement a logic statement. It is usually symbolized in shorthand by an overscore of the logic statement or statements you want to invert. Let's see how the logic statements used for the AND and OR operators can be rewritten.

The statement A refers to the fact that "the weather is nice"; therefore, $\overline{A}$ represents the statement that "the weather is not nice." Similarly, B represents the statement that "he has the day off from work," and $\overline{B}$ means that "he does not have the day off from work." Since A is represented by the binary 1, $\overline{A}$ would then be binary 0 ($A = 1$, $\overline{A} = 0$). Common pronunciations of $\overline{A}$ are "not A" and "A bar," although you may run across others. The truth table for the NOT function is very simple since it operates on one item at a time:

A	$\overline{A}$
0	1
1	0

Each of the logic operators has a physical equivalent that you might find easy to relate to. The simplest example is probably the common water faucet. If you take a look at your kitchen sink, you'll probably see something like the arrangement shown in Fig. 2.1. On top of the sink are two faucets (call them A and C) and a common spout. Under the sink you'll find two valves (call

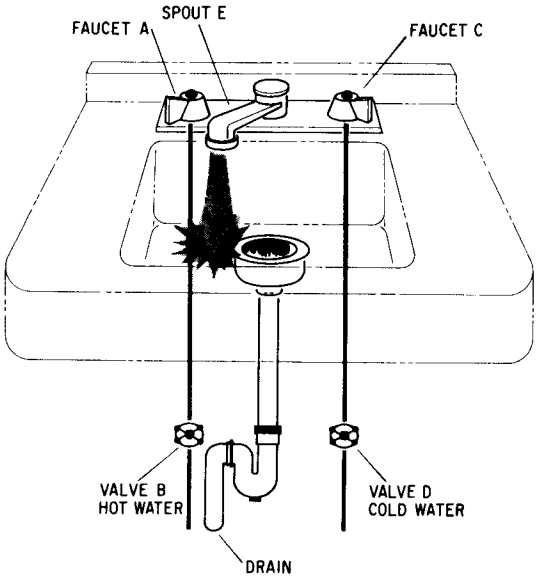


Fig. 2.1 A kitchen sink with two faucets, a common spout, and two emergency valves underneath is a good analogy for combined AND/OR logic statements.

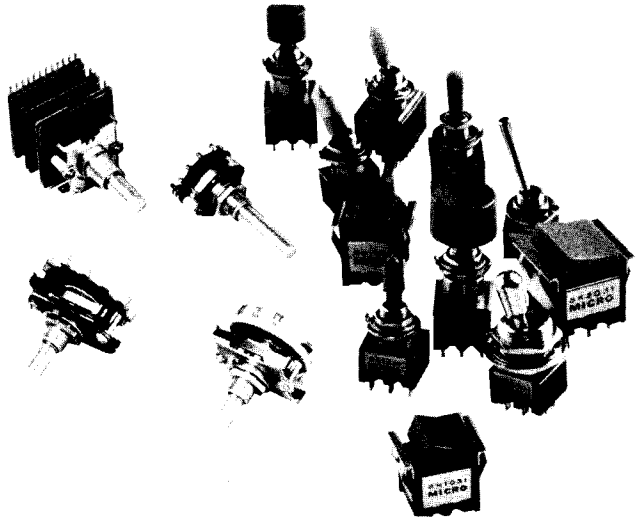


Fig. 2.2 Just as valves are used to start and stop water flow, switches (toggle, push button, rotary, etc.) are used to start and stop electrical current flow. (*Courtesy* Centralab and Micro Switch)

them B and D), one on the hot water line and one on the cold.

For water to flow out of the tap (call this statement E), the valve under the sink AND its respective faucet must be opened: $A \cdot B + C \cdot D = E$. This logic statement combines both the AND and OR operators since the tap is common to both hot and cold water lines. Water will flow if both valves A AND B are opened OR valves C AND D are open.

Just as we use valves to start and stop water flow, we can use electrical switches to start and stop electrical current flow (Fig. 2.2). Several switches connected in series perform an AND function (Fig. 2.3a). Switches in parallel perform an OR function (Fig. 2.3b), and a simple switch set up to function in a normal way can simulate the NOT function (Fig. 2.3c).

Combining Logic Operators

By mixing the three basic operators AND, OR, and NOT together, several other logic functions can be created. An AND combined with a NOT makes a NOT-AND, or NAND function; an OR combined with a NOT makes a NOT-OR, or NOR function; and two NOT functions cancel each other out. The truth tables for these functions and several others are shown in Fig. 2.4. Only two truth statements are used as determining elements in the examples although any number of statements can be used.

Multiple statements can be combined into one logic equation and there is no limit to the number of statements that can be linked. You can have two, three, four, eight, or more logic statements or expressions combined on one operator. The electronic equivalent to the logic operator is called a *gate*. It is possible to buy AND, OR, NAND, NOR, NOT, Exclusive-OR (XOR), AND/OR, and many other types of gates, each with different numbers of possible inputs. Many of the common symbols for logic gates are shown in Fig. 2.5.

The mathematics of combining these different logic gates is what we call Boolean algebra. Every form of mathematics has some basic theorems, postulates, and underlying truths. In the short space of this book, you'll just get an overview and rudimentary understanding of the basic concepts. (For more about Boolean algebra, see Appendix A for additional reading.)

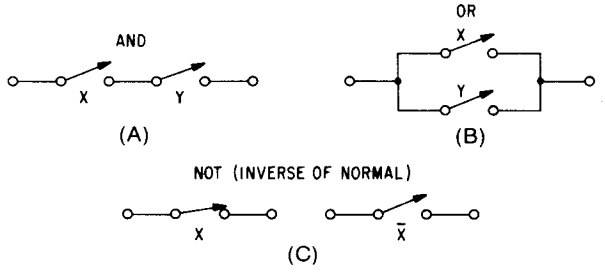


Fig. 2.3 Switch connections showing (a) AND, (b) OR, and (c) NOT (inverse of normally open) equivalents of logic gates.

A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

(A)

A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

(B)

A	$\bar{\bar{A}}$
0	0
1	1

(C)

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

(D)

Fig. 2.4 Truth tables for (a) NAND, (b) NOR, (c) NOT-NOT, and (d) Exclusive-OR (XOR) gates.

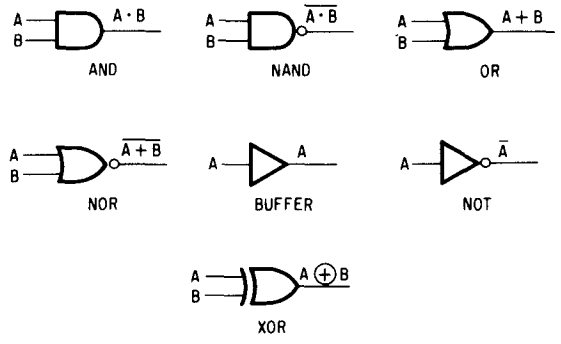


Fig. 2.5 Commonly used symbols for logic gates.

The Basic Rules of Boolean Algebra

Thus far we've looked at truth statements, and if one was true we assigned it a value of 1, and if false a value of 0. Let's see what happens if we try combining statements.

When two true statements are ANDed together, the outcome is always true. If one of the statements is false, the outcome is always false. In symbolic form, some universal statements may be made, as follows:

1. The outcome of a logic AND equation where all but one term is known to be true depends directly on the unknown term:

$$A \cdot 1 = A$$

2. The outcome of a logic AND equation where any of the terms is false is always false:

$$A \cdot 0 = 0$$

3. Any logic statement ANDed with itself is equal to the original logic statement:

$$A \cdot A = A$$

4. Any logic statement ANDed with its complement is always false (see statement 2):

$$A \cdot \overline{A} = 0$$

5. Any logic statement upon which a double NOT operation has been performed is equal to the original statement without any operations performed:

$$\overline{\overline{A}} = A$$

6. The outcome of a logic OR operation where any number of statements are ORed with at least one true statement will always be true:

$$A + 1 = 1$$

7. The outcome of a logic OR equation where one or more terms are known to be false depends on the remaining statements:

$$A + 0 = A$$

8. Any logic statement ORed with itself is equal to the original logic statement:

$$A + A = A$$

9. Any logic statement ORed with its complement is always true:

COLUMN 1			2	3	4	5
A	B	C	$\overline{A+B+C}$	$\overline{A}\cdot\overline{B}\cdot\overline{C}$	$\overline{A\cdot B\cdot C}$	$\overline{A}+\overline{B}+\overline{C}$
0	0	0	1	1	1	1
0	0	1	0	0	1	1
0	1	0	0	0	1	1
0	1	1	0	0	1	1
1	0	0	0	0	1	1
1	0	1	0	0	1	1
1	1	0	0	0	1	1
1	1	1	0	0	0	0

Fig. 2.6 Truth table for DeMorgan's theorem using binary numbers.

$$A + \overline{A} = 1$$

Boolean algebra also follows some common mathematical laws, which can be illustrated as follows:

Commutative Laws

$$A \cdot B = B \cdot A$$
$$A + B = B + A$$

Associative Laws

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$
$$(A + B) + C = A + (B + C)$$

Distributive Laws

$$A + B \cdot C = (A + B) \cdot (A + C)$$
$$A \cdot (B + C) = A \cdot B + A \cdot C$$

Two theorems were developed after Boole formulated his basic postulates. These theorems, developed by DeMorgan, bear his name. The DeMorgan theorems simply state that:

1. The inverse of any series of OR operations is equivalent to an identical series of inverted AND statements:

$$\overline{A + B + C} = \overline{A} \cdot \overline{B} \cdot \overline{C}$$

2. The inverse of any series of AND operations is equivalent to an identical series of inverted OR operations:

$$\overline{A \cdot B \cdot C} = \overline{A} + \overline{B} + \overline{C}$$

Let's go through the truth table for these two theorems (Fig. 2.6). When all three statements are false, the NOT-OR combination (column 2), the NOT-AND combination (column 4), and their DeMorgan equivalents (columns 3 and 5) are all true. However, for the NOR function (column 2) and its equivalent in column 3, whenever any one or more of the logic statements in column 1 are true, the outcome is false. On the other hand, the NAND function shown in column 4 (and its equivalent in column 5) remains true for all combinations of the logic statements except for the case of all three statements being true. When A, B, AND C are true, the AND function requires that the output be true; however, since a NOT operator covers the entire expression, the outcome will be opposite to that normally expected.

CHAPTER 3

Introductory Electronics and Logic Functions

Computers are built from a wide variety of electronic components and hardware—resistors, capacitors, integrated circuits, transistors, diodes, transformers, cabinets, switches, sockets, indicators, printed circuit boards, and a multitude of other items. However, unless you are going to troubleshoot the computer when it breaks down, or you intend to build your own computer boards, you really don't have to know how the various components work. This chapter will provide a brief summary of the operation of most of the major components encountered in a computer. For detailed explanations and theoretical discussions consult Appendix A for additional reading.

Very simply, electricity is the flow of electrons from one point to another. Materials that permit an easy flow of electrons are called *conductors* and they offer a *low resistance* to the flow. Materials that totally block or tremendously impede the electron flow are called *insulators*; they offer a *high resistance* to the flow. There are also materials that fall somewhere in between the conductors and insulators; these we call *semiconductors*, and they form the basis for all solid-state circuits.

The force that moves electrons is called the electrical potential (often referred to as voltage); it is measured in units called *volts*. The flow of electrons caused by the voltage is called *current* and is measured in units called *amperes*. Opposition to the flow of current is called *resistance*; it is measured in units called *ohms*. The triad of volts, amperes, and ohms forms a simple mathematical relationship known as Ohm's law:

$$\text{voltage} = \text{current} \times \text{resistance}$$

Standard abbreviations for voltage, current, and resistance are as folllows:

V = volt (unit of voltage)
A = ampere (unit of current)
Ω = ohm (unit of resistance; symbolized by the capital Greek letter omega)

The basic numerical relationship between these three terms can be expressed with the help of symbols:

$$1\ \text{V} = 1\ \text{A} \times 1\ \Omega$$

However, whenever numbers are not used, the letter E represents the voltage potential.

Electricity takes two forms: alternating current (ac) and direct current (dc). In your home, office, and most places of business, the power available at the wall outlet is a form of sinusoidally alternating current. The potential that forces the current to flow follows a pattern such as the one shown in Fig. 3.1. The voltage starts at a zero level, increases to a positive maximum value, then decreases through zero to a maximum negative value, and finally rises back to zero. This cyclic operation repeats many times a second and each complete variation is called a cycle. The ac power supplied in the United States provides 60 cycles every second at an average of 115 V. To measure the number of cycles per second, we use the unit Hertz (abbreviated Hz); thus the ac power is said to be 115 V, 60 Hz.

Dc voltages don't vary in cyclic patterns; they are constant. A simple example of a dc source is the com-

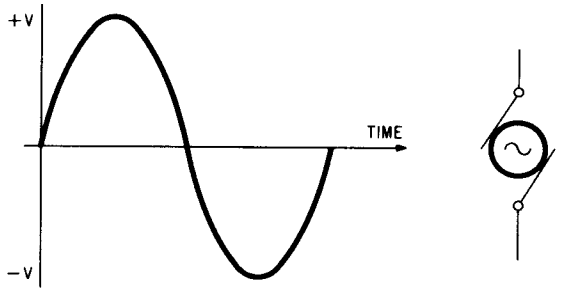


Fig. 3.1 Representation of an ac power source.

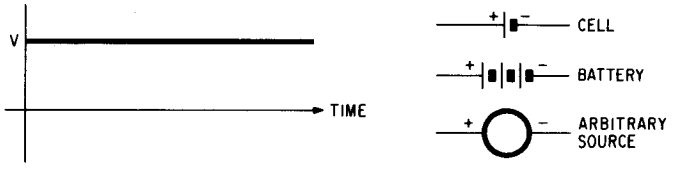


Fig. 3.2 A dc power source shown in graphic and schematic representations can be as simple as an ordinary flashlight battery or as complex as an entire rack of equipment.

mon flashlight battery. Its voltage remains almost constant, but slightly decreasing, as shown in Fig. 3.2 on the previous page, until the chemical processes inside the cell can't produce any more electricity. When that happens, the cell has served its useful life and is either discarded or recharged.

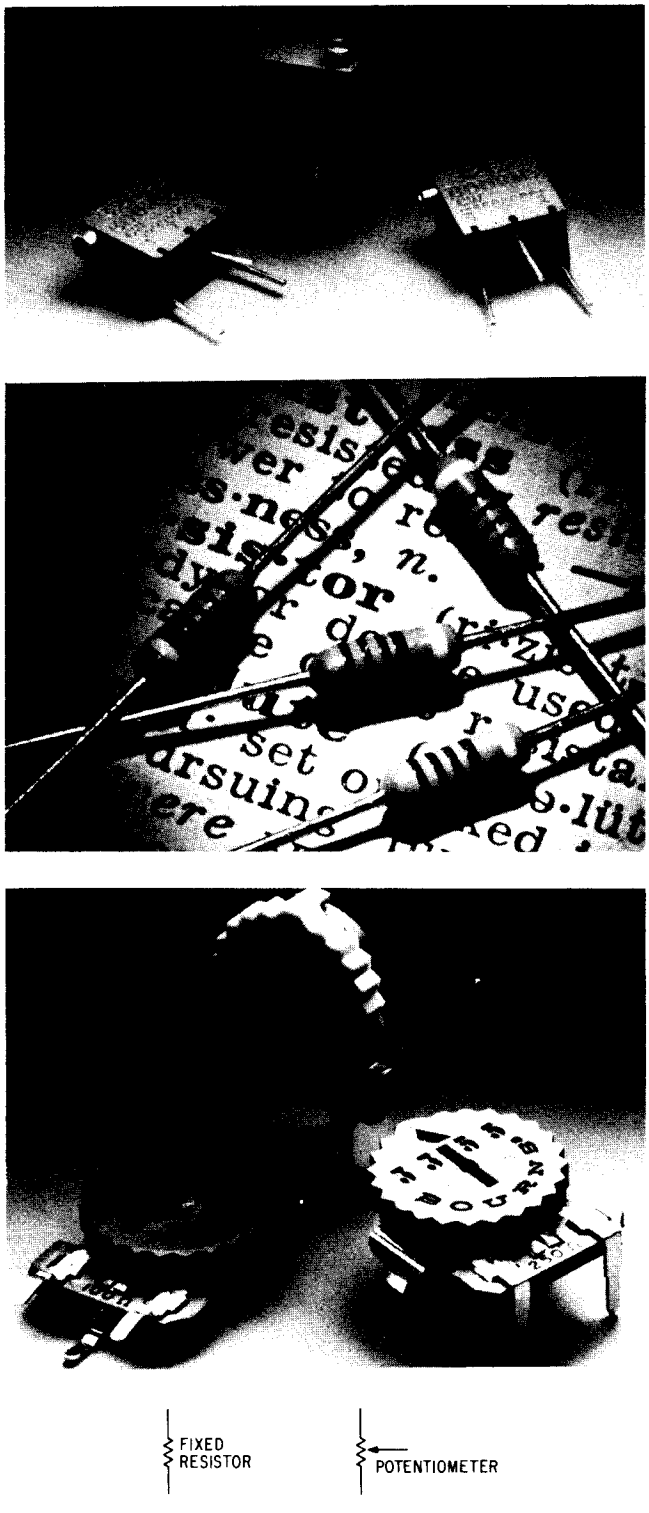


Fig. 3.3 Some typical resistors and their various schematic symbols. (*Courtesy* Allen-Bradley and Bournes)

Let's Look at the Components

Components specially designed to provide resistance, called resistors, form an important part of every electronic circuit. They are available in many different forms, in both fixed and adjustable types, as shown in Fig. 3.3. The type of resistor to be used depends on many design criteria, from resistance value to operating temperature. For many applications, low-cost resistors made from molded carbon are quite sufficient. However, when high power-handling capability or high precision is needed, more expensive metal-film or wire-bound resistors are usually selected. Many resistors are often used in the same circuit, and sometimes several resistors must be combined either in series or in parallel to make a larger or smaller value. Combinations of resistors are often used to split currents and divide voltages.

Every electronic component has resistance, even conductors. Wire, for example, has a very low resistance—typically thousandths of an ohm for short lengths. And, in most cases the resistance can be ignored. But it can't be ignored in the power connections within the computer. Here, voltage losses of half a volt might occur, and that, combined with heat build-up from the power loss (I^2R), can cause problems. Basically, wire resistance depends on four factors—length, material, temperature, and diameter—but is most often directly compared to diameter. The thicker the wire the lower the resistance.

Another component often encountered is the capacitor, which is represented by the symbol shown in Fig. 3.4a; some representative samples are shown in Fig. 3.4b. The capacitor has a characteristic called capacitance, which is a sort of storage capability for electrons, and is measured in units called farads, F. Most capacitance values today are small compared to the farad, and in many cases capacitance is specified in millionths or millionth-millionths of a farad—μF (microfarads) and pF (picofarads), respectively.

In its simplest form a capacitor consists of two closely spaced parallel conducting plates separated by some form of insulating material. The type of insulator used to separate the plates has a lot to do with the storage capacity. Commonly used insulators include ceramics, mica, glass, oil, and even waxpaper. Most small-valued capacitors (under 1 μF) "don't care" as to the type of voltage (ac or dc) connected to them since the materials used are not sensitive to positive or negative voltages. However, larger-valued capacitors, known as electrolytics, use a special combination of materials and chemicals to obtain the high capacitance (tenths of a farad). Because of their special construction, the capacitor terminals have a fixed voltage polarity and can literally explode if polarities are not observed.

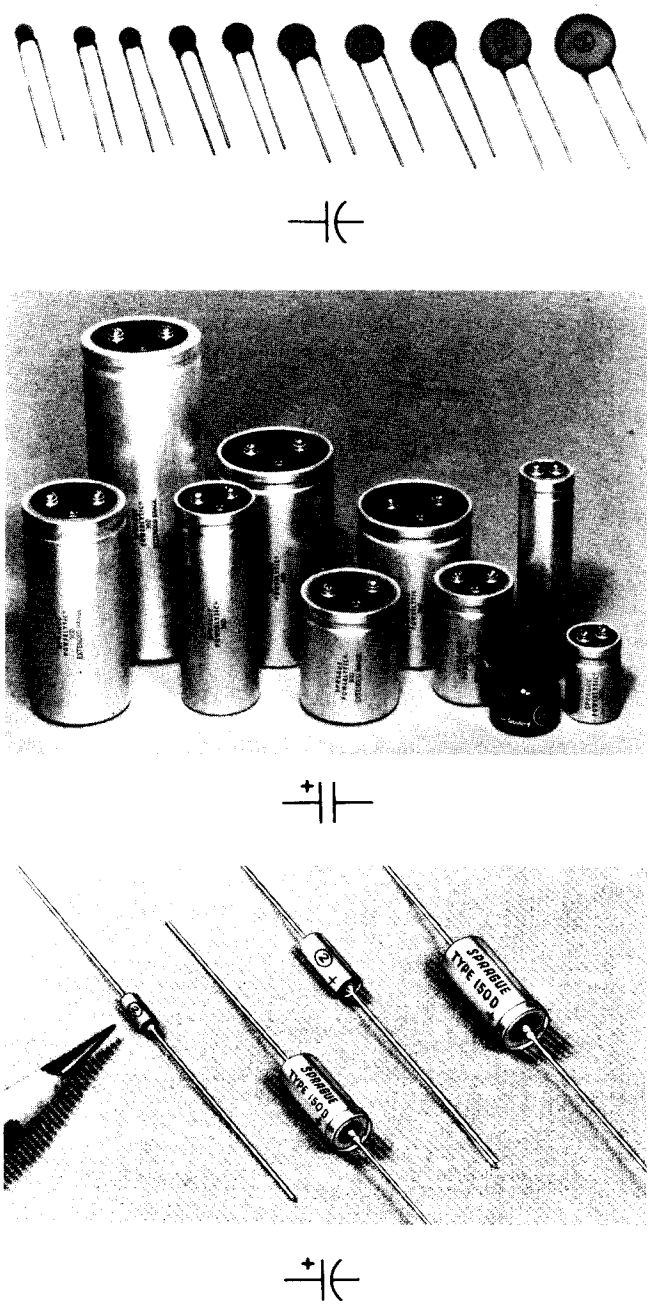


Fig. 3.4 Some typical capacitors and their schematic symbols. (*Courtesy* Sprague)

Capacitors are not only rated for their capacitance, but for their maximum operating voltage as well. Because of the nature of the materials used, the larger the capacitance value, the larger the physical size and the lower the voltage rating. A typical electrolytic capacitor used in a computer power supply might have a rating of, say 5000 μF at 35 V and have a physical size of 5 in. long by 2 in. diameter. In contrast, a tiny disc capacitor, about 0.5 in. diameter and only 0.1 in. thick, might have a rating of 0.01 μF at 1000 V.

Capacitors act as an open circuit for dc voltages; since there is no connection between the plates, no current flows. The amount of voltage the capacitor can withstand and the capacitance value are determined by the separation and size of the plates and the insulating material used. In ac circuits, the voltage, which is constantly cycling, seems to pass right through the capacitor.

Resistors and capacitors are known as passive components since they cannot perform any control function and do not require a source of power aside from the voltage connected to them. Another class of components—active devices—require a power source in addition to the signal coming in. Active devices include such components as electronic tubes, transistors, diodes, silicon-controlled rectifiers, and integrated circuits.

The Basics of Solid-State Technology

Except for tubes, all active components are built from semiconducting material—nowadays silicon, although some early devices in the 1950s and 1960s were built from germanium. Appendix A lists many books that discuss the history of semiconductors and their theory of operation. The rest of this chapter will provide a capsule view of some electronic components to familiarize you with some symbols and terms.

The semiconducting material used is made from specially processed silicon that has been refined to extremely pure levels and then made impure with special materials. Silicon with an excess of electrons is called *n-type* material, and silicon with a deficiency of electrons is called *p-type* material.

A diode consists of two layers of silicon grown on top of each other—one p type and one n type—as shown in Fig. 3.5 (note its symbol). When an alternating voltage is placed across the diode, the electrons are pushed from the n-type material only for the first half of the cycle. On the other half of the cycle, no electrons flow, since electrons are forced to go back into the material. The diode is often called a *rectifier*, since for ac signals half of the signal is removed, as shown in Fig. 3.6. This process is called *rectification*.

The diode is biased so that current will flow if the p section is more positive than the n section. Because of the nature of silicon (not discussed in this book), about

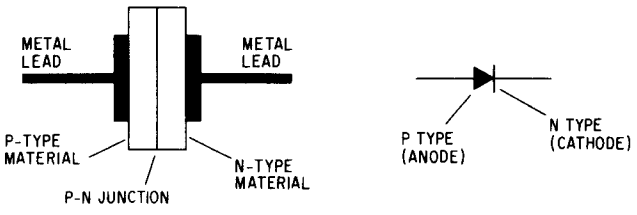


Fig. 3.5 A semiconductor diode consists of two layers of silicon grown on top of each other—one p type and the other n type.

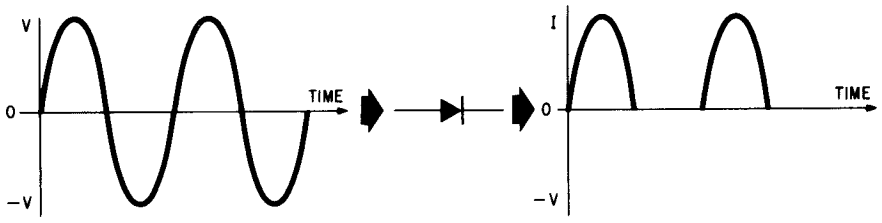


Fig. 3.6 When an ac signal is imposed on a diode, the voltage causes current to flow in only one direction.

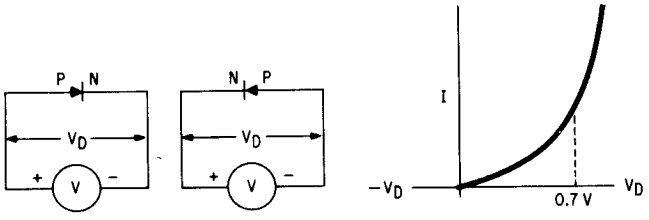
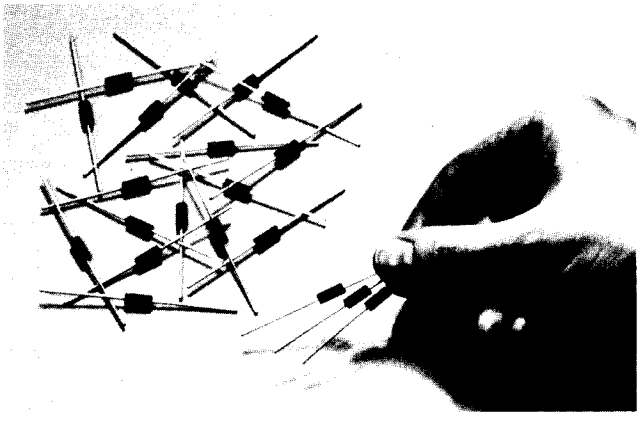


Fig. 3.7 Diodes must be biased so that current will flow if the p section is biased more positively than the n section.



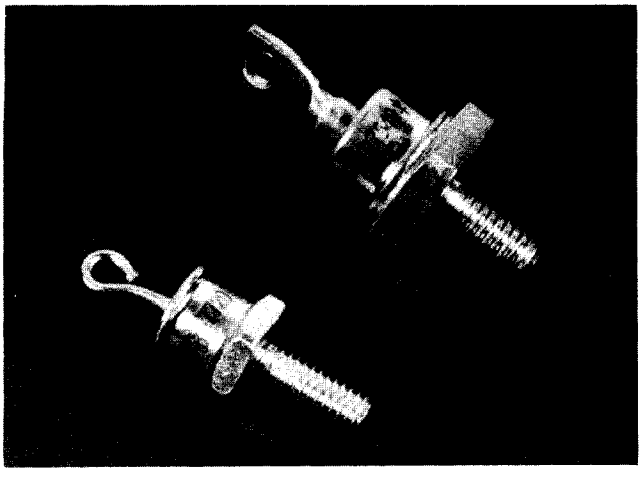


Fig. 3.8 Diodes are hard to tell apart due to their similar packaging. However, their functions vary—signal handling, power rectification, voltage reference, light emitting, light sensing, switching, etc. (*Courtesy* Motorola)

0.7 V is required to *forward bias* the diode (Fig. 3.7). Germanium diodes, although not common now, are still in use; they require a forward voltage bias of 0.4 V.

There are also several families of diodes—signal handling, power rectification, voltage reference, light emitting, switching, and others—but if you look at the packages shown in Fig. 3.8 you can see they are hard to tell apart.

Voltage-reference diodes, called *zeners*, are used to set voltage levels. When forward biased, these diodes behave just like normal diodes and permit current flow. However, when they are reverse biased at a point above what is known as the *breakdown region* the p–n junction goes into an avalanche, or zener, mode, and the potential across the p–n junction remains at an almost constant voltage, known as the *zener voltage*, Vz. (Fig. 3.9). (More zener diode information is available in the reference books listed in Appendix A.)

Another major family of diodes used in building a computer is the light-emitting diode, better known as an LED. These devices, when forward biased, emit colored light (either red, green, yellow, or orange). Typically, bias voltages of 1.5 to 2 V are necessary to make the diodes emit light, with currents ranging from 5 to 50 milliamperes. Some LEDs are available prepackaged with a series resistor so that they can operate from a higher voltage without burning out, as shown in Fig. 3.10. LEDs are used as indicators in many electronic applications; more about how to use them will be discussed in succeeding chapters.

Transistors: Semiconductor Control Elements

If two diodes are placed back to back so that both p regions are connected or so that both n regions are connected as shown in Fig. 3.11, they roughly approximate the structure of a bipolar transistor. Actually, a transistor is made up of three regions. If it's an npn

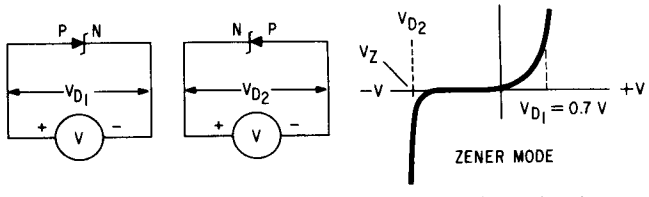


Fig. 3.9 Zener diodes are used to set voltage levels. They behave like a normal diode when forward biased, but when reverse biased at a point above the breakdown region, the p–n junction goes into avalanche (the zener mode). The potential across the p–n junction then remains almost constant. This voltage is known as the zener voltage.

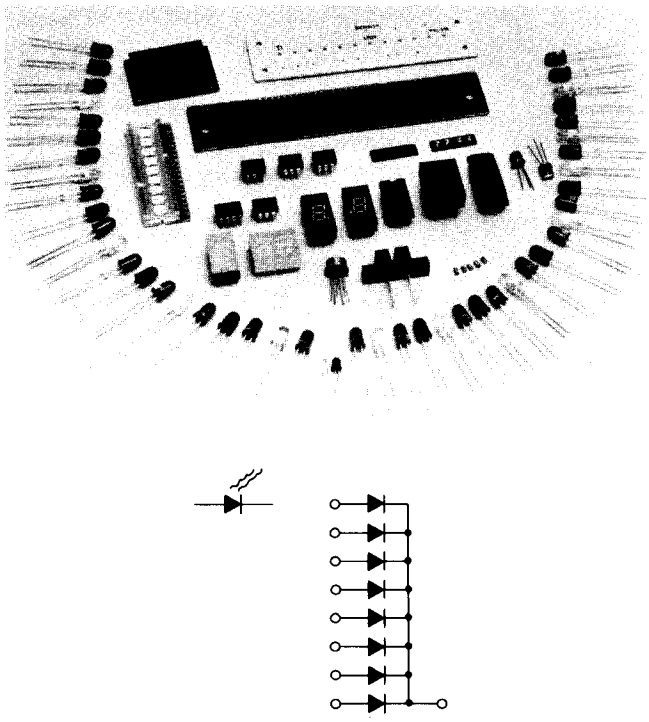


Fig. 3.10 Various forms of light-emitting diodes and their basic schematic symbols. (*Courtesy* Texas Instruments)

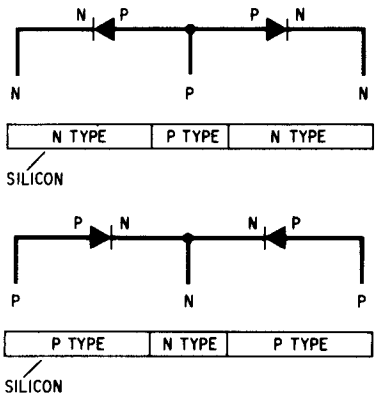


Fig. 3.11 The basic transistor structure can be simulated by placing two diodes anode-to-anode or cathode-to-cathode (pnp and npn structures).

transistor, it has a p material sandwiched between two n-type materials; if it's a pnp transistor, it has an n material sandwiched between two p-type materials. The symbols for npn and pnp transistors, along with some typical devices, are shown in Fig. 3.12.

The central region (where both diode p or n regions combine) is called the *base* of the transistor and serves as the control terminal. The leg of the device with the arrow superimposed is called the *emitter*, and the leg without the arrow is called the *collector*.

Because the base acts as a control element, much like a water faucet, the transistor can act as a switch.

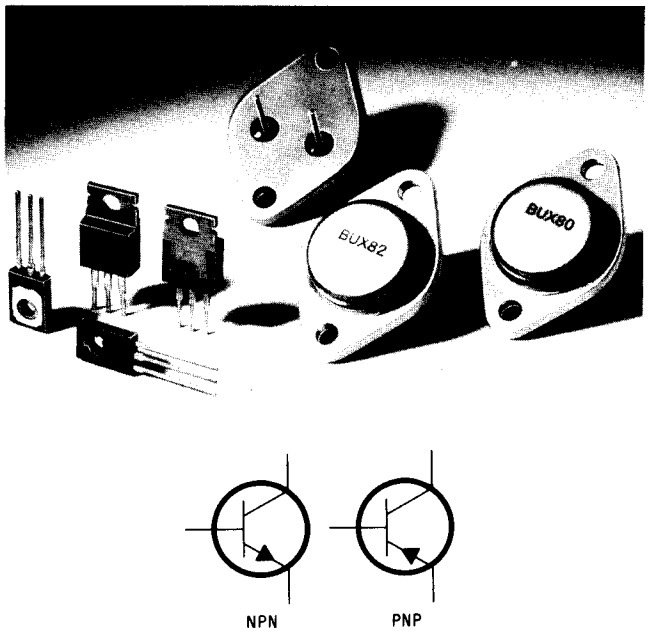


Fig. 3.12 Various transistor packages and the two basic transistor symbols. (*Courtesy* Amperex)

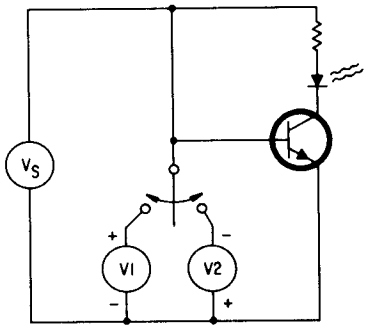


Fig. 3.13 A simple transistor switch, controlled by two power supplies, can turn an LED on or off depending on the polarity of the supply connected to the base terminal.

When the base–emitter junction is reverse biased, no current can flow in the collector circuit; but when it is substantially forward biased, current can easily flow in the collector circuit. For more about basic bipolar transistor operation, see the books listed in Appendix A.

The circuit shown in Fig. 3.13 can be used to illustrate the switch concept simply. In the base–emitter circuit is a switch that can select either a forward or negative bias. When connected to the negative bias V2, the LED will not light up, but if the switch connects to the forward bias source, the LED lights, thus showing current flow. This general principle is used in all digital computer circuits to indicate the ones and zeros of binary arithmetic.

There are many types of three-terminal control semiconductors other than bipolar devices. Some use an electric field-effect to permit or stop current flow.

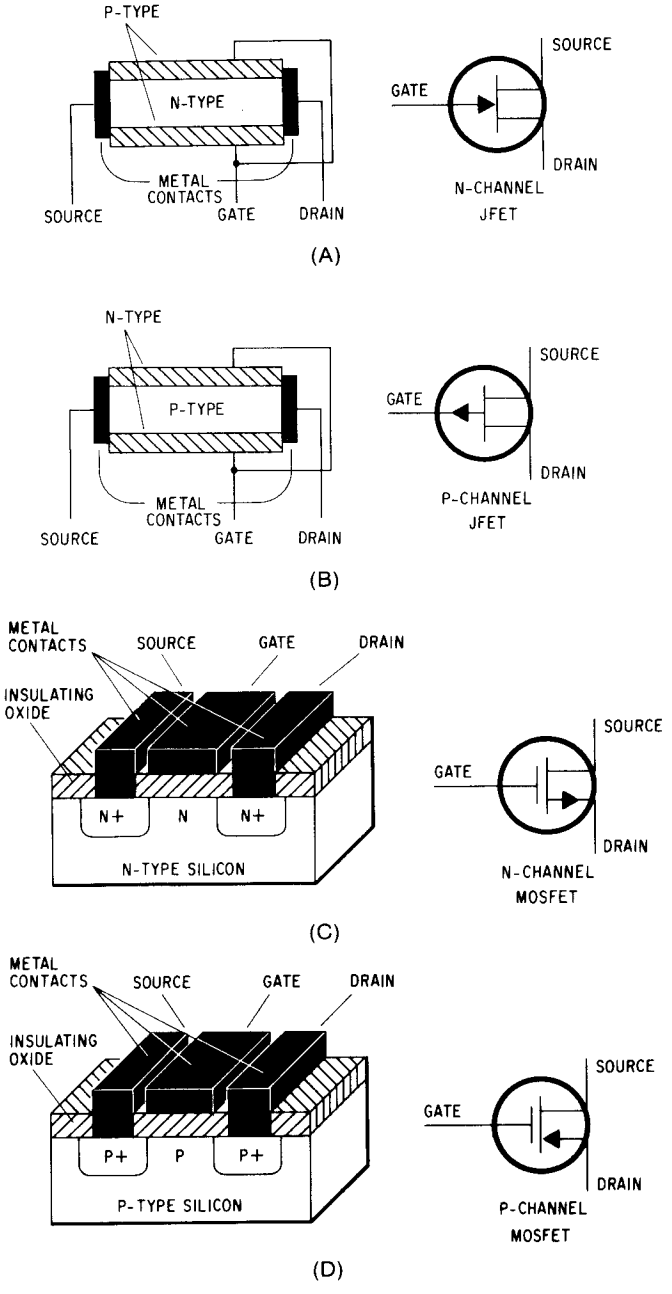


Fig. 3.14 Field-effect transistors are available in two forms: the junction FET (JFET) in both p- and n-channel versions (a and b), and the insulated gate FET (c and d), which is better known as the metal-oxide FET (MOSFET).

These devices, called *field-effect transistors* or FETs, are available in two forms: the junction field-effect transistor (JFET) and the insulated-gate field-effect transistor (IGFET), which is more commonly known as a metal-oxide semiconductor transistor, or MOSFET. FETs do not have a collector, emitter, or base. Instead, their terminals are called the *source*, *drain*, and *gate*, respectively. Both types of FETs are shown in Fig. 3.14, along with their schematic representations.

Other types of devices such as silicon-controlled rectifiers (SCRs) and triacs are also used in computer systems, mainly to perform necessary control functions that require high current-handling capability. The SCR and the triac are three-terminal devices like transistors, but that's where the similarities end. The terminal used as the control element is referred to as the gate, and when a control voltage is applied to the gate, the SCR or triac is turned on like a switch. If the control voltage is then removed, the SCR or triac will continue conducting as long as the voltage difference between the anode and cathode remains at a minimum positive value. The symbols and basic structures of both devices are shown in Fig. 3.15.

Basically, the SCR is a controlled diode. If a voltage is placed across the SCR's anode and cathode so that the "diode" structure is forward biased, no conduction will take place until the gate is also brought positive with respect to the cathode (triggered). When the SCR is "triggered," it will conduct current in the forward direction until the current drops below the minimum needed to support conduction. When the current drops below the minimum value, the SCR or triac resets itself to the blocking condition and will not conduct again until another voltage is placed on the gate. Although operation of the triac is similar to that of the SCR, the triac can conduct current in both directions and thus handle ac current as well as dc. The gate can be triggered each time the voltage difference between anode 1 and 2 or between 2 and 1 goes above the minimum value necessary to permit conduction. Current capabilities for both SCRs and triacs range from tens of milliamps to hundreds of amps.

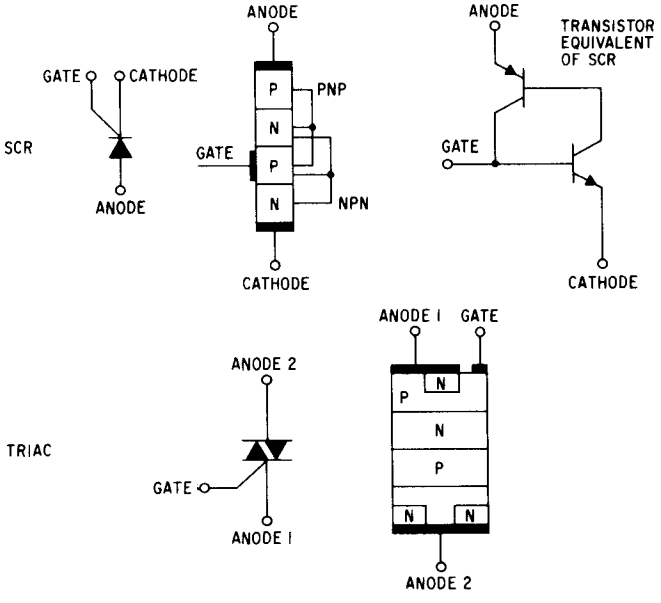


Fig. 3.15 Silicon-controlled rectifiers and triacs are also three-terminal devices like transistors and are often housed in transistor-like packages. Their characteristics, however, are quite different than those of transistors.

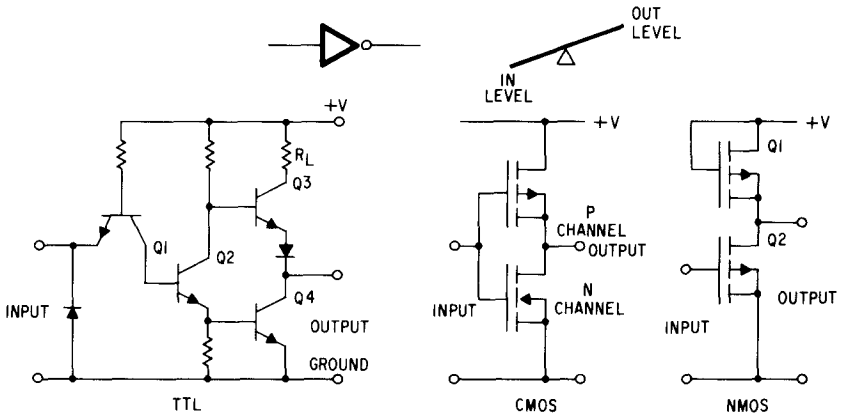


Fig. 3.16 The inverter function is similar to a seesaw—whatever the input state is, the output state is the opposite.

The Integrated Circuit

Almost all the different components discussed so far—the resistor, capacitor, diode, and transistor—can be combined in various forms within a single, minute piece of silicon less than 0.2 in. per side. Circuits formed in this way are called *integrated circuits* and form the basis for all modern computer structures. There are four major classes of logic circuits in use today:

1. complementary metal-oxide semiconductor (CMOS) logic circuits
2. n- and p-channel MOS (NMOS, PMOS) logic circuits
3. transistor–transistor logic (TTL) circuits
4. emitter-coupled logic (ECL) circuits

Each of these logic families can be broken down into three levels of circuit complexity: small-scale integration (SSI), which usually consists of simple gate functions; medium-scale integration (MSI), which consists of storage elements and multiple-gate arrays; and large-scale integration (LSI), which consists of large arrays of storage elements, processing systems, and complex control circuits.

The basic building block of digital computers is the *logic gate*; it is the "glue" that holds all the more complex circuits together. In the previous chapter some basic guidelines were set up for logic notation, and they bear some repeating:

1. A *logic ONE level* is the most positive voltage level used in the system.
2. A *logic ZERO level* is the lowest or most negative voltage level used in the system.

Many circuits in use today have been "standardized" to work within predetermined voltage changes. Almost all forms of TTL, CMOS, and NMOS can operate from a power supply of 5 V dc and have logic ONEs and ZEROs that obey the following rules:

Logic 1 = any voltage above 2 V
Logic 0 = any voltage below 0.8 V

Let's take a look at the simple NOT, or inverter, circuit and see how the input logic level controls the output logic level. Basically, the inverter can be thought of as the pivot of a seesaw (Fig. 3.16). Whatever the condition of the input, the output is the opposite. Thus, when the input is a logic ZERO, the output is a logic ONE, and vice versa.

More Complex Circuits

The simple inverter is just the beginning. There are many types of standard logic circuits, as mentioned in Chapter 2 in the discussion of basic Boolean functions. Some of the more common circuits include the 7400 quad 2-input NAND gate, the 7404 hex inverter, the 7410 triple 3-input NAND gate, the 7402 quad 2-

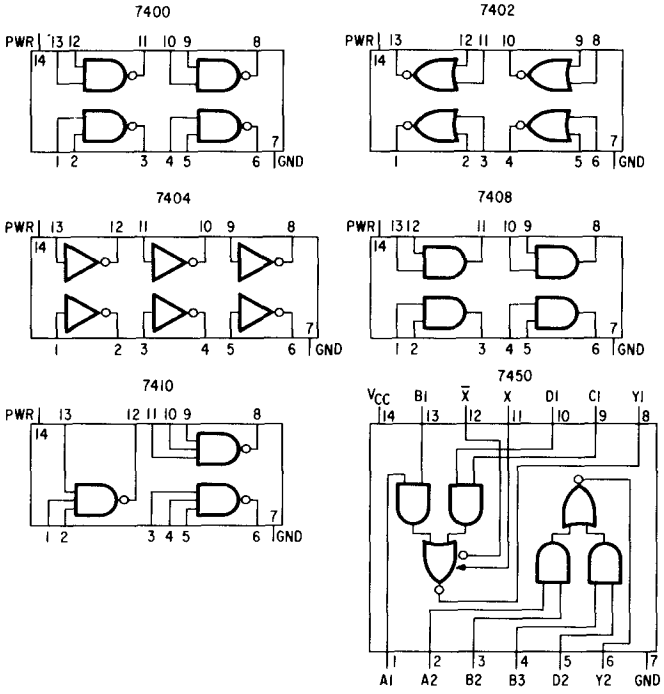


Fig. 3.17 Package layouts of some commonly used logic gate functions.

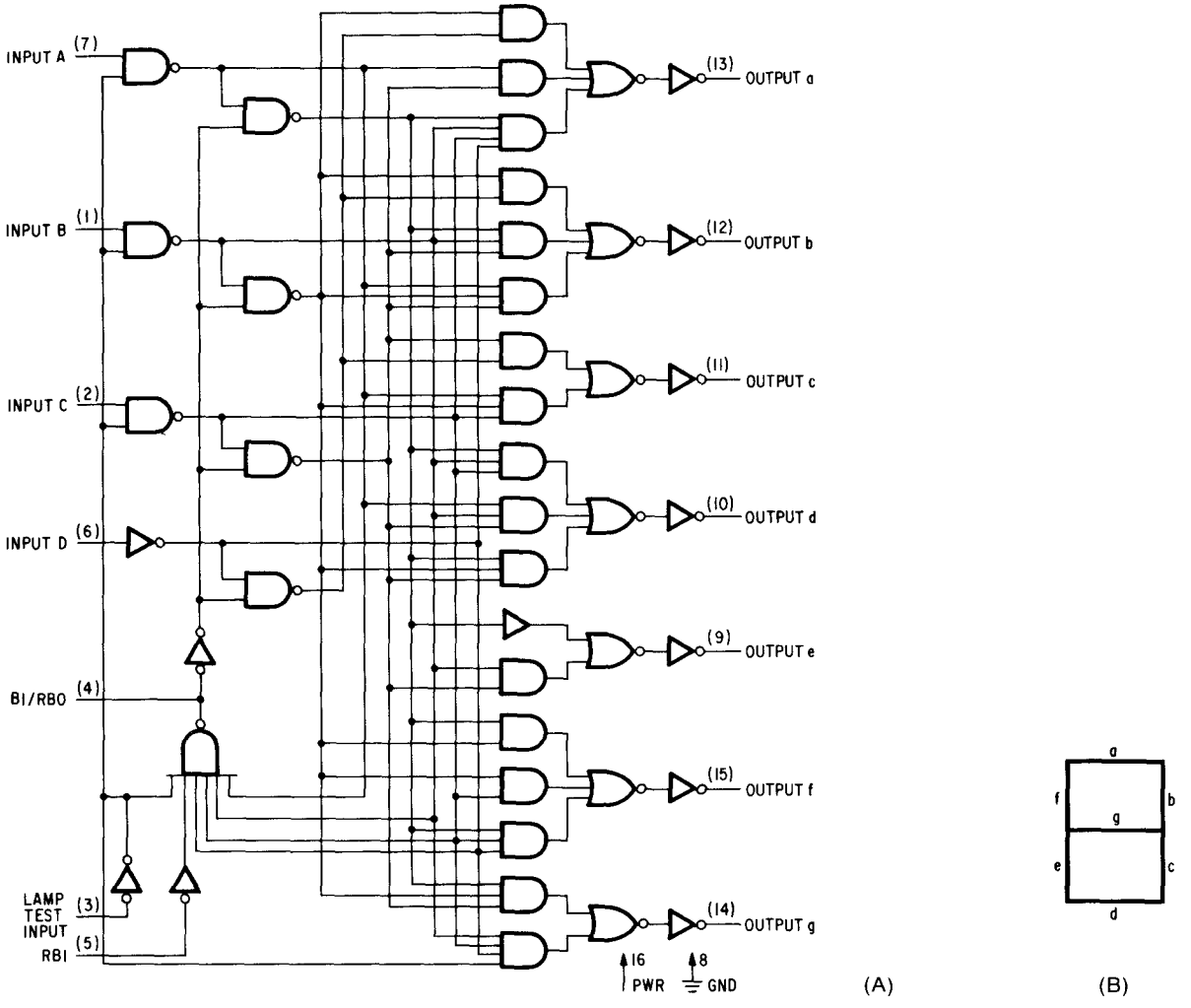


Fig. 3.18 A typical decoding circuit (a) accepts a 4-bit BCD input code and provides seven outputs that can supply power to a seven-segment display, thus forming the numbers 0 to 9 (b).

input NOR gate, the 7408 quad 2-input AND gate, and the 7450 expandable dual 2-input AND-OR-INVERT gate (see Fig. 3.17 on p. 23). These TTL circuits are often referred to as SSI (small-scale integration), and they form the logic "glue" that binds all the more complex circuits together.

A form of TTL circuit that uses a newer technology, the low-power Schottky TTL circuit, has just about replaced almost all of the older TTL circuits for new computer designs. These circuits, referred to as the 74LS00 family are, for the most part, directly substitutable in circuits for the older 7400 family. Basic operation of the functions is identical; however, the 74LS00 family components require less power and operate faster.

More complex arrays of gates can be built to decode binary bit patterns or to encode one bit pattern into another. A typical combination decoding and drive circuit is the 7447 (Fig. 3.18). It accepts a four-bit BCD input code and provides seven outputs that can power the correct lamps in a seven-segment display to form the numbers 0 through 9. The four-bit input at A, B, C, and D determines which of the seven outputs stays HIGH or goes LOW. Each output of the seven inverters has an open collector transistor that can handle up to 20 mA and whose structure is similar to that shown in Fig. 3.19.

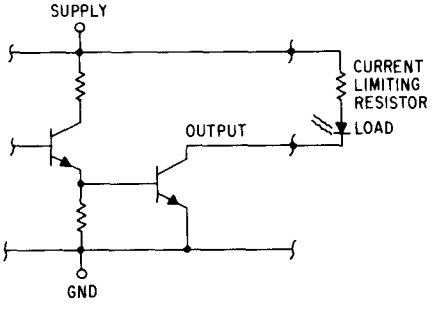


Fig. 3.19 The open-collector output structure of many logic circuits permits the output voltage and current to be defined by the application.

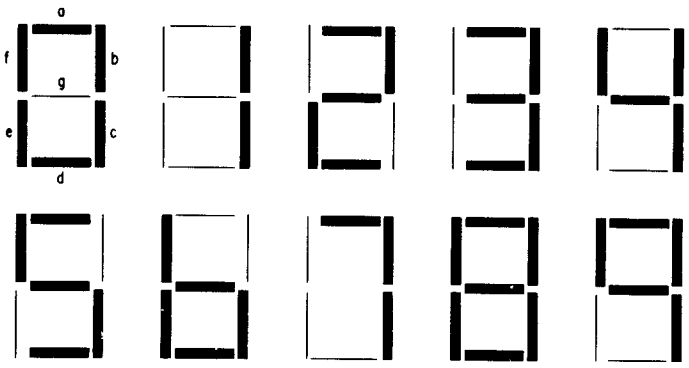


Fig. 3.20 To form the number 0 on an LED display, all segments but the g segment must be turned on and lit up. Other numbers can be formed by lighting the appropriate segments.

Whenever the output transistor is turned on, any load connected from the supply to the output will have current flowing (logic ZERO). When the transistor is turned off (logic ONE), the current stops flowing.

For the input code 0000, all outputs but g go LOW and cause bars a, b, c, d, e, and f to light up on the display, generating the number 0 (Fig. 3.20). For a 0001 input, all outputs except the b and c lines stay HIGH, thus displaying the number 1. For the code 0010, the f and c outputs stay HIGH, thus displaying the number 2; for 0011, f and e stay HIGH, thus displaying 3; for 0100, a, e, and d stay HIGH, thus displaying 4; for 0101, b and e stay HIGH, thus displaying 5; for 0110, a and b stay HIGH, thus displaying 6; for 0111, f, g, e, and d stay HIGH, thus displaying 7; for 1000, no lines stay HIGH, thus displaying 8; and for 1001, e and d stay HIGH, thus displaying 9.

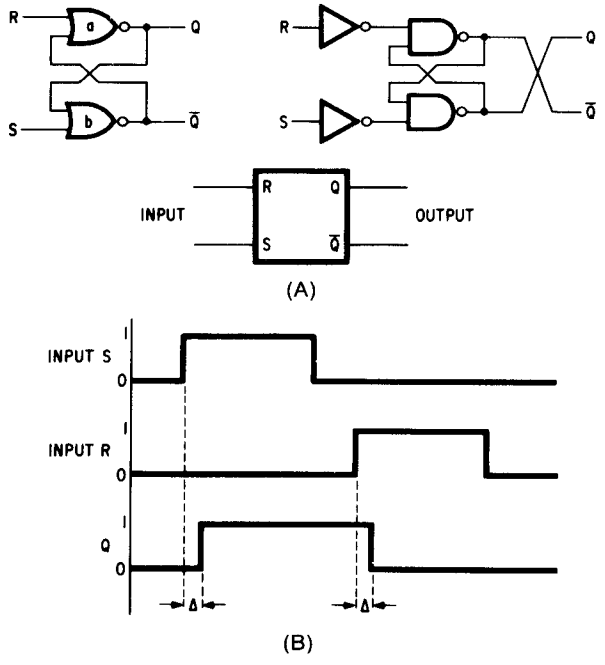


Fig. 3.21 The R-S flip-flop (a) is the simplest form of storage element that can be made by cross coupling two gates. The flip-flop's Q output will go HIGH when S goes HIGH and LOW when R goes HIGH (b).

All gate circuits discussed so far, however have outputs that depend directly on the present state of the input signals. If the input signals are changed, the output will follow the input in accordance with the gate logic. All combinatorial logic circuits have this shortcoming—a lack of memory. If you need a circuit whose output depends on previous states, the circuit must be able to remember, or store, the result of the previous input. Another logic element called the *bistable multivibrator* (better known as the *flip-flop*) is needed to hold information about the previous input. Usually, a flip-flop has two outputs, each the complement of the other. When one output is LOW, the other is HIGH, and vice versa. The condition of the flip-flop outputs is often referred to as the *state of the output*; in many cases, the outputs are labeled Q and $\overline{Q}$.

The most basic form of flip-flop, the R-S, can be formed by cross-coupling two NOR or NAND gates (Fig. 3.21). Input leads R and S refer to the control signals RESET and SET, which determine the state of the Q and $\overline{Q}$ outputs. If the R and S signals are assumed to be voltage levels corresponding to the logic 1 and 0 values, the R and S inputs work as follows: Regardless of the output state, whenever a logic 1 appears at the reset input, the flip-flop's output will always go to $Q = 0$ and $\overline{Q} = 1$. And, when a logic 1 appears on the set line, the output will always go to $Q = 1$ and $\overline{Q} = 0$.

For this simple flip-flop, there is one condition for which the output cannot be defined—when R = 1 and S = 1. Since this is an illogical request anyway—trying to set and reset the flip-flop at the same time—the output can be said to be indeterminate. The operation of a flip-flop can be summarized in what is called a transition table—a form of truth table for flip-flops (Fig. 3.22). In the first column is the present state of the Q output, and in the next two columns are the input conditions of the R and S lines. The last column indicates the new Q output after the input signal has been fed in. The first column can also be said to indicate the flip-flop's output at time t and the last column at time $t + \tau$, where τ is an incremental period of time needed for the flip-flop to change its state. The table can be reorganized to simply indicate what input conditions are needed to make the output change in the fashion desired (Fig. 3.23). This type of table can be called an

Q(t)	R	S	Q (t + τ)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	INDETERMINATE
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	INDETERMINATE

Fig. 3.22 Transition table for a R-S flip-flop.

STATE CHANGE		REQUIRED INPUT	
FROM	TO	R	S
0	0	d*	0
0	1	0	1
1	0	1	0
1	1	0	d*

*DON'T CARE STATE.

Fig. 3.23 The excitation table for the R-S flip-flop.

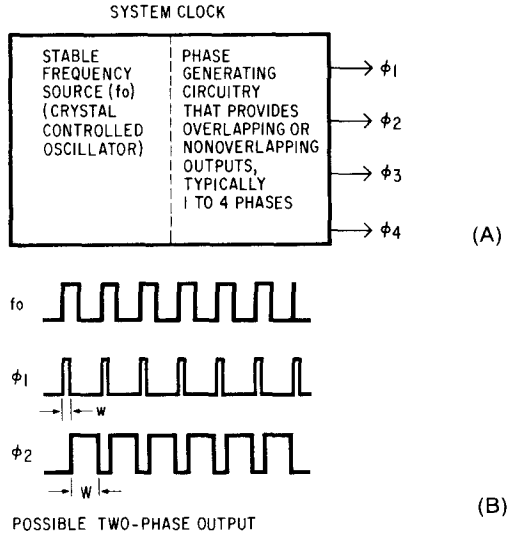


Fig. 3.24 In a computer system, the source of all the timing signals is often referred to as the system clock (a). The clock often generates either a square or pulsed type of signal in from one to four phases (b).

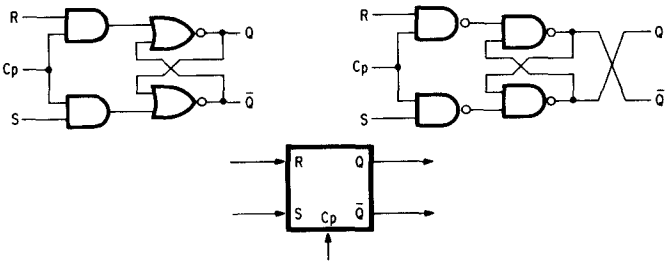


Fig. 3.25 The R-S flip-flop can readily be modified to respond to inputs only when a clock is present.

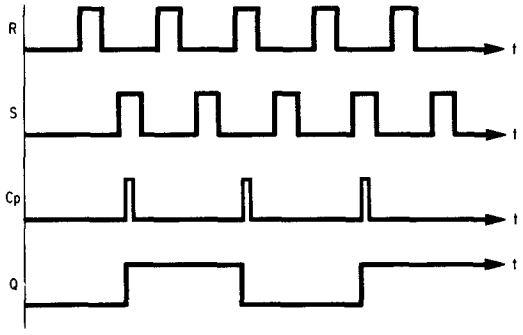


Fig. 3.26 The timing diagram of the clocked R-S flip-flop shows that the output of the flip-flop will only change when an input and a clock pulse are present at the inputs.

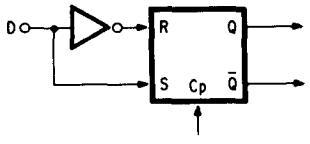


Fig. 3.27 By adding an inverter on the R input of an R-S flip-flop, the circuit becomes a D flip-flop.

excitation table. In the table, d indicates don't-care states, where it doesn't matter whether the signal is a ONE or ZERO.

Let's step back for a minute and look at how the NOR gate flip-flop works. To start with, assume an output condition, say $Q = 1$, $\overline{Q} = 0$. If a signal of $R = 0$, $S = 0$ is input, gate A has an input of 00 and its output stays at 1, and gate B has an input of 01 and its output stays at 0. However, if the input changes to $R = 1$, $S = 0$, the input to gate A becomes 10 and its output changes to 0. Now, the input to gate B changes to 00 and its output changes to 1. The time it takes for gate A to change can be called $\Delta 1$ and for gate B $\Delta 2$. The total flip-flop transition time previously called τ can be represented by $\Delta 1 + \Delta 2$. Since these times are often identical: $\tau = 2\,\Delta 1$.

An equivalent to the NOR circuit built with NAND gates and two inverters works similarly. With both of these circuits, any time a signal comes along on the input lines, the flip-flop will trigger. By adding a gate at each input that can be synchronized to some form of timing generator in the overall system, however, you can effectively lock out signals that occur outside of the window you allow. The timing generator most often used is called the *system clock*. It is just a circuit that generates a square wave or pulse train at a precise frequency and pulse width (Fig. 3.24). The width of the positive portion of the square wave or pulse (W or w, respectively) is the window. Figure 3.25 shows the actual modification to the flip-flop circuit. Now no matter what happens on the R and S inputs, the outputs will remain unaffected until the clock signal goes HIGH. The timing diagram of Fig. 3.26 shows what happens when the clock signal locks out unwanted set and reset signals. The flip-flop can change only when the clock pulse is present.

If an inverter is now added to the clocked R-S flip-flop, as shown in Fig. 3.27, another type of flip-flop with only one input can be built. This type of flip-flop, called a *D flip-flop* or *latch*, is used very often for temporary storage of data. The inverter prevents both the R and S inputs from getting logic one inputs simultaneously. Also, there are only two possible input conditions—a 0 or 1 on the D line. As you can see from the transition table of Fig. 3.28, whenever the input is 1, the output will become 1 after the clock pulse comes along.

Sometimes the basic R-S flip-flop is needed, but it must have the capability to respond to an $R = 1$, $S = 1$

D	Q_t	$Q_t+\Delta$
0	0	0
0	1	0
1	1	1
1	0	1

Fig. 3.28 The response of the D flip-flop is described by this transition table.

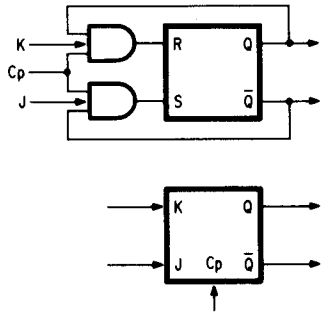


Fig. 3.29 A J-K flip-flop can be created by adding two feedback inputs to the AND gates of the clocked R-S inputs of the circuit in Fig. 3.25.

condition. By adding two feedback inputs to the AND gates of the clocked R and S inputs, as shown in Fig. 3.29, another flip-flop called a J-K flip-flop can be created. The K input (clear) replaces the reset function and the J input does the same job as the S input. In general, the J-K flip-flop functions similarly to the R-S except that both inputs can become 1 simultaneously. When this occurs, whatever the output states are, they get reversed. The J-K flip-flop must have a clock signal to operate, just like the clocked R-S flip-flop.

If Q = 0, K = 0, J = 0, and a clock pulse is applied, nothing happens since the inputs are 0. When the inputs change to JK = 01 and a clock pulse comes along, nothing will happen if Q = 0 since the flip-flop is already reset. However, if Q = 1, the AND gate connected to the R input has a 1 output when the clock pulse occurs and thus triggers the reset (Q goes to 0). Similarly, when Q = 0, JK = 10, and a clock pulse is applied, the AND gate feeding the S input has a 1 output, thus causing the Q output to go to 1. However, if the flip-flop originally

J	K	Q_t	$Q_t+\Delta$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	1
0	1	1	0
1	0	1	1
1	1	1	0

Fig. 3.30 The excitation table of the J-K flip-flop shows all possible outputs for all possible input conditions, even a 1-1 input for the J-K input terminals.

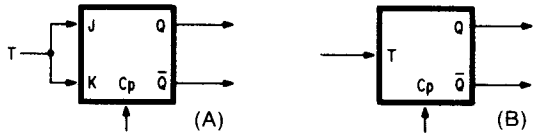


Fig. 3.31 By connecting the J and K inputs together, a T flip-flop can be made from a J-K flip-flop (a). The T flip-flop (b) changes its output state every time an input signal coincides with the clock.

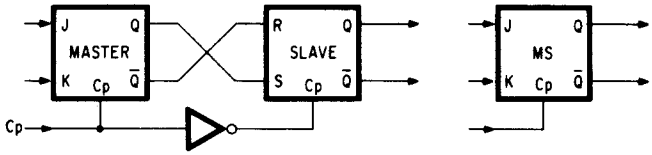


Fig. 3.32 If a J-K and an R-S flip-flop are cascaded, a master-slave J-K flip-flop is created. It is often used when triggering is desired only on the falling edge of an input pulse.

had an output of Q = 1 and it got inputs of JK = 10 and a clock pulse, nothing gets through the AND gates and the output remains at Q = 1. The special case of JK = 11, Q = 0, and an applied clock pulse lets a 1 feed into the set input, thus reversing the flip-flop's output, setting Q = 1. If with Q = 1, the same input conditions are applied, a 1 feeds into the R input, also causing the flip-flop to reverse its output, and resetting Q to 0. A complete excitation table is shown in Fig. 3.30.

By connecting the J and K inputs together, the flip-flop can be made to reverse state every time a 1 input is applied along with a clock pulse (Fig. 3.31). This type of flip-flop, called a clocked T flip-flop because the reversing action is called *toggling*, is very handy when counting circuits are to be made.

The last type of flip-flop to be discussed here is a combination of two flip-flops, as shown in Fig. 3.32. When two flip-flops are used to act as one flip-flop, the arrangement is referred to as a *master-slave combination*. In this circuit, when the clock input is a logic 1 to the master flip-flop, the J-K inputs control its output. However, nothing takes place at the slave output since the inverter changes the clock to logic 0 for the slave flip-flop. When the master's clock goes to logic 0, the slave clock input goes to logic 1, and the output of the slave flip-flop obeys its R-S control inputs. Thus, the J-K/R-S master-slave flip-flop is said to trigger on the falling edge of the input clock pulse.

Cascaded Flip-Flops Count

If flip-flops are connected as shown in Fig. 3.33, they can be used to count the number of input pulses, divide frequencies, shift data, or just store data. Circuits like those shown are indispensable for all computer applications. Let's take a quick look at how some of them work.

Starting with the counting circuitry of Fig. 3.34 and the timing and output diagram of Fig. 3.35, assume that all the flip-flops are J-K master-slave types. If you

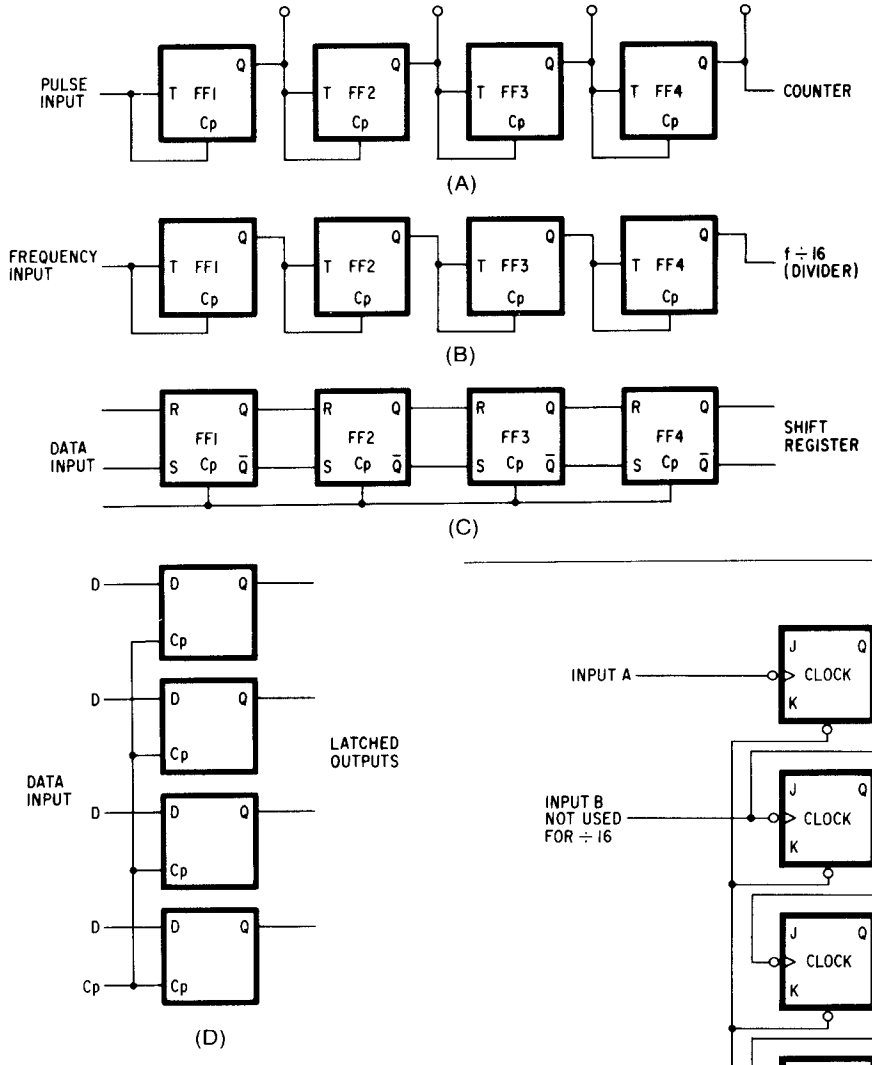


Fig. 3.33 Several flip-flops can be connected together to (a) count the number of input pulses, (b) divide frequencies, (c) shift data, and (d) store data.

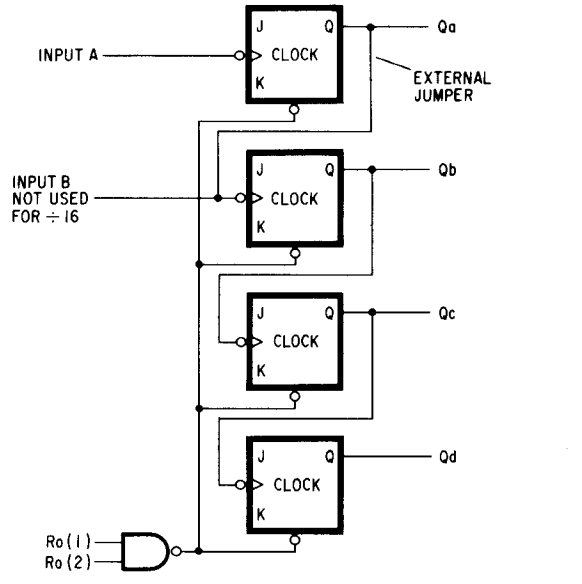


Fig. 3.34 Detailed wiring diagram of a divide-by-16 counter circuit, a 7493.

look at all four outputs of the counting circuit at the same time, the digital signals that appear on those lines can represent a four-bit code for the binary numbers 0000 to 1111 (decimal 0 to 15). Before any pulses arrive, all flip-flops are set to zero, and if you look at their voltage outputs, all outputs would be LOW (logic 0). After the first count pulse, FF1 changes state and its output is a 1. When the next count pulse comes, FF1 changes its state again, and in doing so, it triggers FF2 so that the output of FF2 goes to logic 1 and FF1's output is at logic 0. The next count pulse makes FF1's output rise to logic 1 and has no other effect on the remaining flip-flops, since J-K flip-flops only respond on the falling edge of a waveform—not on the rising edge.

The fourth pulse has a more devastating effect that ripples all the way to FF3. On the falling edge of count pulse 4, FF1's output goes to logic 0, thus triggering FF2 also to change its output from 1 to 0. The change

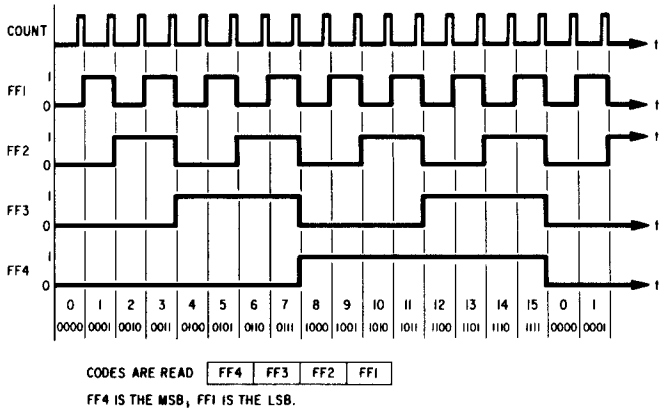
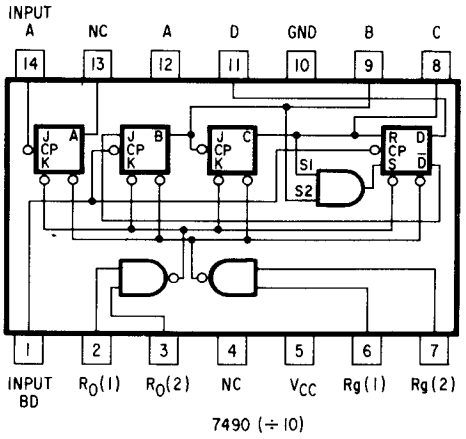


Fig. 3.35 Timing diagram and output pattern for a four-stage counter.



Flip-flops can also be used to transfer data from one place to another. If several flip-flops are cascaded so that both the inputs and both the outputs are connected, as shown in Fig. 3.37, data entered on one end can be pulled through to the other end. This type of circuit is called a *shift register*. There are about seven different varieties of shift registers: unidirectional shifters, bi-directional shifters, recirculating shifters, parallel input/parallel output, parallel input/serial output, serial input/parallel output, and serial input/serial output.

Basically, a serial shift register is like a hollow tube filled with Ping-Pong balls (Fig. 3.38). If a new ball is pushed in one end, the far ball inside the tube will be forced out. Let's use shaded balls to represent zeros

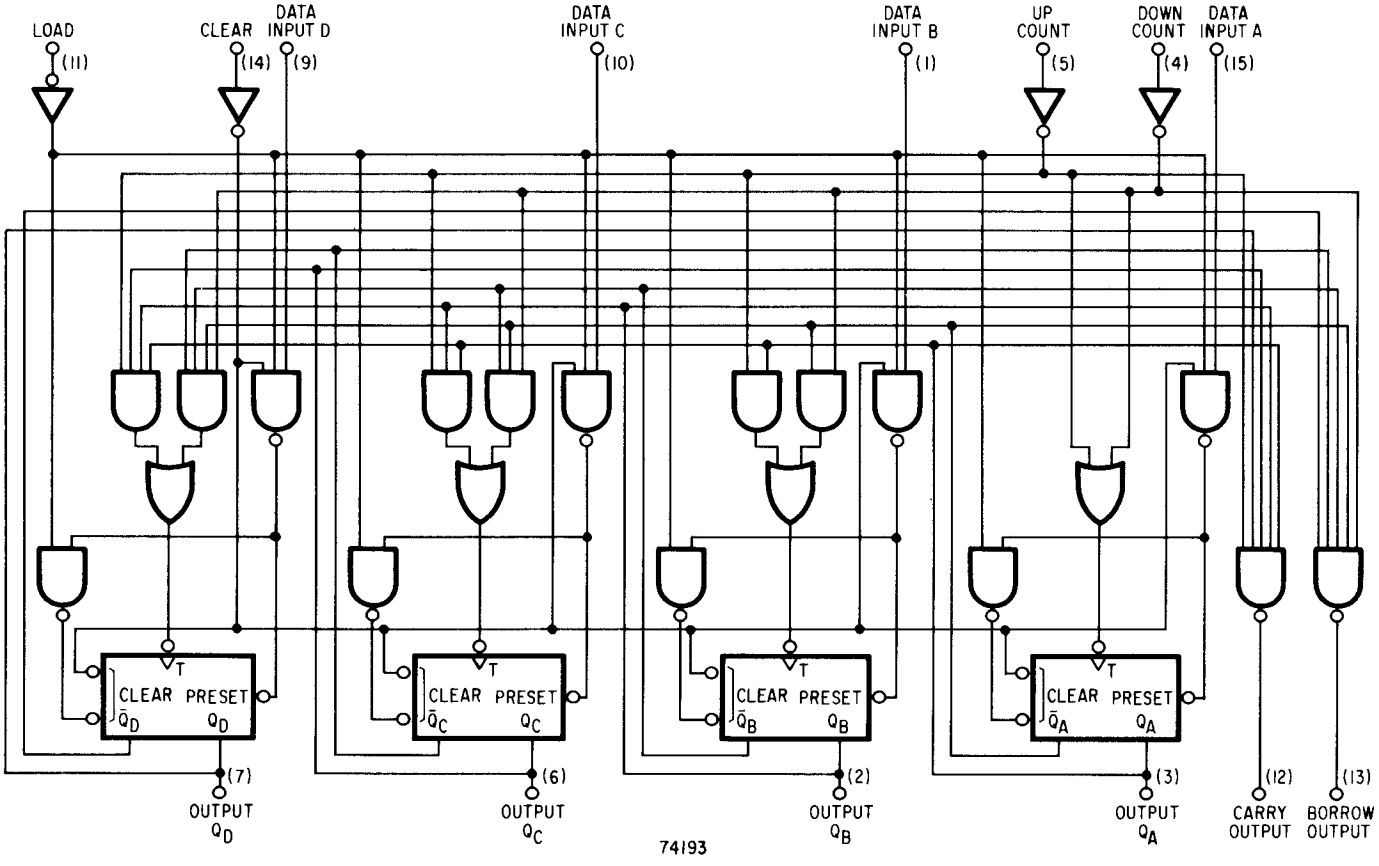


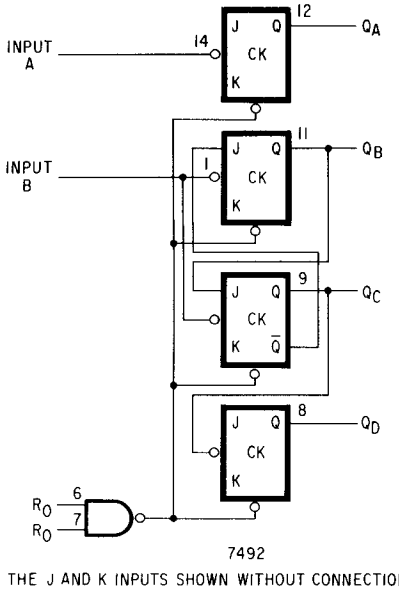
Fig. 3.36 Some commonly used multiple flip-flop counter circuits.

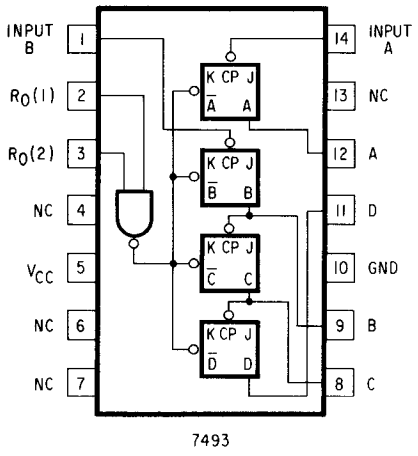
in FF2 causes FF3 to reverse its output state and make its output logic 1. This rippling effect continues with all the ensuing count pulses, and, unless the pulses stop, the counting circuit continues to cycle from 0000 to 1111, back to 0000 and so on.

If you measure the frequency of the signals on the output from each flip-flop, you'll note that each successive flip-flop divides the input clock frequency by 2. So if you needed to divide by 2, 4, 8, or 16, all you have to do is employ a few flip-flops and just use the tap you want off the chain. There are many different types of counting/dividing circuits available; some of the most common types are shown in Fig. 3.36.

and white balls to represent ones. Inside the tube then, is the bit pattern 1010101. If a zero is entered on the left, then all bits shift right by one place, and the new bit pattern is 0101010 (the one that was forced out is discarded). When the output of the register is fed back to the input, the data will not be lost, but rather go in circles—thus the term *recirculating*. Some registers include circuitry that permits control of the direction of data flow—from left to right, right to left, serial to parallel, or parallel to serial. Commonly used devices are shown in Fig. 3.39.

Large arrays of flip-flops can be ganged together to make memories for computer systems and temporary





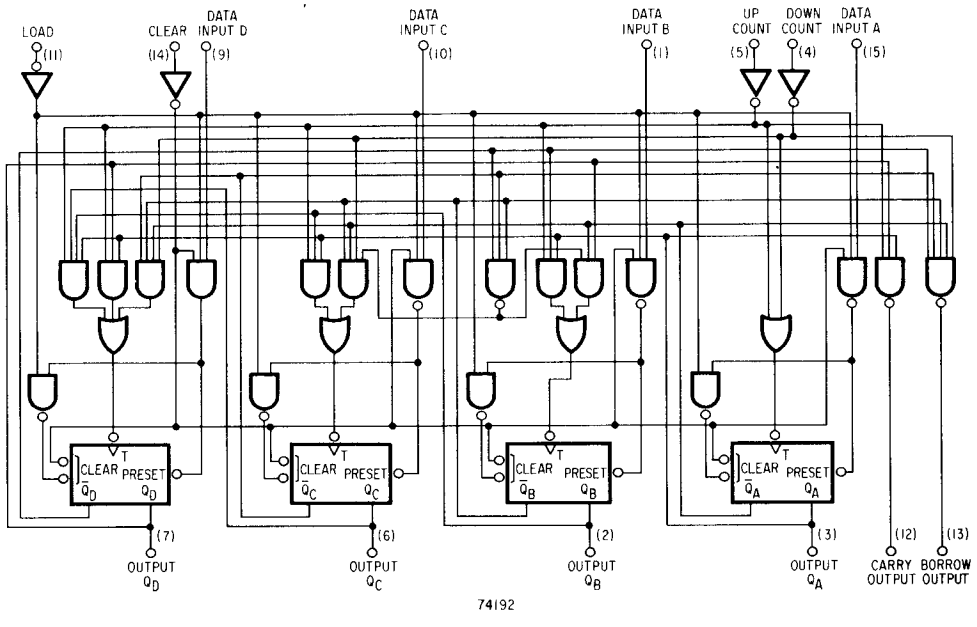


Fig. 3.36 (cont'd) Some commonly used multiple flip-flop counter circuits.

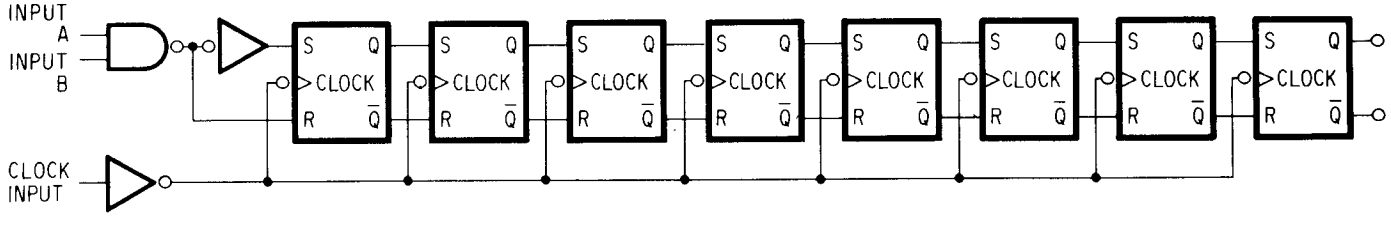


Fig. 3.37 When the output of one flip-flop is directly connected to the input of another, and another, the cascaded array of flip-flops is called a shift register.

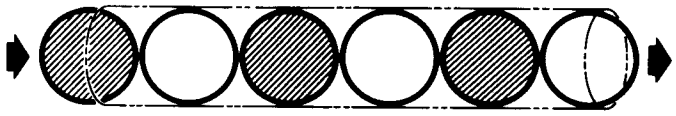
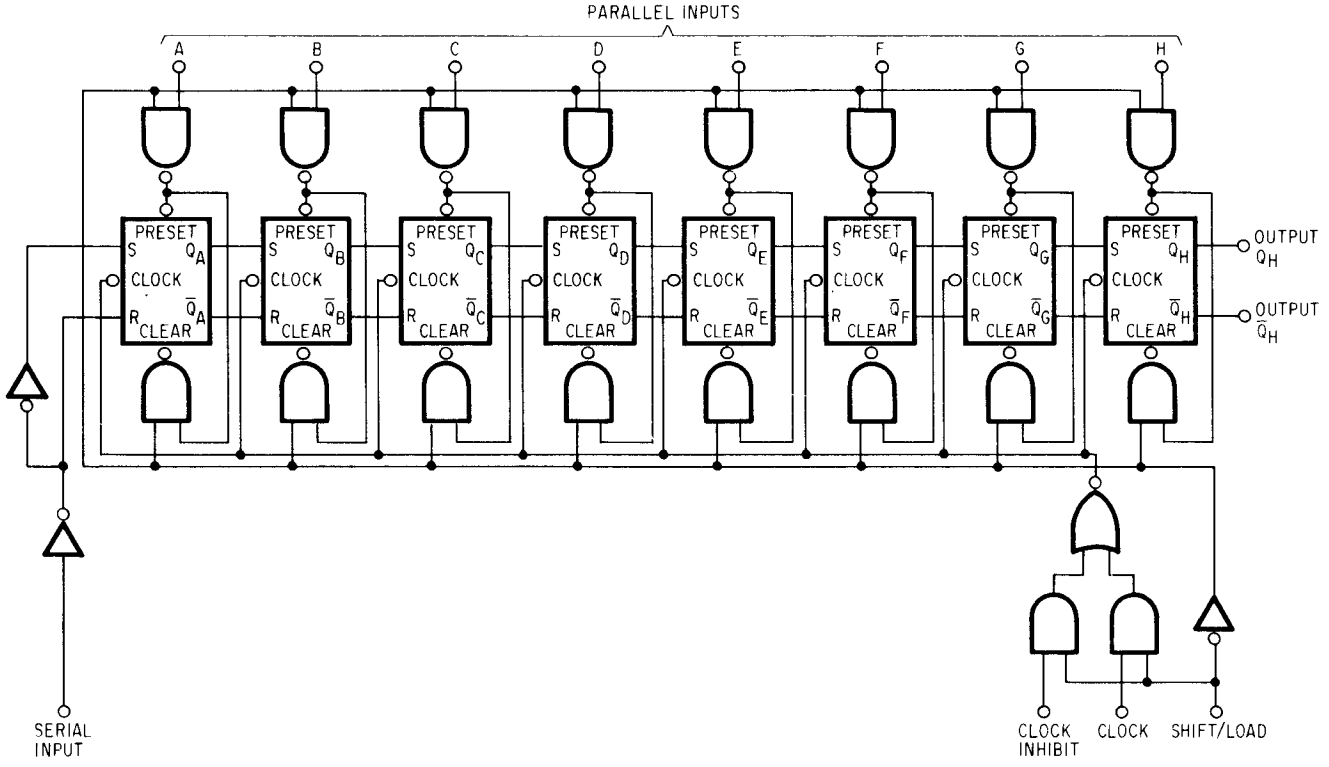


Fig. 3.38 A serial shift register can be likened to this tube filled with black and white Ping-Pong balls. Whatever comes in on one end forces something out the other.

turers, and probably close to the same number of CMOS, PMOS, and NMOS circuits. In the TTL family of circuits, the most prevalent series is the 7400 series and its different variations—the 74L00, 74S00, 74H00, and 74LS00. They are designed to function over a 0 to 70°C temperature range and from a 5-V power supply.



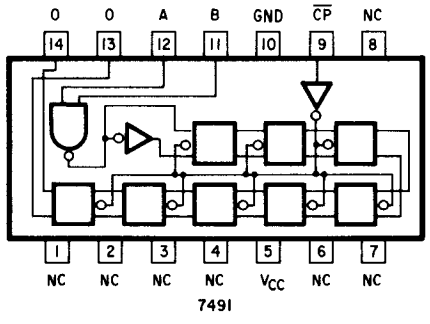


Fig. 3.39 Some commonly used shift registers from the TTL circuit family.

data storage areas for data being transferred or to be displayed. The D flip-flop is ideal for temporary storage, and arrays of four, six, or eight flip-flops in a single package are commonly available.

Modern digital design treats these logic circuits as building blocks; just consider their function and the number of inputs and outputs. There are over 200 standard TTL circuits available from various manufac-

Standard TTL and 74LS circuits are listed in Appendix C. The other 74 series families are performance variations of the original 7400 series: L refers to low power, S to Schottky, H to high speed, and LS to low-power Schottky. Don't get too confused by the part numbers if you see a 5400 series; the parts are the same as for the 7400 series except that they can operate over a wider temperature range. Standard CMOS circuits include the 4000 series and a 74C series; both are available from many vendors. Typical packages and sizes of circuits are shown in Fig. 3.40, and a list of most available types is given in Appendix B.

Connecting the circuits to each other requires some special checks and considerations. Starting with TTL, here are some guidelines to follow when using logic circuits:

TTL

1. Each circuit requires a +5 V dc source and a ground connection.
2. Use a capacitor of about 0.01 μF connected across the power supply and ground lines for every three to five packages.

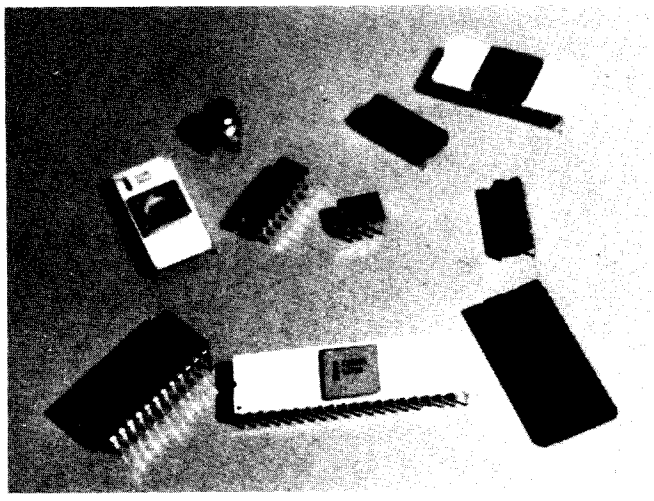


Fig. 3.40 Typical IC packages used in computer systems.

3. Each logic input or output can handle up to a maximum of 10 TTL inputs or outputs; don't overload a gate or other circuit.

CMOS

1. Each circuit usually requires a +5 to +15 V dc source and a ground connection.
2. Same as Rule 2 for TTL.
3. Each logic circuit input or output can handle about 100 or more other CMOS circuit inputs or outputs, but since they are designed for low power operation, they may not be able to handle the comparatively high current of TTL circuits.

NMOS and PMOS

1. Check the specifications carefully; most PMOS and NMOS circuits made today feature TTL compatibility, which means that they operate from a 5 V supply and can accept or deliver enough voltage and current to handle one TTL input. Some of the older devices require special voltages and interface circuits.
2. Same as Rule 2 for TTL.
3. Since these circuits have very different interconnection requirements and drive capabilities, read the data sheets carefully.

A newer form of TTL and MOS logic circuit uses what is now referred to as *three-state logic*. This logic form has, in addition to the normal HIGH and

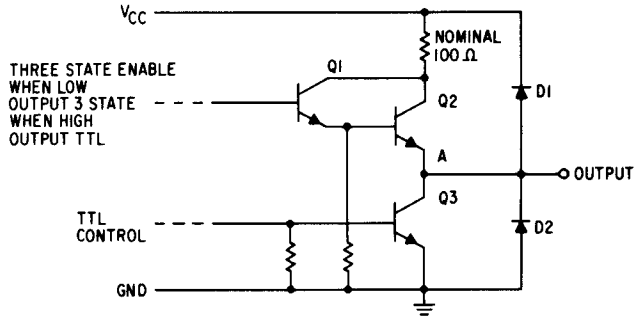


Fig. 3.41 A typical circuit for three-state logic devices.

LOW states, a third state that has no voltage associated with it but instead merely presents a very high resistance to the signal lines connected to the line. The high impedance reduces the current that flows in the circuit and thus minimizes the power required and the loading caused by many circuits being connected together.

Without three-state capability, a normal gate such as a 7400 can have up to 10 other logic circuit lines connected to its output before the output becomes overloaded. Each connected line requires 40 μA for a HIGH level and −1.6 mA for a LOW level. So, when 10 lines are ganged together, the total current can reach 400 μA for a HIGH signal and −1.6 mA for a LOW signal. However, what is the purpose of that much current if the circuits aren't really used all the time? That's where three-state logic comes in. By disabling the output of the gate driving the 10 lines or the inputs of the gates, almost no current is used until the signals are actually needed. This arrangement permits loads of up to 100 logic circuits to be connected to a single gate's output without overloading the gate.

The typical output circuit for three-state circuits is shown in Fig. 3.41. When the base of Q1 is kept near zero, Q1 does not conduct and thus keeps Q2 turned off, which, in turn, leaves point A in a "floating" state. The impedance of the point is thus very high (equivalent to that of a reverse-biased diode). If the base of Q1 is raised to near V_{CC}, Q1 is turned on and it, in turn, turns on Q2, which then brings point A to a voltage (logic 1) near V_{CC} also (assuming that the input to the base of Q3 is near zero). Now, however, if the base of Q3 is brought HIGH, Q3 conducts and brings the voltage at point A down to near zero (logic 0). At any time, the input to Q1 can be brought LOW, thus forcing point A back to the high-impedance third state and disabling the logic circuit.

CHAPTER 4

The Basic S-100 Bus and the Computer Mainframe

The computer systems that we'll look at throughout the rest of the book use the same interconnect structure (bus) between the various boards used to compose the computer. This bus structure, as mentioned in Chapter 1, was started by MITS (now part of Pertec Computer Co.) in its Altair 8800 microcomputer and has been adapted by over 50 other manufacturers, who also offer products that connect to the bus.

Basically, the Altair bus (more commonly referred to as the S-100 bus) consists of 100 parallel lines either made of wire or an etched pattern on a printed-circuit card. When made in the form of a printed-circuit card, such as the one shown in Fig. 4.1, taps are made every inch or so for a connector (Fig. 4.2) to be inserted so that the various computer cards can be plugged in and thus interconnected by the bus. The printed-circuit board that has all the connectors on it (typical boards hold from four to 22 connectors) is often referred to as the "motherboard." Although both the Altair and Imsai motherboards use 100-pin connectors, there are some differences that prevent the same connector from being plugged on both. The pin-to-pin spacing on the rear of the connectors for the Altair and Imsai motherboards is 0.125 in., while the row-to-row spacing for the Altair motherboards is 0.14 in. and 0.25 in. for the Imsai motherboard.

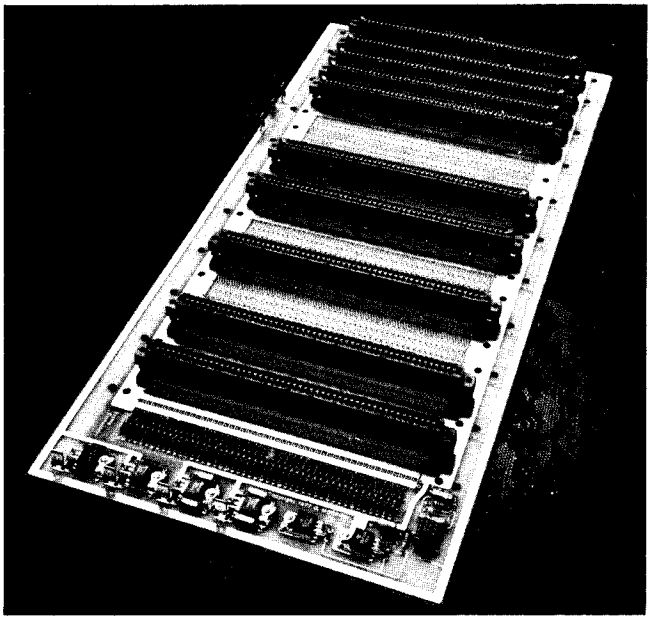


Fig. 4.1 A typical S-100 bus motherboard. (*Courtesy* Sigma Computers)

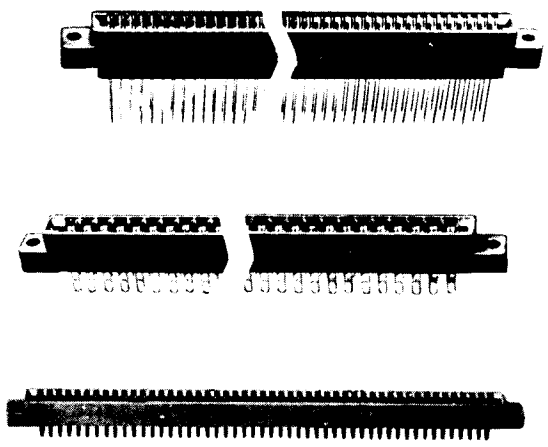


Fig. 4.2 Commonly used edge-card connectors for the S-100 boards. (*Courtesy* Vector Electronics)

Almost every one of the 100 pins of the bus has a predefined function. Table 4.1 lists the functions as defined by MITS for their 8800b computer system. (A new standard definition for the bus has been sponsored by the Institute of Electrical and Electronic Engineers. See Appendix D for a capsule summary of the proposed standard.) Many of the signal lines will have no meaning to you at this point, but as our use of the bus grows, you'll need all the functions. Basically, the pins of the connector can be divided into five groups:

1. Power and ground lines
2. Address lines
3. Data lines
4. Control signal lines
5. Undefined lines (spares)

Table 4.1 Definitions of the S-100 Pins

The Altair 8800b system bus has 100 lines. These are arranged 50 on each side of the plug-in boards. The following general rules apply to the Altair 8800b bus.

SYMBOLS — a "P" prefix denotes a processor command/control signal. "S" denotes a processor status signal.

LOADING — All inputs to a card are loaded with a maximum of one TTL low power load except for the Turnkey Module.

LEVEL — All bus signals except those for the power supply are TTL compatible. Signals whose names are barred (DO DSBL, for example) are active low (0 volts). All others are active high (+5 volts).

In the listing below, those signal names accompanied by * are ineffective or not used in the Altair 8800b Turnkey computer.

Number	Symbol	Name	Function
1	+8V	+8 volts	Unregulated input to 5 volt regulators
2	+18V	+18 volts	Positive unregulated voltage
3	XRDY	External Ready	For special applications: pulling this line low causes the processor to enter a wait state and allows the status of the normal ready line (PRDY) to be examined.
4	VI 0	Vectored Interrupt	Line 0
5	VI 1	Vectored Interrupt	Line 1
6	VI 2	Vectored Interrupt	Line 2
7	VI 3	Vectored Interrupt	Line 3
8	VI 4	Vectored Interrupt	Line 4
9	VI 5	Vectored Interrupt	Line 5
10	VI 6	Vectored Interrupt	Line 6
11	VI 7	Vectored Interrupt	Line 7
12	XRDY2	Extra Ready line	For special applications.
13 to 17			To be assigned
18	$\overline{\text{STA DSB}}$	$\overline{\text{STATUS DISABLE}}$	Puts buffers for the 8 status lines in their high-impedance third state. In this state, no information can be transferred.
19	$\overline{\text{C/C DSB}}$	$\overline{\text{COMMAND/CONTROL DISABLE}}$	Puts the buffers for the 6 command/control lines in their high-impedance third state.
*20	UNPROT	UNPROTECT	Input to the memory protect flip-flop.
*21	SS	SINGLE STEP	Indicates that the computer is in the process of performing a single step.
22	$\overline{\text{ADD DSB}}$	$\overline{\text{ADDRESS DISABLE}}$	Puts the buffers for the 16 address lines in their high-impedance third state.
23	$\overline{\text{DO DSBL}}$	$\overline{\text{DATA OUT DISABLE}}$	Puts the buffers for the 8-data out lines in their high-impedance third state.
24	$\phi 2$	Phase 2 clock	
25	$\phi 1$	Phase 1 clock	
26	PHLDA	Hold Acknowledge	Processor output signal which appears in response to the HOLD signal indicates that the data and address buffers will go to the high-impedance third state.
27	PWAIT	WAIT	Processor output indicates that the processor is in the WAIT state.
28	PINTE	Interrupt Enable	Indicates interrupts are enabled; displays contents of the CPU interrupt flip-flop. This flip-flop may be set or reset by the EI or DI instructions. When reset, it prevents the CPU from acknowledging interrupt requests.
29	A5	Address Line 5	
30	A4	Address Line 4	
31	A3	Address Line 3	
32	A15	Address Line 15	
33	A12	Address Line 12	
34	A9	Address Line 9	
35	DO1	Data Out Line 1	
36	DO0	Data Out Line 0	
37	A10	Address Line 10	
38	DO4	Data Out Line 4	
39	DO5	Data Out Line 5	
40	DO6	Data Out Line 6	
41	DI2	Data In Line 2	
42	DI3	Data In Line 3	
43	DI7	Data In Line 7	
44	SM1	M1	Status output that indicates that the processor is in the fetch cycle for the first byte of an instruction.
45	SOUT	OUT	Indicates that the address bus contains the address of an output device and the data bus will contain the output data when PWR is active.
46	SINP	INP	Indicates that the address bus contains the address of an input device.

Table 4.1 (cont'd) Definitions of the S-100 Pins

Number	Symbol	Name	Function
47	SMEMR	MEMR	Indicates that the data bus will carry memory read data.
48	SHLTA	HLTA	Acknowledges a HALT instruction.
49	CLOCK	Clock	Inverted output of the 2 MHz oscillator that drives the clock.
50	GND	Ground	
51	+8V	+8 volts	
52	−18V	−18 volts	
*53	SSW DSB	SENSE SWITCH DISABLE	Disables the data input buffers so that the inputs from the sense switches may be strobed onto the bidirectional data bus at the processor.
54	EXT CLR	EXTERNAL CLEAR	Clear signal for I/O devices.
55	RTC	Real Time Clock	
56	STSTB	STATUS STROBE	
*57	DIG1	DATA INPUT GATE #1	Front panel control line. It is an output signal from the Display/Control logic that determines which set of Data Input drivers have control of the CPU board's bidirectional data bus. If DIGI is high, the CPU drivers have control; if low, the Display/Control logic drivers have control.
*58	FRDY	Front Panel READY	
59 to 67		To be assigned	
68	MWRT	MEMORY WRITE	Indicates that the current data on the Data-Out bus is to be written into the memory location currently on the address bus.
69	PS	PROTECT STATUS	Indicates the status of the memory protect flip-flop on the memory board currently being addressed.
*70	PROT	PROTECT	Input to the memory protect flip-flop on memory board currently being addressed.
*71	RUN	RUN	Indicates that the RUN/STOP flip-flop is reset.
72	PRDY	READY	Input that controls the run state of the processor. If the line is pulled low the processor will enter a WAIT state until it is released.
73	PINT	INTERRUPT REQUEST	The processor recognizes a request on this line at the end of the current instruction or while halted. If the processor is in the HOLD state or the Interrupt Enable flip-flop is reset it will not honor the request.
74	PHOLD	HOLD	Requests the processor to enter the HOLD state. This allows an external device to gain control of the bus as soon as the processor has completed its current machine cycle.
75	PRESET	RESET	While activated, the contents of the program counter are cleared and the instruction register is set to 0.
76	PSYNC	SYNC	Provides a signal to indicate the beginning of each machine cycle.
77	PWR	WRITE	Used for memory write or I/O control. Data on the data bus are stable while PWR is active.
78	PDBIN	DATA BUS IN	Indicates to external devices that the data bus is in input mode.
79	A0	Address Line 0	
80	A1	Address Line 1	
81	A2	Address Line 2	
82	A6	Address Line 6	
83	A7	Address Line 7	
84	A8	Address Line 8	
85	A13	Address Line 13	
86	A14	Address Line 14	
87	A11	Address Line 11	
88	DO2	Data Out Line 2	
89	DO3	Data Out Line 3	
90	DO7	Data Out Line 7	
91	DI4	Data In Line 4	
92	DI5	Data In Line 5	
93	DI6	Data In Line 6	
94	DI1	Data In Line 1	
95	DI0	Data In Line 0	
96	SINTA	INTA	Acknowledge signal for interrupt request.
97	SWO	WRITE OUT	Indicates that the operation in the current machine cycle is a WRITE memory or output function.
98	SSTACK	STACK	Indicates that the address bus holds the push down stack address from the Stack Pointer.
99	POC	Power-On-Clear	
100	GND	Ground	

The power and ground lines provide the voltage needed by the computer. There are six pins assigned to provide the three unregulated voltages and a ground return: Pins 1 and 51 have the +8 V supply, pins 50 and 100 have the ground return, pin 2 has the +18 V supply, and pin 52 has the −18 V supply. There are also 16 address lines that are used to provide the memory address to the memory circuits. These lines are scattered such that lines A0 to A2 are on pins 79 to 81; A3 to A5 are on pins 31, 30, 29; A6 to A8 are on pins 82 to 84; A9 is on pin 34; A10 is on pin 37; A11 is on pin 87; A12 is on pin 33; A13 and A14 are on pins 85 and 86; and A15 is on pin 32.

For the transfer of instructions and data to and from the microprocessor, 16 other lines are put aside—eight for data that feed into the processor (DI0 to DI7) and eight for data that are fed out of the processor (DO0 to DO7). These lines are also scattered on the connector, with DI0 on pin 95, DI2 and DI3 on pins 41 and 42, DI4 to DI6 on pins 91 to 93 and DI7 on pin 43. The DO lines are set up so that DO0 is on pin 36, DO1 is on pin 35, DO2 and DO3 are on pins 88 and 89, DO4 to DO6 are on pins 38 to 40, and DO7 is on pin 90.

Before looking at the fourth group, the control signal lines, let's quickly look at the pins left as spares, with no preassigned functions. Connector pins 13 to 17 and 59 to 67 have no assigned functions and can be used by you to carry any signal desired when a custom circuit is designed. However, some manufacturers of S-100 compatible boards have already selected a few of these lines so that their own boards can take advantage of these available signal paths. So far, 52 of the 100 pins have been used by basic interconnections to the boards. The remaining 48 bus lines are used to carry timing and control signals for the various sections of the computer.

Pins 4 to 11 of the connector are assigned for interrupt signals. These signals, generated by external conditions, can interrupt the processor so it stops whatever it is doing and starts to perform a specific program in answer to the signal. The clock signals generated by the processor's timing circuit are available on pins 24 and 25 as phase 2 and phase 1, respectively, and on pin 49 (the inverse of phase 2).

Two pins are assigned an external ready function, which permits an external signal fed in to either of these computer lines (pins 3 and 12) to stop the microprocessor (when the signal is logic 0), force it to enter a WAIT (pause) state and allow the status of the processor's normal ready line (PRDY) to be examined.

Each of the remaining 35 control lines has a different function and the simplest way to examine what each line does is to go in numerical sequence starting with pin 18. This is the status disable line and when the line is brought to a logic 0 condition, it disables the buffers for the eight status bits (pins 44 to 48, 96 to 98) and puts the buffer outputs in their high-impedance state, thus preventing any status information from being transmitted from the processor board to any other part of the computer. Pin 19 serves as the command control disable line. When it is brought to a logic 0 condition it places the three-state buffers for the six command/control lines (pins 26 to 28 and 76 to 78) in their high-impedance third state, thus preventing the lines from performing any command or control functions.

Pin 20 is a memory unprotect/protect control line—when kept at a logic 1, it permits information to be written into or read from the computer's memory; when brought to a logic 0, it prevents information from being written into the memory but information can still be read out. This control line is usually manually set to either its 1 or 0 state by a switch. Connector pin 21 buses a signal that tells the processor to execute only a single instruction (often known as single step). When pin 22 is brought to a logic 0 state, it causes the 16 address buffer outputs on the processor card to go to the high-impedance third state, thus preventing the memory from being addressed by the processor. Pin 23, when brought to logic 0, performs the same job on the eight DO buffer outputs.

One of the six command/control outputs from the processor, pin 26 goes to logic 1 when the processor receives a HOLD signal, otherwise pin 26 stays at a logic 0 level. Pin 27, also a processor output, goes to a logic 1 to indicate when the processor has been placed in the WAIT state; otherwise the line stays at a logic 0 level. The interrupt enable line, pin 28, is also an output from the processor. When at a logic 1, it indicates that the processor interrupt is enabled and that an input to any of pins 4 to 11 or pin 73 will interrupt the processor. This line (pin 28) can be controlled by the EI and DI instructions, which set the processor's interrupt flip-flop to logic 1 or 0, respectively. When at logic 0, it prevents the processor from acknowledging interrupt requests.

The next five pins (44 to 48) are status outputs from the processor. Pin 44, when at logic 1, is used to indicate that the processor is in the fetch portion of an instruction cycle. Pin 45, when at logic 1, indicates that the address bus contains the address of an output device and the data bus will contain the output data when PWR line (pin 77) is at logic 0. Pin 46, when at logic 1, indicates that the address bus contains the address of an input device. Pin 47, when at logic 1, indicates that the data bus has data that have been read from memory. And pin 48, when at logic 1, acknowledges that the processor received a HALT instruction, and is stopping.

There are still 21 control lines left, so let's forge onward. Pin 53, when at logic 0, disables the data input buffers to the processor so that the inputs from manually set control panel sense switches can be loaded onto the processor's data bus. On pin 54 is an EXTERNAL CLEAR signal, which when brought to logic 0 clears input/output devices. Pin 55 is used for a REAL-TIME CLOCK signal, that can be used to time events external to the computer. The STATUS STROBE sig-

nal on pin 56 is an output from the central processor board that tells the rest of the computer system that status information is available on the status lines.

A control line whose state is determined by front-panel operations, pin 57, combines with other signals on the CPU card to enable the data-input buffers so data can flow from the data bus to the input of the processor. Pin 58 of the bus handles a signal coming from the front panel that indicates to the CPU when the computer is ready for operation. To tell the memory that data are to be written into the array, pin 68 carries a MEMORY WRITE strobe signal generated by the combination of the signals on bus pins 45 and 77 or the front-panel DEPOSIT switch. Pin 69, when at logic 0, indicates that the state of a program-controlled flip-flop on MITS memory boards is in the PROTECT mode, thus preventing data from being loaded into the memory. When at logic 1 the line indicates that the memory is unprotected.

The next pin, connector pin 70, provides a manual control path for a front panel switch to the memory protect control flip-flop on the MITS memory boards. This permits a front-panel switch to protect or unprotect the memory. Pin 71 shows the state of the front panel's RUN flip-flop. When the line is at logic 0 the processor is in the halted state, and when at logic 1 the processor is running a program. The next line, pin 72, is a signal generated by the front panel and it controls the RUN state of the processor. When pulled to logic 0, the line forces the processor to enter a WAIT state and stay there until the line goes to logic 1. When at logic 1, the line permits the processor to run normally.

Pin 73 has a signal that controls the interrupt capability of the processor. When brought to logic 0 the processor recognizes it as a request so that at the end of its current instruction the processor goes to handle the interrupt. If the processor is in the HOLD state or the interrupt enable flip-flop in the processor is set, the request will not be honored. When kept at logic 1, the line has no effect on the processor. The next line, pin 74, is used to request that the processor enter the HOLD state when brought to logic 0. This permits another device to control its bus as soon as the processor has finished its current machine cycle. Clearing the processor, line 75 initializes the program counter and instruction register of the processor when brought to logic 0.

An output from the processor, pin 76, provides a pulse to the rest of the system each time the processor begins a machine cycle. Pin 77 is used by the processor to transmit the memory write output operations. Data on the data bus are stable when pin 77 goes to logic 0. The next line, pin 78, handles an output signal from the processor. When at logic 1, it indicates to external devices that the data bus is in an INPUT mode. Skipping down to pin 96, this processor output line, when at logic 1, acknowledges that the processor has received an interrupt request. Pin 97, also an output from the processor, is used to indicate that the operation in the current machine cycle is a write to memory or an output function when at logic 0. The next signal, a processor output on pin 98, when at logic 1, indicates that the address bus holds the push-down stack address from the stack pointer. Finally, the last signal line, pin 99, is an output from the processor that occurs when power is turned on. When the line goes to logic 0, it generates a CLEAR signal to initialize the entire computer system.

Control and Signal Lines Hold the System Together

All 100 bus lines interconnect the various cards that comprise an Altair or Imsai computer system. And each system consists of a box that contains the various boards that perform the various functions—central processor, memory, input/output, and peripheral control. On the front of the computer is a control panel that contains switches and indicators that permit you to manually control all computer operations. Often referred to as a front panel, the array of control switches (see Fig. 1.9) can basically be grouped into three areas—address/data switches, basic machine control, and secondary control.

Address and data switches permit data to be manually loaded into selected memory locations or information in memory locations to be examined. Basic machine control switches include functions such as RUN/STOP, RESET/EXTERNAL CLEAR, POWER ON/OFF, MEMORY EXAMINE/EXAMINE NEXT, and MEMORY DEPOSIT/DEPOSIT NEXT. Other control functions such as SINGLE STEP or SLOW RUN, MEMORY PROTECT/UNPROTECT, ACCUMULATOR DISPLAY/LOAD and ACCUMULATOR INPUT/OUTPUT are not absolutely necessary for manual machine operation, but they are helpful.

The RUN/STOP switch provides direct control of the computer by controlling the state of the READY line of the bus. The RESET/EXTERNAL CLEAR switch provides a pulse that resets the central processor when flipped to the RESET position and when moved to the EXTERNAL CLEAR position provides a CLEAR command to all external input/output equipment. The POWER ON/OFF control is self-explanatory. The MEMORY EXAMINE/EXAMINE NEXT permits addresses to be set on the address switches and then the contents of that location examined by flipping the switch to the MEMORY EXAMINE position. The next sequential memory location can be examined by then flipping the switch to the EXAMINE NEXT position. Once the desired memory address has been set and examined, data can be set on the address/data switches and then loaded into that location by flipping the MEMORY DEPOSIT/DEPOSIT NEXT switch into the DEPOSIT position. Now data can then be set

on the data/address switches and loaded into the next sequential memory location by flipping the switch to the DEPOSIT NEXT position.

Non-basic functions such as the SINGLE STEP or SLOW RUN switches permit the processor to execute either a single instruction or operate at a rate of about two machine cycles per second (normal operation is 500,000 cycles per second). A MEMORY PROTECT/UNPROTECT switch permits manual control of the bus MEMORY PROTECT/UNPROTECT line, thus preventing undesired memory-write operations when in the PROTECT position, and permitting write operations in the UNPROTECT position. Accumulator controls are handy for displaying the contents of the accumulator and altering the data if necessary. Accumulator input/output controls permit data present at an I/O device to be loaded into the accumulator to an output device.

Front-panel indicators are used to display address, data, and several status signals (HOLD, WAIT, RUN, INTERRUPTS ENABLED). However, when the computer operates at full speed all indicators that are actually flickering on and off will appear to be on all the time. Signals from the front panel connect to the bus and from there are distributed to the various circuit cards that make up the computer system. The main card controlled by the various front-panel switches is, of course, the central processor. So, without any further delay, let's examine the central processor card of the Altair and Imsai computer systems.

The CPU Works like This . . .

The basic design of the Altair and Imsai CPU cards is very similar—both are built around an 8080A microprocessor and provide buffering on the output signal lines so that they can drive signals on the S-100 bus. The actual circuitry on the two boards really isn't important, but the signals entering and leaving the board are. Basically, each board contains the microprocessor, a crystal controlled clock that generates all the timing signals, a latch to hold all the status information, and the various line drivers and receivers to connect with the bus (Fig. 4.3). Also on the board are the voltage regulators needed to supply constant voltages to the circuits.

The bus interface signals can be grouped into seven classes of signals:

1. Address bus
2. Data bus
3. Timing
4. Power
5. Input control lines
6. Output control lines
7. Status outputs

Not all of the 100 bus pins are used by the CPU card to do its job. However, more than half of the pins are

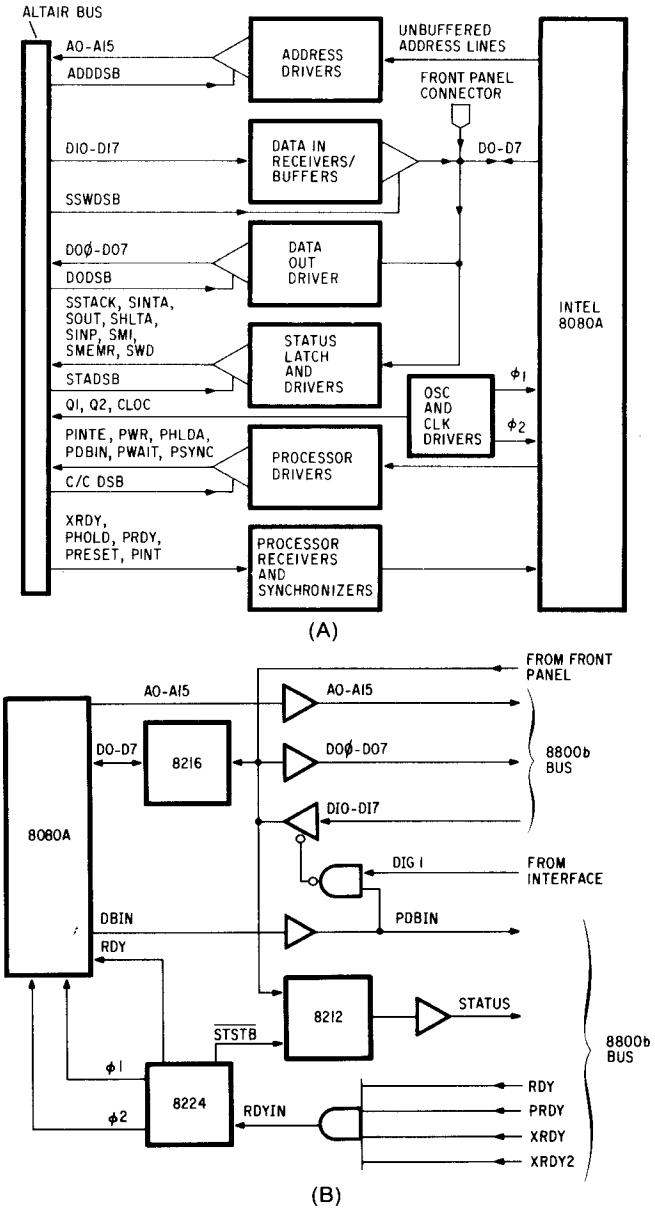


Fig. 4.3 The heart of either the Altair or Imsai systems is the 8080-based central processor board. Both boards are identical in basic operation and their block diagrams, (a) Altair and (b) Imsai, show similar functions performed. (*Courtesy* Pertec and Imsai)

needed. The address and data bus account for 32 lines, timing accounts for only three, power lines include six pins, and the other functions add up to another 27 or 28 lines, depending on whether the board comes from Imsai or MITS. The actual schematics of the CPU boards (Altair 8800b and Imsai 8080) are shown in Appendix C.

The CPU board is the main source of most of the control signals on the S-100 bus. To start with, all three clock signals originate on the CPU card (pins 24, 25, and 49), all nine status signals (pins 44 to 48, 56, and 96 to 98), the POWER ON CLEAR signal (pin 99), and the HOLD, WAIT, INTERRUPT, SYNC, POWER,

and DATA bus in signals (pins 26 to 28 and 76 to 78) all come from the CPU. Inputs to the board include an ADDRESS DISABLE line (pin 22), a DATA BUS OUT DISABLE (pin 23), a DATA BUS IN DISABLE (pin 57), a STATUS DISABLE (pin 18), a PRESET INPUT (pin 75), two EXTERNAL READY lines (pins 3 and 12), two READY lines (pins 58 and 72), an INTERRUPT input (pin 73), a HOLD line (pin 74), and a COMMAND CONTROL DISABLE input (pin 19). The difference between the Altair and Imsai 8080 CPU boards is pin 12—the Altair board uses that pin for the second EXTERNAL READY signal, which is not available on the Imsai board.

In addition to the S-100 bus interface, each CPU card contains another interface bus that connects to the board that contains the switches and displays of the front panel. This eight-line interface is just a buffered version of the data bus (MITS board) or an unbuffered version (Imsai board) that permits data to be manually loaded into the CPU by the front panel.

The 8080 CPU boards from Pertec and Imsai are not the only CPU boards that can mate with the S-100 bus. Other companies offer boards that perform the same function—and some of the boards don't even use the 8080 processor. Since the S-100 bus was introduced in 1974, many new microprocessors such as the Z-80, the 6800, the 6502, and newer versions of the 8080 (the 8085), have been adapted to mate with the S-100 bus. These next-generation CPU boards offer faster operating speeds, more or different instructions, and more flexibility for the user.

Basically, any of the other S-100 CPU boards contains the same circuitry as the MITS or Imsai boards. There will be some subtle differences in the way processor control lines are interfaced to the bus, but the board will still accept data and instructions, perform the operations, and deliver data back out. If the microprocessor on the board is not an 8080, there will be some extra logic on the board to supply the extra control signals needed by the rest of the system that are not generated by the new processor.

There are more than 20 different CPU boards available for the S-100 bus system, from as many manufacturers. In addition to the schematics for the 8080-based Altair and Imsai boards in Appendix C, a diagram for the Xitan (Technical Design Labs) Z-80 board is included.

The CPU and its front panel are needed for a minimal computer system but it's still lacking two major elements to perform any useful functions—memory and input/output sections. The memory section is used to hold instructions and data that the CPU will operate with while the I/O section permits the CPU to communicate with external machines to provide data entry or output.

CHAPTER 5

Computer Memory Systems

Without some form of memory, the computer is just a useless pile of electronic circuits. For a computer to work it must be able to store instructions and data. And to store the instructions and data there are three main types of memories that can be used to hold information:

1. Solid state memories, with no moving parts. These devices typically consist of read/write and random-access memories built from flip-flops for temporary storage, and programmable read-only random-access memories for permanent storage.
2. Magnetic memories, using cassettes, cartridges, or flexible magnetic discs. These provide permanent but alterable storage for data and instructions.
3. Paper tape storage, using holes punched in paper tapes to hold non-alterable data or instructions that must be loaded into the computer.

This chapter will discuss only the first family of memory devices, and subsequent chapters will cover the other memory types. Within the family of solid-state memories there are many different types that store various amount of information. As mentioned back in Chapter 2, the basic memory element of solid-state memories is the flip-flop and by connecting the flip-flops in various ways, large arrays of serial memories (shift registers) or randomly accessible memories can be built.

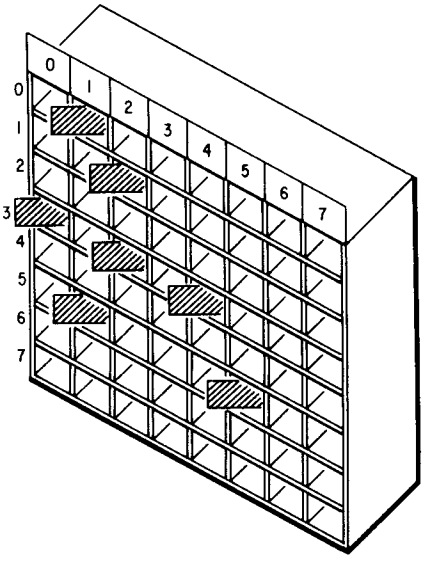


Fig. 5.1 The random access memory of a computer is very similar to this mailman's sorting bin set-up—any item can be retrieved just by knowing the row and column number.

The Random-Access Memory: What Is It?

Computer memory arrays are often called random-access memories because the same amount of time is required to reach any location within the array. The array can be likened to a mailman's letter-sorting box with a different slot for each person on the delivery route (Fig. 5.1). Letters in the slots can be said to represent 1s and no letter (an empty box) can represent 0s. If the mailman's route consists of 64 blocks set up in a grid of 8 × 8 blocks, any one block of mail can be accessed in the same amount of time just by knowing the row and column number (street and avenue). Much the same way, an array of flip-flops can be accessed by defining the row and column of the desired flip-flop (Fig. 5.2).

For most microprocessor memory systems each memory location contains eight bits of data, much like the eight vertical columns of letters slots. The horizontal rows then represent words of memory. To better familiarize yourself with the terminology of memory circuits, here are some definitions of many of the commonly used terms:

Memory Cell. A device or circuit subsection that is used to hold a single bit of computer data. A single flip-flop can be called a cell.

Memory Word. A word consists of several computer bits delivered simultaneously to a system. Common numbers of bits used to make up a word in most computers start at four bits and go as high as 40 or more.

Byte. A collection of eight binary bits, sometimes equivalent to a complete word, as in many microcomputer systems, or used to signify an 8-bit portion of a larger word. When the word size is the same size as a byte the two are often used interchangeably.

Nibble. A fairly recent addition to the designer's vocabulary, this word refers to a 4-bit grouping of binary numbers, or half of an 8-bit byte.

Random-Access Memory. This type of memory circuit permits information to be written to or read from any storage cell or word in the same amount of time. Commonly abbreviated as RAM.

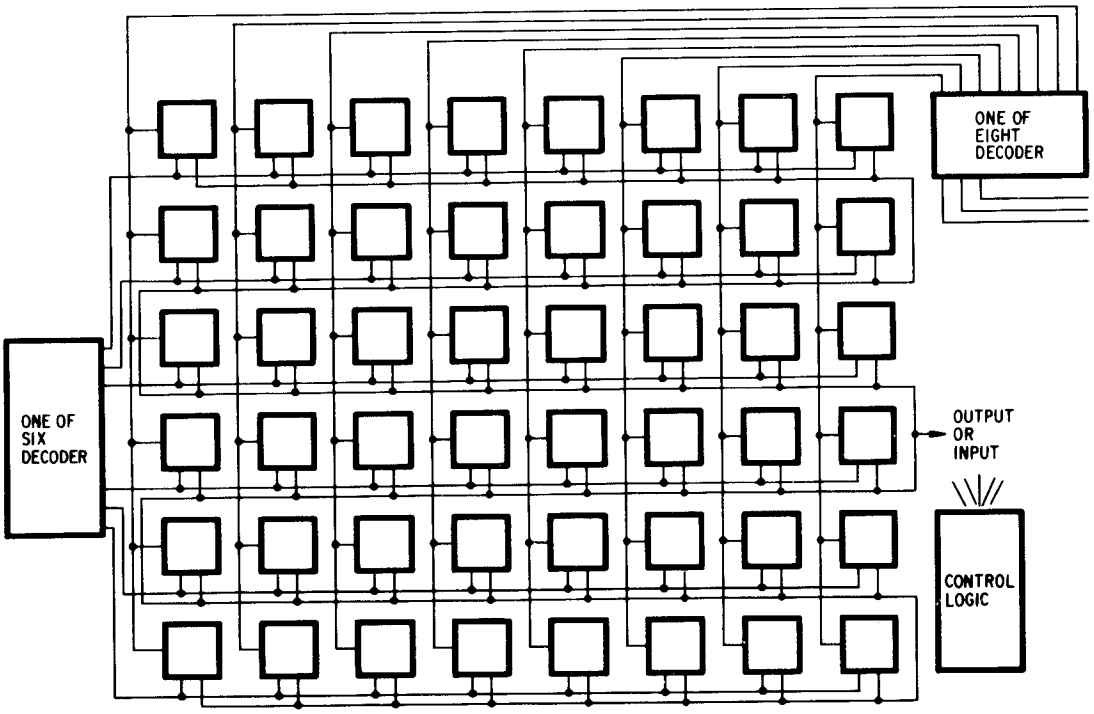


Fig. 5.2 Inside a semiconductor memory, the storage cells are arranged just like the mail sorting bins.

Read-Only Memory. This is a special form of random-access memory from which information can only be read out. Information in these memories cannot be altered once it is put in by the memory manufacturer. It is commonly abbreviated as ROM, and it is a form of RAM.

Programmable Read-Only Memory. A form of ROM, the programmable ROM, or PROM, lets the user enter in data or instructions to be permanently stored. There are different types of PROM, some of which use microscopic metal links (called fuses) that must be burned away by large current pulses for programming, and once burned cannot be restored. Other types store electrical charges in a microscopic dielectric material to represent binary bit patterns and can be wiped clean (erased) by shining an ultraviolet light through a clear quartz window on top of the package and then reprogrammed. Whether the fused-link PROM or the UV EPROM is used, they are indispensable for microcomputer programming.

Memory Size. The number of bits or bytes that are available in the memory circuits. The qualifier bits or bytes should be present to make clear the capacity of the memory. For instance, the 8080A or Z-80 can handle a memory size of 65,536 bytes (often rounded off to 64k or 65k), or 524,288 bits.

Memory Bank. A term often used to refer to the entire processor memory, although individual boards that contain 4096, 8192, 16,384, 32,768 or 65,536 bytes (4k, 8k, 16k, 32k, and 64k) are often referred to as 4k banks, 8k banks, etc.

Memory Address. This is the designation used to determine where the desired memory word is located in a large array of words. It is much like the street address of your home, telling someone where to look.

Access Time. This is the time required by the memory to output the contents of a location and is measured from the time it first received a command to look up a word until it presents the word at its output.

Cycle Time. The cycle time is the access time plus the time necessary for the memory circuit or circuits to prepare themselves for the next request.

Read (Fetch), or Write (Store) Operation. This is the sequence of signals that locates the memory cell or cells from which or into which information will be transferred (written or stored).

Most semiconductor read/write RAMs lose all the information stored in them when power is shut off; all ROMs will retain their stored data and when power is returned data will be present. ROMs, in general, require no timing signals to control the memory. Usually, a single control signal is needed to tell the memory to get ready to be addressed. Once addressed, the memory outputs the contents of the location and that's that.

Some read/write RAMs perform similarly. These types are referred to as *static* RAMs. There are, though, many other types of RAMs that require special timing signals to control the memory and make sure all the internal circuits operate in the proper sequence. These memory circuits are called *dynamic* memories because timing signals are constantly present to replenish the data stored within the memory cells. This is necessary

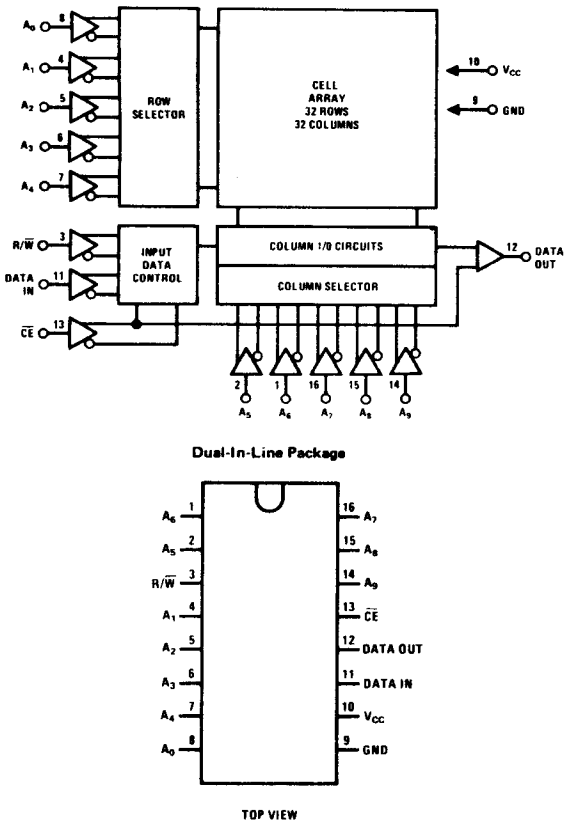


Fig. 5.3 Pinout and block diagram of the 2102, a 1024-bit static memory circuit.

because in dynamic RAMs data bits are stored in transistor-capacitor cells that are slightly "imperfect"—the charge in the capacitor that represents the data leaks away. Thus, every fraction of a second, the charge must be replenished—this process is referred to as the *dynamic memory refresh* and is very important.

Although what goes on inside RAM is important to the RAM designer, as a user you don't really have to know about these processes as long as you supply the proper signals to the inputs of the circuits. However, when you gather together components for a memory system you must make some important decisions before buying components.

1. Decide whether you will use static or dynamic memories in each of the banks. All circuits in each bank (on each board) should be the same type, although you can often mix banks if each bank appears identical to the processor circuits.

2. Decide on the speed of the memory circuits in the memory system (access and cycle times). The maximum speed is often determined by the speed of the processor or the memory itself—the faster the processor clock, the shorter the access time required. For instance, the 8080A CPU card operates at a 2 MHz clock rate and therefore, for the memory to be ready every clock cycle, it must have a cycle time of 1/(2 MHz) or 500 ns. Normally, access times are shorter or equal to the cycle time so all you have to look for is the minimum cycle time specification to pick the right speed memory. If the processor runs faster than the memory, special allowances can be made by programming the processor to wait for the memory to finish.

3. Lastly, decide on how large a memory array you want on a memory board. Typical sizes are 4096, 8192, 12,288 and 16,384 bytes for static memory boards, and 16,384 to 65,636 bytes for dynamic memories on a single board.

Start with the Static Memories

The most common static memory circuit available is the 2102 RAM (Fig. 5.3). This circuit, housed in a 16-pin DIP, contains 1024 storage cells arranged in an array of 1024 × 1. Each of the 1024 cells can be accessed by supplying a binary number to the 10 address lines (A0 to A9). Data to be fed into the memory must be sent to the D_{in} input and data to be read out appear at the D_{out} pin. Another line called the R/W line controls whether the RAM is performing a read or write operation. When the line is HIGH a read operation can be done and when LOW a write operation can be performed. The last control line is a chip-select line which,

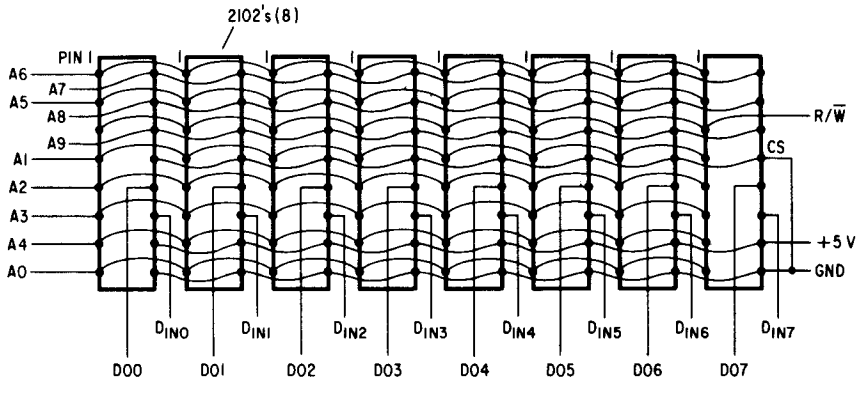


Fig. 5.4 To access eight bits at a time, eight 2102s must be connected so that the same location in each is accessed at the same time.

when HIGH, puts the data output line into a high-impedance state, permitting many RAM output lines to be wire-ORed together without loading the outputs. When LOW, the output assumes either its 1 or 0 state, as determined by the state in the addressed cell. The other two pins of the package are used to supply power to the array (+5 V and ground).

If eight of the 2102s are connected so that all the outputs appear at the same time you have the bare essentials of a 1k × 8 bit memory (Fig. 5.4). All the address lines must be connected in parallel so that all the A0s are connected, all the A1s, etc. Next, all the chip-select lines can be connected together and wired to a ground line since there are no other banks of RAMs to select from. All the R/W lines can also be connected together and this common connection is used to control the simultaneous operation of all eight RAMs. Each data input line from the RAMs must be kept separate, just like the data output lines. Both sets of lines are eventually connected to the DI and DO buses of the mainframe bus.

Actual memory boards used in the S-100 systems have a lot of other circuitry included. For starters, the array of eight 2102s is repeated seven more times and then some special circuitry must be added to control the flow of data, to decode the memory address and to buffer the memory outputs (Fig. 5.5). A typical 8-kbyte memory board, then, might have the following structure: Eight 1-kbyte banks of 2102s are set in an array so that all 10 address lines are connected in parallel and the chip-select lines from each array of eight 2102s are connected to a decoder (typically a BCD-to-decimal) and enable circuit. The decoder determines which of the eight banks is accessed by decoding address lines A10, A11, and A12 from the S-100 bus.

Another circuit, possibly consisting of three exclusive-OR gates, three switches, and some NAND gates can be used to decode address bits A13, A14, and A15 to place the 8k bank at any one of eight possible memory areas in the overall 64k address range. For instance, if all three switches are open the board will respond to address locations 0000 to 8191. However, if an address greater than 8191 appears, the decoding circuit does not enable any of the RAMs in the bank, and the entire 8k bank is disabled and another bank set to respond to this address will output the data.

Signals that flow into and out of memory boards are very straightforward—the 16 address inputs, the eight data inputs, and the eight data outputs for starters. Next are the control signals such as the MEMORY WRITE input, the DATA BUS INPUT ENABLE, a MEMORY READ ENABLE signal, and input and OUTPUT ENABLE signals inputs. Control outputs from the board include a status indicator line (PROTECT or UNPROTECT) and a READY line. Most memory boards use the same control line and have the same address and data buses. Depending on the memory circuits used, the board may require just the 8 V connection to the supplies or also the ±18 V connections as well. Some boards also have provisions for an external battery backup so if power fails, data held on the board won't be lost.

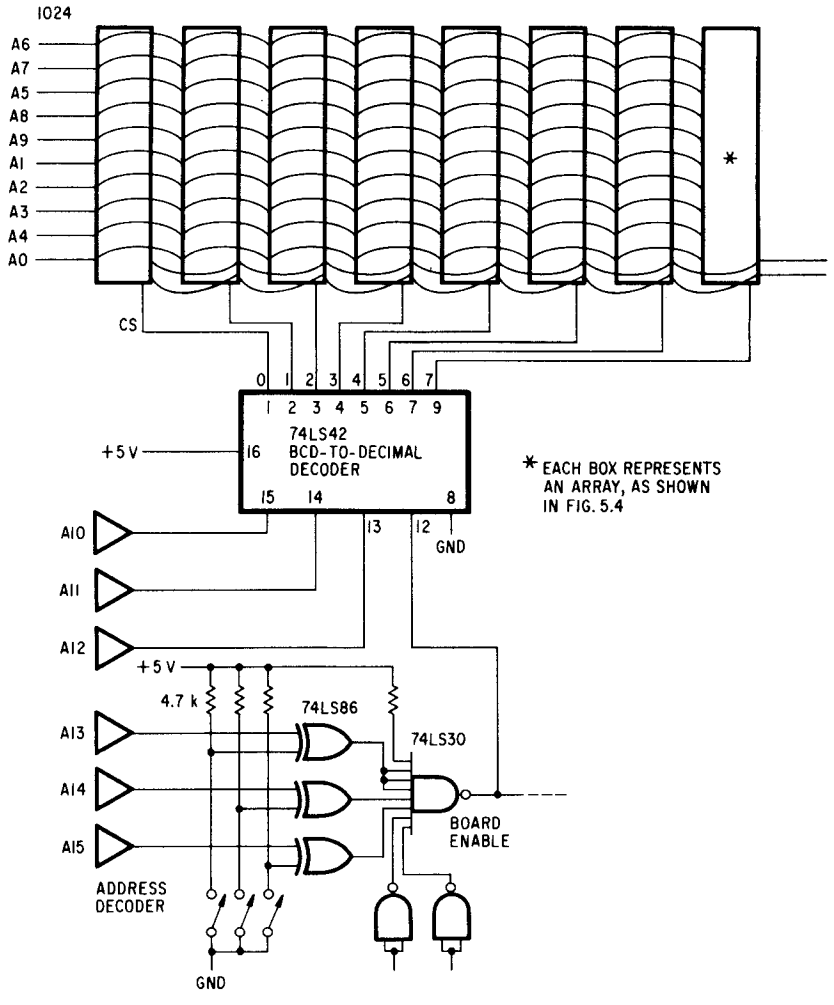
Recently developed static memory circuits can squeeze even more into a single package. For instance, just now available in large quantities are 4096 × 1 bit memory circuits. With these arrays only 16 packages are needed to make an 8192 × 8 bit memory. Since the packages (18, 20, or 22 pin) don't require much more space, 32 of them can be comfortably put on a single card that plugs into the computer mainframe, thus permitting up to 16,384 bytes on a single card. Pertec, Seals Electronics, and Xitan (Technical Design Labs) are just some companies that offer the boards (Fig. 5.6).

Memory boards that use dynamic memory circuits require several additional control signals for operation. In most cases, timing signals from the S-100 bus must be used to time refresh operations so they don't interfere with the computer when it is trying to access or write data. Typical additional signals then would include the clock phase 1 or phase 2, the RUN, WAIT, and HALT lines, the M1 status line, and the SYNC line, which can all be used by the memory board to handle timing.

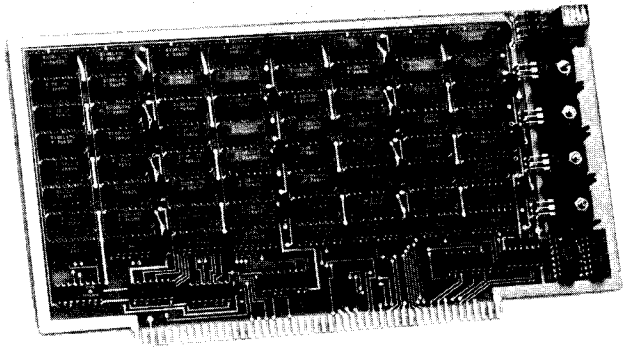
The one disadvantage of dynamic and static RAMs is that when power is turned off the RAMs lose all the information held inside, whether that information is data or instructions. When the computer is set up with a CPU card as described in Chapter 4 and some static and dynamic RAM cards you are ready to enter a program via the front-panel switches and test it out. The user may play a short game with the indicator lights, or learn how to perform basic machine operations such as loading data from the switches, reading the data back out, or doing simple mathematical and logic routines. However, after all the switch flipping and light blinking you shut off the power and the next time you turn the power on you must start all over again.

Read-only memories can save you many of the switch-flipping operations necessary to get the computer ready for operation. Without the ROMs, to prepare the computer you must do many different switch operations, depending on the function you want the computer to perform. If, for instance, you want the computer to accept information from a CRT terminal or teletypewriter or send information to the terminal, not only must you put instructions into the computer's memory to tell it how to handle the incoming or outgoing data but a special interface circuit must be plugged into the main bus to change the serial information from the terminal into parallel information for storage in the computer's memory or the reverse. (More about the interface circuit and the actual instructions in later chapters.)

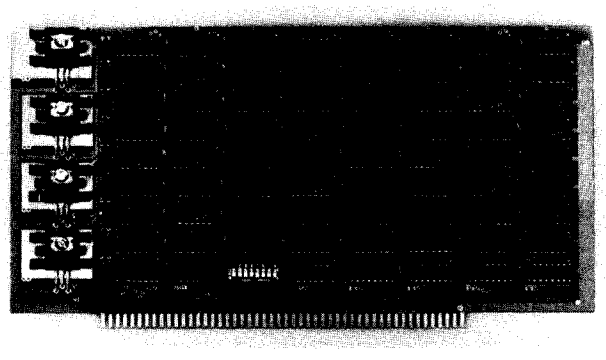
Every time you turn power on you'll have to initialize the system by hand—a tedious and unnecessary



(A)



(B)



(C)

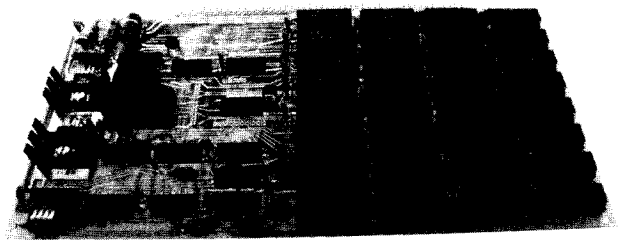
Fig. 5.5 Multiple banks of eight 2102s must be connected to a decoding circuit so that one of eight banks can be enabled when the computer addresses the memory and tries to read or write data (a). A complete memory circuit for an 8 k × 8-bit array, designed by Vector Graphic Corp., is shown in (b), and an equivalent version from Seals Electronics is shown in (c).

job if you take advantage of any of the forms of read-only memories. There are many available preprogrammed ROMs that companies offer for many different applications, and not just for the 8080A and Z-80. The ROMs that hold the special operating programs do not work by themselves; in addition to the ROM, a RAM must be used to hold the information being entered or outputted and intermediate computational results.

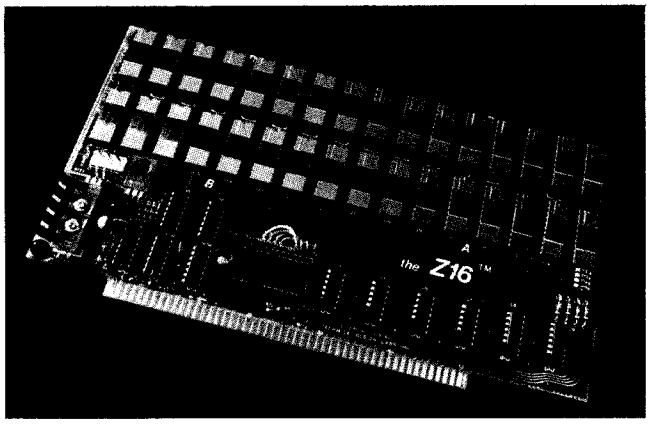
When the instruction is executed it usually causes the program counter inside the processor to change value and access another memory location that starts the initialization procedure. This process is often called vectoring since one address points to another address. Once the initialization procedure is completed (it usually occurs so fast that when the button is pressed you can start immediately), a program designed to accept inputs and deliver outputs to a terminal and handle other functions such as displaying the data and loading information is usually loaded into the RAM from the ROM storage area. This type of program is usually called a monitor program since it oversees the operation of the computer and its peripherals.

The process of loading information from ROM into RAM is usually referred to as bootstrap loading and is used in almost every computer system to initialize the system and load the preliminary programs to handle the terminal, possibly a cassette tape interface, floppy disc drive, and a printer. Inclusion of a board that can hold the ROMs and possibly a small amount of RAM is essential for any system. For instance, the PROM/RAM board made by Vector Graphic can hold 4096 words of permanent storage in the form of 1702A ultraviolet-erasable programmable read-only memories and another 1024 words of static RAM in the form of 2102 RAMs (Fig. 5.7). Other boards, made by MITS, Solid State Music, Seals Electronics, and Pertec offer up to 16 kwords of PROM but no RAM (Fig. 5.8).

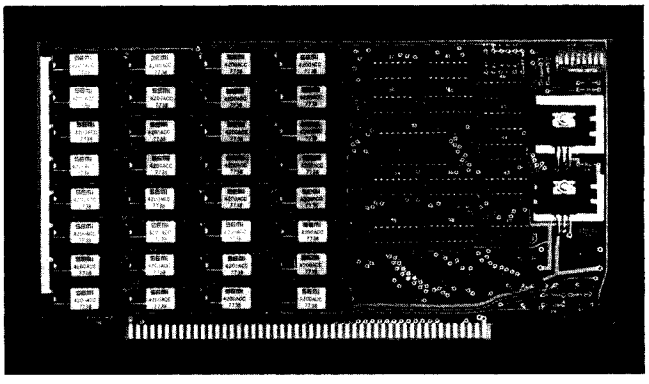
There are even combination boards that can take the place of the front-panel switches, load programs into RAM, and even contain some of the input and output interface circuitry to handle the peripheral equipment. One such example is the System Monitor Board made by Xitan (TDL) (Fig. 5.9).



(A)



(B)



(C)

Fig. 5.6 Some typical larger memory cards: (a) Pertec 88-16 MCD, (b) Xitan/Technical Design Labs Z16, and (c) Seals Electronics 16k. (*Courtesy* Pertec, Xitan/Technical Design Labs, and Seals Electronics)

ROMs: ICs That Remember

The read-only memory, as mentioned at the beginning of this chapter, is a form of random-access memory. However, data stored within the circuit can be looked at (read out) but not altered. There are several methods available to put data into a read-only memory. Manufacturers that use thousands of the same product will usually tell the IC supplier exactly what to put inside the circuit and will obtain a very low-cost part. However, if the manufacturer makes a mistake he'll be stuck with literally thousands of useless memories.

To avoid this problem of ordering a large number of parts before the program or data stored in the ROM are tested, IC suppliers developed several types of memories that the manufacturer (you) can program and then test in the computer or other circuit. The two major types of programmable read-only memories (PROM) are the bipolar fusible link PROM and the ultraviolet

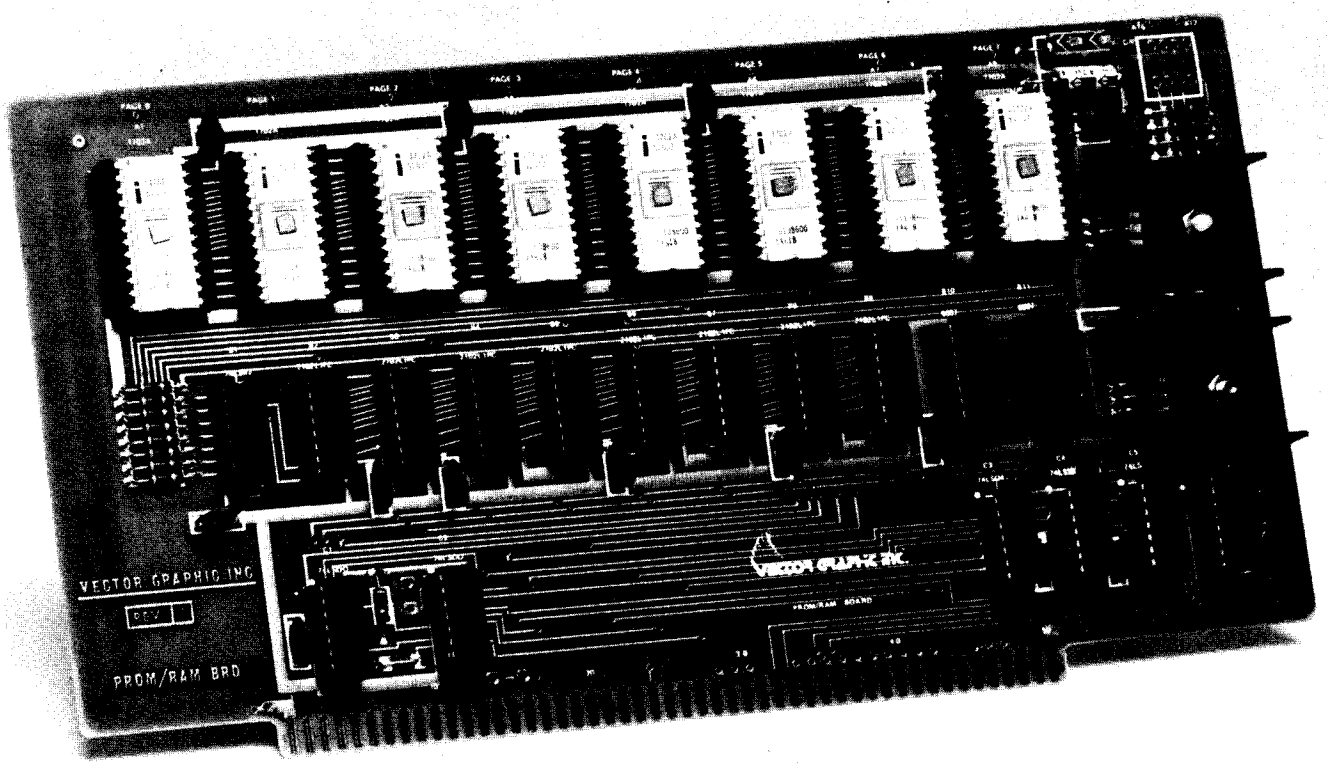
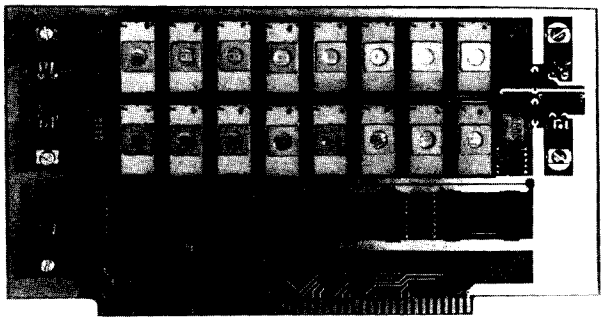
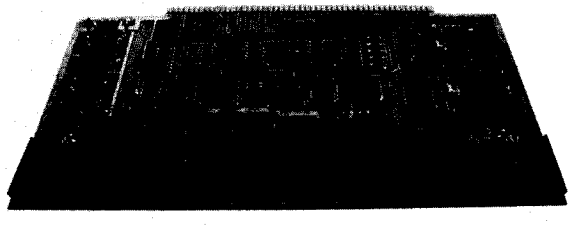


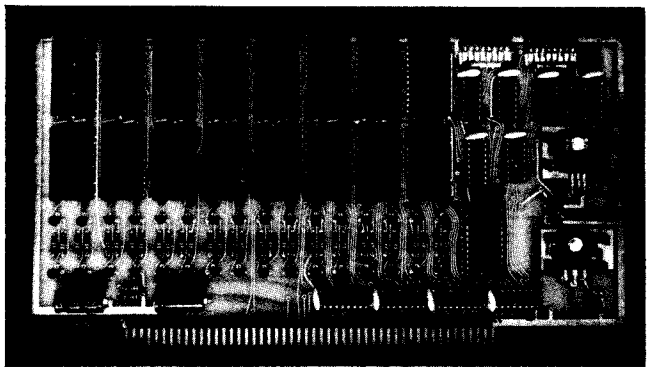
Fig. 5.7 The PROM/RAM board made by Vector Graphic Corp. holds up to 2048 bytes of PROM/EPROM/ROM and an additional 1024 bytes of RAM.



(A)



(B)



(C)

Fig. 5.8 Various PROM boards: (a) from SSM (Solid State Music), (b) Pertec, and (c) Seals Electronics. (*Courtesy* SSM, Pertec, and Seals Electronics)

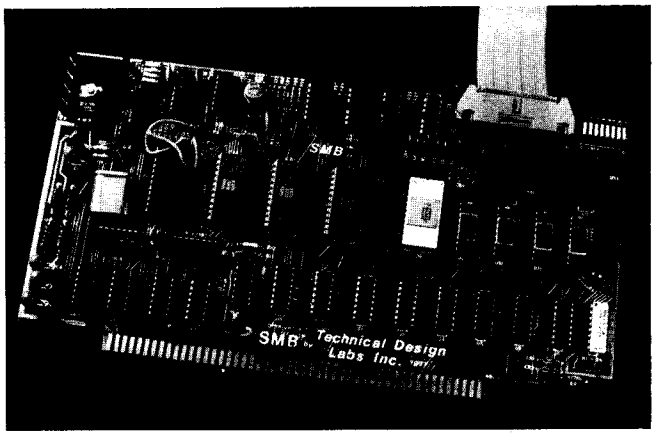
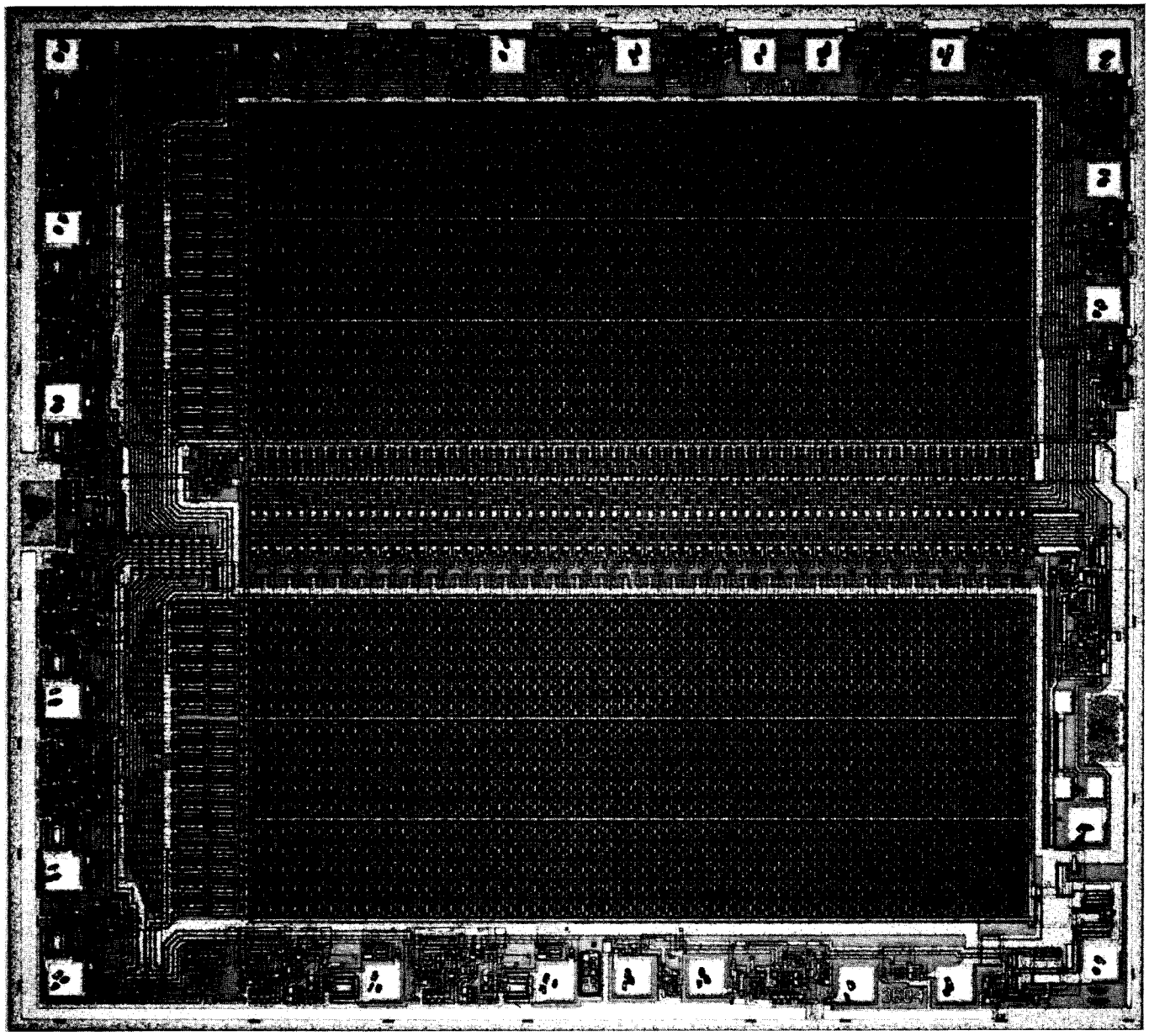
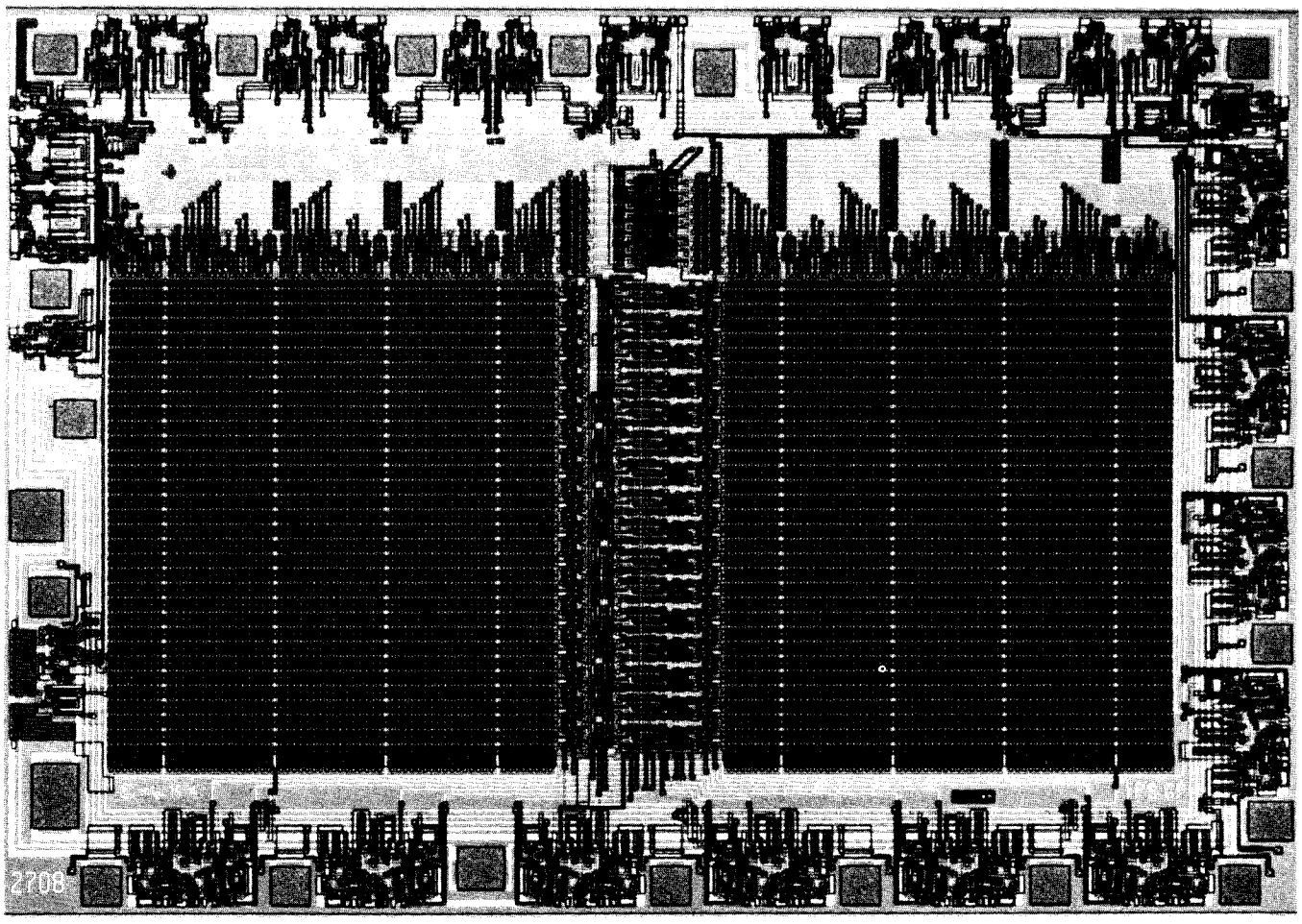


Fig. 5.9 The System Monitor Board (left) designed by Xitan/Technical Design Labs not only holds the ROM and RAM, but also contains serial and parallel interfaces for peripheral equipment.

Fig. 5.10 Able to store data permanently, the 3604 (a) is an electrically programmable ROM that holds up to 4096 bits and is organized as a 512 × 8-bit memory array. The other circuit is a 2708 ultraviolet-erasable, electrically programmable ROM (b). It can hold up to 8192 bits of data and is organized as a 1024 × 8-bit array. (*Courtesy* Intel Corp.)



(A)



(B)

Fig. 5.10 (cont'd) Able to store data permanently, the 3604 (a) is an electrically programmable ROM that holds up to 4096 bits and is organized as a 512 × 8-bit memory array. The other circuit is a 2708 ultraviolet-erasable, electrically programmable ROM (b). It can hold up to 8192 bits of data and is organized as a 1024 × 8-bit array. (*Courtesy* Intel Corp.)

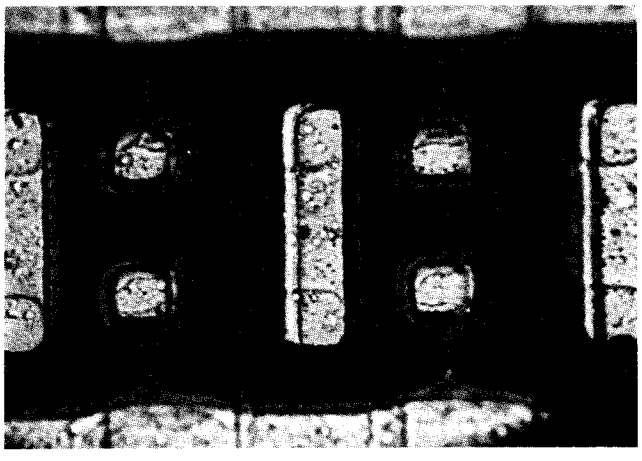


Fig. 5.11 A close-up shot of good and blown fuses in a bipolar PROM. (*Courtesy* Intel Corp.)

erasable PROM (Fig. 5.10). Fusible link PROMs are available in array sizes as small as 256 × 4 and as large as 2k × 8, and even larger sizes are in the works. The UV EPROMs are available in array sizes from 256 × 8 to 4k × 8 bits on a single chip.

Fusible-link PROMs access about 10 times faster than the UV EPROMs; their access times are typically 50 to 100 ns minimum compared with 350 ns to 1 μs for the UV EPROMs. The difference in speed is mainly due to the basic process differences; most fusible-link PROMs are made with bipolar transistors while the UV EPROMs are made from MOSFET transistors. However, even though the bipolar PROMs are faster, they have some disadvantages versus the MOS UV EPROMs. Once the bipolar PROMs are programmed, information stored in the circuit cannot be altered since the programming process burns away some of the actual circuitry inside the PROM. The UV PROMs, on the other hand, can be wiped clean by shining a high-intensity ultraviolet light for about 10 minutes through a transparent lid on the top of the case. After the EPROM is erased new information can be stored inside until the new information is no longer needed; this process can be repeated a great many times.

The most commonly available type of bipolar PROM uses metal links to make or break a connection between transistors within the chip (Fig. 5.11). In a 4096 bit PROM there are 4096 links, one for each bit. Each metal link is often referred to as a fuse and during the programming process the link is either blown apart or left intact to indicate the value stored in the cell. Each manufacturer's family of bipolar PROMs has its own special procedure to burn the links and be programmed.

In MOS UV EPROMs, the memory cell consists of a MOSFET transistor whose gate is isolated from the source and drain by a layer of silicon dioxide, a good insulator (Fig. 5.12). During programming, a large charge is placed on the gate, via a technique referred to as avalanche injection, to represent a bit value. The ultraviolet light during erasure causes a photocurrent (light-dependent current) to flow between the gate and the base silicon of the chip, thus neutralizing the charge stored and wiping clean the memory. Programming procedures for a UV EPROM are vastly different from those for a bipolar device and thus you must really make a choice as to which type of PROM will be used early in your selection.

MOS PROMs, although slow, offer more advantages than disadvantages, especially since they're reuseable if the information stored within them becomes obsolete. There are four commonly used UV EPROMs: the 1702A, the 2704, the 2708, and the 2716 now available at a cost of about $3, $10, $15, and $35, respectively, from various surplus dealers. These PROMs hold 256, 512, 1024, and 2048 8-bit words, respectively. The 1702A pinout differs considerably from those of the 2704, 08, and 16 EPROMs (Fig. 5.13). The 2704, 2708, and 2716 can almost fit in the same socket; only one wire has to be changed to use a 2708 in a 2704 socket and similarly for the 2716. Voltage requirements for the 1702A are +5 V and −9 V for operation after programming; the 2704 and 2708 require +12 V and +5 V supplies, while most of the 2716s require just a 5 V supply. To store the charges inside the PROMs, a programming time of 120 seconds is required for the 1702A and slightly less for the newer 2704, 2708, and 2716 PROMs even though they hold many times the number of bits. That is, if you have a computerized programmer.

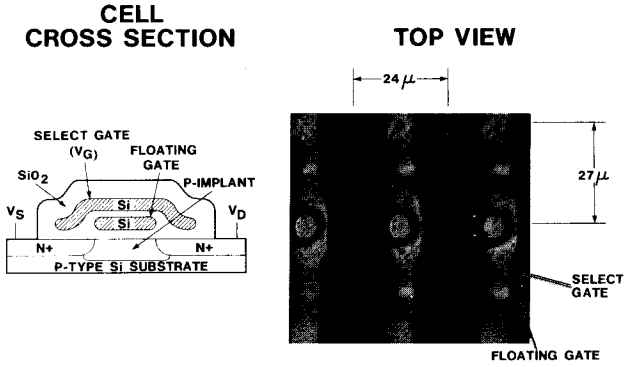
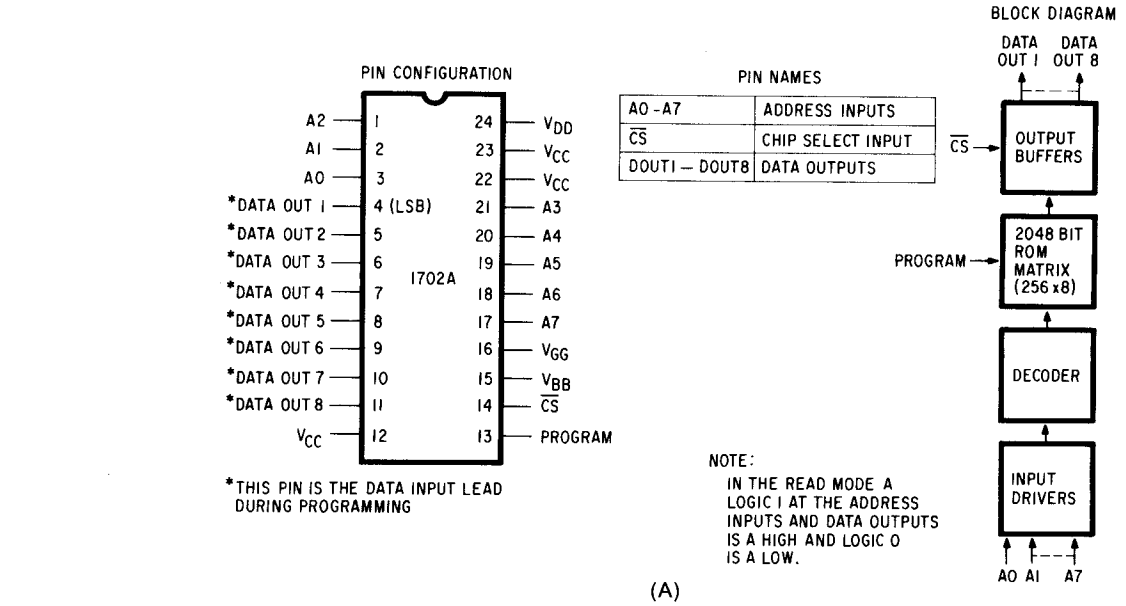


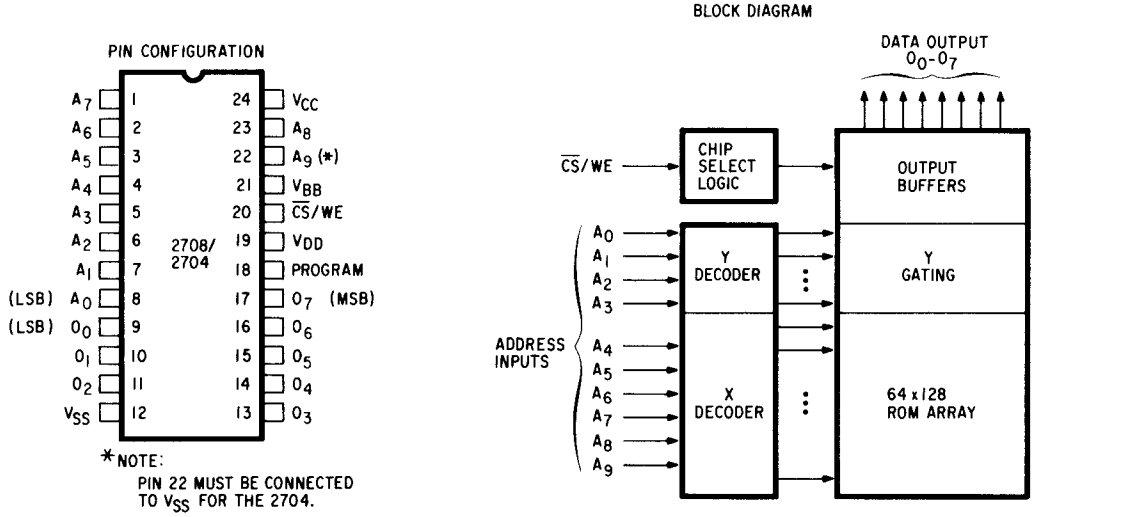
Fig. 5.12 Close-up microphotograph of individual memory cells in a 2708 EPROM. (*Courtesy* Intel Corp.)

The PROMs Are Programmed like This . . .

In the 1702A PROM, all 2048 bits are initially (or after erasing) in the ZERO output state; information is introduced by selectively programming ones into the desired bit locations. To program the 1702A, the complement of the address of the word to be programmed is first placed on the PROM's address bus, and while the bit levels are held constant the V_{dd} and V_{gg} power lines to the chip must be brought to their negative levels (Fig. 5.14). The address bits must be held in their complement state for a minimum of 25 μs before a negative program pulse is applied to the program input pin. All addresses should be programmed in sequence from 0 to 255, for a minimum of 32 times to ensure valid information. The eight data outputs D0 to D7 are used as the information inputs for each word. A negative level on the data lines will cause a one to be programmed and a ground level will cause a zero. All eight bits of each word are programmed simultaneously. During programming, V_{gg} and V_{dd} and the program pulse are pulsed signals, V_{cc} must be kept at ground, V_{bb} must be biased at 12 V, and $\overline{CS}$ must be kept at ground.

The 2704, 2708, and 2716 have similar programming characteristics and initially (or after erasure) come with all bits in the ONE output state. Information is stored by selectively programming zeros into the desired locations. The PROMs are set up for programming by first raising the $\overline{CS}$/WE input pin to 12 V (Fig. 5.15). The word address is selected in the same way as in the READ mode (the binary address is presented to the address inputs) and then data to be programmed are placed on the 8-bit data outputs of the PROM. Voltage levels on the address and data lines are standard TTL levels: 0.65 V for a zero and 3 V for a one. After the address and data are present a 10-μs set-up period must be included and then a positive program pulse of about 25 to 27 V should be applied to the program input line. Again, all addresses should be programmed in sequence (0 to 511 or 0 to 1023) and one pass through all addresses is called a programming loop. The number of loops N required to guarantee programming is a function of the programming pulse width t_{pw} such $N \times t_{pw} \geqslant 100$ ms. However, you are somewhat restricted in that the pulse width must be in the range of 0.1 to 1 ms, therefore N can range from 100 to 1000. There must be N successive loops through all 512 or 1024 addresses to guarantee programming. N pulses should not be applied to one address before the next address is accessed. At the end of the N-loop sequence the $\overline{CS}$/WE falling-edge transition must occur before the first address tran-





PIN NAMES

A_0-A_9	ADDRESS INPUTS
O_1-O_8	DATA OUTPUTS/INPUTS
$\overline{CS}$/WE	CHIP SELECT/WRITE ENABLE INPUT

(B)

PIN CONNECTION DURING READ OR PROGRAM

	PIN NUMBER							
MODE	DATA I/O 9-11, 13-17	ADDRESS INPUTS 1-8, 22, 23	V_{SS} 12	PROGRAM 18	V_{DD} 19	$\overline{CS}$/WE 20	V_{BB} 21	V_{CC} 24
READ	D_{OUT}	A_{IN}	GND	GND	+12 V	V_{IL}	-5 V	+5 V
DESELECT	HIGH IMPEDANCE	DON'T CARE	GND	GND	+12 V	V_{IH}	-5 V	+5 V
PROGRAM	D_{IN}	A_{IN}	GND	PULSED +26 V	+12 V	V_{IHW}	-5 V	+5 V

Fig. 5.13 Pinouts and block diagrams for the (a) 1702A, (b) 2704/2708, and (c) 2716 UV EPROMs. (*Courtesy* Intel Corp.)

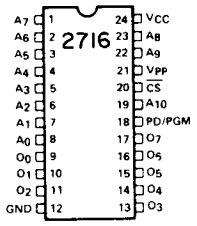
sition when changing from a PROGRAM to a READ cycle. The program pins should also be pulled to ground by an active rather than a passive load.

The actual circuitry necessary to program the UV EPROMs is not overly complex and can be built into a small, separate cabinet or on a card that plugs into the computer. Many companies offer stand-alone programmers and programming cards that fit right in the S-100 bus. However, if you're not doing your own programming this accessory may not be needed. System and end use needs will be the determining factors.

Design the Board That Remembers

Most of the ROM/PROM memory boards from various companies are designed to use the 1702A PROM

PIN CONFIGURATION*



PIN NAMES

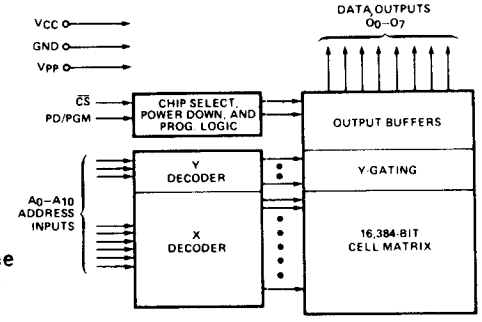
A_0–A_{10}	ADDRESSES
PD/PGM	POWER DOWN/PROGRAM
$\overline{CS}$	CHIP SELECT
O_0–O_7	OUTPUTS

To upgrade from a 2708 to a 2716 in the same socket, pin 19 of the 2708 becomes address line A_{10}.

MODE SELECTION

MODE \ PINS	PD/PGM (18)	$\overline{CS}$ (20)	V_{PP} (21)	V_{CC} (24)	OUTPUTS (9-11, 13-17)
Read	V_{IL}	V_{IL}	+5	+5	D_{OUT}
Deselect	Don't Care	V_{IH}	+5	+5	High Z
Power Down	V_{IH}	Don't Care	+5	+5	High Z
Program	Pulsed V_{IL} to V_{IH}	V_{IH}	+25	+5	D_{IN}
Program Verify	V_{IL}	V_{IL}	+25	+5	D_{OUT}
Program Inhibit	V_{IL}	V_{IH}	+25	+5	High Z

BLOCK DIAGRAM



(C)

Fig. 5.13 (cont'd) Pinouts and block diagrams for the (a) 1702A, (b) 2704/2708, and (c) 2716 UV EPROMs. (*Courtesy* Intel Corp.)

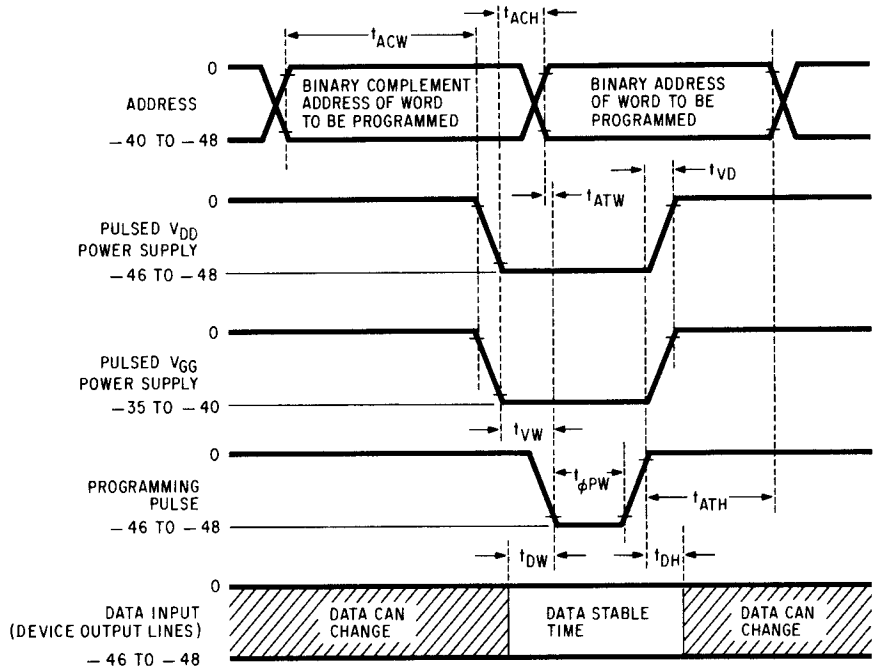
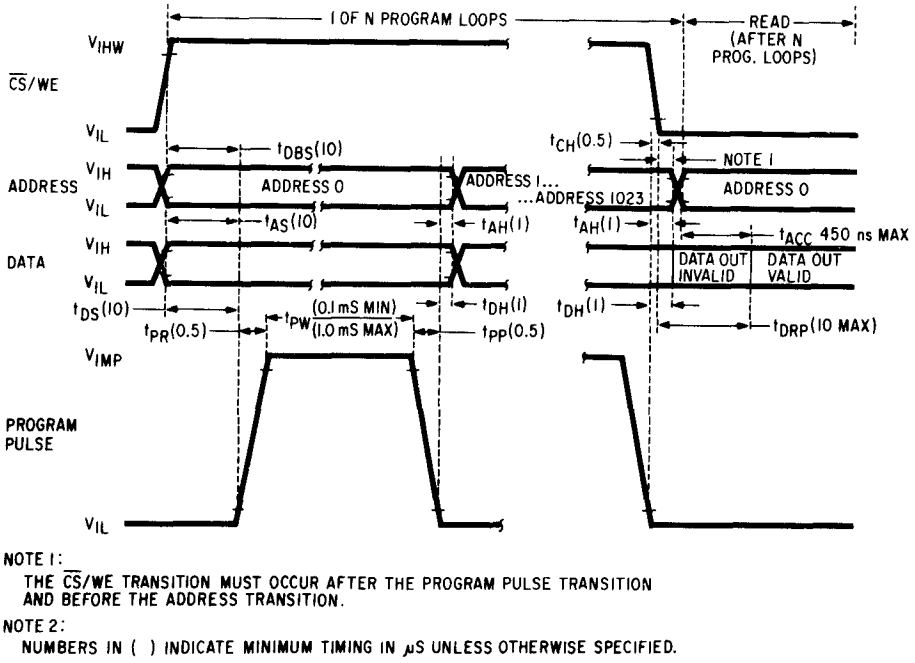


Fig. 5.14 Programming waveforms for the 1702A UV EPROM. (*Courtesy* Intel Corp.)

because it is very inexpensive compared to the larger UV EPROMs such as the 2708 or 2716. However, the 2708 prices have recently dropped to below $15 and can hold the equivalent of four 1702A PROMs. Since the MOS PROMS are slow—the access time is about double that of the RAMs, typically 500 ns—the processor may have to be told by the memory that it has to wait until the memory has accessed the information. This is called inserting a WAIT state (WAIT cycle).

One of the simpler cards is the 88-PMC PROM memory card from Pertec. The board can hold eight 1702A PROMs and even has a provision to put into standby all PROMs not in use. Special circuitry on the board can also be set to insert 0, 1, 2, or 3 WAIT states so that different speed memories can be used. Most of the circuits are normal 7400 or 74LS series except for three, which are special circuits immune to high levels of noise, and the memories. Any PROM



NOTE 1:
THE $\overline{CS}/WE$ TRANSITION MUST OCCUR AFTER THE PROGRAM PULSE TRANSITION AND BEFORE THE ADDRESS TRANSITION.

NOTE 2:
NUMBERS IN ( ) INDICATE MINIMUM TIMING IN μS UNLESS OTHERWISE SPECIFIED.

Fig. 5.15 Programming waveforms for the 2708 UV EPROM. (*Courtesy* Intel Corp.)

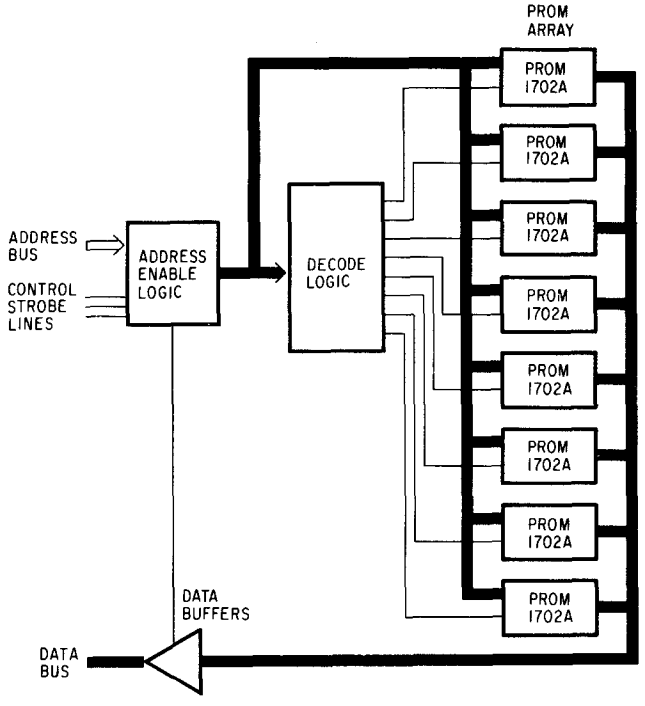


Fig. 5.16 Block diagram of the 88-PMC card made by Pertec.

board is divided into several sections that support the PROMs: the address decoding circuits, the V_{gg} switching circuit, and the wait circuitry (Fig. 5.16).

The interface of the PROM card to the S-100 bus is very simple. There are the 16 address inputs and eight data outputs for starters, the power supply inputs, and eight control lines. The eight control lines are PHASE 1 and PHASE 2 CLOCK, MEMORY READ, POWER, CLEAR, PRESET, SYNC, READY, and the PROTECT STATUS indicator line.

There are, of course, many different memory cards available from over 40 manufacturers. When you put together a system you should make sure all memory boards have an access time faster than the microprocessor cycle time, or that all the boards have provisions for inserting WAIT states. If you really have to pack the most into the least space, several companies offer many high-density memory cards.

For example, several companies offer 64 kbyte RAM boards that provide the entire addressable memory capacity of the computer on a single board. The boards use many of the same pins as the Pertec 4k board—the PHASE 1 and PHASE 2 CLOCK lines, the STATUS OUTPUT line, the SYNC line, the MEMORY WRITE line, the MEMORY READ line, the M1 STATUS line, the STATUS SHORT line, the POWER ON CLEAR line, the POWER ON INDICATOR line, the EXTERNAL READY line, and the DATA BUS INPUT line. And these boards are designed so that when used in an Altair or Imsai computer, they appear like a static memory even though they use dynamic RAMs as the storage elements. Special provisions on some of the boards permit them to be used in applications where up to 16 banks of 64 kbytes can be plugged into the computer and controlled by the processor via a special bank-switching program. Some boards use S-100 bus pins 56 and 66 and 13 to 17 for bank 0 to 15 select signals. These pins on the Imsai computer are undefined. However, on the Pertec 8800b computer pins 13 and 56 to 58 have pre-

defined functions, and thus bank positions 0, 1, 2, and 11 can't be used unless those pins are redefined.

However, few systems can run with just banks of RAM. For large ROM-based programs, Seals Electronics offers a 16-kbyte PROM board that holds up to eight 2716 PROMs. Other companies also offer high-density PROM boards. For instance, Vector Graphic offers a board that holds 12 kbytes of 2708 PROMs and an additional 1 kbyte of RAM, thus permitting minimal memory systems to be built on a single card. A selection of detailed memory board schematics is included in Appendix C.

Using the various types of memory boards in a computer system is very simple. In most cases the boards just have to be set for the address space they are intended to occupy. For example, most 8k memory boards have three switches that are used to specify any one of eight possible memory blocks—0000 to 1FFF, 2000 to 3FFF, 4000 to 5FFF, 6000 to 7FFF, 8000 to 9FFF, A000 to BFFF, C000 to DFFF, and E000 to FFFF. Larger memory arrays such as a 16-kbyte array, and a 32-kbyte array require only two or one switches, respectively, to perform array locating. Full 64k RAM arrays require no switches.

Once the block address on the memory board is set, all that has to be done is to plug in the board and turn on the system. Assuming the board has no defective memory chips, the system should be fully operational. Software techniques for tracing a defective memory chip will be discussed later.

One other aspect of memory system set up is the split in the memory space between RAM and ROM. Many programs are organized so that they must be loaded into RAM starting at address 0000. However, there are just as many programs that are designed to run on the computer with RAM space in the last few thousand bytes of their address range. So, the program you run on the computer has a lot to do with the way the address space is set up. And the program may also determine the amount of RAM or ROM space the computer must contain. Some programs will operate quite nicely with only 8 kbytes of RAM, while others, such as the Extended Disk BASIC offered by Pertec, require 24 kbytes just for the program.

CHAPTER 6

Input/Output Interfaces for the S-100 Bus

The basic S-100 computer system described in Chapters 4 and 5 consists of a system cabinet with a multiple-slot motherboard, a control front panel, a central processor card, and some assortment of memory cards. The next basic section of the computer is the input–output portion of the overall system. Computers are not stand-alone machines—data are fed in from one source, manipulated, and then fed out to another place. Devices that feed information to or accept information from a computer are referred to as computer peripherals.

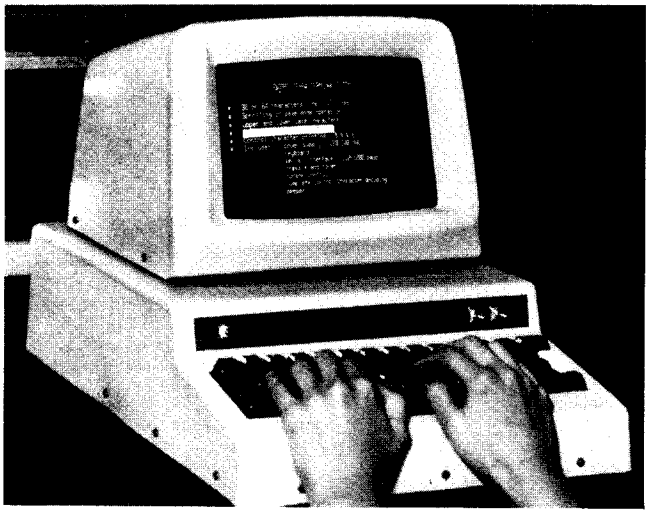
Two of the most commonly used input/output (I/O) devices are the CRT terminal and some form of printing device (Fig. 6.1). The CRT (cathode-ray tube) terminal is a television-like device that has a typewriter-like keyboard. When information is typed on the keyboard it appears on the screen and is sent to the computer. The computer can, in turn, send information back to the terminal so it appears on the screen. A printing device often combines a keyboard with some sort of printing mechanism to permit the operator to enter information and yet have a permanent record of everything entered. The printer can also be a receive-only device and only print information transmitted to it by the computer.

However, no matter what the peripheral device used, there are only two basic ways that data can be transferred back and forth between the peripheral and the computer. The simplest method uses a *serial* interface. This technique, in its minimal form, uses only three wires to connect the peripheral to the computer—one wire is for data transmitted by the peripheral to the computer, another is for data received by the peripheral, and the third line is a ground connection. Information sent back and forth is usually coded into a special format known as ASCII (the American Standard Code for Information Interchange), which is a code that requires eight bits to represent any of the 128 characters plus other information. To transmit the information representing each character over a single wire, special circuits must be used to convert the eight parallel bits into a serial stream of eight bits, and vice versa. These circuits operate at various speeds and in several different modes. There are two basic operating modes—synchronous and asynchronous. The synchronous mode requires that information be transmitted at a specific time along with special timing pulses that are received so that data can be sent at rates of 56,000 bits per second and higher. However, for data to be sent synchronously both the transmitting and receiving units must "lock" their timing signals together.

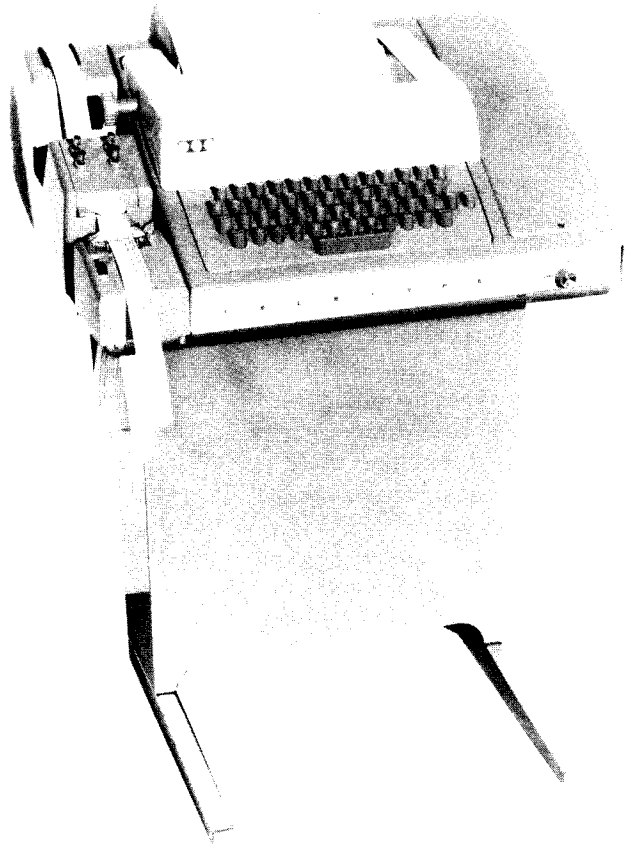
Asynchronous systems, on the other hand, do not require both systems to lock into each other. Instead, each time data are to be sent, the unit sending data interrupts the other unit and sends the information. To make sure data are received properly, special timing signals called start and stop bits are added to each character. Additionally, a signal such as a parity bit can sometimes be added (a parity bit indicates whether the sum of all one bits in the character is odd or even) as an extra safety measure to prevent errors. Typical communication rates during asynchronous operation are 110, 300, 1200, and 9600 bits per second (commonly referred to as baud rates) although many other rates from 75 to 19,200 are also used by various manufacturers. The lower the number, of course, the longer it takes to transmit or receive data.

The other method of transmitting data back and forth is via a *parallel* interface. This method uses at least eight wires for the byte to be transferred and often two or more control lines to signal the computer or peripheral that data are ready to be sent or have been received. Since eight bits are transmitted at one time, and often in a synchronous manner, parallel information transfers are high-speed operations with rates of 1 million bits/second possible.

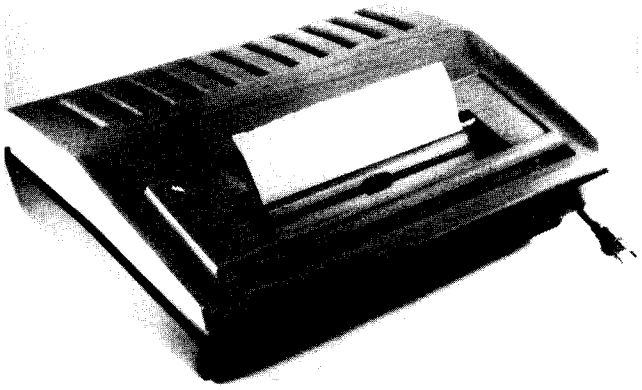
Often, data are transmitted in a parallel ASCII format, but not always. Other than ASCII, there are really no standardized parallel formats. However, serial data do have some standardized transmission formats, including timing as well as signal levels and length of data. One of the most common asynchronous serial formats was developed for use in low-speed communications equipment and it consists of a start bit, five to eight data bits, one or two stop bits, and an even, odd,



(A)



(B)



(C)

Fig. 6.1 The CRT terminal (a) and the teletypewriter (b) are the two most commonly used forms of input and output devices used with computers. The CRT terminal permits data to be entered or output without wasted paper. However, to obtain a permanent record of the computer's output, a hard copy device such as a printer (c) must be used. (*Courtesy* Southwest Technical Products, Teletype Corp., and Texas Instruments)

or no parity bit. Synchronous formats don't require all the extra bits since the equipment is waiting for simplified data.

In addition to the formatted bit pattern, there are three specific voltage requirements for the serial interface, depending on the type of equipment connected to the computer. The three interfaces include the TTY (teletypewriter) 20 mA current loop, the RS-232-C, and a TTL equivalent of the RS-232-C. For a current-loop, interface, an approximate 20 mA current flow indicates a logic 1 while the absence of a current in the loop indicates a logic 0. An RS-232-C interface allows for a −5 to −15 V level to represent a logic 1 and a +5 to +15 V level for a logic 0, although ±12 V levels are commonly used. The TTL RS-232-C equivalent uses standard TTL voltage levels to represent the 1 and 0 values.

The heart of any serial interface is the circuit that will do the parallel-to-serial or serial-to-parallel conversion. This circuit can perform the operation synchronously, asynchronously, or both ways. If it performs only synchronous operations it is often referred to as a USRT (universal synchronous receiver/transmitter). If it performs only asynchronous operations it is referred to as a UART (universal asynchronous receiver/transmitter). And if it performs both it is a USART (universal synchronous/asynchronous receiver/transmitter). These circuits, almost as complex as the microprocessor they interface to, can accept parallel data words on one set of inputs and deliver a serial pulse train on another output. The other half of the circuit accepts a serial pulse train, removes the start, stop, and parity bits, and delivers a parallel data word.

Most UARTs, USARTs and USRTs can also detect some error conditions—they can detect a missing stop bit (usually called a framing error FE), an error in parity (referred to as a parity error PE), or a mismatch of data transmission speeds (called an overrun error OE). When any of these error conditions is detected by the circuit, one of the three error output lines can be used to trigger external logic for an indication to the user. However, the UART, USART, or USRT doesn't

do the job all by itself. It does connect to the computer bus along with appropriate buffers and timing circuits to generate the proper communication rates.

One commonly used circuit is the 8251 USART made by Intel Corp. It is designed to be controlled by the microprocessor and operate with almost any serial data-transmission technique. Housed in a 28-pin DIP, the 8251 requires just a 5-V power supply and a single-phase clock signal (the phase 2 of the computer clock works nicely). Data rates of up to 9600 baud can be handled by the 8251 in its asynchronous mode and up to 56,000 bits/second in the synchronous mode. Most circuit boards that mate to the S-100 bus can hold enough circuitry for at least two complete serial interfaces along with jumper selectable options for the type of serial voltage levels—RS-232-C, TTL, TTY, etc. Jumper wires or switches on the board are often used to hardwire the type of output used. Each input or output to or from the computer provides the computer with a means of communicating with the outside world—these "doorways" or "portals" are referred to as ports. Therefore, a card with two serial interfaces is said to provide two serial I/O ports.

The 8080A microprocessor can directly communicate with up to 256 ports, although it's rare to find a case where more than one tenth that number are being used. Basically, the operation of the 8080A's I/O structure can best be considered an array of memory locations that can be read from or written to. Special instructions provided in the 8080A are just intended to transfer information from the processor to the interface, and have no effect on the memory space. This technique is often referred to as isolated I/O.

Another popular technique to handle I/O ports is to use the memory locations in what is called memory-mapped I/O. This method permits the CPU to manipulate the I/O port with the same instructions it uses to manipulate the data held in a memory location. However, to do this you have to use part of the address space of the memory—usually half of it if you want a simple control circuit. The control signals necessary to provide memory-mapped I/O are generated from the $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ signals and the highest-order address bit A15 (Fig. 6.2). When addresses of less than 32k are being addressed the A15 line is LOW and both gate outputs are HIGH, indicating that the memory is in use. However when A15 goes HIGH the condition of either $\overline{\text{MEMR}}$ or $\overline{\text{MEMW}}$ lines determines whether the output port is in an input or output mode. I/O devices are still considered addressed as ports, but instead of the accumulator in the microprocessor as the only transfer medium, any of the internal registers can be used. And all instructions that could be used to operate on memory locations can now be used in I/O operations.

With both systems of I/O, the addressing of each device can be configured to optimize efficiency and reduce component count. If the non-memory-mapped

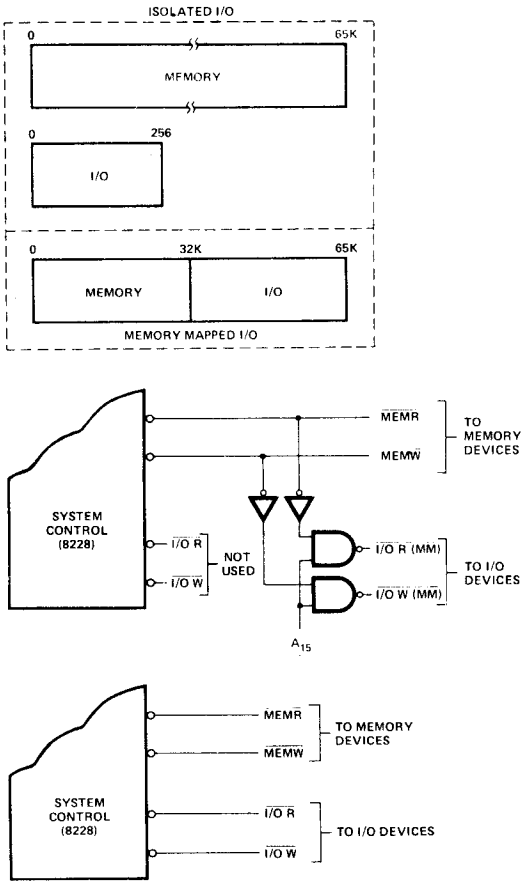
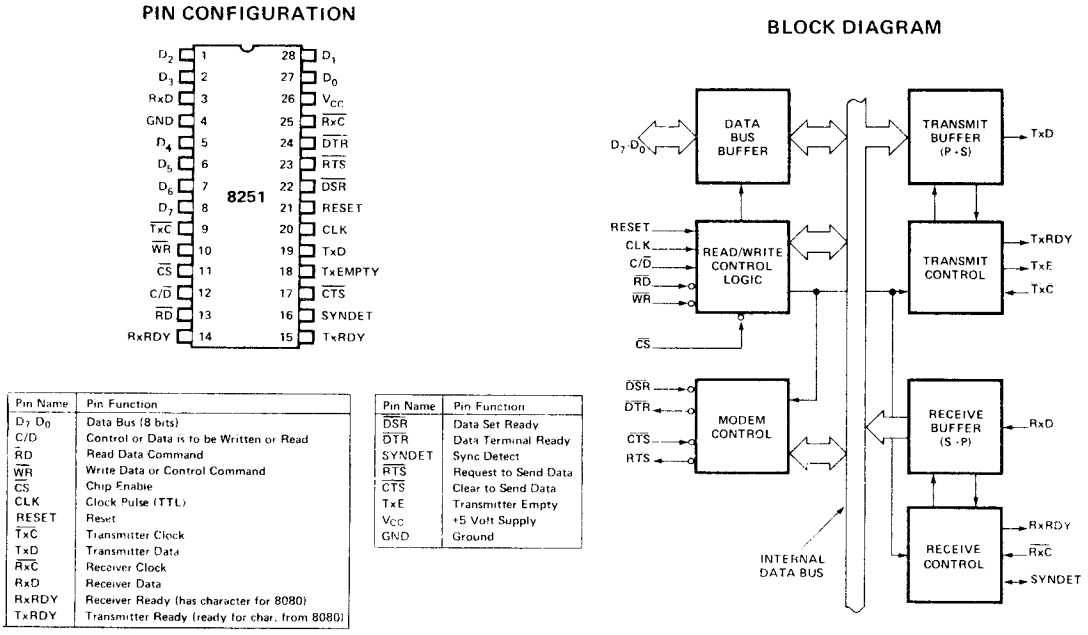


Fig. 6.2 Comparison of memory-mapped vs. isolated I/O circuit techniques.

I/O is used, the port address is delivered to the rest of the system by using the eight high-order address bits on the second machine cycle of the input or output instruction. The address code for the desired port is set up on the bus by the CPU and is decoded by the particular card it is intended for. Let's take a look at what one manufacturer has done to make a dual serial input card for an 8080A- or Z-80-based computer.

The SIO card developed by Imsai provides two serial data ports that can accept either the 20-mA current-loop inputs from a teletypewriter or the TTL level signals from a CRT terminal. Operation of the board requires 16 I/O port address locations, which are selected by address bits 0 to 3. When the board is used with input and output instructions, address bits 4 to 7 form the remainder of the board address and are jumper selectable. The upper byte of the board address is fixed at FE16, thus permitting all the lower eight bits to be used for port addressing—from 0 to 255. A total of 16 port addresses are available to the board so that up to 16 boards can be installed in a system. And, to permit the computer to control the boards, the lower four bits of the address are used to control: the direction of data flow, A3; channel B on or off, A2; channel A on or off, A1; and the command/data line on the 8251, A0.



Pin Name	Pin Function
D_7 D_0	Data Bus (8 bits)
C/D	Control or Data is to be Written or Read
$\overline{RD}$	Read Data Command
$\overline{WR}$	Write Data or Control Command
$\overline{CS}$	Chip Enable
CLK	Clock Pulse (TTL)
RESET	Reset
$\overline{TxC}$	Transmitter Clock
TxD	Transmitter Data
$\overline{RxC}$	Receiver Clock
RxD	Receiver Data
RxRDY	Receiver Ready (has character for 8080)
TxRDY	Transmitter Ready (ready for char. from 8080)

Pin Name	Pin Function
$\overline{DSR}$	Data Set Ready
$\overline{DTR}$	Data Terminal Ready
SYNDET	Sync Detect
$\overline{RTS}$	Request to Send Data
$\overline{CTS}$	Clear to Send Data
TxE	Transmitter Empty
V_{CC}	+5 Volt Supply
GND	Ground

Fig. 6.3 Block diagram and pinout of the 8251 universal synchronous/asynchronous receiver/transmitter (USART) used to form the programmable serial interface on the Imsai dual-channel serial I/O board. (*Courtesy* Intel Corp.)

The USARTs permit extensive program control of the input and output functions including RS-232 control line and SYNC character selection in the synchronous mode and error condition sensing and recovery capability. The board generates an interrupt (a signal to the CPU to stop what its doing and go to a different program) for received characters, empty transmitter buffers, and SYNC characters detected. Provisions are available for jumper selecting the priority of the interrupt, which works in conjunction with another board that can be plugged into the card cage (a priority interrupt controller).

Study the 8251 Before Using Board

Inside the 8251 USART are seven function subsystems (as shown in Fig. 6.3) that manipulate and control the data flow—into or out of the computer. Three of the blocks are buffers but each does a different job. The data bus buffer is a three-state bidirectional buffer that interfaces the 8-bit data bus to all the subsystems in the 8251. The transmit buffer accepts the parallel data or control words coming in from the data bus buffer and, under control of another block, the transmit control, converts the parallel data to serial, sticking in the necessary control bits and parity signals when necessary. Doing the opposite job is the receive buffer, a circuit that accepts a serial data signal and removes the extra control bits under control of the receive control circuits and converts the serial word back to a parallel data format. Eight of the circuit's 28 pins are used for the data bus buffer and one each for the transmit and receive buffers.

Each of the transmit and receive control blocks requires three lines, a TxRDY signal output to indicate that the USART is ready to accept a character from the data buffer, a TxE output line to indicate that the buffer inside the USART is empty and another character can be sent, and a TxC input to accept the clock signal from the system to output data at the desired bit rate. Similarly, the receive control section has three lines—an RxRDY output line to indicate that the receive buffer has a character ready to be shifted into the data bus buffer section, an RxC input line to accept the system clock so that data can be received at the proper rate, and a SYNDET line that is used only in the synchronous operating mode as an input or output to aid in the reception of synchronous data streams.

All the other control lines of the 8251 connect to the modem control block (a modem is a device that can accept serial digital data and convert the data into sound so that transmission over a telephone line is possible). The modem control block has four lines—two that act as input control and two that act as output control. The DSR input signal is an input sent to the 8251 by the CPU and it is used to make sure the data set (the modem itself) is ready to accept data for transmission. In return, the 8251 has a DTR line that can be used to answer back and indicate that the modem is ready for transmission. A CTS signal sent by the computer enables the modem control circuit and permits the modem to transmit data if the TxEN bit in the control byte originally was set HIGH.

As part of the read/write control logic there are three basic functions present that are common to almost every other logic circuit used in a computer—a RESET

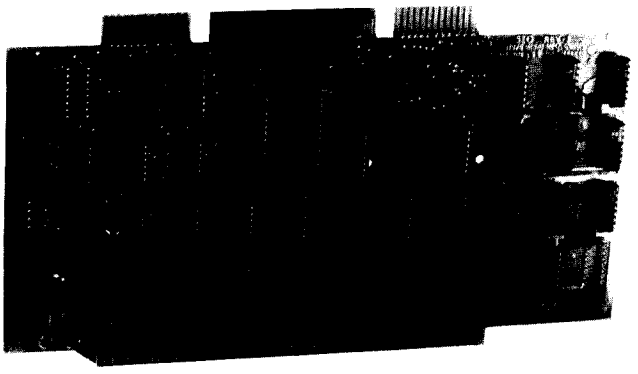


Fig. 6.4 The SIO 2-2 board made by Imsai provides two independent serial I/O channels. (Photo by J. Bierman)

line to put the 8251 into an IDLE mode until it receives a new control byte, a CLK line that accepts the system clock (phase 2) and generates all the 8251's internal timing (the clock must be at least 30 times to receive or transmit clocks during synchronous operation and 4.5 times for asynchronous operation), and a CS signal to enable or disable the 8251—no reading or writing will occur when the 8251 is disabled. The other three control signals on the USART include the C/D line, which in conjunction with the WR and RD lines informs the 8251 that the word on the data bus is either a data character, control word, or status information; the WR line is used to tell the 8251 that the CPU is sending information to the USART for transmission; the RD line indicates that the CPU is waiting for information currently being transmitted by the 8251.

The basic interface circuit then, requires that the 8251 be connected to the data bus and that the address bus control its operation by enabling it at the right time.

On the SIO-2-2 board developed by Imsai (Fig. 6.4), there are two mirror-image sections controlled by some address decoding and buffering circuits (Fig. 6.5). Operation of the board starts with setting up an address to enable the desired circuits. There are over half a dozen

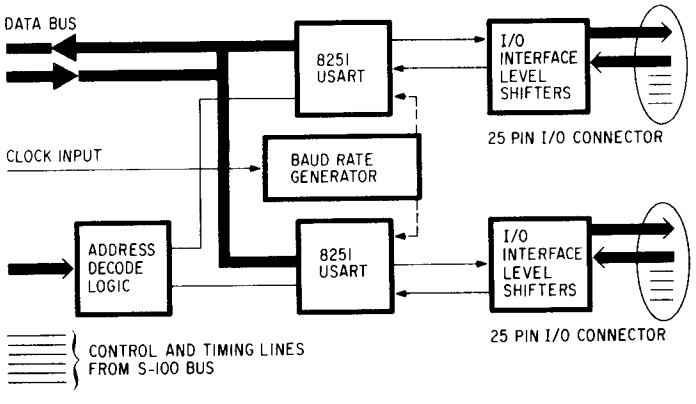


Fig. 6.5 Block diagram of the Imsai SIO 2-2 serial I/O board.

jumper sockets that can be used to not only set up the address, but control the baud rate, select the correct port, and interrupt the processor when a signal is coming in.

Address bits 1 and 2 select which port is being used, with address bit 1 controlling port A and bit 2 controlling port B. Address bit A0 determines whether the 8251 is in the CONTROL or DATA mode. The READ and WRITE strobe signal completes the control sequence and enables the 8251s to read or write data onto the data bus. Each 8251 has four control lines that are used for interrupting the processor. When interrupts aren't needed, the signals can be disabled by two of the output port bits (signals IEA or IEB).

To determine the proper communication rate, the 2 MHz phase-2 clock is divided to provide standard baud rates for jumper selection to ports A and B. Baud rates of 9600, 4800, 2400, 1200, 600, 300, and 150 are available. The 110 baud rate is derived by dividing the 2400 baud signal by 11 and then by 2.

Both data and control outputs of the USARTs are received or transmitted via TTL to RS-232 level converters. TTL data and control levels are also fed to the output port through open-collector driver circuits. Special current-loop input and output circuits for use with teletypewriters are also on the board. These circuits use optical isolators (LED/phototransistor pairs) to isolate the rest of the board circuitry from damaging voltage levels or signal noise generated by electromechanical terminal circuits. Current loops of either 20 mA or 60 mA can be handled.

To initialize the circuits so that a peripheral can be interfaced, line A0 must first be examined since it determines whether the 8251 responds to the current byte as a control or data byte. when A0 is HIGH the 8251 responds to the byte as a control byte, and when LOW as a data byte. Thus, to write a control byte into USART A, the lower four bits of the address would normally contain hex 3 or octal 3 while the normal address for channel B control bytes would be hex 5 or octal 5. Address bit 3 (A3) selects the board control I/O port—when HIGH the port is enabled. Thus, when the port is in use the lower four bits of the address would be hex 8 or octal 10.

Get the Board Up and Running

The control byte is actually divided into two 4-bit control words—the upper half of the byte serves channel B and the lower half serves channel A. These bits control the INTERRUPT ENABLE separately for each channel. When bits 0 and 4 are HIGH, the interrupts are enabled and the processor will receive data and interrupt its processing whenever any of the four USART control lines is active: TxRDY, TxE, RxRDY, SYDET. If bits 0 or 4 on each channel are LOW, no interrupts can be generated for the appropriate channel. Data

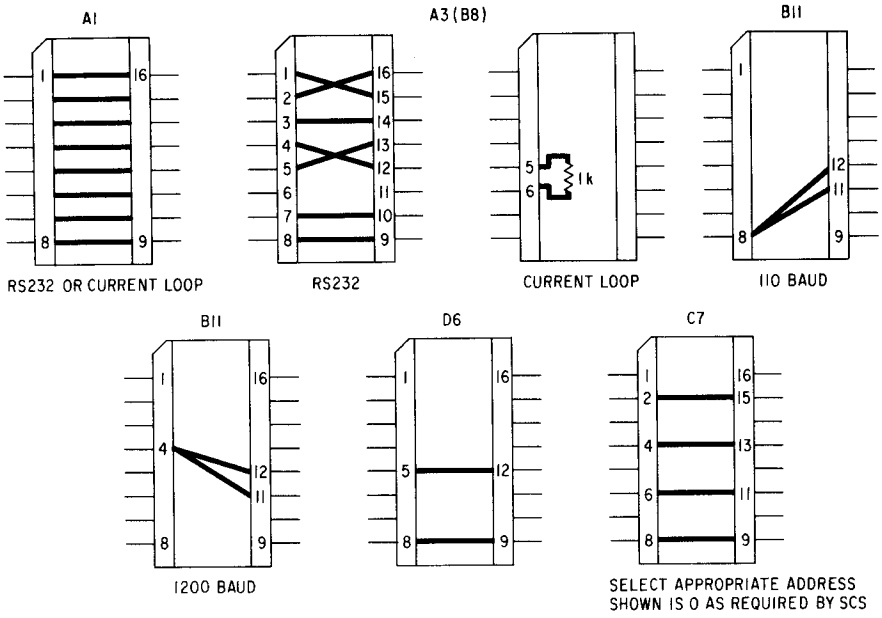


Fig. 6.6 Typical jumper interconnects to set up the Imsai board for a simple serial interface on one of the two channels.

bus lines 1 and 5 are used for channels A and B, respectively, to signify the CARRIER DETECT signal. These lines are active only when a jumper in board socket BJ has selected the board to act as the originator of the CARRIER DETECT signal. (All jumper arrangements are shown in Fig. 6.6.)

Bits 2, 3, 6, and 7 are not used in the OUTPUT mode but are used in the INPUT mode. Similarly, bits 0, 1, 4, and 5 are not used in the INPUT mode but are used in the OUTPUT mode. Lines for bits 2 and 6 read the condition of the CARRIER-DETECT RECEIVE for channels A and B, respectively. This line is only activated when jumpered at socket location BJ so it will read the condition of the CARRIER-DETECT line. Bits 3 and 7 serve channels A and B, respectively, to read the condition of the CTS control signal since the 8251s cannot do that directly through their programmed input.

All the output interface circuits are specialized devices for a particular application. For instance, TTL output levels are available from a 75452 dual peripheral driver that has open-collector outputs pulled up to 5 V with 226 Ω resistors. TTL input lines can accept a 1-TTL-normalized load and have a 1 kΩ pull-up resistor to +5 V. Thus, when they're not used, the inputs are held HIGH and minimize overall power drain.

If the current loop inputs are used, the TTL data input line must be left open and not held HIGH. The current loop input uses optoisolators and can respond to either 20 or 60 mA. A separate transistor is used to switch the current loop through the isolator and it is provided with a transient shunting diode so that it can be used to drive relays without risking damage to the output circuit.

Setting the baud rate for each channel is just a matter of using a few jumpers in socket RJ. Table 6.1 shows the rates and jumper requirements when the 8251 is programmed for a ×16 asynchronous clock rate and a ×1 synchronous clock.

Other jumper areas on the circuit board are used to handle the RS-232 and CARRIER-DETECT functions. Jumper areas CJ-A and CJ-B are used to permit the serial RS-232 ports on channels A and B, respectively, to be wired so that the port can serve as either the terminal end or the computer end of the RS-232 line. This eliminates any special wiring. Each jumper area is actually a 16-pin DIP socket. If, for instance, the board is to serve as the computer end of the RS-232 line, the jumpers must be set as shown in Fig. 6.7. Thus the TRANSMIT DATA circuit is now driving what is received data for the terminal, and the RECEIVE DATA circuit is receiving what is transmitted data from the terminal. Similarly, REQUEST-TO-SEND and CLEAR-TO-SEND lines are reversed and so are the DATA-SET-READY and DATA-TERMINAL-READY lines. Also available on the jumper socket are ground and +5 V tie points to provide permanent mark or space levels to any of the control lines if the CLEAR-TO-

Table 6.1 Baud Rates and Jumper Requirements for the 8251 USART

Socket Pin	1	9600 Asyn
	2	4800 A
	3	2400A (38,400 sync)
	4	1200A (19,200 sync)
	5	600A (9600 sync)
	6	300A (4800 sync)
	7	150A (2400 sync)
	8	110A
	14	75A (1200 sync)

Connect any of the above to socket pins 11 or 12 for sections B and A, respectively.

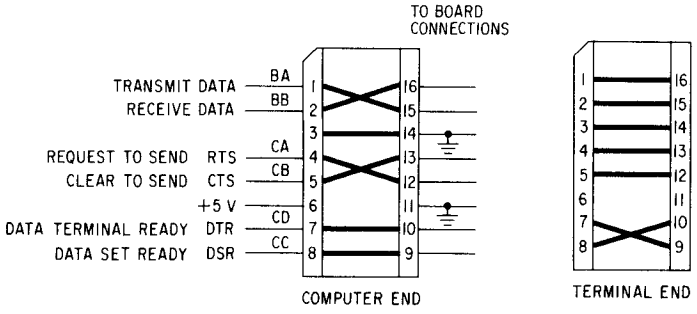


Fig. 6.7 Jumper arrangements on one channel for a complete RS-232 board interface. Jumper connection 3 to 14 must always be made.

SEND is not driven by an external source. If a constant enable signal is needed for the transmitter section of the USART the +5 V signal should be jumpered to pin 6.

Jumper socket BJ is used to set whether the CARRIER-DETECT signal is being originated or received by the SIO board. It is also used to jumper the control lines between channel A and channel B for applications where the control lines are bypassed and data are intercepted and handled. The four primary control lines for both channel A and channel B appear in this jumper socket, and can be connected as desired.

Remember, only one source should drive an RS-232 line at a time. If the control lines are jumpered so that the modem and data terminal are driving the lines, then appropriate jumpers in the CJ sockets should be removed so that the SIO circuits will not try to drive the control lines at the same time. However, if a signal on the DATA-TERMINAL-READY line must be detected, then a jumper must be connected in the BJ socket between pins 5 and 6 for channel A or between pins 11 and 12 for channel B. And, if the board must originate the CARRIER-DETECT line, jumpers should be placed between pins 5 and 7 for channel A and 10 and 12 for channel B, instead. Both ground and +5 V are also available in the BJ socket to provide a permanent mark or space level to any of the control lines.

The interrupt lines from channels A and B are both available on the interrupt select socket. All eight of the possible priority interrupt lines that are allotted on the main computer bus are also wired to the jumper socket. A jumper can then be placed between the appropriate channel's interrupt line and any one of the priority interrupt system lines to provide an interrupt of the desired priority.

For synchronous operation, jumper socket DJ provides facilities for originating and receiving clock signals used for receive and transmit. One half of the socket controls lines for channel A and the other half handles channel B. Pins 1, 2, 3, 4, 13, 14, 15, and 16 serve channel A and pins 5, 6, 7, 8, 9, 10, 11, and 12 are for channel B. When the SIO is used to originate the CLOCK signal, the pins for that channel should be jumpered straight across so that the CLOCK signal from the SIO board is fed into RS-232 drivers and onto the DD and DB lines. The inputs to the data-clock-receive circuits are tied to −12 V to provide an interactive output to the OR gate that supplies the receive CLOCK to the USART. If, instead, the SIO board must receive the CLOCK from the RS-232 lines, DD and DB must be jumpered to the input of the clock receive circuits. When this is done, the data-rate-select socket for the appropriate channel is held at ground, or LOW, to avoid interference between the on-board CLOCK circuit and the incoming CLOCK from the RS-232 lines.

The data-rate-select socket lets you set up different baud rates for both channels A and B from the standard rates provided by the SIO board circuits. Clock lines for channels A and B are completely independent and may be jumpered to the same or different rates. When the 8251 is used in a synchronous mode, the WAIT uses a ×1 clock rate rather than the ×16 rate used in the asynchronous mode. Thus, depending on the mode, the clock rates will be different by a factor of 16.

Two more jumper sockets are used to select the board address and one of the two possible modes. As mentioned earlier, the board can respond to both I/O instructions or memory-access instructions. For the board to respond to I/O commands, the port select 0 socket must be set up so that pins 8 and 9 are connected and pins 5 and 12 are jumpered. Then, the port select 1 socket must be jumpered so when the desired I/O port address appears on the address lines, the inputs to the NAND gate from bits A4 to A7 are HIGH. If, for instance, address bit 5 should be 0 for the board to respond, pins 4 and 13 should be jumpered while pins 3 and 14 should be left open. On the other hand, if bit 6 should be 1, pins 3 and 14 or 3 and 13 should be jump-

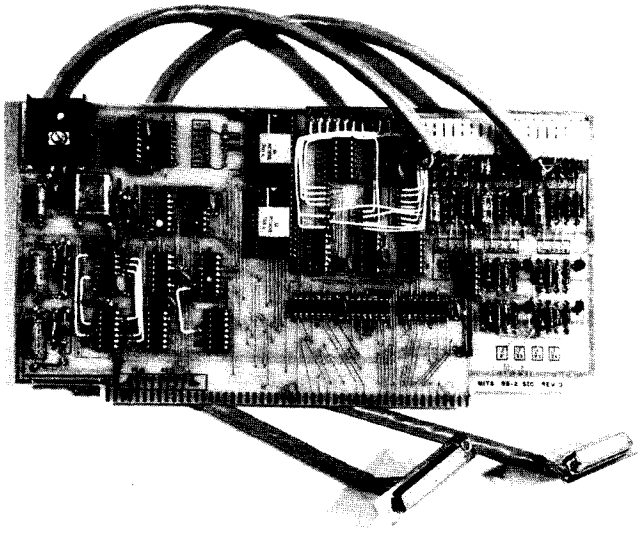


Fig. 6.8 The Pertec 88-2SIO serial interface board. (*Courtesy* Pertec)

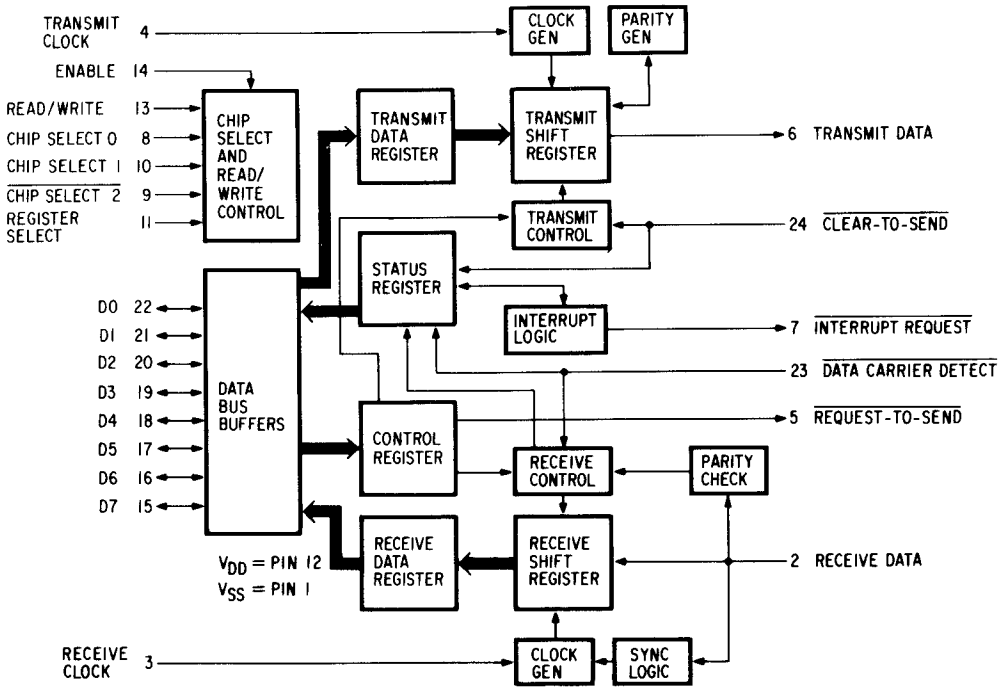


Fig. 6.9 Block diagram of the asynchronous communications adapter used on the Pertec 88-2SIO board. (*Courtesy* Motorola)

ered and pin 4 should be left unconnected. However, when jumpers are used, pins 3 and 13 should be connected so the slanted jumper will stand out to indicate a 1 and pins 4 and 13 jumpered to indicate a 0.

For the board to be used in a memory-mapped I/O format, the I/O port select 0 jumper socket must have jumpers set between pins 7 and 10, and 6 and 11 (the previous jumpers are, of course, removed). The remaining address jumpers are inserted as described in the previous paragraph. When the SIO is used as a memory-mapped I/O board, all instructions that normally affect the memory will operate on the I/O ports. For example, an INCREMENT-MEMORY command would read the data from the addressed input port, increment that data by one, and output it on the same address output port.

Another Path to the Same End

The board developed by Imsai is not the only way to accomplish the task of connecting a serial piece of equipment to the computer. For instance, MITS also offers a board that contains two serial ports, the 88-2SIO (Fig. 6.8). However, their board doesn't use the 8251s; instead it uses two 6850s, an asynchronous communications interface adapter (ACIA) designed by Motorola (Fig. 6.9). the ACIAs contain both the control register and the status register so that most operating options can be programmed into the devices. The only two options that must be jumper selected are the address select and the baud-rate select. And, if necessary, the baud-rate select can be altered under program control. Unlike the board from Imsai, the 88-2SIO can only perform asynchronous operations instead of both asynchronous and synchronous. And, the board is only used in the port I/O addressing mode.

With 256 addresses reserved for I/O devices, the computer uses special IN and OUT instructions from the CPU to distinguish the I/O addresses from normal memory addresses. Each instruction contains two bytes and requires three machine cycles to complete. During machine cycles one and two (M1, M2) bytes 1 and 2 of the instruction are read from the memory by the CPU. Byte 1 is the instruction code (IN = DB_{16}, OUT = $D3_{16}$) and byte 2 is the device (port) address (00 to FF_{16}). During machine cycle 3 (M3) data are transferred as follows: In the first clock period (T1) of M3, the I/O device address (byte 2 of the instruction) is placed on the address bus. Only eight bits are used for I/O device addresses so the address appears on both the lower lines (A0 to A7) and the upper lines (A8 to A15). The status signals of T2 distinguish device address operation from memory operations. During T2, the status information is latched and sent to the system bus. The status signal for the IN command is SINP, and SOUT for the OUT instruction. Except for the I/O status signal, T3 of M3 appears identical to a memory operation.

Data are strobed into the accumulator with PDBIN for an IN operation and strobed out of the accumulator with the $\overline{\text{PWR}}$ signal for an OUT command. The device and interface are responsible for assuring that data are on the bus when the CPU requires it and that the device receives data during $\overline{\text{PWR}}$ in an OUT operation.

Address				
A0	A1	Port No.	Output Function	Input Function
0	0	1	Control register	Status register
1	0	1	Output data	Input data
0	1	2	Control register	Status register
1	1	2	Output data	Input data

Fig. 6.10 Data transfer and control/status channel address definitions.

Address	Function
68	1st port, control/status
69	1st port, data in/out
70	2nd port, control/status
71	2nd port, data in/out

Fig. 6.11 Board address selection chart.

Addresses are selected via jumpers for address lines A2 to A7 (A0 and A1 are already hardwired). The six lines provide 64 possible addresses that are selected in increments of four. All four addresses skipped over are used as the data transfer and control/status channels of port 0 and 1 (Fig. 6.10). Thus, the first board would require addresses 00 to 03, the second 04 to 07, and the last board FB to FF (all numbers in hex). However, address FF is the front-panel sense switch and should not be used. Address lines A0 and A1 cause register selection and port selection, respectively. These lines are software controlled and select registers and ports according to the chart in Fig. 6.10. If, for example, address 44_{16} is selected via jumpers as the starting point, the four board addresses are those shown in Fig. 6.11. Next, if an OUT, 46 instruction (D3 46_{16}) is executed, the contents of the accumulator would be written into the control register of the second port on the board, which is strapped at location 44_{16}.

Each port can be controlled by the computer by feeding an 8-bit control byte into the port's control register. The byte is structured as shown in Fig. 6.12a and the first two bits, 0 and 1, control the internal clock

7	6	5	4	3	2	1	0
In Interrupt	Out Interrupt		Transmission Bits			Clock Divide And Reset	

Note: Data bit LOW = 0
Data bit HIGH = 1

(A)

Bit 1	Bit 0	Function
0	0	÷ Clock by 1
0	1	÷ Clock by 16
1	0	÷ Clock by 64
1	1	Master reset

(B)

Desired Baud Rate	Selected Baud Rate
27.5	110
37.5	150
75.0	300
450	1800
600	2400

(C)

Data Bit			Function		
4	3	2	No. of Data Bits	No. of Stop Bits	Parity
0	0	0	7	2	Even
0	0	1	7	2	Odd
0	1	0	7	1	Even
0	1	1	7	1	Odd
1	0	0	8	2	None
1	0	1	8	1	None
1	1	0	8	1	Even
1	1	1	8	1	Odd

(D)

Data Bit			Function
7	6	5	
X	0	0	$\overline{RTS}$ = LOW, transmitting interrupt disabled.
X	0	1	$\overline{RTS}$ = LOW, transmitting interrupt enabled.
X	1	0	$\overline{RTS}$ = HIGH, transmitting interrupt disabled.
X	1	1	$\overline{RTS}$ = HIGH, transmits a break level on the transmit data output, transmit interrupt disabled.
0	X	X	Receive interrupt disabled.
1	X	X	Receive interrupt enabled.

X = does not matter.

(E)

Octal Code	Function
076 003 323 000	Reset port
076 261 323 000	Set up for 8 data bits, ÷ 16 mode, $\overline{RTS}$ = LOW, transmit and receive interrupts enabled.

(F)

Fig. 6.12 The control byte used to set up the 6850 is broken into many subsections (a). The first two bits control the internal clock divide circuit (b), the frequency of which can be further divided by external circuit connections (c). The next three bits determine word length, parity, and the number of stop bits (d), and the other three bits control system interrupts and I/O handshakes (e). A complete initialization for a teletypewriter with eight data bits, two stop bits, and no parity is shown in (f).

divide circuit and the master reset of the 6850 according to the chart in Fig. 6.12b. A typical sequence would first get both bits equal to 1 and reset the ACIA, next bits 0 and 1 are set to 1 and 0 respectively since the normal incoming clock frequency is 16 times the baud rate. Any of eight baud rates from 110 to 9600 can be jumper selected, and five additional rates can be generated by dividing the selected rate by 64 (Fig. 16.12c). Note that the selected rate is four times greater than the desired rate since the incoming frequency is 16 times the baud rate and is divided by 64 in the ACIA ($64 \div 16 = 4$).

Bits 2 to 4 of the control register determine the word length, parity, and the number of stop bits. The particular peripheral used determines how the control bits are set. All eight codes are shown in Fig. 6.12d. A typical code pattern might be 100—eight data bits, two stop bits, and no parity. The other three bits of the control register control system interrupts and I/O device handshakes, as shown in Fig. 6.12e. For a port set up at address 00, the complete initialization when a teletypewriter is connected with eight data bits, two stop bits, and no parity is shown in Fig. 6.12f. Both receive and transmit interrupts are enabled.

Status Information Is Important Too

Information describing the status of the port is also available by reading the ACIA status register. Information stored in this register indicates the status of the transmit data register, the receive data register, error logic, and the peripheral/modem status inputs of the port. There are eight bits used to indicate different signals for each line.

Bit 0, RECEIVE DATA REGISTER FULL (RDRF): This bit indicates when the received data have been transferred to the received data register. RDRF is cleared after the register has been read or given a MASTER RESET signal. The cleared, or empty state, indicates that the contents of the register are not current. However, if the DATA CARRIER DETECT line is HIGH, the RDRF line will also indicate the register is empty.

Bit 1, TRANSMIT DATA REGISTER EMPTY (TDRE): When this bit is HIGH it indicates that the transmit data register contents have been transferred and that new data can be entered. The LOW state indicates that the register is full and that transmission of a new character has not begun since the last WRITE-DATA command.

Bit 2, DATA CARRIER DETECT (DCD): This bit is HIGH when the DCD input from a modem has gone HIGH to indicate that a carrier is not present. When the DCD input line goes HIGH, it causes an interrupt request to be generated when the RECEIVE INTERRUPT ENABLE is set. It remains HIGH after the DCD input is returned LOW until cleared by first reading the status register and then the data register or until a MASTER RESET signal occurs. If the DCD input remains HIGH after a READ STATUS, READ DATA, or MASTER RESET has occurred, the DCD status bit remains HIGH and will follow the DCD input.

Bit 3, CLEAR TO SEND (CTS): This bit indicates the state of the CLEAR TO SEND input from a modem. A LOW CTS indicates that there is a CTS signal from a modem. When the CTS bit is HIGH the TDRE bit is inhibited. The MASTER RESET signal does not affect the CTS status bit.

Bit 4, FRAMING ERROR (FE): When HIGH, this bit indicates that the received character is improperly framed by a start and a stop bit and is detected by the absence of the first stop bit. A synchronization error, a faulty transmission, or a break condition will all signal an error. The FE line is set or reset during the receive data transfer time. Therefore, this error indicator is present throughout the time that the associated character is available.

Bit 5, RECEIVER OVERRUN (OVRN): This is also an error indicator that lets you know if one or more characters in the data stream were lost. That is, a character or a number of characters were received but not read from the RECEIVE DATA REGISTER prior to subsequent characters being received. The overrun condition begins at the end point of the last bit of the second character received in succession without a read operation performed on the RECEIVE DATA REGISTER. The OVRN signal does not occur in the status register until the valid character prior to OVRN has been read and the RDRF bit remains set until OVRN is reset. Character synchronization is maintained even during the OVRN condition but to reset the OVRN indicator data must be read from the RECEIVE DATA REGISTER or the circuit receives a MASTER RESET signal.

Bit 6, PARITY ERROR (PE): The PE signal, when HIGH, indicates that the number of 1s in the received character does not agree with the preselected odd or even parity. (Odd parity is defined as when all the 1s add up to an odd number and even parity when all the 1s add up to an even number.) As long as the mismatched character is in the RECEIVE DATA REGISTER the PE line will be HIGH. If no parity is selected, then both the transmitter parity generator output and the receiver parity check results are inhibited.

Bit 7, INTERRUPT REQUEST (IRQ): This signal indicates the state of the IRZ output. Any interrupt condition with its applicable enable will be indicated in this status bit. Anytime the IRZ output is LOW, the IRQ bit will be HIGH to indicate the interrupt or service request status.

Connecting the optional jumpers on the board depends on the type of terminal being used, the desired communications rate, and the address of the port. A typical sequence of jumper set up might be the following:

The four octal addresses noted in the chart below represent the control and data channels of Ports 0 and 1. Even-numbered addresses indicate control channels and odd-numbered addresses indicate data channels. The first two addresses are Port 0 and the second two are Port 1, so that for addresses 000-003, 000 is the control channel of Port 0; 001 is the data channel of Port 0; 002 is the control channel of Port 1; and 003 is the data channel of Port 004.

I/O ADDRESS SELECTION CHART

Address Octal	Connections					
000-003	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$
004-007	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	A2
010-013	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	A3	$\overline{A2}$
014-017	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	A3	A2
020-023	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	A4	$\overline{A3}$	$\overline{A2}$
024-027	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	A4	$\overline{A3}$	A2
030-033	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	A4	A3	$\overline{A2}$
034-037	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	A4	A3	A2
040-043	$\overline{A7}$	$\overline{A6}$	A5	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$
044-047	$\overline{A7}$	$\overline{A6}$	A5	$\overline{A4}$	$\overline{A3}$	A2
050-053	$\overline{A7}$	$\overline{A6}$	A5	$\overline{A4}$	A3	$\overline{A2}$
054-057	$\overline{A7}$	$\overline{A6}$	A5	$\overline{A4}$	A3	A2
060-063	$\overline{A7}$	$\overline{A6}$	A5	A4	$\overline{A3}$	$\overline{A2}$
064-067	$\overline{A7}$	$\overline{A6}$	A5	A4	$\overline{A3}$	A2
070-073	$\overline{A7}$	$\overline{A6}$	A5	A4	A3	$\overline{A2}$
074-077	$\overline{A7}$	$\overline{A6}$	A5	A4	A3	A2
100-103	$\overline{A7}$	A6	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$
104-107	$\overline{A7}$	A6	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	A2
110-113	$\overline{A7}$	A6	$\overline{A5}$	$\overline{A4}$	A3	$\overline{A2}$
114-117	$\overline{A7}$	A6	$\overline{A5}$	$\overline{A4}$	A3	A2
120-123	$\overline{A7}$	A6	$\overline{A5}$	A4	$\overline{A3}$	$\overline{A2}$
124-127	$\overline{A7}$	A6	$\overline{A5}$	A4	$\overline{A3}$	A2
130-133	$\overline{A7}$	A6	$\overline{A5}$	A4	A3	$\overline{A2}$
134-137	$\overline{A7}$	A6	$\overline{A5}$	A4	A3	A2
140-143	$\overline{A7}$	A6	A5	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$
144-147	$\overline{A7}$	A6	A5	$\overline{A4}$	$\overline{A3}$	A2
150-153	$\overline{A7}$	A6	A5	$\overline{A4}$	A3	$\overline{A2}$
154-157	$\overline{A7}$	A6	A5	$\overline{A4}$	A3	A2
160-163	$\overline{A7}$	A6	A5	A4	$\overline{A3}$	$\overline{A2}$
164-167	$\overline{A7}$	A6	A5	A4	$\overline{A3}$	A2
170-173	$\overline{A7}$	A6	A5	A4	A3	$\overline{A2}$
174-177	$\overline{A7}$	A6	A5	A4	A3	A2
200-203	A7	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$
204-207	A7	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	A2
210-213	A7	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	A3	$\overline{A2}$
214-217	A7	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	A3	A2
220-223	A7	$\overline{A6}$	$\overline{A5}$	A4	$\overline{A3}$	$\overline{A2}$
224-227	A7	$\overline{A6}$	$\overline{A5}$	A4	$\overline{A3}$	A2
230-233	A7	$\overline{A6}$	$\overline{A5}$	A4	A3	$\overline{A2}$
234-237	A7	$\overline{A6}$	$\overline{A5}$	A4	A3	A2
240-243	A7	$\overline{A6}$	A5	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$
244-247	A7	$\overline{A6}$	A5	$\overline{A4}$	$\overline{A3}$	A2
250-253	A7	$\overline{A6}$	A5	$\overline{A4}$	A3	$\overline{A2}$
254-257	A7	$\overline{A6}$	A5	$\overline{A4}$	A3	A2
260-263	A7	$\overline{A6}$	A5	A4	$\overline{A3}$	$\overline{A2}$
264-267	A7	$\overline{A6}$	A5	A4	$\overline{A3}$	A2
270-273	A7	$\overline{A6}$	A5	A4	A3	$\overline{A2}$
274-277	A7	$\overline{A6}$	A5	A4	A3	A2
300-303	A7	A6	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$
304-307	A7	A6	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	A2
310-313	A7	A6	$\overline{A5}$	$\overline{A4}$	A3	$\overline{A2}$
314-317	A7	A6	$\overline{A5}$	$\overline{A4}$	A3	A2
320-323	A7	A6	$\overline{A5}$	A4	$\overline{A3}$	$\overline{A2}$
324-327	A7	A6	$\overline{A5}$	A4	$\overline{A3}$	A2

I/O ADDRESS SELECTION CHART

Address Octal	Connections					
330-333	A7	A6	$\overline{A5}$	A4	A3	$\overline{A2}$
334-337	A7	A6	$\overline{A5}$	A4	A3	A2
340-343	A7	A6	A5	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$
344-347	A7	A6	A5	$\overline{A4}$	$\overline{A3}$	A2
350-353	A7	A6	A5	$\overline{A4}$	A3	$\overline{A2}$
354-357	A7	A6	A5	$\overline{A4}$	A3	A2
360-363	A7	A6	A5	A4	$\overline{A3}$	$\overline{A2}$
364-367	A7	A6	A5	A4	$\overline{A3}$	A2
370-373	A7	A6	A5	A4	A3	$\overline{A2}$
374-377	A7	A6	A5	A4	A3	A2

Fig. 6.13 Address selection chart for the Pertec 88-2SIO board.

1. Select the desired board address via jumpers A2 to A7, making sure that the address is not used by another board (Fig. 6.13).
2. Wire the 10-pin connector for each port being used for either RS-232, TTL, or 20 mA interfaces to the D-type connector used on the rear of the cabinet (Fig. 6.14).
3. Select the desired baud rate for each port.
4. Decide on the interrupt priority by using a single-level interrupt system, no interrupt at all, or even a multiple-priority interrupt using an optional 88-Vector Interrupt control card.

Don't Want Serial? Go Parallel

Although serial communications circuits are the simplest to interconnect, it takes longer for data words to be transmitted since each 8-bit word is sent along with start and stop bits as well as possibly a parity bit. Therefore, typically 10 clock pulses are required to send each word. Transmission can be sped up by that factor of 10 if all the bits in a data word are transmitted simultaneously (in parallel). Special circuits that can interface the CPU to the outside world can be built to accept parallel data words. For example, to get in and out of the Altair computer system MITS offers the 88-4PIO, a quadruple parallel port (Fig. 6.15). The board uses four 6820s (peripheral interface adapters developed by Motorola) to provide four 8-bit I/O ports (Fig. 6.16).

The basic input and output timing operations are the same as used for the 88-2SIO board discussed on page 61. However, each 4-PIO board requires 16 addresses, four for each port. On the diagram for the board shown in Fig. 6.17, only four jumpers are needed to set up the board address. Thus any one of 16 88-4PIO boards can be selected.

Address lines A2 and A3 enable selection of one of the four 6820s. Each port contains two sections, A and B, and each of the sections contains two channels—one

RS-232 Voltage Levels (±12 Volt Levels)

	Port 0		Port 1	
Signal	Jumper	Jumper	Jumper	Jumper
1. Receive	D3 to I4	I3 to S1-7	E3 to J1	J2 to S2-7
2. Transmit	D5 to N4	N3 to S1-8	E5 to N6	N5 to S2-8
3. Ground*	—	S1-4 to S1-10	—	S2-4 to S2-10
4. Clear to Send (CTS)	D2 to I7	I8 to S1-1	E2 to I2	I1 to S2-1
5. Data Carrier Detect (DCD)	D1 to I5	I6 to S1-2	E1 to J3	J4 to S2-2
6. Request to Send [(RTS), also can be used for Data Terminal Ready]	D4 to N2	N1 to S1-3	E4 to N8	N7 to S1-3

*If Receive, Transmit, and Ground are all that your I/O device requires, ignore the other signal connections.

TTL Voltage Levels

	Port 0		Port 1	
Signal	Jumper	Jumper	Jumper	Jumper
1. Receive	D3 to K12	K11 to S1-9	E3 to K2	K1 to S2-9
2. Transmit	D5 to M1	M2 to S1-5	E5 to M5	M6 to S2-5
3. Ground*	—	S1-4 to S1-10	—	S2-4 to S2-10
4. CTS	D2 to K8	K7 to S1-1	E2 to K6	K5 to S2-1
5. DCD and/or Data Terminal Ready	D1 to K10	K9 to S1-2	E1 to K4	K3 to S2-2
6. RTS, also can be used for Data Terminal Ready	D4 to M3	M4 to S1-3	E4 to M7	M8 to S2-3

*If Receive, Transmit, and Ground are all that your I/O device requires, ignore the other signal connections.

TTY 20 mA Current Loop

	Port 0				Port 1			
Signal	Jumper	Jumper	Jumper	Jumper	Jumper	Jumper	Jumper	Jumper
1. Receive	D3 to K12	K11 to Y7	Y8 to S1-6	Y9 to S1-7	E3 to K2	K1 to Y16	Y17 to S2-6	Y18 to S2-7
2. Transmit	D5 to L1	—	—	Z2 to S1-5	E5 to L3	—	—	Z1 to S2-5
3. Ground*	—	—	—	S1-4 to S1-10	—	—	—	S2-4 to S2-10
4. CTS	D2 to K8	K7 to Y1	Y2 to S1-8	Y3 to S1-9	E2 to K6	K5 to Y10	Y11 to S2-8	Y12 to S2-9
5. DCD and/or Data Terminal Ready	D1 to K10	K9 to Y4	Y5 to S1-1	Y6 to S1-2	E1 to K4	K3 to Y13	Y14 to S2-1	Y15 to S2-2
6. RTS, also can be used for Data Terminal Ready	D4 to L2	—	—	Z4 to S1-3	E4 to L4	—	—	Z3 to S2-3

*If Receive, Transmit, and Ground are all that your I/O device requires, ignore the other signal connections.

Fig. 6.14 I/O connector and interface jumper set-up procedure for the 88-2SIO board from Pertec.

for control and status, and one for data. Address lines A0 and A1 enable the selection of port section A or B and the selection of the control/status or data channel. If the address-selection jumpers are wired for 00 to 0F, the port, section, and channel addresses would appear as shown in Fig. 6.18.

Each port section, A and B, contains three registers, eight data lines, two control lines, and an interrupt-request output. One of the registers is the control/status register—a read/write register that holds eight bits split into four function groups (Fig. 6.19a)—interrupt request, C2 control, DDR control, and C1 control.

Bits 7 and 6, the interrupt-request bits, are unaffected during write operations but are used as interrupt indicators. Bit 7 is an interrupt-status bit that indicates the activity of the external control line C1, the input control line from the external device. C1 affects status bit 7 and the interrupt-request output to the system bus $\overline{IRQ}$. Control bits 1 and 0 define the operation of the C1 line as defined in Fig. 6.19b. Bit 7 and $\overline{IRQ}$ are reset

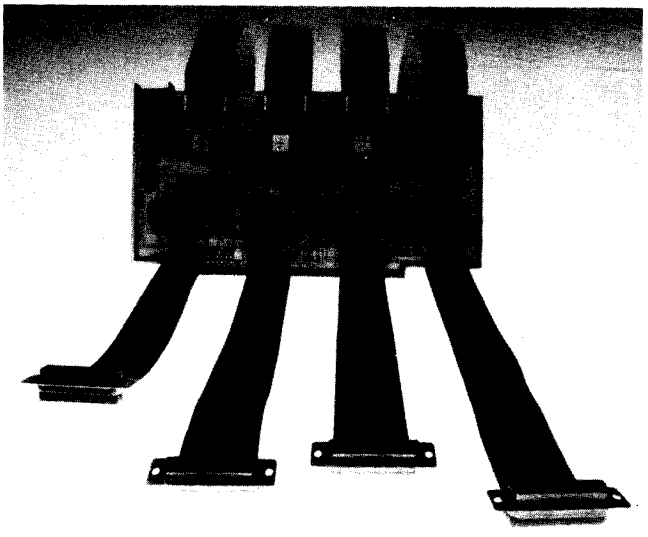


Fig. 6.15 The 88-4PIO board from Pertec offers four 8-bit parallel ports. (*Courtesy* Pertec)

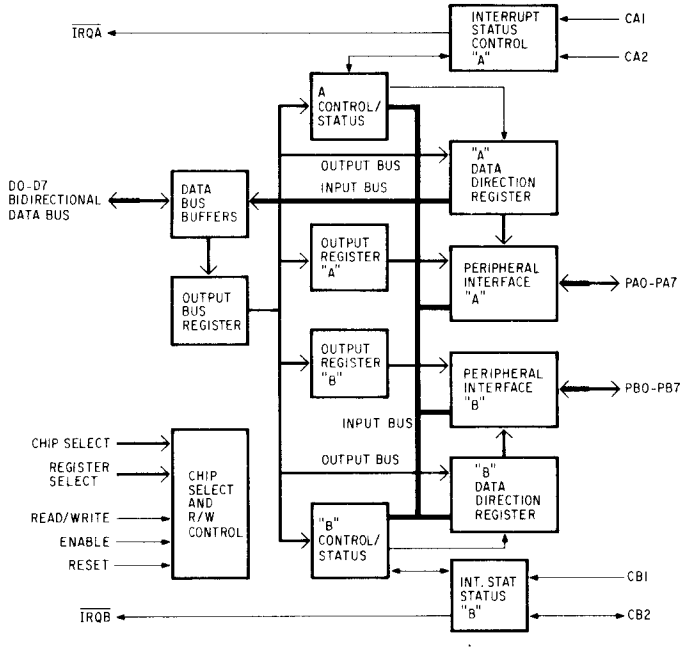


Fig. 6.16 Internal block diagram of the 6821 PIA dual parallel 8-bit port developed by Motorola.

(bit 7 goes LOW and $\overline{\text{IRQ}}$ goes HIGH) when the data register is read by the CPU. The C2 control line can function as either input or output for the I/O device. Control bits 3, 4, and 5 of the control/status word determine the operating mode of C2, as shown in Fig. 6.19c.

Sections A and B operate identically when C2 is set as an input. As an output, C2 functions uniquely for sections A and B. Each section has a C2 control line, marked as CB2 for the sections, respectively. Control bits 5, 4, and 3 are shown in Figs. 6.19d and e. In both cases, the enable pulse E is a strobe signal that locks the word into the port every machine cycle and also partially enables the port to read or write.

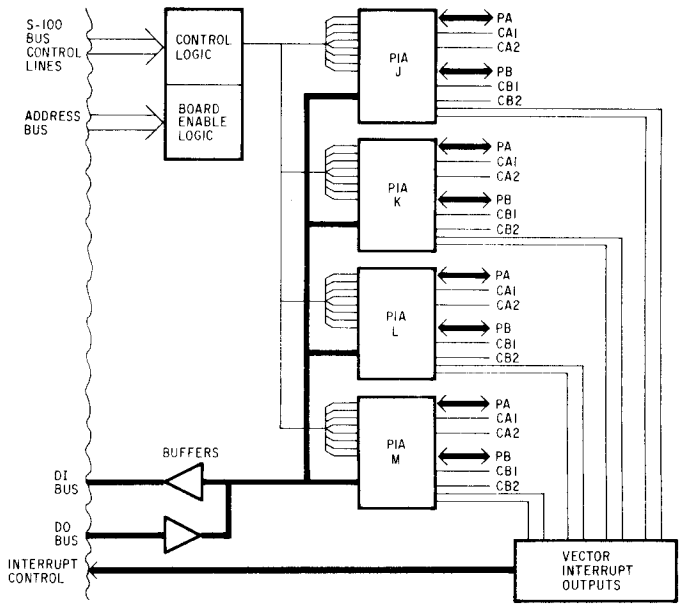


Fig. 6.17 Block diagram of the Pertec 88-4PIO board.

To write into a control register, each accumulator bit must be set according to the chart in Fig. 6.19d or e; next, an OUT instruction must be executed with byte 2 of the instruction equal to the control channel address. Also, data lines must be entered as input or output. All ports are reset when power is first applied, thereby resetting the data lines and the C2 line for both sections as inputs. The data-channel address permits access to either the data register or the data direction registers (DDR). The status of bit 2 in the control register determines whether the data register or the DDR is accessed. If bit 2 is LOW, DDR is accessed, and if HIGH, the data register is accessed. Writing a LOW into a bit of the DDR causes the corresponding data line to act as an input; writing a HIGH into a bit of the DDR causes the corresponding data line to act as an output. The program necessary to initialize a port to interface a parallel I/O device is shown in Fig. 6.20. The board is addressed at location 10 to 1F hex (16 to 31 decimal) and in the example shown, port 0 and addresses 10 to 13 hex are used. The addresses are set up so that 10 is A control, 11 is A data, 12 is B control, and 13 is B data. The A section of the port functions as an input and the B section functions as an output.

This initialization procedure causes the following communication between the I/O device and the CPU:

Input

1. If the I/O device has valid data, a strobe signal from the I/O device pulls the CA1 input LOW; bit 7 of the A control register goes HIGH; $\overline{\text{IRQA}}$ goes LOW; and CA2 goes HIGH. CA2 operates as a "busy" signal back to the I/O device.

ADD		PORT	IC LETTER	SECTION	CHANNEL
Octal	Decimal				
0	0	0	J	A	C
1	1				D
2	2			B	C
3	3				D
4	4	1	K	A	C
5	5				D
6	6			B	C
7	7				D
10	8	2	L	A	C
11	9				D
12	10			B	C
13	11				D
14	12	3	M	A	C
15	13				D
16	14			B	C
17	15				D

C = Control D = Data

Fig. 6.18 Port, section, and channel address definitions for board address jumpered for 00 to 0F.

2. If interrupts are used, an interrupt is generated and the next step is jumped. If the interrupts are not used, the CPU periodically inputs the A control register in order to interrupt the status of bit 7. When bit 7 is HIGH, the next step is entered.
3. Data are input to the accumulator which, in turn, resets bit 7, $\overline{\text{IRQA}}$, and CA2. The transition of CA2 tells the I/O device that new data may be entered.

Output

1. CB1 is pulled LOW by the I/O device when it is ready to receive new data.
2. If interrupts are used, an interrupt is generated and the next step is entered. If interrupts are not used, the CPU periodically inputs the B control register to examine the status of bit 7. When bit 7 is HIGH the next step will be entered.
3. Data are output to the I/O device and thus latched into the port outputs. CB2 goes LOW when the next E pulse goes HIGH and returns LOW when the E pulse goes HIGH again. The CB2 line can also be used to strobe data into the I/O device. Pulse width of the CB2 signal ranges from 1.5 to 3.5 μs, depending on the instruction being executed.

Communication between the I/O device and the CPU can be handled with software by having the CPU program periodically test the status of the I/O device. When a ready bit is detected, the I/O device is serviced. However, in some applications the CPU may not be able to do a periodic check because it's too busy, so the I/O device can interrupt the processor by signaling the processor on the $\overline{\text{IRQ}}$ line. There are two methods of interrupting the processor—the 88-VI board, an optional board, provides eight levels of vectored interrupt and is important in systems where several I/O devices of different priority must be serviced. The other option is just a single interrupt set up on the 88-4PIO board itself by jumpering the particular port interrupt request line(s) (JA, JB, KA, KB, etc.) to the $\overline{\text{PINT}}$ line. When an interrupt occurs the $\overline{\text{PINT}}$ lines goes LOW, and when the CPU finishes executing its current instruction, the program counter contents are pushed onto a stack (a reserved memory location) and the processor jumps to location 39. The I/O device service program must then start at location 39; the return instruction

Bit No.	7	6	5	4	3	2	1	0
Function	Interrupt Request		C2 Control			DDR Control	C1 Control	

Control Bits				
1	0	C1 Input	Status Bit 7	$\overline{IRQ}$ Output
0	1	Active LOW	Set HIGH when C1 is active	Disabled—remains HIGH
0	1	Active LOW	Set HIGH when C1 is active	Goes LOW when bit 7 goes HIGH
1	0	Active HIGH	Set HIGH when C1 is active	Disabled—remains HIGH
1	1	Active HIGH	Set HIGH when C1 is active	Goes LOW when bit 7 goes HIGH

Control Bits					
5	4	3	C2	Status Bit 6	$\overline{IRQ}$
0	0	0	Active LOW	Set HIGH when C2 is active	Disabled—remains HIGH
0	0	1	Active LOW	Set HIGH when C2 is active	Goes LOW when bit 7 is HIGH
0	1	0	Active HIGH	Set HIGH when C2 is active	Disabled—remains HIGH
0	1	1	Active HIGH	Set HIGH when C2 is active	Goes LOW when bit 7 is HIGH

A Section Control Bits			CA2	
5	4	3	Cleared	Set
1	0	0	LOW after E pulse, following read of A data channel	HIGH when CA1 is active
1	0	1	LOW after a read of A data channel	HIGH following next E pulse
1	1	0	Always LOW when bit 3 is low	
1	1	1		Always HIGH when bit 3 is high

B Section Control Bits			CB2	
5	4	3	Cleared	Set
1	0	0	LOW when E pulse goes HIGH, following a write of B data channel	HIGH when CB1 is active
1	0	1	LOW when E pulse goes HIGH, following a write of B data channel	HIGH when next E pulse goes HIGH
1	1	0	Always LOW when bit 3 is LOW	
1	1	1		Always HIGH when bit 3 is HIGH

Fig. 6.19 These five charts define the register set-up procedures for the 6821 parallel interface adapters used on the 88-4-PIO card.

may be used to end the service routine and bring the CPU back by returning the contents of the program counter from the stack.

Combine Functions onto a Single Board

If you don't want your system to be too large, you may not want to use a separate board for serial inputs and another board for parallel inputs. Well, you can often combine the functions on a single board—say two parallel ports, one serial port, and even more. That's what several companies have done—Processor Technology offers a board dubbed the 3P + S that holds three parallel ports and one serial port. Xitan/Technical Design Labs offers a board called the SMB that contains two serial I/O ports, a parallel I/O port, a serial cas-

Location	Instruction	Code	Description
0	MVIA	076	Set bit 2 of both control registers to 0 in order to write to the DDRs.
1	<B2>	000	
2	OUT	323	
3	<B2>	020	
4	OUT	323	
5	<B2>	022	
6	OUT	323	Write all 0s to DDR of A Section to enable A data lines to act as inputs. Write 1s into DDR of B Section to enable B data lines to act as outputs.
7	<B2>	021	
10	MVIA	076	
11	<B2>	377	
12	OUT	323	
13	<B2>	023	
14	MVI	076	Set A control register:*
15	<B2>	045	
16	OUT	323	
17	<B2>	020	
20	MVI	076	
21	<B2>	054	
22	OUT	323	Set B control register:*
23	<B2>	022	

7	6	5	4	3	2	1	0
		1	0	0	1	0	1

7	6	5	4	3	2	1	0
		1	0	1	1	0	0

*See description of C1 and C2 for above settings.

Fig. 6.20 This simple machine code program initializes a port to interface to a parallel I/O device.

sette interface port, as well as 2048 words of ROM-based programs and 2048 words of RAM storage, and Imsai offers a board dubbed the MIO that contains two parallel ports, one serial port, one serial cassette interface port, and one control port. Typically, these boards have all you need for a small system—a serial interface for the CRT or teletypewriter terminal; two parallel interfaces, one for a printer and one for another device; and a cassette interface used to provide bulk storage of programs on standard cassette tapes (more about this in a later chapter).

Other Designs Provide Different Performance

The MIO board offered by Imsai approaches the multiple interface problem for the 8080 from another angle. It contains only one serial I/O port for a teletypewriter or CRT terminal but has two 8-bit parallel I/O ports, one cassette interface that will store or read data on a standard audio cassette, and one control port used for internal and external control. No software is included on the board and there is no RAM workspace. The board is jumper selectable to any one of 64 blocks of four locations of I/O addresses. Additional jumper selection allows each port to be set up in any order within the selected group of four addresses. Along the top of the board are three 26-pin connector finger regions that provide the serial and dual 8-bit ports SIO, PIO2, and PIO1, respectively, from left to right (see Fig. 6.21 on the next page). Just to the left of the PIO2 connector and to the right of the SIO are connections for two cassette tape recorders. Either an RS-232 or 20-mA current-loop is available on the serial port.

Any of the status signals from the I/O ports may be used to generate interrupts. Provisions are included to jumper the status signals to the interrupt lines such as on an optionally available priority interrupt controller board, if used, or the signals can be directly jumpered to the CPU interrupt line for a single level of interrupt.

The serial I/O port can be set so its baud rate is jumper selectable from 45.5 to 9600 baud, the character length can be set, the parity enable and even/odd parity can be jumper selected, the on-board UART can be set to transmit or receive RS-232, 20-mA current-loop, or TTL levels, and the UART status signals can be monitored using the control input port on the interrupt inputs. There are eight status signals that the port can deliver—TRANSMITTER READY, TRANSMITTER READY, RECEIVE READY, RECEIVE READY, PARITY ERROR, OVERRUN ERROR, FRAMING ERROR, and an error indicator SIOS that indicates when either a parity, overrun, or framing error (PE, OE, or FF) has occurred.

Each parallel port accepts or delivers an 8-bit data word and a strobe signal. The strobe is used to indicate to the computer or the external equipment that the data in the port I/O lines are valid. Not only can the ports be programmed to act as inputs or outputs, but the strobe signal can be programmed to indicate valid data in any of four ways:

1. on the rising edge of the strobe;

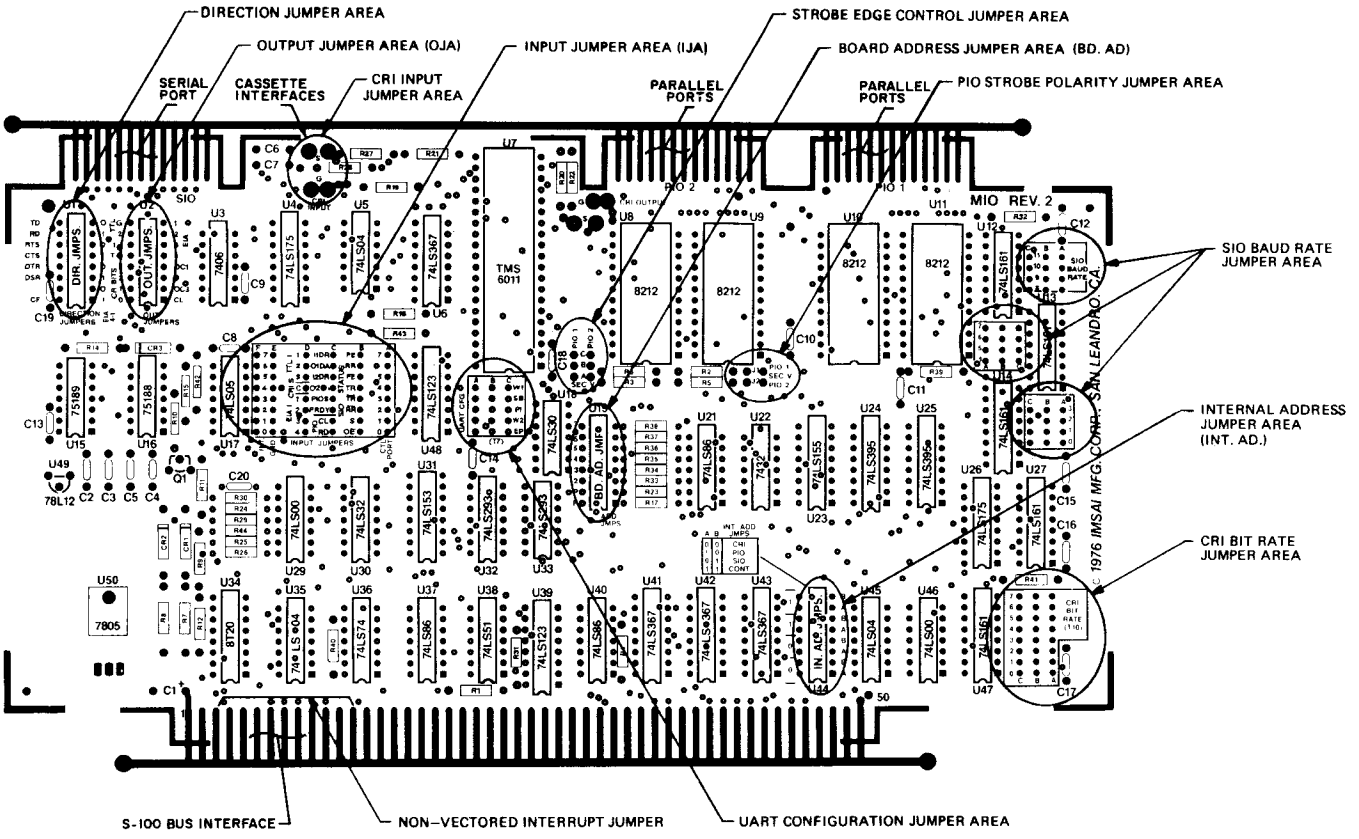


Fig. 6.21 Close-up view of the MIO board organization shows the various I/O interfaces along the top edge of the board. (*Courtesy* Imsai)

2. on the falling edge of the strobe;
3. on the logic HIGH level;
4. on the logic LOW level.

In addition to the strobe signal, there are three more control signals supplied with each port—an ODR (output data ready signal), an IDA (input data accepted) signal, and a PIOS (port I/O status) signal to indicate and control operation of the port. Through the use of four 8212 8-bit latches (one as an input and one as an output for each port) separate input and output lines are available, so the interface to the bus can be simplified. Each output port thus has eight data lines and three control lines. Each input port has eight data lines and two control lines. All signals are available via the 26-finger connector on the upper right of the board.

The cassette interface on the board permits you to record data from the computer's memory onto magnetic tape or load data into the memory using an ordinary tape recorder. Data rates for record or playback can be set from 500 to 62,500 bits per second. The phase of the signal can be jumper selected so that any recorder can be used. Data are stored on tape in a biphase encoded format that is compatible with both the *Byte*/Lancaster (often called the "Kansas City" format) or the Tarbell data formats (more about these in a later chapter). The biphase encoding generates the *Byte*/Lancaster data format by sending alternating ones and zeros when a zero bit is to be recorded and sends all ones when a one bit is to be recorded. In this format, the maximum data rate is 30 bytes per second. For the Tarbell format, one bit of phase encoding is used for each data bit. This format lets you handle data at 187 bytes per second or faster if the recorder used is a high-quality unit. On the board is room to connect two tape recorders, but only one recorder can be used at a time. Full information about cassette interfaces will be discussed in a later chapter.

CHAPTER 7

Peripheral Storage Devices for Microcomputer Systems

Using the serial or parallel interfaces described in the previous chapter permits information to be entered into or fed out of the computer by a combination of software and hardware to control the flow of data. However, the computer system still has two limitations. First, there is no way to permanently store large programs or data before turning off the system for the day. Since most programs or data are held in RAM, when power is shut off, RAMs lose whatever information they hold. And, each time the system is turned on, the previous programs, data, or new programs should be easily loadable into the computer. Second, there is no permanent record (hard copy) of the data so that data or programs can be examined without always using the system.

The first limitation can be overcome in one of three possible alternatives:

1. Use a paper-tape reader/punch for permanent data storage and retrieval. It provides a nonalterable medium once the paper is punched (Fig. 7.1a).

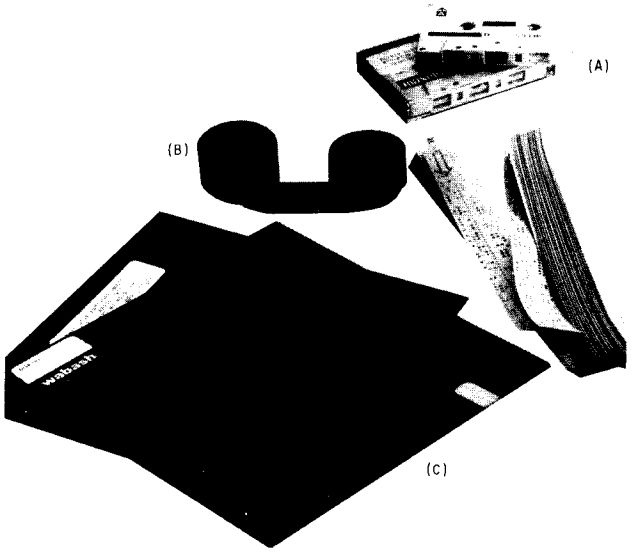


Fig. 7.1 Magnetic tape (a), paper tape (b), and floppy disks (c) are three popular mediums used to store and read in large amounts of data. (Photo by R. Meehan)

2. Use magnetic-tape storage for permanent but alterable storage of data. Depending on the amount of storage, tape drives for either cassettes, cartridges, or open rolls of tape can be used (Fig. 7.1b).

3. Use rotating magnetic disks to store data either on a permanent or temporary basis. These disks, similar to 45 RPM records, are referred to as floppy disks since they are flexible and "flop" from side to side when not in motion (Fig. 7.1c).

If your system is built with some form of printing terminal you already have the second limitation licked. However, if you decided on a low-cost CRT terminal to use with the computer, you might still want some form of hard-copy output. To provide the hard copy, you have a choice of many different types of printers that come in a wide range of capabilities and prices. We'll talk more about printers later in the chapter.

Storing Your Programs—Which Way to Go?

Picking the right type of storage media—paper tape, magnetic tape, or floppy disk—for your system can be a problem unless you have a fairly good idea what sort of use you're planning. Paper tape is best only when you want to load in a program or permanently store information or programs for later use. Cassettes offer more flexibility, permitting you to load, store, and alter data. However, both paper tapes and cassettes can require fairly long amounts of time to load in or store (dump) large amounts of information—typical times range from a few seconds to several minutes. Floppy disks offer the advantages of cassettes but with faster speeds and more of a random-access capability. Any place on the disk can be reached in less than a second while in a high-quality tape drive it can take several seconds to locate the information to be loaded in.

Ideally, then, the floppy disk and its drive offer the best performance in terms of speed, flexibility, and storage capacity. However, there is a catch—floppy disk drives are still very expensive ($400 and up), especially if you include the cost of the complex circuitry needed to control the drive and prepare the data for

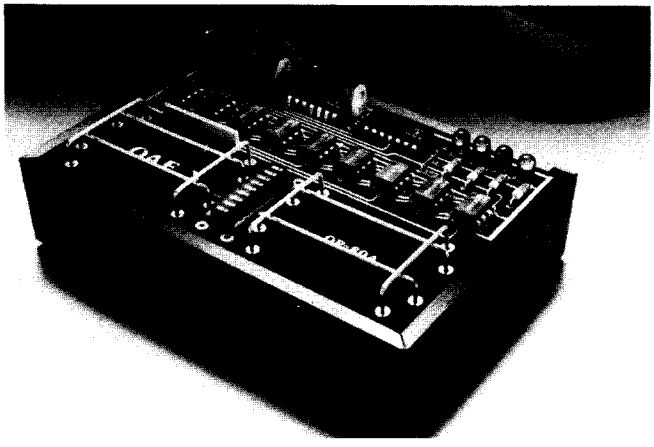


Fig. 7.2 An inexpensive paper-tape reader that requires manual tape movement, this unit, developed by Oliver Audio Engineering, interfaces very simply to any computer system. (*Courtesy* Oliver Audio)

storage or loading. The combination paper-tape reader/punch is probably next on the list of the more expensive program entry/storage devices ($300 and up). And, in addition to the reader/punch you will need either a parallel or serial interface for the computer, depending on the type of I/O required by reader/punch. The audio cassette is the lower cost of digital storage available. By using an audio tape recorder, that costs anywhere from $40 on up, and an interface circuit such as the Kansas City Standard or Tarbell circuits, a complete bidirectional interface can be built for under $200. However it does lack many of the desirable features found in disk systems—no random access capability, hard to keep track of data, hard to relocate data on tape, and hard to rapidly store and retrieve data.

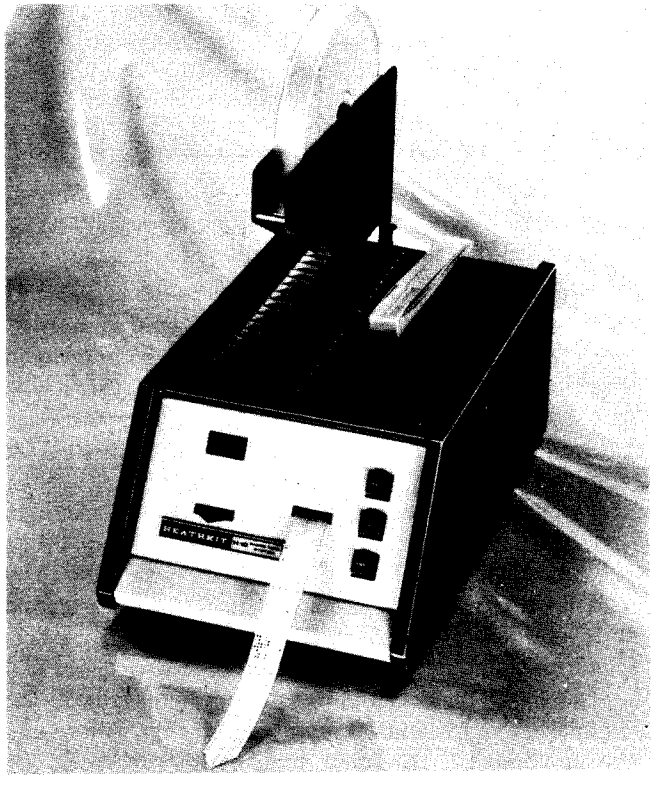


Fig. 7.3 The H-10 paper-tape reader/punch provides computer controllable operation at minimal cost. (*Courtesy* Heath Co.)

There are many low cost ways to enter data if that's all you want to do—inexpensive paper-tape readers are available from several companies for less than $100. One of the most popular units is the Oliver Audio Engineering OP-80A (Fig. 7.2). It is a manually operated paper-tape reader (you must pull the tape through by hand) and is useful only to load programs in the computer through a parallel I/O port. The concept of the paper-tape reader is very simple: light shining in through a series of nine possible holes in the tape hits an equal array of electrically matched phototransistors that, in turn, trigger a series of nine electrically matched timing circuits that provide the 8-bit digital code and the strobe signal to let the port know valid data are present.

However, loading paper tapes into the computer by hand is a tedious process—especially if the tape is long. If the programs to be loaded are less than several thousand bytes, the manual reader is a handy device. But when programs grow beyond that, you'll end up pulling hundreds of feet of paper tape through the reader by hand. To solve that problem some companies have come up with a small battery operated tape winder that can pull the tape through fairly fast and at a constant speed. An even better investment though is a complete paper-tape reader/punch that is electromechanically controlled.

Heathkit has recently introduced a complete kit at a cost of $350 (catalog) that contains an electromechanical paper-tape reader and paper-tape punch with a parallel output—the H-10 (Fig. 7.3). Even at that price it is the lowest cost unit that can punch as well as read paper tapes. There are also many surplus bargains available for the sharp shopper. However, you'll probably have to spend a minimum of 10 to 20 hours just to figure out the appropriate interface necessary to mate the unit to your system. Or if you had the money to invest when you decided on a terminal, for $800 and up you could purchase an ASR-33 teletypewriter to get hard-copy and paper-tape capability.

The major specifications for the H10 include a read speed of 50 characters per second and a punch speed of 10 characters per second. The unit can use standard 1-in.-wide roll or fan-fold eight-level paper tape. Both the punch and reader circuits are completely independent and may thus be operated simultaneously. Features of the H10 include a copy mode for tape duplication, a built-in heavy-duty power supply, and a stepper motor for reliable reader tape drive. The interface to the H10 requires TTL levels and consists of eight data bits, a strobe signal, a control signal for the reader, and another similar set of lines for the punch.

There are three control signals available for the H10 punch—a PUNCH START line, a PUNCH READY line, and a PUNCH READY line. The first line is used to initiate a punch cycle; when it goes LOW and is held LOW for at least 200 ns, the punch cycle will start. If the line is constantly held LOW, the punch will run at maximum speed. The other two punch control lines are complements of each other so only one of the two lines is really needed, depending on your system. On the PUNCH READY line, a HIGH state indicates the punch is ready. A TTL LOW level is present within 200 ns after the leading edge of the PUNCH START input signal and remains LOW until the punch is ready for the next character. The PUNCH READY line must be LOW to indicate the punch is ready. A TTL HIGH level is present within 200 ns after the leading edge of the PUNCH START input signal and stays HIGH until the punch is ready for the next character.

Only two reader control lines are available—a READER START and READER READY line. The READER START line advances the paper tape in the reader one character each time a HIGH-to-LOW transition occurs. A LOW must be held for at least 100 ns. The READER READY line indicates that output data bits are valid. This signal goes HIGH within 200 ns after the leading edge of the READER START signal and remains HIGH for approximately 16.5 ms. The READER START input can only be pulsed when the READER READY line is LOW. Signal lines are basically TTL. The PUNCH READY, PUNCH READY, and READER READY output lines require a LOW to be less than 0.4 V, and they can sink 16 mA each. The outputs are open collector with 1000 Ω pull-up resistors to 4.4 V. The PUNCH START and READER START inputs require a LOW to be less than 0.8 V and can sink 1.6 mA each. HIGHs must be greater than 2 V but less than or equal to 5.5 V. Inputs must not be negative with respect to circuit ground (common).

On the H10 connector there is only one other line—a common ground line so that the reader/punch ground can be connected to the ground in the rest of the system. The front panel of the H10 has four control switches—POWER ON/OFF, READ, FEED, and PUNCH. The POWER switch turns on both the reader and punch and puts all circuitry in a standby mode. When the READ switch is pressed the reader circuitry is enabled and will operate when a READER START signal is received. When not used, the READER READY line at the output connector is HIGH. The FEED switch just advances the tape in the reader by activating only the tape drive mechanism. Lastly, the PUNCH switch, when pressed enables the punch. Data can then be accepted from either the reader circuits or the data input lines, and a tape punched. Another switch mounted on the rear of the H10 determines whether the H10 is copying a tape or accepting data from the input lines.

However, no matter which punch or reader you have, the largest problem with its use is that data cannot be altered once the tape is punched. To correct the error a new tape must be punched—a time-consuming and wasteful process. Magnetic tape offers an easier alternative—and at a lower cost.

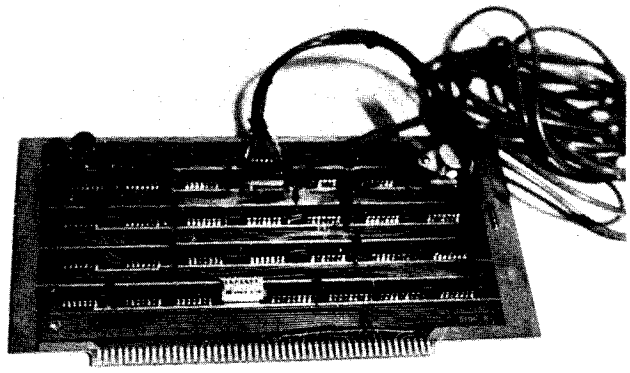


Fig. 7.4 The Tarbell cassette interface, shown here, permits an ordinary cassette tape recorder/player to be used as a computer's memory for permanent record storage. (Photo by J. Bierman)

Storing data on magnetic tape can be done with many different recording techniques and with many methods to code the data bits before they are recorded. The simplest method to store data on tape uses the common audio cassette, a cassette recorder, and an interface that transforms each parallel data word into a serial pulse train which is then transferred into audio frequency variations and finally recorded on tape. To retrieve data already recorded on tape, the audio signal is first transformed back into digital form and then decoded and put back into parallel form. There are currently two, almost "standard," methods in use today by the hobbiest community. They are the Tarbell and the Kansas City Standard cassette interfaces.

Boost Data Entry and Storage Speed with Cassettes

The Tarbell cassette interface (Fig. 7.4) provides a data entry or record speed of up to 540 bytes per second when a high-quality tape recorder is used (over $70 typically). With a simpler recorder such as the J. C. Penney model 6536 (about $40) data rates of 187 bytes per second are possible (three to 20 times the speed of paper tape). And, if you make the necessary modifications, the circuit can also work with data recorded according to the Kansas City Standard at 27 bytes per second. When a higher quality tape drive is used, such as the Phi Deck made by Triple I Corp. (Fig. 7.5), record and playback speeds of up to 1000 bytes per second are possible. The suggested tape for use is the Scotch Low-Noise, High-Density audio tape although any good quality tape can be used. Ideally, the recorder should have

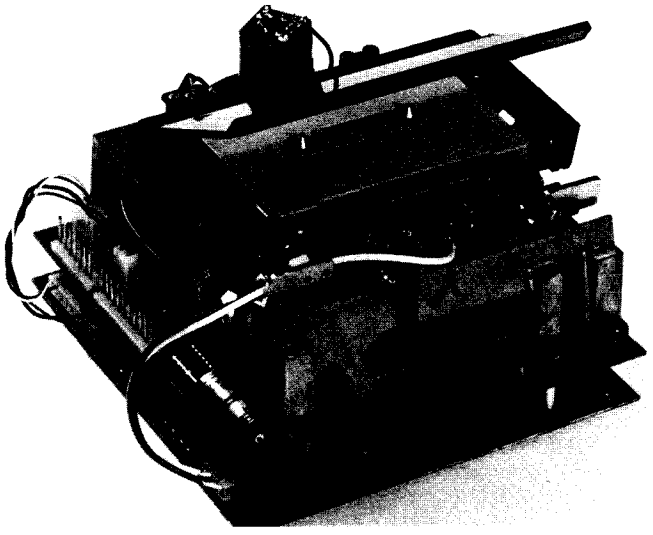


Fig. 7.5 For high data transfer rates (higher than possible with ordinary tape recorders), digital tape decks such as this unit can transfer data at over 1000 bytes/second. (*Courtesy* Triple I Corp.)

a good high-frequency response up to 8000 Hz, a tone control, a digital counter, and operate from the ac line.

Basically, the cassette interface works as follows: when data are to be recorded on the tape (Fig. 7.6a), the parallel data are first loaded from the computer's data bus into a parallel-input shift register. Then the register is clocked and its serial input is Exclusive-ORed with the clock, thus producing a biphase data stream. These data go directly into the cassette recorder's microphone input. The more difficult process is recovering the data. To recover the digital data from the biphase recording, a comparator, a one shot, and an exclusive-OR gate combined in a single chip (an 8T20 made by Signetics) can be used to first transform the played-back signal into digital levels, then restore the correct pulse widths, and then regenerate the proper number of pulses (Fig. 7.6b).

The data transfer speed may not seem too important, but if you use a cassette interface, a good portion of your time will be spent loading and storing data and programs. There is a world of difference between loading a BASIC operating system at the 30 bytes per second (about 4 minutes for 8 kbytes) and at 187 bytes per second (40 seconds), especially if you have to repeat the process several times. Jacks on the tape recorder for a remote control input and a direct line input are invaluable to help simplify the overall interface.

Before looking at exactly how the Tarbell interface works let's look at the Tarbell and Kansas City "standards" used to record data on the tape. The Tarbell interface uses a base frequency of 3000 Hz (when set for a 187 byte per second rate). A ONE is generated by writing a word of all ZEROs (00000000), and a ZERO is generated by writing a word of alternating ONEs and ZEROs (01010101). The Kansas City interface (often referred to as the *Byte*/Lancaster format) writes each 8-bit byte onto the tape with one start bit (a ZERO), eight data bits (ZEROs or ONEs) and two stop bits (ONEs). A ONE is defined as eight cycles at a frequency of 2400 Hz and a ZERO is defined as four cycles at a

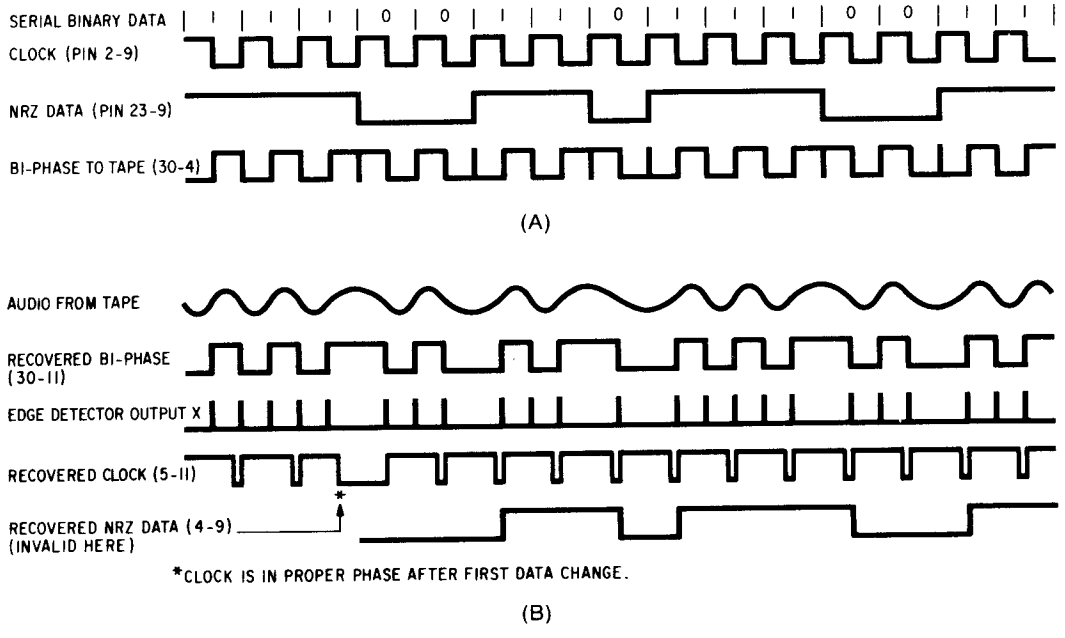


Fig. 7.6 Recording digital data onto magnetic tape requires that the digital information be changed into a frequency shift signal (a). To recover the digital information from the cassette tape, the frequency shift signal must be restored to digital form (b). (*Courtesy* Tarbell Electronics)

```
                CASSETTE INTERFACE OUTPUT ROUTINE

                THIS PROGRAM WRITES A BLOCK OF MEMORY OUT ONTO CASSETTE TAPE. THE PRO-
                GRAM IS ASSEMBLED TO START AT 3100 (HEX), BUT MAY BE REASSEMBLED TO START
                ANYWHERE. THE BLOCK STARTING ADDRESS IS LOCATED AT ADDRESS 3104 (HEX).
                THE BLOCK LENGTH (2 BYTES) IS LOCATED AT ADDRESS 3107 (HEX). THE PROGRAM
                WILL WRITE A "W" ON THE COMMENT DEVICE WHEN IT IS THROUGH WITH ITS DATA
                TRANSFER.

3100  31 43 31        LXI   SP,STAK   SET STACK POINTER.
3103  21 00 00        LXI   H,0       GET BLOCK ADDRESS.
3106  01 00 20        LXI   B,2000H   SET BLOCK LENGTH = 8192.
3109  1E 00           MVI   E,0       SET E = 0.
310B  3E 3C           MVI   A,3CH     GET START BYTE
310D  CD 32 31        CALL  COUT      OUTPUT START BYTE TO CASSETTE.
3110  3E E6           MVI   A,0E6H    GET SYNC BYTE.
3112  CD 32 31        CALL  COUT      OUTPUT SYNC BYTE TO CASSETTE.
3115  7E        LOOP  MOV   A,M       GET A DATA BYTE FROM MEMORY.
3116  CD 32 31        CALL  COUT      OUTPUT DATA BYTE TO CASSETTE.
3119  83              ADD   E         ADD E (CHECKSUM) TO A.
311A  5F              MOV   E,A       PUT NEW CHECKSUM INTO E.
311B  23              INX   H         INCREMENT MEMORY POINTER.
311C  OB              DCX   B         DECREMENT COUNTER.
311D  3E 00           MVI   A,0       MAKE A = 0.
311F  B8              CMP   B         IF B NOT = 0,
3120  C2 15 31        JNZ   LOOP        REPEAT LOOP.
3123  B9              CMP   C         IF C NOT = 0.
3124  C2 15 31        JNZ   LOOP        REPEAT LOOP.
3127  7B              MOV   A,E       OTHERWISE, GET CHECKSUM
3128  CD 32 31        CALL  COUT      AND OUTPUT IT.
312B  3E 57           MVI   A,"W"     WRITE "W" (END OF WRITE).
312D  D3 01           OUT   1         PRINT ON CONSOLE.
312F  C3 2F 31  WAIT  JMP   WAIT      WAIT HERE WHEN DONE.
3132  F5        COUT  PUSH  PSW       SAVE A AND FLAGS.
3133  DB 6E     CLOP  IN    CASC      READ CASSETTE STATUS.
3135  E6 20           ANI   20H       CLEAR ALL BUT BIT 5.
3137  C2 33 31        JNZ   CLOP      TRY AGAIN IF NOT READY.
313A  F1              POP   PSW       RESTORE A AND FLAGS.
313B  D3 6F           OUT   CASD      OUTPUT DATA TO CASSETTE.
313D  C9              RET             RETURN FROM COUT.
313E  00              0
313F  00              0
3140  00              0
3141  00              0
3142  00              0
3143  00        STAK  0
                CASD  EQU   6FH
```

Fig. 7.7 Control program necessary to output data to a tape recorder using the Tarbell interface.

frequency of 1200 Hz. This provides a data transfer rate of 300 baud, or a little less than 30 bytes per second.

The Tarbell interface can be converted to a Kansas City interface by first raising the 3000 Hz frequency to 4800 Hz and, then on the input side, the sync detector circuit must be modified to recognize the alternating bit pattern as a sync byte in addition to the normal sync byte of E6 (hex).

Using the Cassette Interface

The Tarbell interface is designed to plug right into an Altair or Imsai type motherboard and connect to a tape recorder or deck via two shielded cables that plug into the DIP socket on the Tarbell board. Before plugging the board into the main bus, the seven switches of the DIP switch must be set—switch 1 off, 2 off, 3 on, 4 off, 5 off, 6 on, and 7 off (input phase inversion). Switches 1 to 6 correspond to address bits 2 to 7 and off is a ONE, on is a ZERO. Address bit 1 can be either ONE or ZERO since it isn't used by the interface, and address bit 0 is ZERO for status/control and ONE for data. Therefore, the switch settings correspond to device address 011011XX, where X indicates the bit can be either 1 or 0. This is the device select code used in all software supplied by Tarbell.

After the switches are set, the board can be plugged into the bus and the cables connected between the DIP

```
                CASSETTE INTERFACE INPUT ROUTINE

                THIS PROGRAM READS A BLOCK OF BYTES FROM CASSETTE INTO MEMORY. THE PRO-
                GRAM IS ASSEMBLED TO START AT 3100 (HEX), BUT MAY BE REASSEMBLED TO START
                ANYWHERE, ALTHOUGH CARE SHOULD BE TAKEN TO INSURE THAT THE DATA IT IS
                READING DOES NOT WRITE OVER THE PROGRAM ITSELF. THIS MAY BE ACCOMPLISHED
                BY LOCATING THE PROGRAM IMMEDIATELY BELOW OR A BLOCK LENGTH ABOVE THE
                DATA TO BE READ IN. THE STARTING ADDRESS FOR THE BLOCK IS LOCATED IN AD-
                DRESS 3185 (HEX). THE BLOCK LENGTH IS LOCATED IN ADDRESS 3188 (HEX) (TWO
                BYTES).

3180  3E 10        MVI   A,10H     SET BIT 4 OF A = 1.
3182  D3 6E        OUT   CASC      RESET INTERFACE.
3184  21 00 00     LXI   H,0       GET STARTING ADDRESS.
3187  11 00 20     LXI   D,2000H   GET BLOCK LENGTH.
318A  06 00        MVI   B,0       SET CHECKSUM = 0.
318C  DB 6E  LOOP  IN    CASC      READ CASSETTE STATUS.
318E  E6 10        ANI   10H       LOOK AT BIT 4.
3190  C2 8C 31     JNZ   LOOP      WAIT IF NOT READY.
3193  DB 6F        IN    CASD      READ DATA FROM CASSETTE.
3195  77           MOV   M,A       PUT DATA INTO MEMORY
3196  80           ADD   B         ADD CHECKSUM TO A.
3197  47           MOV   B,A       PUT IT BACK IN B.
3198  23           INX   H         INCREMENT MEMORY POINTER.
3199  1B           DCX   D         DECREMENT COUNTER.
319A  3E 00        MVI   A,0       CLEAR A.
319C  BA           CMP   D         IF D NOT = 0,
319D  C2 8C 31     JNZ   LOOP        READ MORE.
31A0  BB           CMP   E         IF E NOT = 0,
31A1  C2 8C 31     JNZ   LOOP        READ MORE.
31A4  DB 6E  CHEK  IN    CASC      READ STATUS.
31A6  E6 10        ANI   10H       LOOK AT BIT 4.
31A8  C2 A4 31     JNZ   CHEK      WAIT IF NOT READY.
31AB  DB 6F        IN    CASD      READ CHECKSUM.
31AD  B8           CMP   B         COMPARE TO B.
31AE  3E 45        MVI   A,"E"     PUT CODE FOR "E" IN A.
31B0  C2 B5 31     JNZ   ERR       IF CHECKSUMS NOT EQUAL, ERROR.
31B3  C6 02        ADI   2         ADD A 2 TO MAKE "G" IF EQUAL.
31B5  D3 01  ERR   OUT   CRTD      PRINT "E" FOR "G."
31B7  C3 B7 31 END   JMP   END     WAIT HERE WHEN DONE.
                CASC  EQU   6EH    CASSETTE STATUS/CONTROL PORT.
                CASD  EQU   6FH    CASSETTE DATA PORT.
                CRTD  EQU   01H    CONSOLE DATA PORT.
```

Fig. 7.8 Control program necessary to input data to the computer from a tape recorder using the Tarbell interface.

socket and the tape recorder. The Tarbell kit comes with a test tape which should be inserted into the recorder. If the recorder has a tone control make sure it is turned to maximum (to get best frequency response) and then turn the recorder's volume control to a middle setting. Next, set the potentiometer of the board's input section (R8) to a middle setting, turn on the computer, hit the RESET switch and then press the PLAY button on the cassette recorder. If the LED on the interface does not light up after a few seconds, try adjusting the recorder's volume control and R8 until the light comes on.

However, if the LED still doesn't come on, try reversing switch number 7 (put it on) on the DIP switch and repeat the potentiometer adjustments. Now if the LED doesn't come on the problem is either the tape recorder or the receive section of the interface. Assuming the LED does come on, it tells you that the receiver is operating properly and is detecting the continuous stream of sync bytes recorded on the test tape. Now, adjust both the interface pot and the volume control so there is some leeway on either side of center where the LED will remain on. During normal operation the LED will flicker since it is indicating sync bytes, not data.

Next, the output section must be tested. This is done by programming the computer to generate a continuous stream of E6 bytes, the same as recorded on the tape. First, put a blank cassette tape into the recorder and then load the following program into the computer via the front-panel switches or your terminal. (For background information on software, see Chapter 8.)

```
0000  DB  6E      Loop  IN    CASC   Read status
0002  E6  20            ANI   20H    Look at bit 5
0004  C2  00 00       JNZ   LOOP   Wait until ready
0007  3E  E6            MVI   A,0E6H Get sync byte
0009  D3  6F            OUT   CASD   Write it onto cassette
000B  C3  00,00         JMP   LOOP   Repeat
                CASC  EQU   6EH    Status port
                CASD  EQU   6FH    Data port
                      END
```

After the program is loaded, put all address switches to address 0000, hit the RESET switch and then the RUN switch. Next, turn on the tape recorder so it records data onto the blank tape. If the light doesn't come on and stay on, the recording level may be too high or too low. Try several levels until you find the best place. You may also find the recording is opposite in phase to the playback; if so, change the jumper on IC23 from pin 9 to pin 8.

Assuming both the input and output sections check out, the interface is now ready to be used. To perform a save operation from the computer to the tape recorder first requires that the computer have in its memory a routine that tells it how to store the data. Such a routine is shown in Fig. 7.7 (p. 75). This program writes information stored in a block of memory onto the cassette. It is designed to start at location 3100 (hex) but may be rearranged to start anywhere. The block starting address is located at address 3104 and the length at address 3107. After the transfer is completed the program writes a W onto the terminal to indicate the transfer is complete.

Once the program is in the computer, set the recorder's volume control, set the tape to the desired data storage location, and start the recorder. If the volume control affects the record level, slowly increase the volume until the correct recording level is shown. Wait about five seconds to record a leader on the tape and then push the button that starts the output routine on the computer (this could be the carriage return after CSAVE in BASIC or the front-panel RUN button for stand-alone programs). When the program indicates transfer is complete, stop the cassette recorder.

Similarly, to load data from a cassette you also must first tell the computer how to handle the incoming data. Such a routine is shown in Fig. 7.8 (p. 76). This program reads a block of bytes from a cassette into memory. It is also intended to start at 3100 (hex), but may be rearranged to start anywhere, although care should be taken to ensure that the data being loaded don't write over the program itself. The starting address for the block is located at address 3185 and the block length is located at addresses 3188 and 3189.

After the computer has this program in its memory the tape that has the desired program should be loaded into the recorder, the volume control should be set for the desired level, and the computer should be set so all it takes is the push of one button. Start the cassette recorder in the playback mode and press the button on the computer that starts the input routine (in BASIC this could be the carriage return after CLOAD or the front-panel RUN button for stand-alone programs). When the program indicates it has completed the load, stop the cassette recorder. Remember, the memory into which data are read must be unprotected.

For simple program load operations, another, much shorter program can be used. The cassette bootstrap pro-

CASSETTE BOOTSTRAP PROGRAM

```
2F00  3E 10          MVI   A,10H    SET BIT 4 of A = 1.
2F02  D3 6E          OUT   CASC     RESET INTERFACE.
2F04  21 00 00       LXI   H,0      PUT STARTING ADDRESS IN H,L.
2F07  DB 6E    LOOP  IN    CASC     READ STATUS.
2F09  E6 10          ANI   10H      CLEAR ALL BUT BIT 4.
2F0B  C2 07 2F       JNZ   LOOP     WAIT IN LOOP UNTIL READY.
2F0E  DB 6F          IN    CASD     READ A DATA BYTE.
2F10  FB             EI             SIGNAL OPERATOR.
2F11  77             MOV   M,A      PUT DATA INTO MEMORY.
2F12  23             INX   H        INCREMENT MEMORY POINTER.
2F12  C3 07 2F       JMP   LOOP     REPEAT THE ABOVE OPERATION.
                CASC  EQU   6EH     CASSETTE STATUS PORT.
                CASD  EQU   6FH     CASSETTE DATA PORT.
                      END
```

NOTE: IF YOU HAVE AN IMSAI OR ALTAIR WITH AN OUTPUT PORT ON THE FRONT PANEL (8 LED'S), YOU CAN USE THE BOOTSTRAP PROGRAM FOR TROUBLE-SHOOTING THE INPUT SECTION WITH THE FOLLOWING MODIFICATION:

AT	INSTEAD OF	SUBSTITUTE
2F10	EI (FB)	CMA (2F)
2F11	MOV M,A (77)	OUT (D3)
2F12	INX H (23)	LEDS (FF)

Fig. 7.9 For minimal tape loading efforts, this simple bootstrap routine permits data to be loaded into the computer via the Tarbell interface.

gram shown in Fig. 7.9 (p. 77) loads data starting at address 0000 and keeps on going. There is no count of bytes and no error checks. It is assembled to run at address 2F00 (hex) but can be reassembled to run anywhere provided that it does not load data over itself. Use the seconds indicator on your watch to determine how long to wait until stopping the program. Allow about 45 seconds to load an 8-kbyte block of data.

Sometimes, programs must be written for a particular interface. These programs are often referred to as drivers and are usually attached or linked to other programs, such as the cassette input or output routines. The Tarbell interface is a synchronous device, which means that data or programs are most efficiently written as a contiguous block, rather than as separate bytes. A few basic recommendations for writing software include:

1. The first byte must be a start byte—any byte except 00, FF, or E6.
2. The second byte must be a sync byte—E6.
3. The output software must be able to deliver bytes to the interface as fast as the interface can accept them, which is 187 bytes per second at the standard speed. This means that any loop that the program goes through (between bytes) must last no more than 5.3 ms. (On the 8080 this means about 5300 instruction cycles between bytes of data—normally not a problem.)
4. The input software must also have a similar constraint—incoming data should be accepted as fast as it is being readied by the interface.

There are times, though, when data cannot be provided or accepted fast enough by the software. One example of this is generated data from a program running in BASIC. To solve the speed problem, data are read in or sent out a line at a time, with nulls between. However, there are other methods to solve the problem—you can send each byte as a separate block with its start and sync bytes or you can have the bytes accumulate in a buffer area of the computer memory and start and stop the cassette recorder under computer control when it's time to empty the buffer.

Some other things to consider when you write programs and store them on cassettes include:

1. Since tape is an imperfect recording medium it is best to include an error-checking scheme such as checksums to indicate if the tape loaded properly. However, there are hundreds of error-checking routines and each provides slightly different protection against a bad bit or two.
2. Consider including an identifier, such as a name, along with the file on the cassette. This makes accessing the file much simpler in some cases.
3. Try to include the length of the file at the beginning of the tape. A one- or two-byte header to indicate file or block size is sometimes used.
4. Another code in the header could classify the type of software on the tape, the particular format, or the speed of recording. One byte should be sufficient.

Control the Recorder with the Computer

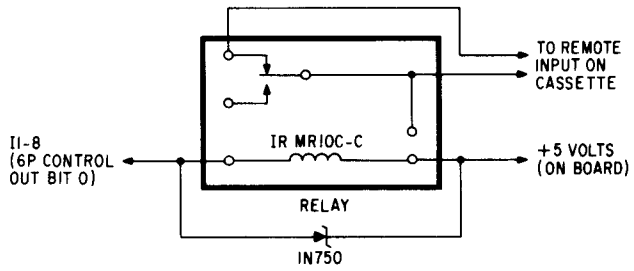
So far, the tape recorder has to be turned on and off manually every time you want to read or write data. By using the microphone switch available on most small recorders you can start or stop the recorder by first building an interface to the computer that can handle the current of the tape recorder's motor. The ability to control the recorder is important if the amount of data on tape is too much for the computer's memory or if data generated by the computer only comes in blocks with large time separations between the blocks. And, in some cases, more than one tape recorder can be controlled, thus permitting you to read from one and write to another in order to modify, assemble, and store programs all at the same time.

The simple circuit shown in Fig. 7.10a along with the software assembly listings shown in Fig. 7.10b can control the recorder. In the START routine there is a DELAY subroutine requested (but not shown) since there must be a delay of about one to two seconds for the tape to get up to speed from a dead stop before data can be read or recorded. The delay depends on the type of recorder used and should be longer before a write operation than before a read. A module that allows the control of up to four cassette recorders with the Tarbell interface is available from RO-CHE Systems, Van Nuys, California.

The ordinary cassette recorder is still limited by the mechanical controls for record and playback. To get more computer control, the Phi Decks, made by Triple I Corp., for example, can eliminate almost all manual intervention with the use of a control board that connects to the Tarbell interface and the computer's bus. The basic tape transport without any support electronics has three motors, a read/write head, a beginning/end of tape sensor, and an engage/disengage control. There are five basic transports available:

1. Model 1: Fixed-speed unidirectional-play deck with play speeds of 1 to 6 ips (with pulley change). Operation is from a 12 V dc, 900 mA supply.
2. Model 2: Variable-speed, bidirectional play with tape speeds of 0.5 to 10 ips (voltage controllable). Operation is from 12 V dc, 900 mA supply.
3. Model 3: Variable-speed bidirectional play with tape speeds of 0.4 to 20 ips (voltage controllable). Start and stop speeds are less than 100 ms. Operation requires five supplies: +18 dc at 25 mA, −18 V at 120 mA, 11 V at 100 mA, +7 dc at 500 mA, and 5 V at 400 mA.
4. Model 4: Fixed-speed, unidirectional-play ac motor driven unit with speed regulation of 0.2% and tape speeds of $^{15}/_{16}$, $1^{7}/_{8}$, $3^{3}/_{4}$, 5, 6, 7, and 8 ips.
5. Model 5: Fixed-speed, unidirectional-play ac motor driven unit with speed regulation of 0.2% and tape speeds of 7.5, 10, 12, 14, and 16 ips.

Any of these decks would make a nice addition to a computer system but the Model 3 offers the most flex-



(A)

```
START   LDA   CTLS     GET CONTROL STATUS BYTE.
        ORI   01       SET BIT 0 = ONE.
        STA   CTLS     UPDATE CONTROL STATUS BYTE.
        OUT   CASC       START THE TAPE.
        CALL  DELAY      WAIT FOR TAPE TO GET UP TO SPEED.
        RET              RETURN (NEXT DO YOUR I/O).

STOP    LDA   CTLS     GET CONTROL STATUS BYTE.
        AND   0FEH     SET BIT 0 TO ZERO.
        STA   CTLS     UPDATE CONTROL STATUS BYTE.
        OUT   CASC     STOP THE TAPE.
        RET            RETURN FROM I/O ROUTINE.

CTLS    DB    0        CONTROL STATUS BYTE.
CASC    EQU   6EH      CASSETTE CONTROL PORT.
```

(B)

Fig. 7.10 This simple circuit permits the Tarbell interface to start and stop a cassette tape recorder (a). The software necessary to control the cassette recorder is a simple 13-line assembly code routine (b).

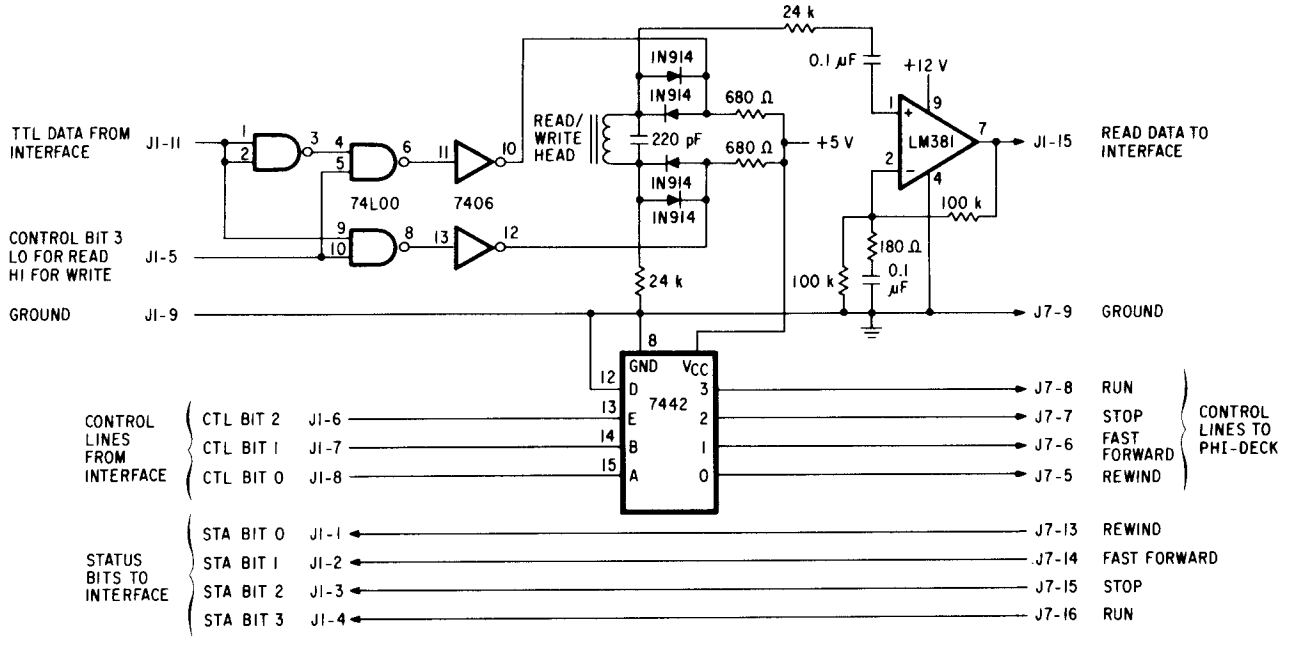
ibility and highest speed although the Model 2 runs a close second. Just the bare deck can be purchased for about $170 and $124, respectively. The motion control electronics package costs $119 more. And additional read/write electronics boards add another $120 to the package cost. One option that should be included is the solenoid head engage feature so the computer can control all operation except for tape insertion or removal.

Interfacing the Phi Deck to the Tarbell board is relatively simple, as the circuit in Fig. 7.11a shows. Basically, the read/write head is fed by several gates with the digital data to be recorded on tape. To recover the data the output of the head is fed into an amplifier, which in turn feeds the signal to the Tarbell board. To control the drive a one-of-ten decoder uses control bits from the Tarbell interface to provide the four basic deck control functions—RUN, STOP, FAST FORWARD, and REWIND. Four return lines from the motion board provide status information to the Tarbell board. The simple program shown in Fig. 7.11b can be used to get the deck working with the interface. To use the program, flip sense switch 2 on the computer's front panel up momentarily to pulse the control line. Sense switch 3 should be up for read and down for write. Data transfer rates of over 1000 bytes per second are possible.

Consider the Kansas City Standard

Even though the KC Standard tape interface is slow in comparison to the Tarbell interface, it is extremely popular and there are many available sources of prerecorded programs. Basically, the KC Standard uses frequency-shift modulation at a data rate of 300 baud. The technique allows for long- and short-term tape speed variations, bandwidth limitations, and tape misalignment. A recorded character consists of a space as a start bit, eight data bits, and two or more marks as stop bits. The interval between characters consists of an unspecified amount of time at the logic ONE frequency (2400 Hz). The eight data bits are organized least-significant bit first, most-significant bit last, and optionally followed by a parity bit for a total of eight bits. Data are organized in blocks of arbitrary length preceded by a minimum of five seconds of marks. To avoid errors due to splices and tape damage common at the beginning of the tape, the first recorded data block will occur no sooner than 30 seconds from the beginning of a clear leader.

Basically, the recording and data recovery methods used in the Tarbell interface are the same. The only differences are the frequencies used to represent ONEs and ZEROs and the format of data as the bytes are stored on the tape. Pertec offers an interface for its Altair computer system. The interface, known as the 88-ACR, uses a serial interface board as described in Chapter 8 and a "piggyback" circuit that transforms the digital data into signals that can be recorded on an ordinary tape recorder (Fig. 7.12). When no signal is present the output of the 88-ACR is a constant-amplitude 2.4



(A)

The simple program below is handy for experimenting. Flip sense switch 2 up momentarily to pulse the control line.

```
LOOP IN   FFH   DB FF   Read Sense Switches.
     OUT  6EH   D3 6E   Write to Control Port.
     IN   6EH   DB 6E   Read Status Lines.
     OUT  FFH   D3 FF   Write To Display Lights.
     JMP  LOOP  C3 00 00  Do it all over again.
```

Control Table

s1	s0	function
0	0	RUN
0	1	STOP
1	0	FF
1	1	REW

Sense switch 3 should be up for read, down for write.

NOTE: A PHI-DECK kit, which includes a board with control *and* read/write electronics, is available from MECA, 7344 Wamego Trail, Yucca Valley, Calif. 92284. MECA provides information about connecting their unit to the Tarbell Cassette Interface.

(B)

Fig. 7.11 The Tarbell interface can also be used to control a more complex tape unit such as the PHI-DECK from Triple I with the circuitry shown here (a). This simple program can provide minimal operating interface control for a digital tape deck (b).

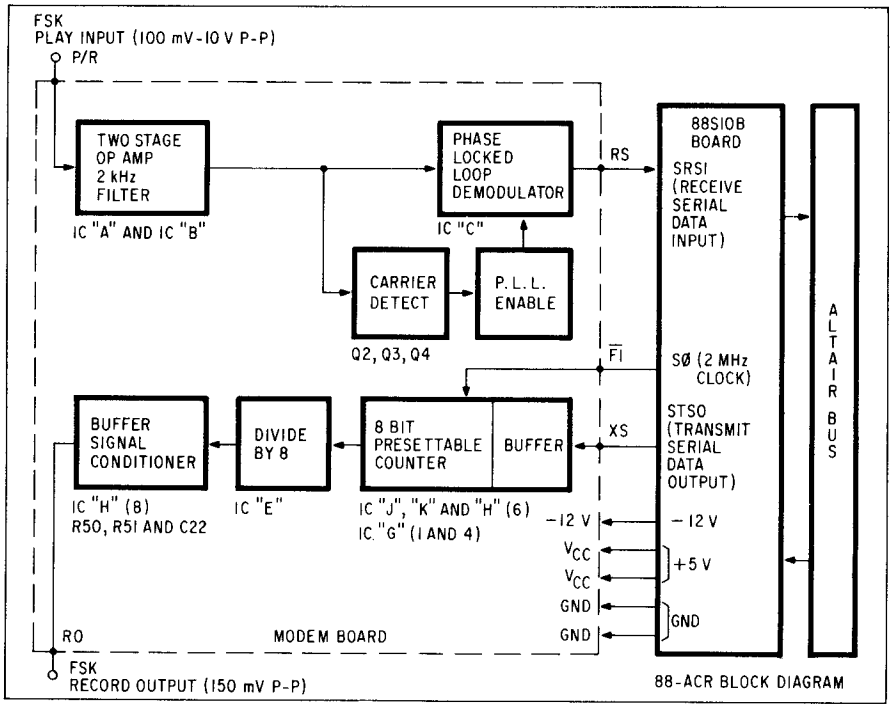
kHz sinewave (logic ONEs), and when the digital input begins changing, indicating ONEs and ZEROs, the output frequency drops to 1.85 kHz to indicate a ZERO.

The piggyback board used on the 88-ACR contains the modulator/demodulator circuits that convert digital data into tones and vice versa. When an output program is running, parallel data are fed into the 88-ACR and are transformed to serial data by the SIO portion of the 88-ACR. The serial digital data stream is then fed out of the main board to the "piggyback" board. The digital data stream is converted into square-wave tone signals that are next fed to the buffer signal conditioner where they are changed from TTL levels to a 100 mV peak-to-peak sawtooth wave suitable for the microphone input of the recorder. From the conditioner, the signal is fed to the output cable and to the tape recorder.

When a tape is being played to load data, the audio output of the recorder is fed via shielded cable into the 88-ACR's audio input (signal levels should be 35 mV to 3.5 V rms for proper operation). The audio signal is filtered and then fed to the demodulator and into the carrier detector circuit. The demodulator compares the frequency of the incoming signal with the frequency of the internal oscillator, which is set at the halfway mark. When the input frequency is over the halfway point the output is a logic 1 and when the input frequency is under the halfway point the output is a logic 0. The demodulator output is then adjusted to TTL levels, which are then fed from the piggyback board to the input of the main board.

The 88-ACR has several jumper areas that must be set up before any operations begin. As with the I/O boards described in earlier chapters, the board address must be set by strapping gate inputs. The eight lower-order address bus lines are fed to the select logic. Depending on the state of A0, either the control or data channel will be enabled (A0 LOW enables the control

(A)



(B)

Fig. 7.12 The 88-ACR tape interface developed by Pertec consists of two boards (a). One board performs the serial-to-parallel and parallel-to-serial conversion, and the other board performs the digital to frequency shift keying (FSK) to digital conversion (b). (*Courtesy* Pertec)

channel, A0 HIGH enables the data channel). Of the two device addresses on the board, the control channel is always an even number and the data channel is always an odd number.

Serving a dual purpose, the control channel enables/disables the hardware interrupt capability for the I/O device, and it tests the status I/O device. After an IN command is executed with the control channel address, SINP goes HIGH, which, in turn, enables the DATA IN lines.

The $\overline{\text{SWE}}$ (STATUS WORD ENABLE) line is always enabled except when inputting data; this results in the status being inputted to the DATA IN lines and into the CPU's accumulator. The eight data bits are defined in Table 7.1. When an OUT instruction is executed with the control channel address, data bits 0 and 1 are gated to the I/O interrupt flip-flops. (For example, to enable the input device and disable the output device interrupts, load the accumulator with xxxxxx01 and then execute an OUT instruction with the control channel address.) The data channel transfers the data between the device and the CPU. As soon as the CPU puts the data from the accumulator onto the data out bus, the PWR line goes LOW, causing the parallel data on the bus to be loaded and then transmitted serially. The UART, similar in function to the unit described in the serial interface in the previous chapter, does all the serial to parallel and parallel to serial conversion. Both the receive and transmit sections of the UART require a clock input that is 16 times the baud rate. Baud rates of 110, 150, 300, 600, 1200, 2400, 4800, 9600, and 19,200 can be set by jumping pins to 5 V or ground according to Table 7.2. Any other frequency can be determined by the following formula:

$$\text{Present frequency} = 4100 - \frac{(\text{period of output frequency in } \mu\text{s})}{0.5\ \mu\text{s}},$$

where the maximum frequency is 400 kHz and the maximum rate is 25,000 baud.

Table 7.1 Status Word Bit Definitions for the 88-ACR from Pertec

Data Bit	Logic Zero	Logic One
7	Output device ready (Emitter buffer empty). Also causes a hardware interrupt to occur if the interrupt is enabled.	Not ready
6	Not used	Not used
5	Not used	
4		Data overflow (a new word of data has been received before the previous word was inputed to the accumulator).
3		Framing error (data word has no valid stop bit)
2		Parity error (received parity does not agree with selected parity).
1	Not used	
0	Input device ready. Data are available for computer to input	Not ready

Table 7.2 Baud Rate Jumper Selection for the 88-ACR

Baud Rate	Preset Count											
	11	10	9	8	7	6	5	4	3	2	1	0
110	1	0	1	1	1	0	0	1	0	1	0	0
150	1	1	0	0	1	1	0	0	0	0	1	1
300	1	1	1	0	0	1	1	0	0	0	1	1
600	1	1	1	1	0	0	1	1	0	1	0	0
1200	1	1	1	1	1	0	0	1	1	1	0	0
2400	1	1	1	1	1	1	0	1	0	0	0	0
4800	1	1	1	1	1	1	1	0	1	0	1	0
9600	1	1	1	1	1	1	1	1	1	0	0	0
19200	1	1	1	1	1	1	1	1	1	1	1	0

Table 7.3 Set-up Chart for Stop, Parity, and Data Word Size on the 88-ACR

Stop bits:	NSB to GND = 1 stop bit
	NSB to +V = 2 stop bits
Parity:	NPB and POE to GND = odd parity
	NPB to GND and POE to +V = even parity
	NPB to +V and POE to either = no parity
Data bits/char:	NDB1 and NDB2 to GND = 5 bits/char
	NDB1 to +V and NDB2 to GND = 6 bits/char
	NDB1 to GND and NDB2 to +V = 7 bits/char
	NDB1 and NDB2 to +V = 8 bits/char

Also included in the board is a hardware interrupt capability that can be jumper selected on three pads, OUT, IN, and BH. These represent the output device, the input devices, or both devices, respectively. Any of the pads can be jumpered to the VI pads (numbered 0 to 7). The VI pads represent the vector interrupt lines and the pads 0 to 7 correspond to the eight priority levels, with 0 being the lowest and 7 the highest priority. These interrupt levels are intended to be used with the 88-VI interrupt controller board. If the board is not used in the system, the processor has an input interrupt line that can be controlled by the INT pad on the 88-ACR. When this pad is used, one level of interrupt to the processor is available, causing the CPU to immediately jump to location 70 octal (38 hex), and begin execution. The interrupt service routine can then be stored in locations 70 to 77 octal (38 to 3F hex).

Other jumper connections must be made to determine the number of stop bits, parity (even, odd, none), the number of bits per character (adjustable from 5 to 8), and just some interconnect that could not be done via the printed wiring on the board. To set the stop bits, parity, and data bits, jumpers must be inserted according to Table 7.3. For a typical system set-up the 88-ACR should be jumpered for address 006 (octal), the baud rate set for 300, and the UART set for eight data bits, one stop-bit, and no parity bit. (Jumpers POE, NDBL, NDBZ, and NPB should go to +V and NSB should go on GND.) To set the address, jumpers must be set according to Table 7.4.

To actually use the 88-ACR, the computer must have instructions in its memory that tell it how to transfer the data back and forth, from memory to tape or vice versa. Assuming the basic control programs are in the computer's memory and you want to store data on the tape, the RECORD OUT line of the 88-ACR must be connected to the microphone input of the recorder. With the computer in the stopped condition, but set to output the data, place the recorder in the RECORD mode. Allow the recorder to run for at least 10 seconds to provide a sufficient leader before hitting the computer's RUN switch. If you have a tape counter built into the recorder, start recording at 000, hit the RUN switch at 015, and carefully note the start and stop counts. To input data connect the earphone jack of the tape recorder to the PLAY-IN line of the 88-ACR. Again, the computer should be in the STOP mode but preset so only the RUN switch must be hit. Start the tape at the beginning and then hit the RUN switch when the tape counter indicates the program is loaded.

The test tape used to align the 88-ACR can be made using the 88-ACR in an output mode. Listed in Table 7.5, the program shown records a test byte (125) until the program is manually stopped. It is written using I/O address 006 for status and address 007 for data. To play back the test program once it is recorded on the tape, another program must be used (Table 7.6). However, to actually perform everyday I/O operations some short programs that can be used to output or load the main program are shown in Table 7.7.

Table 7.4 Address Set-up Selection for the 88-ACR

Address (Octal)	Connections							Address (Octal)	Connections						
	I7	I6	I5	I4	I3	I2	I1		I7	I6	I5	I4	I3	I2	I1
000	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$	200	A7	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$
002	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	A1	202	A7	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	A1
004	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	A2	$\overline{A1}$	204	A7	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	A2	$\overline{A1}$
006	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	A2	A1	206	A7	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	A2	A1
010	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	A3	$\overline{A2}$	$\overline{A1}$	210	A7	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	A3	$\overline{A2}$	$\overline{A1}$
012	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	A3	$\overline{A2}$	A1	212	A7	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	A3	$\overline{A2}$	A1
014	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	A3	A2	$\overline{A1}$	214	A7	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	A3	A2	$\overline{A1}$
016	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	A3	A2	A1	216	A7	$\overline{A6}$	$\overline{A5}$	$\overline{A4}$	A3	A2	A1
020	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	A4	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$	220	A7	$\overline{A6}$	$\overline{A5}$	A4	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$
022	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	A4	$\overline{A3}$	$\overline{A2}$	A1	222	A7	$\overline{A6}$	$\overline{A5}$	A4	$\overline{A3}$	$\overline{A2}$	A1
024	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	A4	$\overline{A3}$	A2	$\overline{A1}$	224	A7	$\overline{A6}$	$\overline{A5}$	A4	$\overline{A3}$	A2	$\overline{A1}$
026	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	A4	$\overline{A3}$	A2	A1	226	A7	$\overline{A6}$	$\overline{A5}$	A4	$\overline{A3}$	A2	A1
030	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	A4	A3	$\overline{A2}$	$\overline{A1}$	230	A7	$\overline{A6}$	$\overline{A5}$	A4	A3	$\overline{A2}$	$\overline{A1}$
032	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	A4	A3	$\overline{A2}$	A1	232	A7	$\overline{A6}$	$\overline{A5}$	A4	A3	$\overline{A2}$	A1
034	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	A4	A3	A2	$\overline{A1}$	234	A7	$\overline{A6}$	$\overline{A5}$	A4	A3	A2	$\overline{A1}$
036	$\overline{A7}$	$\overline{A6}$	$\overline{A5}$	A4	A3	A2	A1	236	A7	$\overline{A6}$	$\overline{A5}$	A4	A3	A2	A1
040	$\overline{A7}$	$\overline{A6}$	A5	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$	240	A7	$\overline{A6}$	A5	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$
042	$\overline{A7}$	$\overline{A6}$	A5	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	A1	242	A7	$\overline{A6}$	A5	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	A1
044	$\overline{A7}$	$\overline{A6}$	A5	$\overline{A4}$	$\overline{A3}$	A2	$\overline{A1}$	244	A7	$\overline{A6}$	A5	$\overline{A4}$	$\overline{A3}$	A2	$\overline{A1}$
046	$\overline{A7}$	$\overline{A6}$	A5	$\overline{A4}$	$\overline{A3}$	A2	A1	246	A7	$\overline{A6}$	A5	$\overline{A4}$	$\overline{A3}$	A2	A1
050	$\overline{A7}$	$\overline{A6}$	A5	$\overline{A4}$	A3	$\overline{A2}$	$\overline{A1}$	250	A7	$\overline{A6}$	A5	$\overline{A4}$	A3	$\overline{A2}$	$\overline{A1}$
052	$\overline{A7}$	$\overline{A6}$	A5	$\overline{A4}$	A3	$\overline{A2}$	A1	252	A7	$\overline{A6}$	A5	$\overline{A4}$	A3	$\overline{A2}$	A1
054	$\overline{A7}$	$\overline{A6}$	A5	$\overline{A4}$	A3	A2	$\overline{A1}$	254	A7	$\overline{A6}$	A5	$\overline{A4}$	A3	A2	$\overline{A1}$
056	$\overline{A7}$	$\overline{A6}$	A5	$\overline{A4}$	A3	A2	A1	256	A7	$\overline{A6}$	A5	$\overline{A4}$	A3	A2	A1
060	$\overline{A7}$	$\overline{A6}$	A5	A4	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$	260	A7	$\overline{A6}$	A5	A4	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$
062	$\overline{A7}$	$\overline{A6}$	A5	A4	$\overline{A3}$	$\overline{A2}$	A1	262	A7	$\overline{A6}$	A5	A4	$\overline{A3}$	$\overline{A2}$	A1
064	$\overline{A7}$	$\overline{A6}$	A5	A4	$\overline{A3}$	A2	$\overline{A1}$	264	A7	$\overline{A6}$	A5	A4	$\overline{A3}$	A2	$\overline{A1}$
066	$\overline{A7}$	$\overline{A6}$	A5	A4	$\overline{A3}$	A2	A1	266	A7	$\overline{A6}$	A5	A4	$\overline{A3}$	A2	A1
070	$\overline{A7}$	$\overline{A6}$	A5	A4	A3	$\overline{A2}$	$\overline{A1}$	270	A7	$\overline{A6}$	A5	A4	A3	$\overline{A2}$	$\overline{A1}$
072	$\overline{A7}$	$\overline{A6}$	A5	A4	A3	$\overline{A2}$	A1	272	A7	$\overline{A6}$	A5	A4	A3	$\overline{A2}$	A1
074	$\overline{A7}$	$\overline{A6}$	A5	A4	A3	A2	$\overline{A1}$	274	A7	$\overline{A6}$	A5	A4	A3	A2	$\overline{A1}$
076	$\overline{A7}$	$\overline{A6}$	A5	A4	A3	A2	A1	276	A7	$\overline{A6}$	A5	A4	A3	A2	A1
100	$\overline{A7}$	A6	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$	300	A7	A6	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$
102	$\overline{A7}$	A6	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	A1	302	A7	A6	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	A1
104	$\overline{A7}$	A6	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	A2	$\overline{A1}$	304	A7	A6	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	A2	$\overline{A1}$
106	$\overline{A7}$	A6	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	A2	A1	306	A7	A6	$\overline{A5}$	$\overline{A4}$	$\overline{A3}$	A2	A1
110	$\overline{A7}$	A6	$\overline{A5}$	$\overline{A4}$	A3	$\overline{A2}$	$\overline{A1}$	310	A7	A6	$\overline{A5}$	$\overline{A4}$	A3	$\overline{A2}$	$\overline{A1}$
112	$\overline{A7}$	A6	$\overline{A5}$	$\overline{A4}$	A3	$\overline{A2}$	A1	312	A7	A6	$\overline{A5}$	$\overline{A4}$	A3	$\overline{A2}$	A1
114	$\overline{A7}$	A6	$\overline{A5}$	$\overline{A4}$	A3	A2	$\overline{A1}$	314	A7	A6	$\overline{A5}$	$\overline{A4}$	A3	A2	$\overline{A1}$
116	$\overline{A7}$	A6	$\overline{A5}$	$\overline{A4}$	A3	A2	A1	316	A7	A6	$\overline{A5}$	$\overline{A4}$	A3	A2	A1
120	$\overline{A7}$	A6	$\overline{A5}$	A4	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$	320	A7	A6	$\overline{A5}$	A4	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$
122	$\overline{A7}$	A6	$\overline{A5}$	A4	$\overline{A3}$	$\overline{A2}$	A1	322	A7	A6	$\overline{A5}$	A4	$\overline{A3}$	$\overline{A2}$	A1
124	$\overline{A7}$	A6	$\overline{A5}$	A4	$\overline{A3}$	A2	$\overline{A1}$	324	A7	A6	$\overline{A5}$	A4	$\overline{A3}$	A2	$\overline{A1}$
126	$\overline{A7}$	A6	$\overline{A5}$	A4	$\overline{A3}$	A2	A1	326	A7	A6	$\overline{A5}$	A4	$\overline{A3}$	A2	A1
130	$\overline{A7}$	A6	$\overline{A5}$	A4	A3	$\overline{A2}$	$\overline{A1}$	330	A7	A6	$\overline{A5}$	A4	A3	$\overline{A2}$	$\overline{A1}$
132	$\overline{A7}$	A6	$\overline{A5}$	A4	A3	$\overline{A2}$	A1	332	A7	A6	$\overline{A5}$	A4	A3	$\overline{A2}$	A1
134	$\overline{A7}$	A6	$\overline{A5}$	A4	A3	A2	$\overline{A1}$	334	A7	A6	$\overline{A5}$	A4	A3	A2	$\overline{A1}$
136	$\overline{A7}$	A6	$\overline{A5}$	A4	A3	A2	A1	336	A7	A6	$\overline{A5}$	A4	A3	A2	A1
140	$\overline{A7}$	A6	A5	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$	340	A7	A6	A5	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$
142	$\overline{A7}$	A6	A5	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	A1	342	A7	A6	A5	$\overline{A4}$	$\overline{A3}$	$\overline{A2}$	A1
144	$\overline{A7}$	A6	A5	$\overline{A4}$	$\overline{A3}$	A2	$\overline{A1}$	344	A7	A6	A5	$\overline{A4}$	$\overline{A3}$	A2	$\overline{A1}$
146	$\overline{A7}$	A6	A5	$\overline{A4}$	$\overline{A3}$	A2	A1	346	A7	A6	A5	$\overline{A4}$	$\overline{A3}$	A2	A1
150	$\overline{A7}$	A6	A5	$\overline{A4}$	A3	$\overline{A2}$	$\overline{A1}$	350	A7	A6	A5	$\overline{A4}$	A3	$\overline{A2}$	$\overline{A1}$
152	$\overline{A7}$	A6	A5	$\overline{A4}$	A3	$\overline{A2}$	A1	352	A7	A6	A5	$\overline{A4}$	A3	$\overline{A2}$	A1
154	$\overline{A7}$	A6	A5	$\overline{A4}$	A3	A2	$\overline{A1}$	354	A7	A6	A5	$\overline{A4}$	A3	A2	$\overline{A1}$
156	$\overline{A7}$	A6	A5	$\overline{A4}$	A3	A2	A1	356	A7	A6	A5	$\overline{A4}$	A3	A2	A1
160	$\overline{A7}$	A6	A5	A4	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$	360	A7	A6	A5	A4	$\overline{A3}$	$\overline{A2}$	$\overline{A1}$
162	$\overline{A7}$	A6	A5	A4	$\overline{A3}$	$\overline{A2}$	A1	362	A7	A6	A5	A4	$\overline{A3}$	$\overline{A2}$	A1
164	$\overline{A7}$	A6	A5	A4	$\overline{A3}$	A2	$\overline{A1}$	364	A7	A6	A5	A4	$\overline{A3}$	A2	$\overline{A1}$
166	$\overline{A7}$	A6	A5	A4	$\overline{A3}$	A2	A1	366	A7	A6	A5	A4	$\overline{A3}$	A2	A1
170	$\overline{A7}$	A6	A5	A4	A3	$\overline{A2}$	$\overline{A1}$	370	A7	A6	A5	A4	A3	$\overline{A2}$	$\overline{A1}$
172	$\overline{A7}$	A6	A5	A4	A3	$\overline{A2}$	A1	372	A7	A6	A5	A4	A3	$\overline{A2}$	A1
174	$\overline{A7}$	A6	A5	A4	A3	A2	$\overline{A1}$	374	A7	A6	A5	A4	A3	A2	$\overline{A1}$
176	$\overline{A7}$	A6	A5	A4	A3	A2	A1	376	A7	A6	A5	A4	A3	A2	A1

Table 7.5 Program Listing that Makes the 88-ACR Record a Test Byte on the Tape

The following is a listing of the output program used to write test data onto tape. This will be used for the alignment of the 88-ACR Demodulator. The program may be used with any memory size, 256 words or larger.

This program will record the test byte (125) until the program is manually stopped. The program is written using I/O address 6 for status and I/O address 7 for data. If your board address has been wired differently, change the program accordingly.

Address Location (octal)	Octal Code	Mnemonic	Description
200	333	IN	Input
201	006	–	I/O Port Status Address
202	007	RLC	Rotate accumulator left
203	332	JC	Jump if carry
204	200	–	Low \ Address jumped to if zero
205	000	–	High / accumulator
206	076	MVI	Move immediate to A
207	125	–	TEST BYTE
210	323	OUT	Output
211	007	–	I/O Port Data Address
212	303	JMP	Jump unconditional
213	200	–	Low \ Starting address of routine
214	000	–	High /

Table 7.6 Program that Permits the 88-ACR to Retrieve Data from a Cassette Tape

The following is a listing of the program for playback of the Output Test Program. This program will also be used for the alignment of the 88-ACR Demodulator. It is written using the same I/O port addresses as the other program and should be changed accordingly if necessary.

Address Location (octal)	Octal Code	Mnemonic	Description
000	333	IN	Input
001	006	–	I/O Port Status Address
002	017	RRC	Rotate accumulator right
003	332	JC	Jump if carry
004	000	–	Low \ Address jumped to if zero
005	000	–	High / accumulator
006	333	IN	Input
007	007	–	I/O Port Data Address
010	356	XRI	Exclusive Or Immediate with A
011	125	–	Exclusive Or Test Word
012	312	JZ	Jump on zero
013	300	–	Low \ Address jumped to if zero
014	000	–	High / accumulator (Hi Addr. Test Prog.)
015	303	JMP	Jump unconditional
016	000	–	Low \ Address jumped to if zero
017	000	–	High / accumulator
300	257	XRA	Exclusive Or register with A
301	062	STA	Store A direct
302	376	–	Low \ First address to be zeroed
303	000	–	High / out
304	062	STA	Store A direct
305	377	–	Low \ Second address to be
306	000	–	High / zeroed out
307	072	LDA	Load A direct
310	376	–	Low \ Address of data for above
311	000	–	High /
312	306	ADI	Add immediate to A
313	001	–	Data to be added
314	062	STA	Store A direct
315	376	–	Low \ Address for above to be
316	000	–	High / stored
317	322	JNC	Jump on no carry
320	337	–	Low \ Address to be jumped to
321	000	–	High / for above
322	072	LDA	Load A direct
323	377	–	Low \ Address of data for above
324	000	–	High /
325	306	ADI	Add immediate to A
326	001	–	Data to be added
327	062	STA	Store A direct
330	377	–	Low \ Address for above to be
331	000	–	High / stored
332	356	XRI	Exclusive Or immediate with A
333	006	–	Data to be Ex-Ored
334	312	JZ	Jump on zero
335	000	–	Low \ Address jumped to if zero
336	000	–	High / accumulator
337	333	IN	Input
340	006	–	I/O Port Status Address
341	017	RRC	Rotate accumulator right
342	332	JC	Jump if carry
343	307	–	Low \ Address jumped to if zero
344	000	–	High / accumulator
345	333	IN	Input
346	007	–	I/O Port Data Address
347	356	XRI	Exclusive Or immediate with A
350	125	–	Data to be Ex-Ored (Test Byte)
351	312	JZ	Jump on zero
352	300	–	Low \ Address jumped to if zero
353	000	–	High / accumulator
354	303	JMP	Jump unconditional
355	000	–	Low \ Jump to this address if A
356	000	–	High / is not zero

The output program specifies the start address of the data and the ending address, then a test byte is written (000 in this case) followed by data output. The last portion of the program tests to see if the program has transmitted the last byte of data. If it has, the program jumps to the last position in memory, which can be observed by a change in the address lights on the front panel. If the program has not outputted the last byte, the H and L registers of the CPU are incremented by one and the program outputs the next byte. Intended to run in the upper portion of a 4k memory, the program has a starting address of 017 000 (octal). When record-

Table 7.7 Normal I/O Routines for the 88-ACR Interface

Write Program (38 bytes)

Writing data on tape through the 88-ACR is accomplished by first specifying the start address of data and the end address of data. Then a test byte (000 in this program) is written, followed by data output. The last portion of the program tests to see if the program has transmitted the last byte of data. If it has, the program jumps to the last positions in memory, and is observed by a change in the address lights on the front panel. If the program has not outputted the last data byte, the H & L registers are incremented by 1 and the program outputs the next byte. This program is placed in the upper portion of 4K memory with a starting address of 017,000. The location may be changed, but be sure to change all jump addresses accordingly. After recording data that include program information, write down the start and end address on the tape cartridge along with the name and test byte of the program for identification.

When recording data at the beginning of a cassette tape, record at least 15 seconds of steady tone before running the write program (to get past the plastic leader and wrinkles in the beginning of the tape). Also, if recording more than one batch of data, leave at least 5 seconds of steady tone between batches. This program is written for 88-ACR addresses of 6 and 7.

Tag	Mnemonic	Address	Octal Code	Explanation
	LXI	017,000	041	Load immediate H & L register pair
		1	xxx	Low starting address of
		2		High data to be written
	LXI	3	001	Load immediate B & C register pair
		4	xxx	Low end address of
		5	xxx	High data to be written
	MVI	6	076	Move immediate to accumulator
		7	000	Test byte to be written at beginning
	OUT	017,010	323	Output data from accumulator
		11	007	Data channel No. of 88-ACR
Test	IN	12	333	Input data to accumulator
		13	006	Status channel No. of 88-ACR
	RLC	14	007	Rotate accumulator left, test for D7 true
	JC	15	332	Jump if carry (D7 not true)
		16	012	To "TEST"
		17	017	
	MOV	017,020	176	Move contents of memory specified by H & L register to accumulator
	OUT	21	323	Output data from accumulator
		22	007	Data channel No. of 88-ACR
	MOV	23	175	Move contents of L register to accumulator
	CMP	24	271	Compare accumulator vs B register
	JNZ	25	302	Jump if not zero (L ≠ B)
		26	040	To "NEXT"
		27	017	
	MOV	017,030	174	Move contents of H register to accumulator
	CMP	31	270	Compare accumulator vs C register
	JNZ	32	302	Jump if not zero (H ≠ C)
		33	040	To "NEXT"
		34	017	
	JMP	35	303	Jump (if L = B and (H = C)
		36	375	To "END"
		37	017	
NEXT	INX	017,040	043	Increment register pair H & L
	JMP	1	303	Jump
		2	012	To "TEST"
		3	017	
END	JMP	017,375	303	Jump (loop to self)
		376	375	To "END"
		377	017	

Read Program (48 bytes)

As in the write program, start and end addresses of incoming data are specified first. Next, the program looks for the test byte (000 in this program). Once the test byte is detected, the program inputs data and stores them in memory as specified by the H & L registers. The next portion of the program tests to see if the end memory address has been filled. If it has, the program jumps to the last positions in memory, and is observed by a change in the address lights on the front panel. If it is not the end, then the program increments H & L by 1 and jumps back to input another data byte. This program is placed in the upper portion of 4K of memory with a starting address of 017,000. The location may be changed, but be sure to change all jump addresses accordingly. When reading data back in, the tape and program should be started a few seconds before the start of data.

Tag	Mnemonic	Address	Octal Code	Explanation
	LXI	017,000	041	Load immediate H & L register pair
		1	xxx	Low starting address of
		2	xxx	High data to be read
	LXI	3	001	Load immediate B&C register pair
		4	xxx	Low end address of

Table 7.7 (cont'd) Normal I/O Routines for the 88-ACR Interface

Tag	Mnemonic	Address	Octal Code	Explanation
		5	xxx	High data to be read
TSTBT	IN	6	333	Input data to accumulator
		7	006	Status channel # of 88-ACR
	RRC	017,010	017	Rotate accumulator right (test DØ true)
	JC	11	332	Jump if carry (DØ not true)
		12	006	To "TSTBT"
		13	017	
	IN	14	000	Input data to accumulator
		15	007	Data channel No. of 88-ACR
	CPI	16	376	Compare immediate with test byte vs accumulator
		17	000	Test byte
	JNZ	017,020	302	Jump if not zero (test byte ≠ input byte)
		21	006	To "TSTBT"
		22	017	
TEST	IN	23	333	Input data to accumulator
		24	006	Status channel # of 88-ACR
	RRC	25	017	Rotate accumulator right (test DØ true)
	JC	26	332	Jump if carry (DØ not true)
		27	023	To "TEST"
		017,030	017	
DATA	IN	31	333	Input data to accumulator
		32	007	Data channel # of 88-ACR
	MOV	34	175	Move contents of L register to accumulator
	CMP	35	271	Compare accumulator vs B register
	JNZ	36	302	Jump if not zero (L ≠ B)
		37	051	To "NEXT"
		017,040	017	
	MOV	41	174	Move contents of H register to accumulator
	CMP	42	270	Compare accumulator vs C register
	JNZ	43	302	Jump if not zero (H ≠ C)
		44	051	To "NEXT"
		45	017	
	JMP	46	303	Jump (if L = B and H = C)
		47	375	To "END"
		017,050	017	
NEXT	INX	51	043	Increment H & L register pair
	JMP	52	303	Jump
		53	023	To "TEST"
		54	017	
END	JMP	017,375	303	Jump (loop to self)
		376	375	To "END"
		377	017	

ing data at the beginning of a cassette tape, record at least 15 seconds of steady tone before starting the program, and when recording more than one program on a tape, leave at least five seconds of steady tone between batches.

Similarly, the read-data program requires that the start and end addresses of incoming data be specified. Next, the read program looks for the test byte (000 for this example) and once the byte is detected, the program looks for data and stores the data in memory as specified by the H and L registers. The next portion of the program tests to see if the ending address has been reached; if it has, the program jumps to the last positions in memory and indicates the abrupt address change on the front panel. If the program is not fully entered, the program increments the H and L registers by one and jumps back to input another data byte. This input routine is also intended to start at address 017 000 (octal). Remember, when reading data back in, the tape and program should be started a few seconds before the start of the data.

Get Larger Storage Capability with a Floppy

If the access-time or storage limitations of the cassette drives or tape recorder rule them out as a viable medium to record and retrieve data, your next step up must be to one of the many floppy-disk drives available from about 20 manufacturers. There are four basic types of floppy-disk drives—the mini floppy and the dual-sided mini floppy, which both use a 5 in. diameter flexible disk as the storage media; and the full floppy and dual-sided full floppy, which both use an 8 in. diameter medium. Developed to reduce the space and cost requirement in small systems, the mini floppy (Fig. 7.13) has become a very popular medium due to the low cost of the drive and the system flexibility it offers. For instance, in kit form several companies offer single or dual drive systems that cost well under $1500, but when interfaced to your computer system provide you with the computing power equivalent to a minicomputer. If larger storage than the mini floppy is needed, the full floppy drives can be used in similar multidrive systems.

Storage capacities of the floppy disks depend on the format used to store the data on the disk. IBM has developed a format that permits up to 300 kbytes of data on a full floppy disk and about 100 kbytes on a mini floppy. Other recording techniques and data formats are available that provide double the recording density

Fig. 7.13 The mini-floppy disk has become the most popular low-cost mass storage system for microcomputers. (*Courtesy* Icom and Pertec)

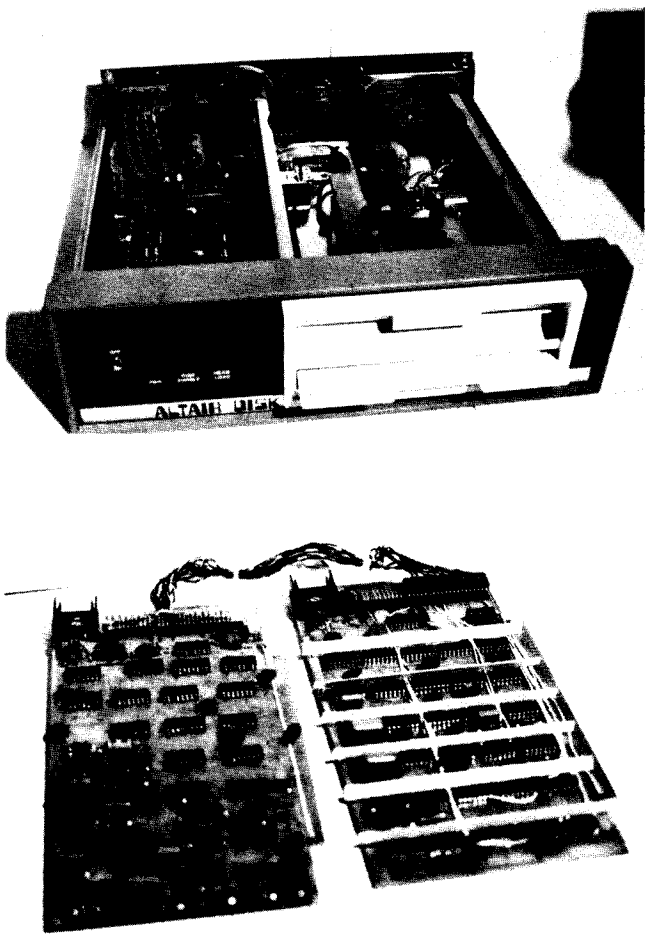


Fig. 7.14 Providing control of up to 20 full-sized floppy disk drives, the 88-DCDD disk controller system developed by Pertec consists of two circuit cards that plug into the computer. (Photo by J. Bierman)

on the same size disks and thus twice the recording capacity. The biggest problem with floppy-disk systems available today is the problem of making the data compatible if you remove a disk from a system used with an Altair, for instance, and use the data in a system with another computer. And the floppy drives themselves are not even standardized—each requires a different set of control signals, a different size cabinet, and different power requirements. Even the software that controls each company's drive must be written just for your computer, or at least slightly modified depending upon the way your computer system is organized. The port addresses must be set as well as the basic control instructions that load in the basic drive control instructions.

Of course, the computer must have enough RAM to hold the basic control instructions and still permit some workspace for you to develop new programs. Many disk-operating systems (the program necessary just for the computer to control the disk drive and format the data) typically require anywhere from 8 to 24 kbytes. So, not only will you have to pay more money for a more complicated memory storage system, but you'll have to boost the computer's RAM space up to at least 24 kbytes and possibly 32 kbytes to provide ample workspace for program development. The disk-operating-system program usually resides on the disk itself and a shorter, bootstrap program stored in ROM or PROM or on paper tape is first loaded into the computer to provide minimal control of the disk to load the larger control programs into the computer.

Floppy-disk operating systems are too complex to build from scratch unless you have an excellent technical background and are willing to spend hundreds of hours getting the system to work. There are, though, many S-100 bus compatible controllers for various types of floppy-disk drives. One such system is the 88-DCDD offered by Pertec for the Altair computer system (Fig. 7.14). It requires a minimum of 24 kbytes of RAM, preferably 32 kbytes, and consists of a full-sized Pertec floppy-disk drive with all necessary control electronics. The control portion of the system consists of two boards that plug into the computer bus and the basic control circuitry built into the drive. The manual that describes how the system operates contains 150 pages—too much to explain here—so only a brief summary of how the drive and controller operate will be given here.

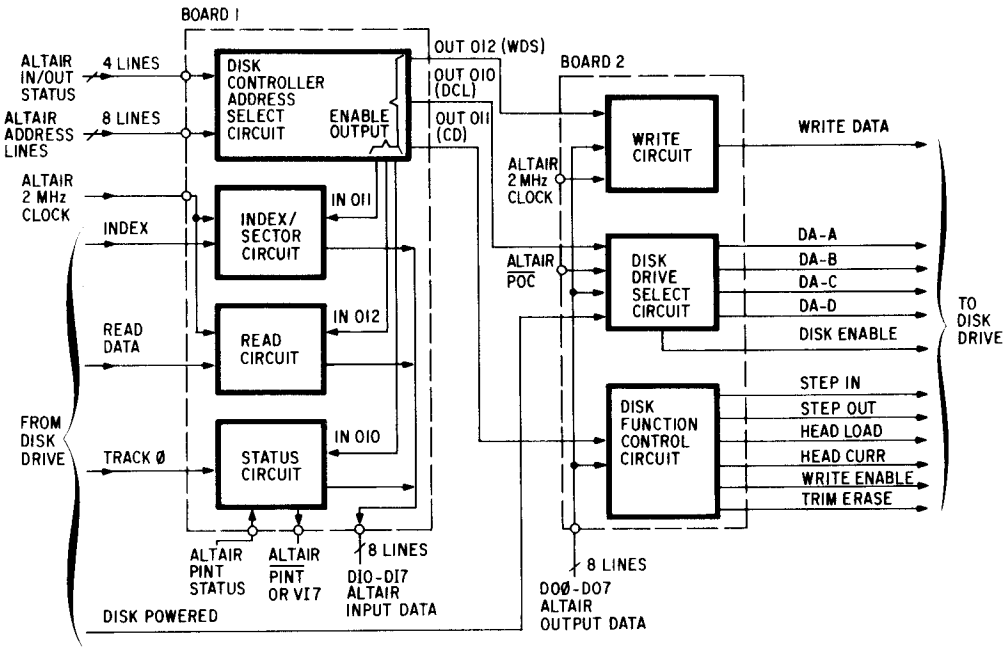


Fig. 7.15 One of the most complex subsystems used in the computer, the 88-DCDD controller card's functions are split into two halves, as this block diagram indicates.

The 88-DCDD system capacity is over 300 kbytes of storage on each disk, with an access time of less than a second for any data stored on the disk. The boards can control up to 16 disk drives (Model FD-400 from Pertec), thus providing a total storage capacity of almost 5 Mbytes. Connections to the drives from the controller cards are made in a daisy-chain fashion via 37-pin connectors. The controller boards connect to the first drive and then a cable from the first drive goes to the second drive, and so on.

Transferring data serially to and from the disk drive at 250 kbits/second, the controller converts the serial data to parallel words and converts the parallel words to serial data, one word every 32 μs. The controller also controls all mechanical functions of the drive as well as presenting status information to the computer. All timing functions are performed by hardware, thus leaving the computer free for other tasks.

The disk-operating system is subdivided into several functional blocks, as shown in Fig. 7.15. Controller board 1 performs all input functions to the Altair bus (read data, sector data, status information) as well as control addressing of all disk drives to the computer I/O. Board 2 performs all the output functions from the bus (write data, disk control, disk enable, and drive selection).

For the computer to use the disk system, the computer must select and enable the disk drive and the controller. The desired disk address (000 to 017, octal) must first be output on I/O channel 010 (octal). The timing track on the disk, track 0, is found by the computer by first moving the disk's read/write head out to the outermost track and testing the status. After the appropriate reference for track 0 is located, the computer moves the head back to the desired track and then loads the read/write head on the surface of the disk. Software keeps track of the track the disk head is on once track 0 is located. The correct part of the track (sector) is located by performing an input from the sector channel and comparing the desired sector number with the sector count from the controller circuit. After the disk reaches the correct rotational position (sector), the computer performs either a read or write operation. When the computer has finished accessing the disk, the disk control is cleared, enabling the drive and causing all disk functions to stop. Turning off the drive's power, disconnecting the cable, or opening the disk-drive door also clears the disk control. Also, when changing access from one drive to another, the disk control must be cleared before the new drive is enabled. This ensures that the controller circuits are reset before accessing a new drive.

Assuming you've elected to use a full floppy-disk drive in a computer system with 32 kbytes of RAM, two serial I/O ports, two parallel I/O ports, a cassette-tape interface, and a bootstrap PROM board, you've invested about $3000, not including any type of terminal or printer. Add another $1000 to $3000 for a terminal or printer, and you have a complete computer system minus just one more part—the *software*. Empty card slots within the computer can be used to expand the computer's memory, I/O capability, or provide specialized functions such as speech synthesis, speech recognition, music generation, process control, analog interface, etc.—your imagination is the only limit.

The system, once a bootstrap ROM card is used, becomes very simple to operate—after power is turned on, the initial address for the desired bootstrap program must be examined by use of the front-panel switches

and display, and then the RUN switch can be pressed to start execution of the bootstrap program. Once the bootstrap program is loaded, it prompts the user via a terminal so the next program can be loaded in from the disk. Of course, dual- or multiple-disk systems provide even more flexibility—the main operating system programs are typically stored on one disk drive and all programs being developed are stored on the other. This keeps the amount of RAM needed in a large system to a minimum and can prevent serious data loss due to power-downs, since data entered and stored on the disk cannot be easily erased, written over, or lost.

The only missing hardware component for the computer system is some form of hard-copy output—assuming, of course, you opted not to buy a printing terminal or you need a faster delivery of the printout than your terminal can provide. As with picking a terminal, selecting a printer is just as difficult. The three most basic decisions you'll probably have to make are:

1. Do I want upper or lower case capability? The ability of a printer to handle upper and lower case characters is handy for systems that will be used to produce letters and other printed material. However, printers with upper and lower case capability tend to be much more expensive than upper case only units since they require more internal control circuitry and a more complex mechanism.

2. How fast a printer do I need? Printer speeds run the gamut from 10 characters per second to well over 500 characters per second. If you expect to use the printer to produce lengthy program listings or large amounts of other types of copy, a high-speed unit is best, ideally in the 30 to 100 characters per second range. Printers in this range typically cost $1500 to $3000. Units in the 30 to 50 cps range are probably the best choice for the money.

3. What type of interface do I want? Printers, like almost any other peripheral can be purchased with a wide variety of interface circuits. The less you pay for a printer, the less you get—even to the point of a bare printer mechanism that has absolutely no circuitry. Typical options include parallel interfaces (seven or eight data bit lines plus some control lines), serial interfaces such as RS-232 or TTL levels, or some other type of interface.

Of course, an overall limiting factor is how much you can afford to spend for the unit. Sharp shoppers can even pick up some excellent bargains from various surplus dealers. There are many different types of printers available from dozens of dealers, so probably the easiest way to make a decision about selecting a unit is to make your own check list and see how many units can match what you want. Three points have already been narrowed down—speed, interface, and character set. Next, the printing method used should be included—printers can produce characters in three basic methods and each method has various pros and cons.

1. Thermal printers. This type of printer uses tiny dots to form a character on specially treated heat-sensitive paper. Much in the way dots are used to form characters on a CRT screen, the heated dots form a thermal pattern on the paper, which in turn changes the color of the paper. Advantages of this method include almost silent operation combined with fewer moving parts for higher reliability. However, the disadvantages include a limited print speed (typically 30 to 50 cps), the problem of making multiple copies (printouts must be repeated to make more than one copy) and the use of the more expensive thermally sensitive paper.

2. Impact printers. The impact printers are probably the most common variety of printer and are also the noisiest since small solinoids are used to drive hammers against an inked ribbon that, in turn, hits the paper and leaves an inked impression. Impact printers do have the advantage of being able to make multiple copies when the original is being printed, and they do not require any special paper. However, they are noisy and need servicing on regular intervals to replace the ribbon and make sure the hammers are unjammed.

3. Electrostatic printers. This type of printer is capable of completely silent operation, much like the thermal printer. It also has the drawback of not being able to make multiple copies of a printout and requires special metalized paper. However, it can deliver a lot of data in a short amount of time; rates of 100 cps are not uncommon.

Of course, if you need even faster printing speeds, there are other types of printers available that use other techniques to produce printed data. Typical costs of faster than 100 cps printers range from about $3000 to over $10,000 each.

Once you've selected the speed of a printer and the character set, next decide on the interface you will use. If speed wasn't a critical factor, a 20 mA current loop or RS-232 serial interface would provide the least interconnect trouble since only four wires, at the most, would be needed. However, for simplest interconnection and highest speed, a parallel interface should be used. In most parallel interface applications, eight lines are used to transfer the data, and any number of other control lines can be connected to control the printer and computer operation.

Printer Control Signals Are Important

Depending on the type of printer you finally select, the control signals may differ considerably. For a printer with a parallel data input, only three basic control lines are needed in addition to the data lines and a common ground line. The three lines include a Ready line, a Start line, and a Busy line. The Ready line is an output from the printer and signals the computer that the printer is turned on and is functional. Depending

on the printer, the signal can be either high or low, although many printers use a high level to indicate that power is on. The Start line is an input to the printer and when brought high or low (depending on the model) by the computer, it initiates the printer's print sequence. (This assumes a valid character is present at the printer's data input.) While the printer is printing, its last basic control line, the Busy line changes states and outputs its signal to the computer, telling the computer not to send another character on the data lines until the Busy line changes back to the state it was in before the Start line initiated the printer.

Other control lines are possible—it's just a matter of how many control lines you want to have going back and forth. One often available extra line is a control input to the printer that increments the line (advances the paper by one line) and brings the printing mechanism back to the beginning of the new line, thus saving a two character transmission (carriage return and line feed).

Serial input printers have a more limited control capability, since all control is via a three-wire interface. However, the number of connections is vastly reduced and the interface itself is very standardized, typically being an RS-232 or teletypewriter 20 mA current loop.

CHAPTER 8

Programming the 8080 CPU in the S-100 Bus Microcomputer

Before going any further with the use of a microcomputer system, the software (programs) necessary for the basic machine and its interfaces have to be developed. However, before even the basic operating programs for the computer are developed, you have to know how to program. But what is programming?

A program is simply a set of instructions that the computer will obey to perform a specific task, and the "art" of programming is just the process of putting the instructions in the desired sequence. As mentioned earlier, the 8080A has 78 basic commands it will obey, and the Z-80 microprocessor has over 150. Each instruction causes the processor to perform a specific operation. But putting the instructions in the right sequence is a process that must be learned because sometimes thousands of simple instructions make up a program, and just one instruction in the wrong place or an incorrect instruction will cause the entire program to go haywire.

Those of you familiar with large computer systems such as made by IBM, Burroughs, Control Data, and other companies, know that programming is done in almost English-like commands. However the microcomputer, in the form described thus far in the book, comes with nothing as simple. Programming is done in its simplest form—ones and zeros entered into the computer. Each instruction is represented by one or more 8-bit groups of ones and zeros (one or more bytes) and must be loaded into the computer's RAM or permanently stored in ROM where it is pulled from memory when needed. This simple form of programming is called machine-language programming since the computer is being told what to do in its own language of ones and zeros. To some people this is the hardest form of programming since there is no indication of what each instruction is unless you know what every binary, octal, or hexadecimal code represents. And the use of binary, octal, or hexadecimal can also bring even more confusion since many companies use only one of the three. For instance, all programs listed in the Imsai manuals are printed in hexadecimal notation while programs shown in the Pertec manuals are in octal formats. However, for the rest of this chapter all programs, unless specially noted, will appear in hexadecimal format.

To make programs easier to write, each instruction is represented in an abbreviated "English" form called a mnemonic. For instance, the 8080A has an instruction that performs no operation. Its mnemonic is NOP and is represented by the hexadecimal code 00, the octal code 000, or the binary code 00000000. Another instruction, HLT, causes the 8080A to stop what its doing (HALT). The hexadecimal code is 76, the octal code is 166, and the binary code is 01110110. As you can see, it's much easier to read or write and understand mnemonics than it is to read number codes. Mnemonic programming is called assembly-language programming because after the mnemonics are put together into proper sequence (assembled), the numeric codes must be inserted (compiled) so that the computer will receive the machine-language instructions (often referred to as machine code.)

Even more English-like methods are available to help you put together a program. Larger systems can use more English-like statements such as GO TO, A + B =, LET, etc., with languages such as BASIC or FORTRAN to represent whole strings of mnemonic instructions much as in a complex math example where you use letters to represent numbers. For example, let the numbers 1, 2, 3, and 4 be represented by A, B, C, and D (much in the same way that mnemonics represent the binary numbers). Now, if another letter, say E, is used to represent the sum of all the other letters, we've gone one step higher and instead of saying that the sum is 1 + 2 + 3 + 4, or A + B + C + D, we say the sum is E. In much the same way, an English command such as GO TO could represent a string of several mnemonic or numeric instructions. The English-like language is often referred to as a high-level language since, like the letter E, each statement can be broken down into smaller statements. The most commonly used language is BASIC—in various forms—since most microcomputers can use it without burning up much valuable memory space to hold the programming system. (There are versions of BASIC that

can operate with as little as 4096 words of memory space although many of the versions need at least 8k and some 16k.)

Start by Flowcharting the Problem

Writing a program for a computer is simple if you think very logically. If you don't, the best way to develop a program is to first diagram the problem you want to solve, showing all the major steps the computer should take to do the job. This process is referred to as developing the flowchart—a map of how the computer will perform its job. There are two major symbols used on flowcharts to distinguish some important operations. A diamond-shaped box (Fig. 8.1a) indicates that the processor is making some sort of decision or doing a comparison. Usually the diamond will have one input and two outputs—two outputs because the decision is often in the form of yes or no. The program will perform one set of tasks if the decision is yes and a different set of tasks if the decision is no; this is sometimes called a branch point. The other symbol is just a rectangular box (Fig. 8.1b) that represents a cluster of instructions that don't require the computer to make a decision but must be performed. By combining these symbols and some words within the boxes, programs can be flowcharted.

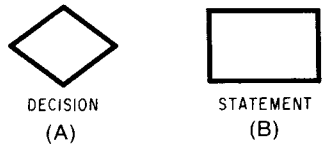


Fig. 8.1 To make a flowchart, the diamond-shaped box (a) is used to represent a decision point and the rectangular-shaped box (b) represents a statement or a group of statements that do not involve a decision.

A typical day in your life can be represented in flowchart form and would appear very similar to a computer program. For instance, the flowchart shown in Fig. 8.2 could be someone's daily routine. Note that right at the beginning of the program there are some important decision blocks that can drastically alter the sequence of events. This flowchart is very simple compared to the actual complexities and number of jobs to be performed.

Let's go back now to the actual computer system and go through some extremely simple instructions to familiarize you with flowcharting and some of the actual instructions for the 8080A. For the first example, let's see how the computer would add two numbers together (Fig. 8.3). As you can see, the process is just a sequence of rectangular boxes describing each operation. Data (the numbers to be added) can either be stored in the

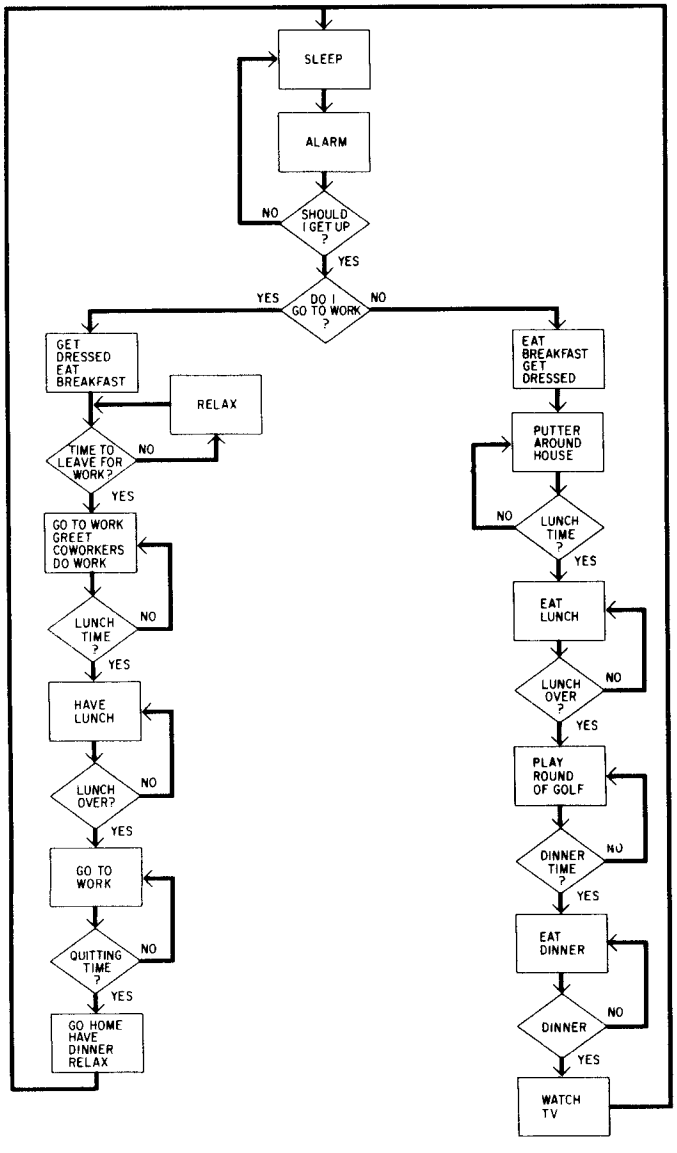


Fig. 8.2 A simplified flowchart of someone's daily routine.

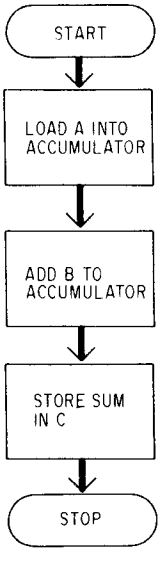


Fig. 8.3 Simple flowchart for a two number addition program.

LDA	A	3A 01 00
LHDL	B	2A 02 00
ADD	M	86
STA	C	32 00 00

Note: This assumes that the numbers to be added are already in A and B. LHDL is an alternate instruction to the two MVI instructions shown in the text.

Fig. 8.4 Mnemonic and machine code listing for the simple addition program of Fig. 8.3.

memory or loaded into the processor from an outside source when needed. Let's look at one possible method the computer can use to add several numbers. The sequence of commands shown in Fig. 8.4 assumes that the numbers to be added (say A and B) are stored in memory locations 0001 and 0002. Since all mathematic and logic operations take place in the 8080A's accumulator, the numbers in the memory must be added to the accumulator.

At this point, there are two different ways to get the first number into the accumulator. A Load Accumulator Direct (LDA) instruction can be used. This instruction tells the computer to use the two bytes following the instruction as the low order and high order address bytes of the information to be loaded in. Thus, the sequence of commands in assembly language and machine language would start off with

LDA A or 3A 01 00

The next step in the sequence is to get the number B added to the accumulator. To do that, an ADD M instruction will be used. However, to use an ADD M instruction requires that the address of the data be stored in the H and L registers of the 8080A. Thus, two instructions must be combined—first the address of B must be loaded into the H and L registers by using two MVI (Move Intermediate) instructions and then the ADD instruction must be used. This sequence looks like:

MVI L	26 02
MVI H	2E 00
ADD M	86

Now the sum of the two numbers A and B is held in the accumulator. To store that sum somewhere, say location 0000 (call it C), another command that stores the accumulator in the memory (STA) must be used. Its sequence would be:

STA C 32 00 00

As you can see, programming the 8080A to perform the simple job of adding two numbers requires five instructions. However, before looking at more programs and the instruction set, let's take a look at the 8080A and its registers.

Get to Know the 8080A Programming Model

Inside the 8080A are seven registers called working registers. They are numbered and referenced by the numbers 0, 1, 2, 3, 4, 5, and 7 or by the letters B, C, D, E, H, L, and A (for the accumulator), respectively. In these registers, all operations take place. Sometimes these registers are accessed in pairs and are referred to as register pairs B (B and C), D (D and E), H (H and L), and PSW (A and a special register that holds various status flags). Also included in the 8080A are two special registers called the program counter, PC, and the stack pointer, SP. The PC is a 16-bit register that is used to direct the processor to the memory location holding the next instruction that is to be executed. The SP is also a 16-bit register and it is used by the processor to access a reserved area of memory called a stack in which data and addresses are stored and retrieved. Stack operations are performed by several of the 8080A instructions and are most often used to handle program subroutines (programs within programs) and program interrupts. All seven of the 8-bit registers and both 16-bit registers are accessible via instructions.

A processor's program contains a sequence of instructions, where each instruction performs a simple operation such as an arithmetic or logic operation, the movement of a data byte, or a change in instruction execution sequence. The memory address of the next instruction to be executed is held in the PC. Just before each instruction is executed, the PC is incremented to the next instruction address. Execution proceeds sequentially unless a jump, call, or return instruction occurs to cause the program counter to be set to a specific address and then execution proceeds sequentially starting with the new address.

However, when the processor examines the contents of a memory byte, it has no way of determining whether the binary bit pattern represents an instruction or a data word. Thus, it is important that the program logic tells the processor whether to expect an instruction or data. When first turned on and reset, the processor expects to retrieve an instruction. Every program has a starting address (the address of the first instruction byte to be executed) that the machine must be directed to either via the front-panel switches or another program. Since 8080A instructions require one, two, or three bytes, the program counter must keep track of which instruction is being executed so it increments at the proper time (Fig. 8.5). To avoid errors, make sure that a data byte does not follow an instruction when another instruction is expected by the processor.

A group of the 8080A's instructions, often referred to as transfer-of-control commands, cause program execution to branch to a set of instructions that are out of numerical sequence (located in a different part of the memory). The address specified to jump to must contain another instruction since the processor is waiting to

Memory address (Hex) bytes of instruction	Inst #	PC contents
0212	1	0213
0213 0214	2	0215
0215	3	0216
0216 0217 0218	4	0219
0219	5	021A
021A 021B	6	021C
021C 021D 021E	7	021F
021F	8	0220
0220	9	0221
0221	10	0222

Fig. 8.5 Depending on whether the instruction is one, two, or three bytes long, the program counter will increment by one, two, or three.

get an instruction, not data. Thus, as you can see, addressing the memory is an important part of any processor's program. And there are several ways that memory locations can be accessed.

Direct Memory Addressing uses the two bytes immediately following the instruction to point to the memory location being accessed. For instance, the instruction LDA XY tells the processor to load the contents of memory address XY into the accumulator (X represents the eight LSB's of the address and Y represents the eight MSB's). One point to especially note—when using the 8080A, all addresses are entered in reverse byte order—the lower byte first and then the higher byte.

A memory address may also be specified by the contents of a register pair (*Register-Pair Addressing*). For many 8080A instructions the H and L registers are used; the H register holds the eight MSB's of the address and the L register holds the eight LSB's. An instruction such as ADD M, adds the contents of a memory location specified by the H and L registers to the number already in the accumulator. There are, though, two instructions that use either the B and C or D and E registers to address the memory. These instructions, STAX and LDAX, will store the contents of the accumulator or load the accumulator to or from the location specified by register pairs B and C or D and E.

In a reserved portion of memory called a stack, memory words can also be accessed by using the SP register to point to the location. This *Stack-Pointer Addressing*, though, is not as flexible and there are only two operations possible—data can be put onto the stack (a push operation) or retrieved from the stack (a pop operation). A stack is very similar to a pile of cafeteria trays; old trays get piled on the top and when removed are taken away in reverse order (last in, first out). During a push operation, 16 bits of data are transferred to the stack from either a register pair or the PC. The actual address accessed is determined by the value of the SP register. The eight MSB's are stored at the memory address one less than the value of the SP register, the eight LSB's are stored at the memory address two less than the value of the SP, and the value of the SP ends up decremented by two. For a pop operation, 16 data bits are transferred from the memory to a register pair or to the PC. The SP register determines which locations are accessed. First, the second register of the pair (or the eight LSB's) of the PC is loaded from the memory address specified by the SP. Next, the first register of the pair (or the eight MSB's of the PC) is loaded from the memory address one greater than the value of the SP, and lastly, the SP ends up incremented by two.

Another addressing mode, *Immediate Addressing*, combines the instruction and the data to be handled in sequential bytes. For instance, the instruction Load the Accumulator with the value 2A (MVI 2A) has the operation (Load the Accumulator) and the data (2A) in two sequential bytes. Immediate instructions do not access the memory, nor are they as flexible as some other commands.

Often, when writing programs there are groups of instructions that are repeated several times during execution. Instead of repeating the instructions each time they are needed, a master set of the instructions can be accessed. This master set is called a subroutine since it is usually only part of a larger, more complex program (Fig. 8.6). Several instructions are available that can

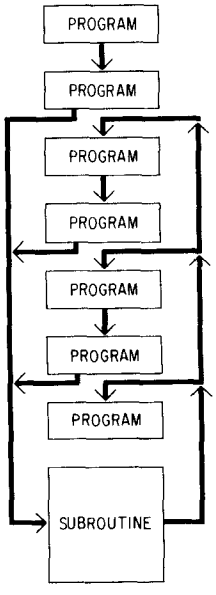


Fig. 8.6 A subroutine can be used each time the main program must perform the same routine over and over again at different points in its operation.

divert the main program to the subroutine and then go back to the main program (Call and Return). When the Call instruction is executed, the address of the next instruction, which is held in the PC, is pushed onto the stack, the address of the starting instruction is loaded into the processor, and the subroutine is then executed. The last executed instruction of the subroutine is usually a Return instruction which pops an address off the stack and loads it back into the PC, thus bringing the main program back into action and execution continues with the "next" instruction. Sometimes there can even be subroutines within subroutines and they can be stacked (often called nested) one within the next within the next, etc., up to any amount, limited only by the amount of memory available for the stack.

Condition Bits in the 8080A Tell You Its Secrets

As part of the 8080A, there are five special bits known as condition or status bits that are available as indicators to reflect the results of data operations. All but one of these bits (the auxiliary carry bit) can be tested by program instructions, which, in turn, can alter the flow of the program depending on the state of the bit. If a bit is set, its logic value is 1; if it is reset, its logic value is 0. The five bit names are the carry bit, auxiliary carry bit, sign bit, zero bit, and parity bit.

The carry bit can be set or reset by various data operations, and its condition (0 or 1) can be tested by an instruction. Operations that can alter the bit are addition, subtraction, rotate, and logic commands. If a carry is generated from the MSB of the bytes being operated on, the carry bit is set; if no carry is generated, the bit is reset. The auxiliary carry bit indicates a carry from bit 3 to bit 4 of an operation in the accumulator. It is set or reset by addition, subtraction, increment, decrement, and compare instructions. As mentioned before, it cannot be tested by a program instruction. It is present only to enable one useful instruction, DAA (decimal adjust accumulator), to do its job.

The sign bit is used to indicate the value of a byte of data within the range of -128_{10} to $+127_{10}$. If bit 7 of a word is 1, the number represented by bits 0 to 6 is within the range of -128_{10} to -1. If bit 7 is 0, the number is in the range 0 to 127_{10}. After certain 8080A instructons have been executed, the sign bit will be set to the condition of the MSB of the byte being examined. The zero bit is set when the result generated by the execution of certain instructions is zero. If the result is not zero, the zero bit is reset. And a result that has a carry but a zero answer will also set the zero bit. After certain operations, byte parity is checked and the number of 1 bits in a byte are counted. If the total is odd, odd parity is flagged (parity bit set to 0); if the total is even, even parity is indicated (parity bit set to 1).

Fig. 8.7 Although programming in assembly language provides you with readable source code, the final program used by the computer must be in object code.

Just to summarize what has been said so far, programming can be done at any of three levels—machine language, assembly language, or high-level language. For this chapter, we will look at machine and assembly language programming and how to develop programs for the 8080A. Also, we've seen that inside the 8080A are seven accessible, general-purpose registers (including the accumulator), a stack pointer, and a program counter. With these registers and five condition bits, all aspects of microprocessor operation can be observed and controlled.

Programs can, of course, be written in the simplest form—machine language. However, unless you know all the instructions by heart you'll be hard pressed to remember what you've written, and which byte represents an instruction or data. So for now, programming in assembly language is the best route to follow. When you write programs for your computer system in assembly language, the programs must be translated into executable code either by hand on a line-by-line basis or by a program called an assembler that is also stored in the computer. The assembly language program written by you is referred to as a *source* program which is then converted by an assembler into an *object* program that can be loaded into the processor's memory and executed (Fig. 8.7). So, before going any further with the discussion of programming, let's first look at the various 8080A instructions and what they do.

Examine the 8080A Instruction Set Carefully

There are 78 basic instructions in the 8080A command set and the only way to find out what each one does is to examine each one. Tables 8.1 and 8.2 summarize all the instructions (Table 8.1 in increasing op code value and Table 8.2 in alphabetical order), but to examine each one, let's pull apart each instruction going in alphabetical order.

Mnemonic and definition; op code

ACI Add immediate data to accumulator with carry; CE.

The byte immediately following the instruction is assumed to be data and is added to the contents of the accumulator plus the contents of the carry

Table 8.1 8080A Instructions by Increasing Op Code Value

Op Code	Mnemonic	Op Code	Mnemonic	Op Code	Mnemonic	Op Code	Mnemonic	Op Code	Mnemonic	Op Code	Mnemonic
00	NOP	2B	DCX H	56	MOV D, M	81	ADD C	AC	XRA H	D7	RST 2
01	LXI B, D16	2C	INR L	57	MOV D, A	82	ADD D	AD	XRA L	D8	RC
02	STAX B	2D	DCR L	58	MOV E, B	83	ADD E	AE	XRA M	D9	–
03	INX B	2E	MVI L, D8	59	MOV E, C	84	ADD H	AF	XRA A	DA	JC Adr
04	INR B	2F	CMA	5A	MOV E, D	85	ADD L	B0	ORA B	DB	IN D8
05	DCR B	30	–	5B	MOV E, E	86	ADD M	B1	ORA C	DC	CC Adr
06	MVI B, D8	31	LXI SP, D16	5C	MOV E, H	87	ADD A	B2	ORA D	DD	–
07	RLC	32	STA Adr	5D	MOV E, L	88	ADC B	B3	ORA E	DE	SBI D8
08	–	33	INX SP	5E	MOV E, M	89	ADC C	B4	ORA H	DF	RST 3
09	DAD B	34	INR M	5F	MOV E, A	8A	ADC D	B5	ORA L	E0	RPO
0A	LDAX B	35	DCR M	60	MOV H, B	8B	ADC E	B6	ORA M	E1	POP H
0B	DCX B	36	MVI M, D8	61	MOV H, C	8C	ADC H	B7	ORA A	E2	JPO Adr
0C	INR C	37	STC	62	MOV H, D	8D	ADC L	B8	CMP B	E3	XTHL
0D	DCR C	38	–	63	MOV H, E	8E	ADC M	B9	CMP C	E4	CPO Adr
0E	MVI C, D8	39	DAD SP	64	MOV H, H	8F	ADC A	BA	CMP D	E5	PUSH H
0F	RRC	3A	LDA Adr	65	MOV H, L	90	SUB B	BB	CMP E	E6	ANI D8
10	–	3B	DCX SP	66	MOV H, M	91	SUB C	BC	CMP H	E7	RST 4
11	LXI D, D16	3C	INR A	67	MOV H, A	92	SUB D	BD	CMP L	E8	RPE
12	STAX D	3D	DCR A	68	MOV L, B	93	SUB E	BE	CMP M	E9	PCHL
13	INX D	3E	MVI A, D8	69	MOV L, C	94	SUB H	BF	CMP A	EA	JPE Adr
14	INR D	3F	CMC	6A	MOV L, D	95	SUB L	C0	RNZ	EB	XCHG
15	DCR D	40	MOV B, B	6B	MOV L, E	96	SUB M	C1	POP B	EC	CPE Adr
16	MVI D, D8	41	MOV B, C	6C	MOV L, H	97	SUB A	C2	JNZ Adr	ED	–
17	RAL	42	MOV B, D	6D	MOV L, L	98	SBB B	C3	JMP Adr	EE	XRI D8
18	–	43	MOV B, E	6E	MOV L, M	99	SBB C	C4	CNZ Adr	EF	RST 5
19	DAD D	44	MOV B, H	6F	MOV L, A	9A	SBB D	C5	PUSH B	F0	RP
1A	LDAX D	45	MOV B, L	70	MOV M, B	9B	SBB E	C6	ADI D8	F1	POP PSW
1B	DCX D	46	MOV B, M	71	MOV M, C	9C	SBB H	C7	RST 0	F2	JP Adr
1C	INR E	47	MOV B, A	72	MOV M, D	9D	SBB L	C8	RZ	F3	DI
1D	DCR E	48	MOV C, B	73	MOV M, E	9E	SBB M	C9	RET Adr	F4	CP Adr
1E	MVI E, D8	49	MOV C, C	74	MOV M, H	9F	SBB A	CA	JZ	F5	PUSH PSW
1F	RAR	4A	MOV C, D	75	MOV M, L	A0	ANA B	CB	–	F6	ORI D8
20	–	4B	MOV C, E	76	HLT	A1	ANA C	CC	CZ Adr	F7	RST 6
21	LXI H, D16	4C	MOV C, H	77	MOV M, A	A2	ANA D	CD	CALL Adr	F8	RM
22	SHLD Adr	4D	MOV C, L	78	MOV A, B	A3	ANA E	CE	ACI D8	F9	SPHL
23	INX H	4E	MOV C, M	79	MOV A, C	A4	ANA H	CF	RST 1	FA	JM Adr
24	INR H	4F	MOV C, A	7A	MOV A, D	A5	ANA L	D0	RNC	FB	EI
25	DCR H	50	MOV D, B	7B	MOV A, E	A6	ANA M	D1	POP D	FC	CM Adr
26	MVI H, D8	51	MOV D, C	7C	MOV A, H	A7	ANA A	D2	JNC Adr	FD	–
27	DAA	52	MOV D, D	7D	MOV A, L	A8	XRA B	D3	OUT D8	FE	CPI D8
28	–	53	MOV D, E	7E	MOV A, M	A9	XRA C	D4	CNC Adr	FF	RST 7
29	DAD H	54	MOV D, H	7F	MOV A, A	AA	XRA D	D5	PUSH D		
2A	LHLD Adr	55	MOV D, L	80	ADD B	AB	XRA E	D6	SUI D8		

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.

Table 8.2 8080A Instructions in Alphabetical Order

Mnemonic	Description	Mnemonic	Description
ACI	Add immediate to A with carry	LHLD	Load H & L direct
ADC M	Add memory to A with carry	LXI B	Load immediate register pair B & C
ADC r	Add register to A with carry	LXI D	Load immediate register pair D & E
ADD M	Add memory to A	LXI H	Load immediate register pair H & L
ADD r	Add register to A	LXI SP	Load immediate stack pointer
ADI	Add immediate to A	MVI M	Move immediate memory
ANA M	And memory with A	MVI r	Move immediate register
ANA r	And register with A	MOV M, r	Move register to memory
ANI	And immediate with A	MOV r, M	Move memory to register
CALL	Call unconditional	MOV r1, r2	Move register to register
CC	Call on carry	NOP	No-operation
CM	Call on minus	ORA M	Or memory with A
CMA	Compliment A	ORA r	Or register with A
CMC	Compliment carry	ORI	Or immediate with A
CMP M	Compare memory with A	OUT	Output
CMP r	Compare register with A	PCHL	H & L to program counter
CNC	Call on no carry	POP B	Pop register pair B & C off stack
CNZ	Call on no zero	POP D	Pop register pair D & E off stack
CP	Call on positive	POP H	Pop register pair H & L off stack
CPE	Call on parity even	POP PSW	Pop A and flags off stack
CPI	Compare immediate with A	PUSH B	Push register pair B & C on stack
CPO	Call on parity odd	PUSH D	Push register pair D & E on stack
CZ	Call on zero	PUSH H	Push register pair H & L on stack
DAA	Decimal adjust A	PUSH PSW	Push A and flags on stack
DAD B	Add B & C to H & L	RAL	Rotate A left through carry
DAD D	Add D & E to H & L	RAR	Rotate A right through carry
DAD H	Add H & L to H & L	RC	Return on carry
DAD SP	Add stack pointer to H & L	RET	Return
DCR M	Decrement memory	RLC	Rotate A left
DCR r	Decrement register	RM	Return on minus
DCX B	Decrement B & C	RNC	Return on no carry
DCX D	Decrement D & E	RNZ	Return on no zero
DCX H	Decrement H & L	RP	Return on positive
DCX SP	Decrement stack pointer	RPE	Return on parity even
DI	Disable Interrupt	RPO	Return on parity odd
EI	Enable Interrupt	RRC	Rotate A right
HLT	Halt	RST	Restart
IN	Input	RZ	Return on zero
INR M	Increment memory	SBB M	Subtract memory from A with borrow
INR r	Increment register	SBB r	Subtract register from A with borrow
INX B	Increment B & C registers	SBI	Subtract immediate from A with borrow
INX D	Increment D & E registers	SHLD	Store H & L direct
INX H	Increment H & L registers	SPHL	H & L to stack pointer
INX SP	Increment stack pointer	STA	Store A direct
JC	Jump on carry	STAX B	Store A indirect
JM	Jump on minus	STAX D	Store A indirect
JMP	Jump unconditional	STC	Set carry
JNC	Jump on no carry	SUB M	Subtract memory from A
JNZ	Jump on no zero	SUB r	Subtract register from A
JP	Jump on positive	SUI	Subtract immediate from A
JPE	Jump on parity even	XCHG	Exchange D & E, H & L registers
JPO	Jump on parity odd	XRA M	Exclusive Or memory with A
JZ	Jump on zero	XRA r	Exclusive Or register with A
LDA	Load A direct	XRI	Exclusive Or immediate with A
LDAX B	Load A indirect	XTHL	Exchange top of stack, H & L
LDAX D	Load A indirect		

NOTES: 1. DDD or SSS — 000 B — 001 C — 010 D — 011E — 100H — 101L — 110 Memory — 111 A. 2. Two possible cycle times, (5/11) indicate instruction cycles dependent on condition flags.

bit. All five condition bits can be affected by an ACI operation.

ADC M or **ADC r** Add memory to accumulator with carry or add register to accumulator with carry; 8E or 88, 89, 8A, 8B, 8C, 8D, or 8F (memory or registers B, C, D, E, H, L, or A, respectively).

The specified byte plus the contents of the carry bit are added to the contents of the accumulator. All five condition bits can be affected.

ADD M or **ADD r** Add memory or register to accumulator; 86 or 80, 81, 82, 83, 84, 85, or 87 (memory or registers B, C, D, E, H, L, or A respectively).

The specified byte is added to the contents of the accumulator using two's complement arithmetic. All condition bits can be affected by an **ADD** operation.

ADI Add immediate data to accumulator; C6.

The byte immediately following the instruction is to be added to the contents of the accumulator using two's complement arithmetic. All five condition bits can be affected by the instruction.

ANA M or **ANA r** Logic AND memory or register with contents of accumulator; A6 or A0, A1, A2, A3, A4, A5, or A7 (memory or registers B, C, D, E, H, L, or A, respectively).

The specified byte is ANDed bit by bit with the contents of the accumulator and the carry bit is reset to zero. All five condition bits can be affected by the instruction.

ANI AND immediate data with accumulator; E6.

The byte of data immediately following the instruction is ANDed with the contents of the accumulator and the carry bit is reset to zero. All five condition bits can be affected by the instruction.

CALL Call a subroutine; CD XY (X = low address byte, Y = high address byte).

A **CALL** operation is an unconditional request to the processor to go to a subroutine at address YX. **CALL** pushes the contents of the program counter (the address of the next sequential instruction) onto the stack and then jumps to the address specified. The instruction has no effect on the condition bits.

CC Call if carry; DC XY.

This instruction examines the carry bit and if the carry bit is one a CALL operation is performed to location YX. **CC** pushes the contents of the PC (program counter) onto the stack. No condition bits are affected.

CM Call if minus; FC XY.

This instruction examines the sign bit (which indicates a minus result if it is one) and will perform a **CALL** operation to location YX if the sign bit is one. **CM** pushes the contents of the PC onto the stack. No condition bits are affected.

CMA Complement accumulator; 2F.

This instruction complements the contents of the accumulator, producing the one's complement of the number. No condition bits are affected.

CMC Complement carry bit; 3F.

This instruction complements the value of the carry bit. Only the carry bit (CY) is affected.

CMP M or **CMP r** Compare memory or register with accumulator; BE, or B8, B9, BA, BB, BC, BD, or BF (memory or register B, C, D, E, H, L, or A).

The specified byte is compared to the contents of the accumulator by internally subtracting the contents of the register from the accumulator (leaving both unchanged), and setting the condition bits based on the result. For example, the zero bit is set if the quantities are equal and is reset if unequal. Since a subtraction operation is performed, the carry bit will be set if there is no carry out of bit 7, indicating that the contents of the register is greater than the contents of the accumulator, and reset otherwise. (If the two quantities differ in sign, the sense of the carry bit is reversed.) All condition bits can be affected by the instruction.

CNC Call if no carry; D4 XY.

This instruction examines the carry bit and if the carry bit is a zero, a **CALL** operation to the location YX is performed. No condition bits are affected.

CNZ Call if no zero; C4 XY.

This instruction examines the zero bit and if the bit is zero a **CALL** operation is performed to location YX. No condition bits are affected.

CP Call if plus; F4 XY.

This instruction examines the sign bit and if the bit is zero (indicating a positive result), a **CALL** operation to location YX is performed. No condition bits are affected.

CPE Call if parity even; EC XY.

This instruction examines the parity bit and if the bit is one (indicating even parity), a **CALL** operation to location YX is performed. No condition bits are affected.

CPI Compare immediate data with accumulator; FE.

The byte immediately following the instruction is compared to the contents of the accumulator by subtracting the byte from the accumulator by using two's complement arithmetic. The comparison leaves the accumulator unchanged but sets condition bits, depending on the result. A zero bit is set if the quantities are equal, or a carry bit is set if there is no carry of bit 7, indicating that the immediate data are greater than the contents of the accumulator, and reset otherwise.

CPO Call if parity odd; E4 XY.

This instruction examines the parity bit and if the bit is zero (indicating odd parity), a **CALL** operation to location YX is performed. No condition bits are affected.

CZ Call if zero; CC XY.

This instruction examines the zero bit and if the bit is set, a **CALL** operation to location YX is performed. No condition bits are affected.

DAA Decimal adjust accumulator; 27.

This instruction adjusts the number in the accumulator to form two 4-bit BCD digits in two steps:

1. If the least-significant four bits of the accumulator represent a number greater than nine, or if the auxiliary carry bit is equal to one, the accumulator is incremented by six, otherwise no incrementing occurs.
2. If the four MSB's of the accumulator now represent a number greater than nine, or if the normal carry bit is equal to one, the MSB's are incremented by six, otherwise no incrementing occurs.

If a carry out of the four LSB's occurs during step 1, the auxiliary carry bit is set, otherwise it is reset. Likewise, if a carry out of the four MSB's occurs during step 2, the normal carry bit is set, otherwise it is unchanged. (This instruction is used when adding decimal numbers. It is the only instruction whose operation affects the auxiliary carry bit.) All condition bits can be altered by this instruction.

DAD Double add; 09, 19, 29, or 39 (for the B, D, H, or SP register pairs, respectively).

A 16-bit number held in the specified register pair is added to the 16-bit number held in the H and L registers using two's complement arithmetic. The result replaces the contents of the H and L registers. Only the carry bit can be affected by the operation.

DCR M or **DCR r** Decrement memory or register; 35 or 05, 0D, 15, 1D, 25, 2D, or 3D (for memory or registers B, C, D, E, H, L, or A, respectively).

This instruction increments the contents of the specified memory location or register by one. The zero, sign, parity, and auxiliary carry bits can be affected by the instruction.

DCX r Decrement register pair; 0B, 1B, 2B, 3B (for pairs B and C, D and E, H and L, or the stack pointer, respectively).

This instruction causes the 16-bit number held in the selected register pair to be decremented by one. No condition bits are affected by the instruction.

DI Disable interrupts; F3.

This instruction resets the interrupt-enable flip-flop, causing the processor to ignore any interrupts. The interrupt system is disabled when the processor recognizes an interrupt as well as after a DI instruction. No condition bits are affected.

EI Enable interrupts; FB.

This instruction sets the interrupt-enable flip-flop, thus permitting the processor to recognize and respond to interrupts. The interrupt is delayed one instruction to allow interrupt subroutines to return to the main program before a subsequent interrupt is acknowledged. No condition bits are affected.

HLT Halt; 76.

This instruction causes the PC to increment to the address of the next sequential instruction and then the processor enters a stopped state; no further activity takes place until an interrupt occurs. If a **HLT** instruction is executed while interrupts are disabled, the only way to restart the processor is to apply a Reset sequel.

IN Input; DB Z (Z is an 8-bit number representing the port address).

This instruction tells the processor to access port number Z, read an 8-bit data byte from the port, and load the byte into the accumulator. No condition bits are affected.

INR M or **INR r** Increment memory or register; 34 or 04, 0C, 14, 1C, 24, 2C, or 3C (for memory or register B, C, D, E, H, L, or A respectively).

This instruction increments the specified memory location or register by one. All condition bits except the carry bit can be affected by the instruction.

INX Increment register pair; 03, 13, 23, or 33 (for the B and C, D and E, H and L registers, or the SP, respectively).

This instruction increments by one the 16-bit number held in the specified register pair. No condition bits are affected.

JC Jump if carry; DA XY.

This instruction examines the carry bit and if the bit is one, program execution continues at memory address YX. No conditions bits are affected. This instruction is similar to the **CALL** instruction except that no return address is stored in the stack. (The **CALL** instruction puts the value of the PC in the stack before going to a subroutine.)

JM Jump if minus; FA XY.

This instruction examines the sign bit and if the bit is one (indicating a negative result), program execution continues at address YX. No condition bits are affected.

JMP Jump; C3 XY.

This instruction forces the processor to unconditionally continue its processing at memory location YX. No condition bits are affected.

JNC Jump if no carry; D2 XY.

This instruction examines the carry bit and if the bit is zero (indicating no carry), program execution continues at memory location YX. No condition bits are affected.

JNZ Jump if no zero; C2 XY.

This instruction examines the zero bit and if the bit is zero, program execution continues at memory location YX. No condition bits are affected.

JP Jump if positive; F2 XY.

This instruction examines the sign bit and if the bit is zero (indicating a positive or zero result), program execution continues at memory location YX. No condition bits are affected.

JPE Jump if parity even; EA XY.

This instruction examines the parity bit and if the bit is one (indicating a result with even parity), program execution continues at memory location YX. No condition bits are affected.

JPO Jump if parity odd; E2 XY.

This instruction examines the parity bit and if the bit is zero (indicating a result with odd parity), program execution continues at memory location YX. No condition bits are affected.

JZ Jump if zero; CA XY.

This instruction examines the zero bit and if the bit is one, program execution continues at memory address YX. No condition bits are affected.

LDA Load accumulator direct; 3A XY.

The byte of data at location YX is moved into the accumulator, replacing whatever was in there. No condition bits are affected.

LDAX B or **LDAX D** Load accumulator; 0A or 1A.

This instruction puts the byte addressed by the B and C or D and E registers into the accumulator, replacing whatever was in there. No condition bits are affected.

LHLD Load H and L registers direct; 2A XY.

This instruction accesses location YX and places the byte stored there into the L register. Then the location address is automatically incremented by one to YX + 1 and the contents of that location are placed in the H register. No condition bits are affected.

LXIB, LXID, LXIH, or **LXI SP** Load register pair with immediate data; 01, 11, 21, or 31 (for the B and C, D and E, H and L pairs, or the SP registers, respectively).

These instructions consist of three bytes, the first of which is the actual instruction and the next two are data. When performed, these instructions cause the specified register pair to be loaded with the data contained in the second and third bytes. The second byte of the instruction is loaded into the second register or 8 LSB's of the pair specified, while the third byte is loaded into the first register or 8 MSB's of the pair specified. No condition bits are affected.

MOV M, r or **MOV r, M** or **MOV r, r** Move byte from to; for memory to register: 70, 71, 72, 73, 74, 75, and 77 (for destination registers B, C, D, E, H, L or A, respectively); for register to memory: 46, 4E, 56, 5E, 66, 6E, or 7E (for registers B, C, D, E, H, L or A, respectively); or for register to register: 40, 41, 42, 43, 44, 45, 47 (B to B, C, D, E, H, L, or A), or 48, 49, 4A, 4B, 4C, 4D, 4F (for C to B, C, D, E, H, L, or A), or 50, 51, 52, 53, 54, 55, 57 (for D to B, C, D, E, H, L, or A), or 58, 59, 5A, 5B, 5C, 5D, 5F (for E to B, C, D, E, H, L, or A), or 60, 61, 62, 63, 64, 65, 67 (for H to B, C, D, E, H, L, or A), or 68, 69, 6A, 6B, 6C, 6D, 6F (for L to B, C, D, E, H, L, or A) and 78, 79, 7A, 7B, 7C, 7D, or 7F (for A to B, C, D, E, H, L, or A respectively). Total of 63 possible instruction codes.

This instruction moves one byte of data to the first specified location called the destination register (M or r) from the second specified location called the source register. The byte transferred replaces the contents of the destination register but the source register is left unchanged. The only invalid transfer is a memory to memory transfer; instruction code 76 is not permitted (it is used for a **HALT** instruction). When a memory reference move is requested the memory address is specified by the contents of the H and L registers; the L register holds the 8 LSB's and the H register holds the 8 MSB's.

MVI M or **r** Move immediate data, 36 or 06, 0E, 16, 1E, 26, 2E, or 3E (for memory or registers B, C, D, E, H, L, or A, respectively).

This instruction takes the byte immediately following the instruction and stores it in the memory location or register specified. No condition bits are affected.

NOP No operation; 00.

This instruction performs no operation; execution proceeds with the next sequential instruction. It is useful for generating time delays. No condition bits are affected.

ORA M or **ORA r** Logic OR memory or register with accumulator; B6 or B0, B1, B2, B3, B4, B5, or B7 (for memory or registers B, C, D, E, H, L, or A, respectively).

This instruction performs a logic OR operation between the byte specified by M or r and the contents of the accumulator. The carry bit and auxiliary carry bit are reset to zero. All condition bits can be affected.

ORI OR immediate data with accumulator; F6.

This instruction performs a logic OR operation between the byte of data immediately following the instruction and the contents of the accumulator, storing the result in the accumulator. All condition bits can be affected but the carry bit and auxiliary carry bit are reset to zero.

OUT Output; D3 Z.

This instruction transfers the contents of the accumulator to the output device connected to the port number specified in byte Z following the instruction. No condition bits are affected. This is the opposite of the **IN** instruction.

PCHL Load program counter from H and L; E9.

This instruction transfers the 16 bits of information held in the H and L registers into the PC,

replacing whatever was in the PC. The contents of the H and L registers are unchanged. No condition bits are affected.

POP B or **POP D** or **POP H** or **POP PSW** Pop data off stack; C1, D1, E1, or F1 (for registers B and C, D and E, H and L, or A and PSW, respectively).

These instructions remove two bytes of data from the stack and load them into the specified register pair. The byte of data at the memory address indicated by the SP is loaded into the second register of the pair and the byte of data at SP + 1 is loaded into the first register of the pair. After data have been restored, the SP is incremented by two. If the **POP PSW** instruction is used, all condition bits are affected, otherwise no condition bits are affected.

PUSH B or **PUSH D** or **PUSH H** or **PUSH PSW** Push data onto stack; C5, D5, E5, or F5 (for registers B and C, D and E, H and L, or A and PSW, respectively).

These instructions transfer data from the selected register pair onto the stack. The byte of data from the first register of the pair is saved at memory location SP − 1 (one less than the address indicated by the SP) and the other byte of data is saved at memory location SP − 2. When the PSW byte is saved there are three other bits along with the condition bits that are saved; they are bits 5, 3, and 1 of the word and are 0, 0, and 1 respectively. In any case, after data have been saved, the SP is decremented by two. No condition bits are affected.

RAL Rotate accumulator left through carry; 17.

This instruction shifts the contents of the accumulator one bit position to the left with the higher order accumulator bit replacing the carry bit and the carry bit replacing the low order bit (end around carry). The carry bit is the only condition bit affected.

RAR Rotate accumulator right through carry; 1F.

This instruction shifts the contents of the accumulator one bit position to the right, with the carry bit shifted into the MSB of the accumulator and the LSB of the accumulator shifted into the carry bit. The carry bit is the only condition bit affected.

RC Return if carry; D8

This instruction examines the carry bit, and if the bit is one, a return operation is performed. No condition bits are affected.

RET Return; C9.

This instruction pops two bytes of data off the stack and places them into the PC register. Execution of the program resumes at the new address. It generally causes program execution to proceed with the instruction immediately following the most recent **CALL** instruction. The **RET** command is an unconditional instruction and brings the subroutine being executed to an end. No condition bits are affected.

RLC Rotate accumulator left; 07.

This instruction shifts the contents of the accumulator one bit position to the left, with the carry bit set equal to the MSB of the accumulator before the shift and the MSB then shifted into the LSB position. The carry bit is the only condition bit affected.

RM Return if minus; F8.

This instruction examines the sign bit and if the bit is one (indicating a minus result), a return operation is performed. No condition bits are affected.

RNC Return if no carry; D0.

This instruction examines the carry bit and if the bit is zero, a return operation is performed. No condition bits are affected.

RNZ Return if not zero; C0.

This instruction examines the zero bit and if the bit is zero, a return operation is performed. No condition bits are affected.

RP Return if plus; F0.

This instruction examines the sign bit and if the bit is zero (indicating a positive result), a return operation is performed. No condition bits are affected.

RPE Return if parity even; E8.

This instruction examines the parity bit and if the bit is one (indicating even parity), a **RET** operation is performed. No condition bits are affected.

RPO Return if parity odd; E0.

This instruction examines the parity bit and if the bit is zero (indicating odd parity), a **RET** operation is performed. No condition bits are affected.

RRC Rotate accumulator right; 0F.

This instruction shifts the contents of the accumulator one bit position to the right and before the shift sets the carry bit equal to the value of the accumulator's LSB. When shifted, the accumulator's LSB is transferred into the MSB position. The carry bit is the only condition bit affected.

RST Restart; C7, CF, D7, DF, E7, EF, F7, or FF.

This instruction is normally used in conjunction with up to eight 8-byte routines in the lowest 64 bytes of memory in order to service interrupts to the processor. Each **RST** instruction causes the processor to go to a different starting location: 0000, 0008, 0010, 0018, 0020, 0028, 0030, or 0038. When invoked, this instruction forces the contents of the PC onto the stack, providing a return address for later use by a **RET** instruction. No condition bits are affected.

RZ Return if zero; C8.

This instruction examines the zero bit and if the bit is one, a **RET** operation is performed. No condition bits are affected.

SBB M or **SBB r** Subtract memory or register from accumulator with borrow; 9E or 98, 99, 9A, 9B, 9C, 9D, or 9F (for memory or registers B, C, D, E, H, L, or A, respectively).

The contents of the memory location specified by the HL register pair or the B, C, D, E, H, L, or or A register and the contents of the carry bit are both subtracted from the accumulator and the result is left in the accumulator. All condition bits can be affected by this instruction.

SBI Subtract immediate data from accumulator with borrow; DE Z.

The contents of the second byte of the instruction, Z, and the contents of the carry bit are both subtracted from the accumulator. The result is kept in the accumulator. All condition bits can be affected.

SHLD Store H and L direct; 22 XY.

This instruction stores the contents of register L in the location specified by YX and the contents of the H register are stored in the next higher location (YX + 1). No condition bits are affected.

SPHL Load SP from H and L; F9.

This instruction transfers the contents of the H and L registers into the stack pointer, leaving the contents of the H and L registers unchanged. No condition bits are affected.

STA Store accumulator direct; 32 XY.

This instruction transfers the contents of the accumulator to the memory location specified by bytes YX. No condition bits are affected.

STAX B or **STAX D** Store accumulator; 02 or 12.

This instruction stores the contents of the accumulator in the memory location specified by either the B and C register pair or the D and E register pair. No condition bits are affected.

STC Set carry; 37.

This instruction sets the carry bit to one. Only the carry bit is affected.

SUB M or **SUB r** Subtract memory or register from accumulator; 96 or 90, 91, 92, 93, 94, 95, or 97 (for memory or registers B, C, D, E, H, L, or A, respectively).

This instruction specifies a byte in memory or in a specific register that is to be subtracted from the accumulator by use of two's complement arithmetic. If there is no carry out of the MSB of the accumulator, indicating that a borrow occurred, the carry bit is set; otherwise it is reset. All condition bits can be affected.

SUI Subtract immediate data from accumulator; D6 Z.

This instruction subtracts the second instruction byte, Z, from the contents of the accumulator using two's complement arithmetic. Because this is a subtraction operation, the carry bit sets to indicate a borrow if there is no carry out of the MSB or the accumulator. If there is a carry out, the carry bit is reset. All condition bits can be affected.

XCHG Exchange registers; EB.

This instruction exchanges the data held in the H and L registers with the data stored in the D and E registers. No condition bits are affected.

XRA M or **XRA r** Logic Exclusive-OR memory or register with accumulator (zero accumulator); AE or A8, A9, AA, AB, AC, AD, or AF (for memory or registers B, C, D, E, H, L, or A, respectively).

This instruction performs an Exclusive-OR operation between the specified memory location or register and the accumulator. The carry and auxiliary carry bits are reset to zero. The result of the Exclusive-OR operation is left in the accumulator. All condition bits can be affected.

XRI Exclusive-OR immediate data with accumulator; EE Z.

This instruction performs and Exclusive-OR operation between the immediate data byte, Z, and the contents of the accumulator. The carry bit is set to zero. All condition bits can be affected.

XTHL Exchange stack; E3.

This instruction transfers the contents of the L register to the memory location whose address is held in the stack pointer and transfers what was in the memory location to the L register. And, the contents of the H register are swapped with the byte whose address is one greater than the address held in the stack pointer. No condition bits are affected.

Write Your Programs Systematically

To ease the development of programs, the first thing to do is to set up a procedure to write down all the instructions, and note all the branch points. First, each instruction can be set up so that it is spread apart into four parts called fields.

Field 1 is called a label field and is used to indicate all addresses for the instructions.

Field 2 is called the code field and is used to show the actual code for the instruction that is to be performed.

Field 3 is the operand field and it shows any immediate data or address information needed by the instructions in the code field.

Field 4 is a comment area to permit you to make notes for yourself so that you can easily see what the program is doing.

Label	OP Code	Operand	Mnemonic Code	Comments

Fig. 8.8 If programs are converted to object code by hand, the best method to use is a large chart that has columns for all the intermediate steps and comments to clarify the program flow.

Address Field Label	Op Code	Operand	Mnemonic Code	Comments
0000	3E	03	MVI A,Z	Move next byte into A
0002	D3	10	OUT Z	Output A to port 10
0004	3E	11	MVI A,Z	Move next byte into A
0006	D3	10	OUT Z	Output A to port 10
0008	DB	10 Loop	IN Z	Input data from port 10 to A
000A	0F		RRC	Rotate A right one bit
000B	D2	08 00	JNC XY	Jump to XY if carry bit = 0
000E	DB	11	IN Z	Input data from port 11 to A
0010	D3	11	OUT Z	Output data to port 11 from A
0012	C3	08 00	JMP XY	Jump unconditionally to XY

Fig. 8.9 This simple program permits a terminal to transmit a character to the Altair computer and then has the computer echo back the same character to the terminal.

Address Field Label	Op Code	Operand	Mnemonic Code	Comments
0000	3E	CA	MVI A,Z	Move mode byte into A
0002	D3	03	OUT Z	Output A to port 03
0004	3E	27	MVI A,Z	Move command byte into A
0006	D3	03	OUT Z	Output A to port 03
0008	DB	03 Loop	IN Z	Input byte to A from port 03
000A	E6	02	ANI Z	AND immediate to A to mask all but receive bit
000C	CA	08 00	JZ XY	Jump back to XY if A = 0
000F	DB	02	IN Z	Input data from port 02 to A
0011	D3	02	OUT Z	Output data to port 02 from A
0013	C3	08 00	JMP XY	Jump unconditionally to XY

Fig. 8.10 Performing a similar echo procedure for the Imsai 8080 computer, this simple program requires about the same number of commands.

Thus, when you start to write a program, make up a table, much like that shown in Fig. 8.8. Jump or branch instructions are a very important part of many computer programs since they are capable of altering the sequential program flow. As listed earlier, the 8080A has many jump, return, and call instructions that permit it to examine specific conditions and go to a subroutine just for those conditions. These instructions control many computer operations. As a simple example, let's look at a program that lets you type a character on a terminal, accepts the character, and then sends back the same character to the terminal so it prints it (Fig. 8.9). This routine is called an echo procedure and can be used to test the terminal interface since it is short and can easily be entered via the front panel switches of the computer.

The first instruction loads the number 03 into the accumulator; this is part of the set-up information needed by an ACIA used on a serial interface board by Pertec. The next instruction outputs the 03 to port 10 to actually load the information into the ACIA. The third instruction again loads a number into the accumulator and, again, this number 11 is preliminary set-up information for the ACIA. Now, the computer and serial port are ready to handle data so the next instruction tells the computer to treat the information on the bus from port 10 as an input and load the information into the accumulator. The next instruction rotates the contents right one bit to shift the LSB into the carry bit so it can be checked by the next instruction. If there is no carry the program goes back to the input instruction and loops through the same three instructions until a carry occurs. When a carry finally does occur the program goes to the next instruction which says input data from port 11 into the accumulator. After the data are in the accumulator, they are then output with the next instruction back on port 11, thus echoing the entered character. The final instruction sends the computer back to the waiting loop to detect an entered character.

However, that program is not the only way to echo a character. Imsai's SIO-2 card uses the 8251 and requires different program set-up information (Fig. 8.10). And, instead of rotating the accumulator, the program uses a "mask" word (a mask is a bit pattern designed to cover up or eliminate all bits in a word that are not desired) to check the contents of the accumulator. Depending upon the serial interface circuit, either of the programs will do the job. But this simple program can't do anything more than just echo the characters back—it doesn't even store them in memory. And, even if you

Table 8.3 ASCII Codes

Character	ASCII Binary Code	ASCII Hex Code	Character	ASCII Binary Code	ASCII Hex Code
A	1000001	41	w	1110111	77
B	1000010	42	x	1111000	78
C	1000011	43	y	1111001	79
D	1000100	44	z	1111010	7A
E	1000101	45	0	0110000	30
F	1000110	46	1	0110001	31
G	1000111	47	2	0110010	32
H	1001000	48	3	0110011	33
I	1001001	49	4	0110100	34
J	1001010	4A	5	0110101	35
K	1001011	4B	6	0110110	36
L	1001100	4C	7	0110111	37
M	1001101	4D	8	0111000	38
N	1001110	4E	9	0111001	39
O	1001111	4F	¢	0100000	20
P	1010000	50	!	0100001	21
Q	1010001	51	"	0100010	22
R	1010010	52	#	0100011	23
S	1010011	53	$	0100100	24
T	1010100	54	%	0100101	25
U	1010101	55	&	0100110	26
V	1010110	56	'	0100111	27
W	1010111	57	(	0101000	28
X	1011000	58	)	0101001	29
Y	1011001	59	•	0101010	2A
Z	1011010	5A	+	0101011	2B
a	1100001	61	,	0101100	2C
b	1100010	62	—	0101101	2D
c	1100011	63	.	0101110	2E
d	1100100	64	/	0101111	2F
e	1100101	65	:	0111010	3A
f	1100110	66	;	0111011	3B
g	1100111	67	<	0111100	3C
h	1101000	68	=	0111101	3D
i	1101001	69	>	0111110	3E
j	1101010	6A	?	0111111	3F
k	1101011	6B	@	1000000	40
l	1101100	6C	[	1011011	5B
m	1101101	6D	\	1011100	5C
n	1101110	6E	]	1011101	5D
o	1101111	6F	^	1011110	5E
p	1110000	70	_	1011111	5F
q	1110001	71	`	1100000	60
r	1110010	72	{	1111011	7B
s	1110011	73	|	1111100	7C
t	1110100	74	}	1111101	7D
u	1110101	75	~	1111110	7E
v	1110110	76	≡	1111111	7F

did have the characters entered in memory, they would be in ASCII form and would first have to be converted back to pure binary in order to run in the computer. Let's take a look at how some of the conversion is performed for just the hex numbers.

To start with, look at the ASCII codes that represent the numbers and letters 0 to F (Table 8.3). The numbers follow an increasing binary code pattern as do the letters. However the pattern is different between the numbers and letters, and since the final result must

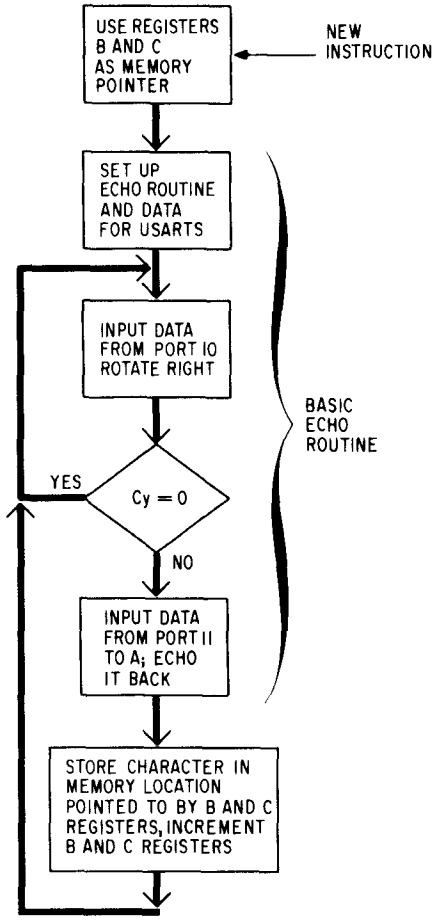


Fig. 8.11 Modifying the echo routine to store characters that are entered requires that an extra few steps be added to keep track of the memory location and to initialize the pointer to a predetermined value.

be a 4-bit code, a different check routine must be used to determine if the input character is a number. The flow chart must start with the basic echo routine with one extra step to set up a register pointer to indicate where in memory the characters are stored (Fig. 8.11). Now the job becomes more complex.

To start with, the data in the accumulator must be temporarily stored so that several tests can be performed on the data without losing the data. So let's store the data in the memory location addressed by registers B and C. Now, even though the value was stored, it is still also in the accumulator and can be used. The next step, then, is to determine whether the data represent a number between 0 and 9 or a letter between A and F (Fig. 8.12). Start this process by comparing the 8-bit number in the accumulator with the lowest value you're concerned with, zero (in ASCII the value is 30). Then, if the value in the accumulator is smaller than 30 the program should go back and wait for another character. If the result of the compare operation is zero or positive, the program then must determine if the character is 0 to 9 or A to F.

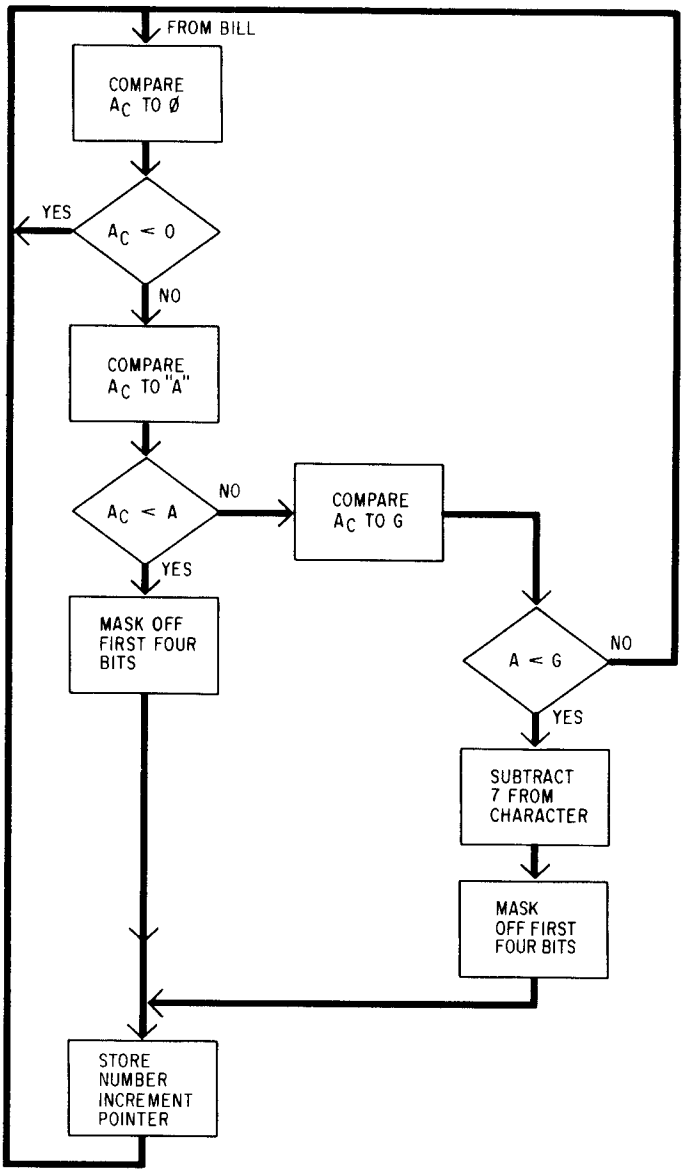


Fig. 8.12 Although the task of determining the value of the number held in the accumulator can be done in different ways, this flowchart suggests one possible direction the program flow can take.

The next step is to compare the value of the accumulator with the ASCII nine (39). If the value of the accumulator is smaller than 39 the contents represent a number between 0 and 9 and the program goes to a subroutine to strip away the ASCII code and store the number in a memory location. If the accumulator is greater than zero the program checks to see if the code is at least that of the letter A (41). If the value of the accumulator is less than the value of A the character isn't one that's being looked for, so the program goes back to wait for another character. When the value of the compare result is equal to or greater than A, the next step is to make sure the ASCII character is only an A, B, C, D, E, or F. So next the program checks for a G or higher. If the value is that of a G or higher, the program again goes to the wait portion to accept the next character entered. When the program recognizes the characters A to F, it also goes to another routine to determine exactly which letter it is.

Next, of course, is the procedure to strip away the ASCII code so all that's left is the 4-bit hex representation of 0 to F. For a number the stripping is easy—the last four bits of the code represent the actual number so all that has to be done is to mask out (eliminate) whatever is in the first four bits. To do the masking an AND immediate instruction with a data byte containing 0F can eliminate the first four accumulator bits and leave the last four bits unaffected. The data byte in the accumulator is then stored in a memory location, whose address is stored in the B and C registers.

Determining the correct conversion for the alphabetic character is more complex. Each character can be compared in a six-step process. First, since the code is in the accumulator it can be compared using a CPI instruction with ASCII for A to F and if a match exits the program can branch. So, the first step could be a CPI command, and then a JZ command to do the branch, bringing the program to the correct conversion step. The code for F does not have to be used since after E it is the only other possibility. So for F, all the program has to do is load the hex code for F into the accumulator and then go to the final subroutine to store the accumulator in memory, increment the memory pointer register, and go back to get another character.

The program that accomplishes all that is shown in Fig. 8.13 and is listed in assembly code in Fig. 8.14. As you can see, it's fairly complex and if you didn't have the comments or the mnemonics to use, it would be almost impossible to understand, as illustrated by the assembled code version of the same program in Fig. 8.15.

Organize Your Work Carefully When Writing Programs

As pointed out earlier, organizing your notes before writing a program can save much time later on when trying to figure out why it doesn't work. To start program organization, the initial attack is to, of course, define what you want the program to do, and possibly divide it into smaller routines that work on their own and then just link the smaller programs together, either by using them like subroutines or by melding the addresses so they run consecutively. Next, after the program has been defined, the program should be flowcharted so that all steps are clearly illustrated.

After the flowchart is drawn, the program must actually be written. To help keep track of the actual program steps, special forms made just for programming provide you with the necessary columns and comment spaces. For instance, Fig. 8.8 shows just such a form developed by Walton Electronics with most of the

```
        LXI B, 00 10     Set contents of B and C register to 1000
        MVI A,Z  )
        OUT Z    )
        MVI A,Z  )
        OUT Z    )       See Fig. 8.9 for explanation
Loop    IN Z     )
        RRC      )
        JNC Loop )
        IN Z             Input keyboard data into accumulator
        CPI 30           Compare accumulator with ASCII code for 0
        JC Loop          If accumulator is less jump back to Loop
        CPI 40           Otherwise compare with ASCII @
        JC Num           If less go to Num subroutine
        CPI 41           Otherwise compare with ASCII A
        JC Loop          If less, go back to Loop
        CPI 46           Otherwise compare with ASCII F
        JNC Loop         If greater than F go back to Loop
        OUT Z            Otherwise output character
        SUI F9           Subtract complement of ASCII 7 from accumulator
Num     ANI 0F           AND contents of accumulator with 0F mask
        STAX B           Store contents of accumulator at address B
        INX B            Increment address B
        JMP Loop         Go back and get another character
```

Fig. 8.13 This is most of a program that will accept ASCII characters from a terminal, strip them to their 4-bit hex values, pack them two to a byte, and then store them in the memory.

Location	Contents													
0000	01	00	10	3E	03	D3	10	3E	11	D3	10	DB	10	0F
0010	D2	0B	00	DB	11	FE	30	DA	0B	00	FF	40	DA	2D
0020	00	FE	41	DA	0B	00	FE	46	D2	0B	00	D3	11	D6
0030	F9	E6	0F	02	03	C3	0B	00						

Fig. 8.14 Machine code listing of the assembly program in Fig. 8.13.

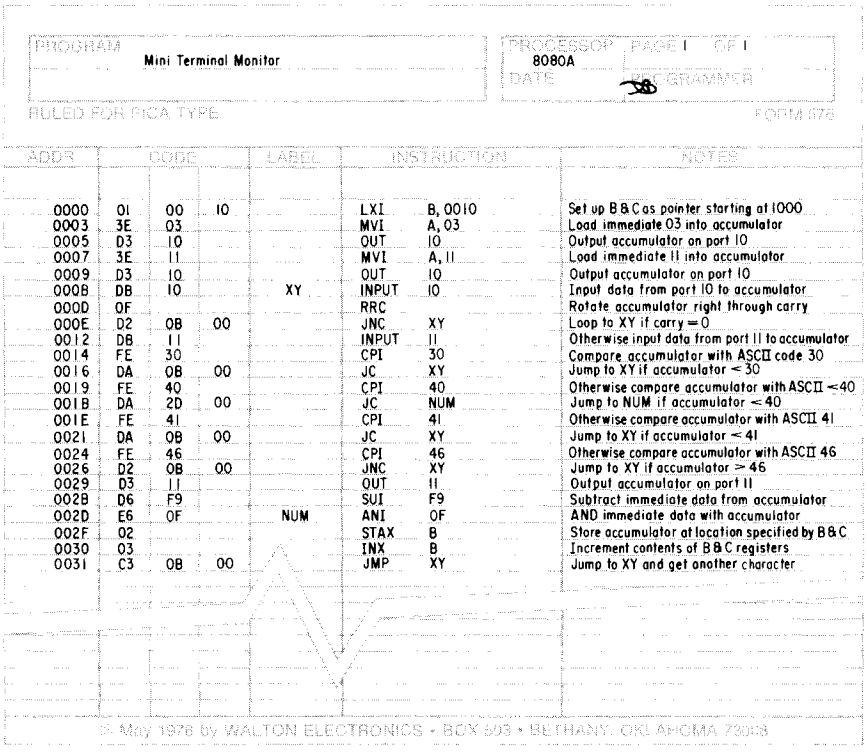


Fig. 8.15 By using specially prepared paper coding forms, such as this one from Walton Electronics, hand assembly and conversion of programs into object code form is not very difficult.

information filled in. This makes it very easy to understand what the program is doing every step of the way. After the program is written in assembly language it must be converted into machine language so it can be stored in the computer's memory and perform its job. To do the conversion you must either assemble the program yourself, sticking in the appropriate numbers for the addresses whenever the program branches, jumps, or returns, or use a special program called an assembler that takes the mnemonics as they are entered via a keyboard, checks for errors, and translates the mnemonics into the instruction codes and inserts the proper addresses.

Assemblers, depending on their complexity, permit you to not only type in the instructions in mnemonic form but to just give one command and get back a coded version of the program ready to run on the computer. Other programs called system monitors were mentioned earlier, but these are used in conjunction with the assembler. The monitor tells the computer how to "talk" to the terminal and any other devices, such as mass memories or a printer, and tells the computer how to store and access the various instructions.

Thus, to handle the input from a serial device such as a CRT terminal, the computer must really have some sort of monitor program. With the basic system from Imsai comes a combination monitor, assembler, and text-editor program originally developed by a company called Microtec. The text-editor portion of the program permits the user to set aside space in the memory for files. And, each line of program statements is prefaced by a fixed line number reference that permits simple search and reference capability. Up to 10,000 lines can exist in each file, assuming the memory is large enough. As lines are typed on the terminal they are entered into the designated file area and assigned a line number. The user can initialize the line sequence by entering a four digit decimal number. An entry to the editor is terminated by hitting the carriage return; no more than 80 characters can be typed on a single line.

CONTROL-X	Kill current line
ENTR	Enter data to memory
DUMP	Display memory data
FILE	Create, assign, or display file information
EXEC	Execute a program
ASSM	Assemble a source file to object code
LIST	List file
DELT	Delete lines of file
1111	Any four numeric digits enters Editor program
PAGE	Move a page of data
BREK	Set or clear breakpoints
PROC	Proceed from breakpoint
CVST	Optional user-defined control at location 2000

Fig. 8.16 The monitor and executive program provided with the Imsai 8080 computer system includes these commands to help develop assembly language programs.

The overall software package requires about 6 kwords of memory, so one 8 k RAM card can be used and it will provide about 2 kwords of program area. Part of the program interacts with the I/O ports, which are set up to respond as follows:

Port	Function
2	TTY Data
3	TTY Status
	Bit 0 indicates TBE
	Bit 1 indicates DAV
FF	Sense-switch input
	Address/program input
	Switch seven controls the file listing

Use Software from Others to Write Your Own

The software in the package from Imsai lets you use the computer system to develop your own software. Doing all the control is the monitor and executive portion of the package. The monitor controls the I/O devices while the executive provides commands to perform different programming operations (Fig. 8.16). To initialize the system, start it at 0000; to restart without initializing it, start at 0003. There is also one error message included—WHAT?—that indicates an improper command or parameter error on the command.

The assembler portion of the software translates the mnemonic assembly code into machine code for the computer. It operates in two passes (it has to be run twice)—once to allocate all storage necessary for the final program and to define the value of all symbols used for program loops, constants, etc., by creating a symbol table for lookup. The second pass evaluates all expressions, symbols, and ASCII characters, gives them absolute values, and places them in the allocated memory. A listing is also produced to indicate exactly what information is in each location.

Features of the assembler include the following:

- Free format source input
- Symbolic addressing, including forward references and relative symbolic references
- Complex expressions may be used as arguments
- Self-defining constants
- Multiple constant forms
- Up to 256 five-character symbols
- Reserved names for 8080 registers
- ASCII character code generation
- Six special operations (assembler directives) referred to as pseudo operations

Translating the lines of code contained in the current file into machine code, the assembler uses the second character following the line number to be the first source (assembly language) code character position. Thus, the character immediately following the line

```
0015    FLOP        MOV M,B      COMMENT
0020    *COMMENT
0025                JMP BEG
0030                CALL FLOP
0035    BEG         ADI 8+6-4
0040                MOV A,B
```

Fig. 8.17 A short listing using the Editor part of the Imsai software.

should normally be a space. Line numbers are not processed by the assembler; they are just duplicated on the listing.

Statements may be assembly language commands and symbolic locations or pseudo operations (pseudo operations direct the assembler to perform functions that do not always result in a translation of assembly code into machine code). Each statement is broken into fields, much as before when using pad and pencil. There are four fields:

Name Operation Operand Comment

The Name field begins in character position one (one space after the line number listing). The symbol used in the field can contain any number of characters, but only the first five are used in a symbol table to uniquely define the symbol. All symbols used in the files must begin with an alphabetic character and may contain no special characters. The Operation field contains either an 8080 mnemonic or a system pseudo operation name. The Operand field contains parameters pertaining to the operation listed in the Operation field. If two arguments are present, a comma must separate them. An example of a short listing is shown in Fig. 8.17. All fields are separated and distinguished from one another by the presence of one or more spaces. The Comment field provides space for explanatory remarks and is reproduced on the listing without being processed. Separate lines that begin with an asterisk are also comment lines.

To assign a symbolic name to a statement, such as FLOP, in the example of Fig. 8.17, just place the name in the Name field. If not, just skip a few spaces and you begin the Operation field. When a symbolic name is used, the assembler assigns it the value of the current location counter. The location counter always holds the address of the next byte to be assembled. And the only exception to this is the EQU pseudo op in which the symbol in the Name field is assigned a value in the Operand field. For instance:

```
0057    POTTS    EQU 128
```

assigns the value 128 to the name POTTS, which can then be used elsewhere in the program, such as in the instruction ADI POTTS.

In addition to the user-defined names, the assembler has some special reserved symbols that have predetermined uses (Table 8.4). These names may not be used except in the Operand field. In addition to the reserved symbols, there is a special character symbol $ that changes in value as the assembly progresses—it is always equated with the value of the program counter after the current instruction is assembled and it may only be used in the Operand field. For example, the program in Fig. 8.18 shows some use of the $ symbol.

Table 8.4 Reserved Assembler Symbols

A—Accumulator
B—Register B
C—Register C
D—Register D
E—Register E
H—Register H
L—Register L
M—Memory location addressed by H and L registers
P—Program status word
S—Stack pointer

If the name of a particular location is known, the assembler program permits a nearby location to be specified using the known name and a numeric offset (Fig. 8.19). In the example shown, the command JMP BEG refers to the MOV instruction while the command JPE BEG + 4 refers to the JNR B instruction. BEG + 4 means the address BEG plus four.

The assembler also allows positive or negative numbers to be written directly, instead of in two's complement form. The numbers will be regarded as decimal constants and their binary equivalent will be used by the program. All assigned numbers are considered positive. Decimal constants can be defined using the description D after the numeric value. However, this is not required except for clarity's sake if you're using octal or hex numbers too. Hex constants are defined

```
JMP $       Means jump to this location after this instruction;
MOV A,B       that is, the MOV instruction.
LDA $+5     Means load the data at the fifth location after this
DB 0          location. In this case, the data have the value
DB 1          of 5.
DB 2
DB 3
DB 4
DB 5
```

Fig. 8.18 Some uses of the $ symbol are shown in these program lines.

```
            JMP BEG
            JPE BEG+4
            CC SUB
            CALL $+48
BEG         MOV A,B
            HLT
            MVI C,'B'
            INR B
```

Fig. 8.19 Sample program showing the use of the JMP BEG command.

using the description H after a numeric value, for instance +10H, 10H, 3AH, 0F4H. A hex constant cannot start with the letters A to F. If needed, a leading zero must be included to enable the assembler to differentiate between a numeric value and a symbol. ASCII constants can be defined by surrounding the character within single quote marks ('C' for example). For double word constants, two characters can be defined with one quote string.

Expressions consist of sequences of one or more symbols, constants, or other statements separated by the arithmetic operators + or −. For example, PAM + 3, ISAB − 'A' + 52, or LOOP + 32H − 5 are expressions. All expressions are calculated using 16-bit arithmetic and all arithmetic is done modulo 65,536. Single byte data cannot contain a value greater than 255 or less than −256. Any value outside their range will result in an assembler error.

Pseudo Operations Add Flexibility to the Assembler

The pseudo operation statements help the assembler perform its job by supplying the programmer with some handy functions. There are six operations available to the user of the Imsai Assembler:

ORG (set program origin): This format for the pseudo is ORG expression.

END (end of assembly): This pseudo op informs the assembler that the last source statement has been read. The assembler will then start on pass 2, terminate the assembly, and then pass control back to the executive. This pseudo op is not needed when assembling from a memory file since the assembler will stop when an end of file indicator has been reached.

EQU (equate symbolic value): This pseudo op has a similar format to the ORG directive where the label is a symbol whose value will be determined from the expression. When evaluated, the expression, will be assigned to the symbol in the same field.

DS (define storage): The form for this pseudo op is also similar to ORG. This pseudo op causes the assembler to advance the assembly program counter, effectively skipping past a given number of memory bytes.

DB (define byte): Also similar in format to ORG, this pseudo op is used to reserve one byte of storage. The content of the byte is specified in the argument field.

DW (define word): This pseudo op defines two bytes of storage. The evaluated argument will be placed in the two bytes; high-order 8-bits in the low-order byte and the low-order 8-bits in the high-order byte, thus conforming to the format used by 8080A systems for storing two-byte addresses. When an error occurs during assembly, error indicators are output by the assembler (Table 8.5). Some of the error indicators are only output during pass 1.

Table 8.5 Error Indicators from the Assembler

O—Operation code error
L—Label error
D—Duplicate label error
M—Missing label error
V—Value error
U—Undefined symbol
S—Syntax error
R—Register error
A—Argument error

While the system has no explicit provisions to save and restore programs, it is possible to do so using an ASR style teletypewriter. To do so requires that the user:

1. Make the file to be saved the current file.
2. Type 'LIST' but don't type carriage return.
3. Turn on the paper-tape punch.
4. Type carriage return. (The program will be listed on the teletypewriter and simultaneously punched on the paper tape punch.)
5. When the 'LIST' is completed, turn off the punch.

To restore the file the procedure is simpler:

1. Make the file you want to restore into the current file.
2. Put the tape into the tape reader.
3. Start reader; the program will be automatically read in.

A similar procedure using the DUMP and ENTR commands may be used to save and restore machine code programs.

The program to do all the editing, assembling, and monitoring would require about twenty pages to print so it is not listed here. However, Imsai does have copies of the programs available for a small fee. As you can see, some sort of supervisory program is essential for simple machine operation. There are simpler programs that do nothing but monitor specific ports and let you enter programs. Let's take a close look at the program used by Vector Graphic in their PROM/RAM board, which was shown in Chapter 5. The program is just a monitor that lets you use a terminal that delivers serial ASCII and a cassette tape-recorder interface.

There are nine commands available in the monitor routine that are called by typing an ASCII character for each routine (G, D, P, T, R, W, V, L, or A). The procedure implemented by each command is as follows:

G XXXX: Go to location XXXX and start execution.

D XXXX YYYY: Display memory contents from location XXXX to YYYY.

P XXXX YYYY: Put information into memory at

location XXXX and ending at location YYYY.

T XXXX YYYY: Test memory starting at location XXXX and go to YYYY and output any errors.

R XXXX YYYY: Read cassette and load data starting at location XXXX and ending at location YYYY.

W XXXX YYYY: Write data onto cassette starting at location XXXX and ending at location YYYY.

V XXXX YYYY: Verify data on cassette by reading the tape and comparing the checksum with that recorded on the tape. No byte-by-byte comparison is made.

L XXXX YYYY: Load program into memory locations XXXX to YYYY and start execution at XXXX if no errors exist.

A XXXX YYYY: Dump ASCII characters from locations XXXX to YYYY.

The assembly code listing for the program is shown in Fig. 8.20. As with almost all monitor programs, the first command is a jump instruction that is executed either upon power on or when the RESET switch is pressed. So now the processor jumps to location C003 where it starts the initialization process and then goes to location C00B where the stack pointer is loaded with the immediate data 00D0 and then calls a subroutine CRLF at location C081 which is used to print a carriage return by first loading 0D (ASCII code for carriage return) into the accumulator and then jumping to another subroutine, PTCN at location C075, which pushes the contents of the accumulator onto the stack at the location loaded into the stack pointer register.

```
C000                    0010 CONC   EQU  0          CONSOLE STAT PORT
C000                    0020 COND   EQU  1          CONSOLE DATA PORT
C000                    0030 CASD   EQU  6FH        CASSETTE DATA PORT
C000                    0040 CASC   EQU  6EH        CASS STAT PORT
C000                    0050 SPTR   EQU  0D000H     STACK POINTER
C000                    0051 *
C000                    0052 *** VECTOR ONE MONITOR - VERSION 1.2(A)
C000                    0053 *FOR SIO REV. 1 AND 3P+S W. INV. STATUS
C000                    0054 ********* COMMAND FORMAT *********
C000                    0055 *G LLLL GO TO LOC LLLL AND EXEC
C000                    0056 *D SSSS FFFF DISPLAY MEMORY
C000                    0057 *P LLLL PROGRAM MEMORY
C000                    0058 *T SSSS FFFF TEST MEMORY
C000                    0059 *R SSSS FFFF READ CASSETTE
C000                    0060 *W SSSS FFFF WRITE CASSETTE
C000                    0061 *V SSSS FFFF VERIFY CASSETTE
C000                    0062 *L SSSS FFFF LOAD AND GO
C000                    0063 *A SSSS FFFF ASCII DUMP
C000                    0064 ********************************
C000                    0070 *
C000 C3 03 C0           0080        JMP  INIT
C003                    0090 INIT   DS   8
C00B 31 00 D0           0100 START  LXI  SP,SPTR
C00E CD 81 C0           0105        CALL CRLF
C011 3E 2A              0110        MVI  A,'*'      PRINT "*"
C013 CD 75 C0           0120        CALL PTCN
C016 CD 8B C0           0130        CALL RDCN       READ KEYBOARD
C019 F5                 0140        PUSH PSW        SAVE INPUT
C01A CD 73 C0           0150        CALL SPCE
C01D F1                 0160        POP  PSW        RESTORE ACC
C01E FE 47              0170        CPI  'G'        IF G
C020 CC 4E C0           0180        CZ   EXEC       EXECUTE A PROGRAM
C023 FE 56              0190        CPI  'V'        IF V,
C025 CC CB C0           0200        CZ   CINR       GOTO INPUT ROUTINE
C028 FE 57              0230        CPI  'W'        IF W
C02A CA 99 C0           0240        JZ   COUTR      GO TO CASS OUT
C02D FE 44              0250        CPI  'D'        IF D
C02F CC 8E C1           0260        CZ   DISP       GO TO MEM DISP
C032 FE 50              0270        CPI  'P'        IF P
C034 CC C6 C1           0280        CZ   PGM        GO TO PROG MEM
C037 FE 52              0290        CPI  'R'        IF R
C039 CC CB C0           0300        CZ   CINR       GOTO CASS IN
```

Fig. 8.20 Excerpt from the monitor program used by Vector Graphic on their PROM/RAM board.

```
C03C FE 4C      0310        CPI   'L'          IF L
C03E CC CB C0   0320        CZ    CINR         DO A LOAD AND GO
C041 FE 54      0330        CPI   'T'          IF T
C043 CC 19 C1   0340        CZ    TMEM         TEST MEMORY
C046 FE 41      0342        CPI   'A'          IF A
C048 CC 8E C1   0344        CZ    DISP         DUMP ASCII
C04B C3 0B C0   0350        JMP   START        START OVER
C04E            0360 *
C04E            0370 *** EXECUTE THE PROGRAM AT THE ADDRESS ***
C04E            0380 *
C04E CD 57 C0   0390 EXEC   CALL  AHEX         READ ADD FROM KB
C051 EB         0392        XCHG
C052 11 0B C0   0394        LXI   D,START
C055 D5         0396        PUSH  D
C056 E9         0400        PCHL  JUMP         TO IT
C057            0410 *
C057            0420 *** CONVERT UP TO 4 HEX DIGITS TO BIN
C057            0430 *
C057 21 00 00   0440 AHEX   LXI   H,0          GET 16 BIT ZERO
C05A 0E 04      0450        MVI   C,4          COUNT OF 4 DIGITS
C05C CD 8B C0   0460 AHEI   CALL  RDCN         READ A BYTE
C05F 29         0470        DAD   H            SHIFT 4 LEFT
C060 29         0480        DAD   H
C061 29         0490        DAD   H
C062 29         0500        DAD   H
C063 D6 30      0510        SUI   48           ASCII BIAS
C065 FE 0A      0520        CPI   10           DIGIT 0-10
C067 DA 6C C0   0530        JC    ALF
C06A D6 07      0540        SUI   7            ALPHA BIAS
C06C 85         0550 ALF    ADD   L
C06D 6F         0560        MOV   L,A
C06E 0D         0570        DCR   C            4 DIGITS?
C06F C2 5C C0   0580        JNZ   AHEI         KEEP READING
C072 EB         0585        XCHG
C073 3E 20      0590 SPCE   MVI   A,20H        PRINT SPACE
C075 F5         0600 PTCN   PUSH  PSW          SAVE REG A
C076 DB 00      0610 PTLOP  IN    CONC         READ PRTR STATUS
C078 E6 80      0620        ANI   80H          IF BIT 7 NOT 0,
C07A C2 76 C0   0630        JNZ   PTLOP        WAIT TILL TIS
C07D F1         0640        POP   PSW          THEN RECOVER A
C07E D3 01      0650        OUT   COND         AND PRINT IT
C080 C9         0660        RET   RETURN       FROM PTCN
C081 3E 0D      0670 CRLF   MVI   A,0DH        PRINT CR
C083 CD 75 C0   0680        CALL  PTCN
C086 3E 0A      0690        MVI   A,0AH
C088 C3 75 C0   0700        JMP   PTCN
C08B            0710 *
C08B            0720 *** READ FROM CONSOLE TO REG A ***
C08B            0730 *
C08B DB 00      0740 RDCN   IN    CONC         READ KB STATUS
C08D E6 01      0750        ANI   1            IF BIT 1 NOT 0
C08F C2 8B C0   0760        JNZ   RDCN         REPEAT UNTIL IT IS
C092 DB 01      0770        IN    COND         READ FROM KB
C094 E6 7F      0780        ANI   7FH          STRIP OFF MSB
C096 C3 75 C0   0790        JMP   PTCN         ECHO ONTO PRINTER
C099            0860 *
C099            0870 *** CASSETTE INTERFACE OUTPUT ROUTINE ***
C099            0880 *
C099 CD 57 C0   0890 COUTR  CALL  AHEX         READ BLOCK LENGTH
C09C CD 57 C0   0910        CALL  AHEX         READ STARTING ADD
C09F 06 00      0920        MVI   B,0          START CHECKSUM = 0
C0A1 CD BF C0   0930        CALL  COUT         START BYTE OUT
C0A4 3E E6      0940        MVI   A,0E6H       SEND SYNC BYTE
C0A6 CD BF C0   0950        CALL  COUT         TO CASSETTE
C0A9 7E         0960 COLOP  MOV   A,M          GET DATA FROM MEM
```

Fig. 8.20 (cont'd) Excerpt from the monitor program used by Vector Graphic on their PROM/RAM board.

```
C0AA CD BF C0    0970         CALL  COUT          SEND TO CASSETTE
C0AD 80          0980         ADD   B             ADD TO CHECKSUM
C0AE 47          0990         MOV   B,A
C0AF CD F5 C1    1000         CALL  BMP
C0B2 C2 A9 C0    1040         JNZ   COLOP         REPEAT LOOP
C0B5 78          1050         MOV   A,B           GET CHECKSUM
C0B6 CD BF C0    1060         CALL  COUT          OUTPUT IT
C0B9 CD 74 C1    1065         CALL  PT2           PRINT CHECKSUM
C0BC C3 0B C0    1070         JMP   START         GET ANOTH COMMND
C0BF F5          1080 COUT    PUSH  PSW           SAVE A AND FLAGS
C0C0 DB 6E       1090 CLOP    IN    CASC          READ CASS STATUS
C0C2 E6 20       1100         ANI   20H           LOOK AT BIT 5
C0C4 C2 C0 C0    1110         JNZ   CLOP          TRY AGAIN?
C0C7 F1          1120         POP   PSW           RESTORE A
C0C8 D3 6F       1130         OUT   CASD          SEND DATA TO CASS
C0CA C9          1140         RET   RETURN        FROM COUT
C0CB             1150 *
C0CB             1160 *** CASSETTE INPUT ROUTINE ***
C0CB             1170 *
C0CB F5          1180 CINR    PUSH  PSW           SAVE CONTROL CHAP
C0CC 3E 10       1190         MVI   A,10H         USE BIT 4 IN PEG A
C0CE D3 6E       1200         OUT   CASC          TO RESET CASS INT
C0D0 CD 57 C0    1210         CALL  AHEX          READ BLOCK LENGTH
C0D3 CD 57 C0    1230         CALL  AHEX          READ STARTING ADD
C0D6 F1          1240         POP   PSW           GET CONTROL CHAP
C0D7 E5          1250         PUSH  H             SAVE START ADD
C0D8 F5          1260         PUSH  PSW           UNDER CONTROL CHAP
C0D9 06 00       1270         MVI   B,0           SET CHECKSUM = 0
C0DB CD 0F C1    1280 CILOP   CALL  CIN           READ FM CONS
C0DE 4F          1290         MOV   C,A           SAVE IT IN REG C
C0DF F1          1300         POP   PSW           GET CONTROL CHAR
C0E0 F5          1310         PUSH  PSW           SAVE IT BACK
C0E1 FE 56       1320         CPI   'V'           IS IT A V?
C0E3 79          1330         MOV   A,C           GET BACK DATA BYTE
C0E4 CA E8 C0    1340         JZ    CINO          IF C, DON'T STORE
C0E7 77          1350         MOV   M,A           IF NOT, STORE
C0E8 80          1360 CINO    ADD   B             ADD TO CHECKSUM
C0E9 47          1370         MOV   B,A
C0EA CD F5 C1    1380         CALL  BMP
C0ED C2 DB C0    1420         JNZ   CILOP         READ MORE
C0F0 CD 0F C1    1430         CALL  CIN           READ LAST BYTE
C0F3 F5          1431         PUSH  PSW
C0F4 CD 74 C1    1432         CALL  PT2           PRINT CHECKSUM
C0F7 CD 73 C0    1434         CALL  SPCE          SPACE OVER
C0FA F1          1435         POP   PSW
C0FB B8          1440         CMP   B             COMP TO CHKSUM
C0FC 3E 45       1450         MVI   A,'E'         PRINT E FOR ERROR
C0FE C2 09 C1    1460         JNZ   CERR          PRINT NOW IF ERROR
C101 F1          1470         POP   PSW           RECOVER CTL CHAR
C102 FE 4C       1480         CPI   'L'           IF NOT L
C104 C2 09 C1    1490         JNZ   CERR          DON'T EXECUTE
C107 E1          1500         POP   H             OTHERWISE, EXECUTE
C108 E9          1510         PCHL  AT            STARTING ADDRESS
C109 CD 75 C0    1520 CERR    CALL  PTCN          PRINT V,E, OR R
C10C C3 0B C0    1530         JMP   START
C10F DB 6E       1540 CIN     IN    CASC          READ STATUS
C111 E6 10       1550         ANI   10H           LOOK AT BIT 4
C113 C2 0F C1    1560         JNZ   CIN           WAIT UNTIL LOW
C116 DB 6F       1570         IN    CASD          READ DATA FM CASS
C118 C9          1580         RET   RETURN        FROM CIN
C119             1590 *
C119             1600 *** MEMORY TEST ROUTINE ***
C119             1610 *
C119 CD 57 C0    1620 TMEM    CALL  AHEX          READ BLK LEN
C11C CD 57 C0    1640         CALL  AHEX          READ ST ADD
```

Fig. 8.20 (cont'd) Excerpt from the monitor program used by Vector Graphic on their PROM/RAM board.

```
C11F 01 5A 5A    1650        LXI   B,5A5AH       INI B,C
C122 CD 4A C1    1660 CYCL   CALL  RNDM
C125 C5          1670        PUSH  B             KEEP ALL REGS
C126 E5          1680        PUSH  H
C127 D5          1690        PUSH  D
C128 CD 4A C1    1700 TLOP   CALL  RNDM
C12B 70          1710        MOV   M,B           WRITE IN MEM
C12C CD F5 C1    1720        CALL  BMP
C12F C2 28 C1    1760        JNZ   TLOP          REPEAT LOOP
C132 D1          1770        POP   D
C133 E1          1780        POP   H             RESTORE ORIG
C134 C1          1790        POP   B             VALUES OF
C135 E5          1800        PUSH  H
C136 D5          1810        PUSH  D
C137 CD 4A C1    1820 RLOP   CALL  RNDM          GEN NEW SEQ
C13A 7E          1830        MOV   A,M           READ MEM
C13B B8          1840        CMP   B             COMP MEM
C13C C4 68 C1    1850        CNZ   ERR           CALL ERROR ROUT
C13F CD F5 C1    1860        CALL  BMP
C142 C2 37 C1    1930        JNZ   RLOP
C145 D1          1940        POP   D
C146 E1          1950        POP   H
C147 C3 22 C1    1960        JMP   CYCL
C14A             1970 *** THIS ROUTINE GENERATES RANDOM NOS ***
C14A 78          1980 RNDM   MOV   A,B           LOOK AT B
C14B E6 B4       1990        ANI   0B4H          MASK BITS
C14D A7          2000        ANA   A             CLEAR CY
C14E EA 52 C1    2010        JPE   PEVE          JUMP IF EVEN
C151 37          2020        STC
C152 79          2030 PEVE   MOV   A,C           LOOK AT C
C153 17          2040        RAL                 ROTATE CY IN
C154 4F          2050        MOV   C,A           RESTORE C
C155 78          2060        MOV   A,B           LOOK AT B
C156 17          2070        RAL                 ROTATE CY IN
C157 47          2080        MOV   B,A           RESTORE B
C158 C9          2090        RET                 RETURN W NEW B,C
C159             2100 *
C159             2110 *** ERROR PRINT OUT ROUTINE
C159             2120 *
C159 CD 81 C0    2130 PTAD   CALL  CRLF          PRINT CR,LF
C15C 7C          2140        MOV   A,H           PRINT
C15D CD 74 C1    2150        CALL  PT2           ASCII
C160 7D          2160        MOV   A,L           CODES
C161 CD 74 C1    2170        CALL  PT2           FOR
C164 CD 73 C0    2180        CALL  SPCE          ADDRESS
C167 C9          2200        RET
C168 F5          2210 ERR    PUSH  PSW           SAVE ACC
C169 CD 59 C1    2220        CALL  PTAD          PRINT ADD.
C16C 78          2230        MOV   A,B           DATA
C16D CD 74 C1    2240        CALL  PT2           WRITTEN
C170 CD 73 C0    2250        CALL  SPCE
C173 F1          2270        POP   PSW           DATA READ
C174 F5          2280 PT2    PUSH  PSW
C175 CD 7C C1    2290        CALL  BINH
C178 F1          2300        POP   PSW
C179 C3 80 C1    2310        JMP   BINL
C17C 1F          2320 BINH   RAR
C17D 1F          2330        RAR
C17E 1F          2340        RAR
C17F 1F          2350        RAR
C180 E6 0F       2360 BINL   ANI   0FH           LOW 4 BITS
C182 C6 30       2370        ADI   48            ASCII BIAS
C184 FE 3A       2380        CPI   58            DIGIT 0-9
C186 DA 75 C0    2390        JC    PTCN
C189 C6 07       2400        ADI   7             DIGIT A-F
```

Fig. 8.20 (cont'd) Excerpt from the monitor program used by Vector Graphic on their PROM/RAM board.

```
C18B C3 75 C0      2410         JMB   PTCN
C18E               2420 *
C18E               2430 *** DISPLAY MEMORY CONTENTS ***
C18E               2440 *
C18E 47            2450 DISP    MOV   B,A          SAVE CONTROL
C18F CD 57 C0      2455         CALL  AHEX         START
C192 CD 57 C0      2470         CALL  AHEX         FINISH
C195 0E 10         2480 ENT1    MVI   C,16         LOC/LINE
C197 CD 59 C1      2490         CALL  PTAD
C19A 78            2492 LP2     MOV   A,B
C19B FE 41         2500         CPI   'A'          IS IT "A"?
C19D 7E            2505         MOV   A,M
C19E CA B2 C1      2507         JZ    ASCD         DUMP ASCII
C1A1 CD 74 C1      2510         CALL  PT2          PRINT OUT
C1A4 CD 73 C0      2515         CALL  SPCE
C1A7 CD F5 C1      2520 LP3     CALL  BMP
C1AA C8            2525         RZ
C1AB 0D            2530         DCR   C
C1AC CA 95 C1      2540         JZ    ENTI         END OF LINE
C1AF C3 9A C1      2600         JMP   LP2          CONTINUE LOOP
C1B2 E6 60         2601 ASCD    ANI   60H          MASK FOR CONTROL
C1B4 C2 BD C1      2602         JNZ   NCON
C1B7 CD 73 C0      2603         CALL  SPCE
C1BA C3 A7 C1      2604         JMP   LP3
C1BD 7E            2605 NCON    MOV   A,M
C1BE E6 7F         2606         ANI   7FH          MASK FOR ASCII
C1C0 CD 75 C0      2607         CALL  PTCN
C1C3 C3 A7 C1      2608         JMP   LP3
C1C6               2610 *
C1C6               2620 *** PROGRAM MEMORY *****
C1C6               2630 *
C1C6 CD 57 C0      2640 PGM     CALL  AHEX         READ ADD.
C1C9 EB            2645         XCHG
C1CA CD 81 C0      2650         CALL  CRLF
C1CD 7E            2660 PGLP    MOV   A,M          READ MEMORY
C1CE CD 74 C1      2670         CALL  PT2          PRINT 2 DIG.
C1D1 3E 2D         2680         MVI   A,'-'        LOAD DASH
C1D3 CD 75 C0      2690         CALL  PTCN         PRINT DASH
C1D6 CD 8B C0      2700 GRIG    CALL  RDCN
C1D9 FE 2F         2710         CPI   '/'
C1DB C8            2720         RZ                 QUIT ON SLASH
C1DC FE 0D         2730         CPI   0DH
C1DE C2 E7 C1      2740         JNZ   CONI         SKIP IF CR
C1E1 CD 81 C0      2750         CALL  CRLF         PRINT CR,LF
C1E4 C3 D6 C1      2760         JMP   CRIG         BACK FO MO
C1E7 EB            2770 CON1    XCHG               H,L > D,E
C1E8 21 00 00      2780         LXI   H,0          GET 16 BIT ZERO
C1EB 0E 02         2790         MVI   C,2          COUNT 2 DIG.
C1ED CD 5F C0      2800         CALL  AHEI + 3     CONV TO HEX
C1F0 73            2820         MOV   M,E          WRITE IN MEM
C1F1 23            2830         INX   H            INC POINTER
C1F2 C3 CD C1      2840         JMP   PGLP         KEEP GOING
C1F5 7B            3000 BMP     MOV   A,E
C1F6 95            3010         SUB   L
C1F7 C2 FC C1      3020         JNZ   GOON
C1FA 7A            3030         MOV   A,D
C1FB 9C            3040         SBB   H
C1FC 23            3050 GOON    INX   H
C1FD C9            3060         RET
```

Fig. 8.20 (cont'd) Excerpt from the monitor program used by Vector Graphic on their PROM/RAM board.

SYMBOL TABLE

AHEI	C05C	AHEX	C057	ALF	C06C	ASCD	C1B2	BINH	C17C	BINL	C180
BMP	C1F5	CASC	006E	CASD	006F	CERR	C109	CILOP	C0DB	CIN	C10F
CINO	C0E8	CINR	C0CB	CLOP	C0C0	COLOP	C0A9	CON1	C1E7	CONC	0000
COND	0001	COUT	C0BF	COUTR	C099	CRIG	C1D6	CRLF	C081	CYCL	C122
DISP	C18E	ENT1	C195	ERR	C168	EXEC	C04E	GOON	C1FC	INIT	C003
LP2	C19A	LP3	C1A7	NCON	C1BD	PEVE	C152	PGLP	C1CD	PGM	C1C6
PT2	C174	PTAD	C159	PTCN	C075	PTLOP	C076	RDCN	C08B	RLOP	C137
RNDM	C14A	SPCE	C073	SPTR	D000	START	C00B	TLOP	C128	TMEM	C119

D 3000 31FF

3000	C3	03	C0	00	00	00	00	00	00	00	00	31	00	D0	CD	81
3010	C0	3E	2A	CD	75	C0	CD	8B	C0	F5	CD	73	C0	F1	FE	47
3020	CC	4E	C0	FE	56	CC	CB	C0	FE	57	CA	99	C0	FE	44	CC
3030	8E	C1	FE	50	CC	C6	C1	FE	52	CC	CB	C0	FE	4C	CC	CB
3040	C0	FE	54	CC	19	C1	FE	41	CC	8E	C1	C3	0B	C0	CD	57
3050	C0	EB	11	0B	C0	D5	E9	21	00	00	0E	04	CD	8B	C0	29
3060	29	29	29	D6	30	FE	0A	DA	6C	C0	D6	07	85	6F	0D	C2
3070	5C	C0	EB	3E	20	F5	DB	00	E6	80	C2	76	C0	F1	D3	01
3080	C9	3E	0D	CD	75	C0	3E	0A	C3	75	C0	DB	00	E6	D1	C2
3090	8B	C0	DB	01	E6	7F	C3	75	C0	CD	57	C0	CD	57	C0	06
30A0	00	CD	BF	C0	3E	E6	CD	BF	C0	7E	CD	BF	C0	80	47	CD
30B0	F5	C1	C2	A9	C0	78	CD	BF	C0	CD	74	C1	C3	0B	C0	F5
30C0	DB	6E	E6	20	C2	C0	C0	F1	D3	6F	C9	F5	3E	10	D3	6E
30D0	CD	57	C0	CD	57	C0	F1	E5	F5	06	00	CD	0F	C1	4F	F1
30E0	F5	FE	56	79	CA	E8	C0	77	80	47	CD	F5	C1	C2	DB	C0
30F0	CD	0F	C1	F5	CD	74	C1	CD	73	C0	F1	B8	3E	45	C2	09
3100	C1	F1	FE	4C	C2	09	C1	E1	E9	CD	75	C0	C3	0B	C0	DB
3110	6E	E6	10	C2	0F	C1	DB	6F	C9	CD	57	C0	CD	57	C0	01
3120	5A	5A	CD	4A	C1	C5	E5	D5	CD	4A	C1	70	CD	F5	C1	C2
3130	28	C1	D1	E1	C1	E5	D5	CD	4A	C1	7E	B8	C4	68	C1	CD
3140	F5	C1	C2	37	C1	D1	E1	C3	22	C1	78	E6	B4	A7	EA	52
3150	C1	37	79	17	4F	78	17	47	C9	CD	81	C0	7C	CD	74	C1
3160	7D	CD	74	C1	CD	73	C0	C9	F5	CD	59	C1	78	CD	74	C1
3170	CD	73	C0	F1	F5	CD	7C	C1	F1	C3	80	C1	1F	1F	1F	1F
3180	E6	0F	C6	30	FE	3A	DA	75	C0	C6	07	C3	75	C0	47	CD
3190	57	C0	CD	57	C0	0E	10	CD	59	C1	78	FE	41	7E	CA	B2
31A0	C1	CD	74	C1	CD	73	C0	CD	F5	C1	C8	0D	CA	95	C1	C3
31B0	9A	C1	E6	60	C2	BD	C1	CD	73	C0	C3	A7	C1	7E	E6	7F
31C0	CD	75	C0	C3	A7	C1	CD	57	C0	EB	CD	81	C0	7E	CD	74
31D0	C1	3E	2D	CD	75	C0	CD	8B	C0	FE	2F	C8	FE	0D	C2	E7
31E0	C1	CD	81	C0	C3	D6	C1	EB	21	00	00	0E	02	CD	5F	C0
31F0	73	23	C3	CD	C1	7B	95	C2	FC	C1	7A	9C	23	C9	00	00

Fig. 8.20 (cont'd) Excerpt from the monitor program used by Vector Graphic on their PROM/RAM board.

VECTOR 1 MONITOR V 1.2
B,C,D,E Patches

Option B

```
0090  INIT    MVI   A,03H
0091          OUT   10H
0092          MVI   A,11H
0093          OUT   10H

P 0600
0600  PTCN    PUSH  PSW
0610  PTLOP   IN    10H
0620          ANI   02
0630          JZ    PTLOP
0640          POP   PSW
0650          OUT   11H
0660          RET   RETURN

P 0740
0740  RDCN    IN    10H
0750          ANI   1
0760          JZ    RDCN
0770          IN    11H
0780          ANI   7FH
0790          JMP   PTCN
```

Option C

```
0090  INIT    MVI   A,0CEH
0091          OUT   03
0092          MVI   A,27H
0093          OUT   03

P 0600
0600  PTCN    PUSH  PSW
0610  PTLOP   IN    03
0620          ANI   01
0630          JZ    PTLOP
0640          POP   PSW
0650          OUT   02
0660          RET   RETURN

P 0740
0740  RDCN    IN    03
0750          ANI   02
0760          JZ    RDCN
0770          IN    02
0780          ANI   7FH
0790          JMP   PTCN
```

Option D

```
0600  PTCN    JMP   0C700H
0620          ANI   01
0630          JMP   RDCN
0640          POP   PSW
0650          OUT   02
0660          RET   RETURN


P 0740
0740  RDCN    IN    0D0H
0750          ANI   81H
0760          JNZ   RDCN
0770          IN    0D1H
0780          ANI   7FH
0790          JMP   PTCN
```

Option E

```
P 0600
0600  PTCN    PUSH  PSW
0610  PTLOP   IN    CONC
0620          ANI   80H
0630          JZ    PTLOP
0640          POP   PSW
0650          OUT   COND
0660          RET   RETURN

P 0740
0740  RDCN    IN    CONC
0750          ANI   40H
0760          JZ    RDCN
0770          IN    COND
0780          ANI   7FH
0790          JMP   PTCN
```

Option B – MITS 2 SIO
Option C – IMSAI SIO 2
Option D – Polymorphic Video Interface
Option E – Processor Technology 3 P + S without inverted status bits

Fig. 8.20 (cont'd) Excerpt from the monitor program used by Vector Graphic on their PROM/RAM board.

Next, the subroutine reads the status of port 00 into the accumulator and ANDs the contents with the byte 80. If the result is not zero the processor loops back to the INPUT command and repeatedly checks the input port for an 80 input code. When the code appears, the value of the accumulator stored in the stack is reloaded into the accumulator and then output on port 01 to a pointer or the terminal. Lastly, the subroutine returns from the subroutine to the main program at location C011 where the accumulator is now loaded with the ASCII code for an asterisk (2A) and subroutine PTCN at location C075 is again called to output the character and then go back to the main program.

After the * is printed or is visible on the terminal, another subroutine, RDCN, is called at location C08B. This subroutine inputs the keyboard status on port 00 to the accumulator and ANDs the contents with 01 (a mask word), which is used to check for the keyboard strobe signal, indicating valid data on the port. The subroutine loops until the strobe is detected and then an input command loads the keyboard data from port 01 into the accumulator and then strips off the MSB by ANDing the word with 7F. Next, the subroutine jumps to the PTCN subroutine to put the character back on the screen and then return to the main program at location C019 where the program status word is saved on the stack. Then the SPCE subroutine is called to load the accumulator with the ASCII code for a space (20), print it on the terminal using the PTCN routine, and then return to the program at location C01D where the PSW data are restored with a POP command.

As you can see so far, the monitor program is nothing more than a bunch of subroutines held together by a master list or executive program that knows when to call the right subroutine. But let's continue the examination.

At this point, the character is in the accumulator and displayed on the terminal. The next step is for the computer to determine what the character is. To find out, a series of compare operations are performed using immediate data to see if the entered character is a command character or just an arbitrary character. If it's a command character the program branches to the appropriate one of the following subroutines: EXEC, CINR, COUTR, DISP, PGM, TMEM, or DISP, depending on the character entered. When none of the subroutines is called, the character entered is not a command character so the main program loops back to START and waits for a command character.

Once one of the command characters is detected, the program branches to the respective subroutine. The G causes the program to jump to location C04E which calls the AHEX subroutine at location C057. The AHEX subroutine first loads the H and L CPU registers with 0000, then loads the C register with 04, and then calls the RDCN subroutine to read a byte of data from the terminal. Once the byte is entered (a key is pressed), the routine reads it into the accumulator, displays it on the terminal, and returns to the AHEX subroutine at location C05F. The DAD H (double add) instruction is the equivalent of shifting the contents of the H and L one bit position left. So, four consecutive DAD H commands shifts the value in the H and L registers four places left. Next the ASCII value for the letter H (48) is subtracted from the accumulator (which still contains the byte of data from the terminal) and then compares the result with the number 10 (decimal). If the value of the number in the accumulator is smaller than the number subtracted the program continues to the SUI 7 instruction, subtracts the number seven to convert the difference value into hex, and then jumps to ALF if a carry is present. When the number in the accumulator is larger than the number subtracted the program branches to ALF, which is used to pack the hex digits two to a byte.

The contents of the H and L registers were originally set to zero so the registers can serve as holding registers. First the L register is added to the accumulator (which is holding the stripped character) and then the result is moved back into the L register. Next, the C register is decremented (the C register originally held the count of 4 to make sure only four address digits are accepted) and if the register is not zero the program loops back to AHEI to read in another byte and perform the stripping routine. However, if the result of the DCR operation makes the contents of the C register zero, the program breaks out of the loop and exchanges the contents of the H and L registers with the contents of the D and E registers. So now the four hex address digits are in the D and E registers.

After all four characters are entered and registers swapped, the program sequences to SPCE (location C073) which is used to print a space and then return from the AHEX routine to the main program at the address C051. The subroutine returns to another XCHG command to put the four address digits back into the H and L registers since the XCHG command used back in the AHEX subroutine is used with other routines to perform specific jobs. Next, the program loads the D and E registers with the return address of START at C00B so the program that will be executed due to the G command will finish by returning the computer back to the monitor mode. After the D and E registers are loaded, their values are stored in the stack with a PUSH command and then the program counter is loaded with the contents of the H and L registers so that the program starting at that address can be executed.

When the program finishes and the computer returns to the monitor mode another command key can be pressed. If the D key is pressed, for instance, the monitor program goes to subroutine DISP at location C18E. This subroutine starts by first saving the contents of the accumulator into register B and then subroutine AHEX is called to load the next four hex char-

acters into the H and L registers and then exchange the contents of the H and L registers with those of the D and E registers. Next, the subroutine goes back to the main subroutine which says go back and load another four characters into the new H and L registers. After the next four characters are entered, the control is returned to the DISP subroutine at address C195.

The next part of the DISP routine loads register C with the number 16 and then calls subroutine PTAD which in turn calls another subroutine—CRLF—which provides carriage return and line feed to begin a new line. Next the PTAD routine loads the accumulator with the contents of H and then calls yet another subroutine—PT2—which stores the contents of the status word in the stack and calls another routine, BINH, which first shifts the accumulator right four bits and then restores the ASCII code to the stripped character and then jumps to PCTN to print the character. After the character is printed, the subroutine returns to the PT2 subroutine, pulls the status word from the stack, and then jumps to the BINL subroutine to print the other half of the word held in the H register. Then the subroutine returns to the PTAD routine at address C160. Now, the contents of the L register are transferred to the accumulator and are then converted back to ASCII and printed and then another subroutine—SPCE—is called.

Subroutine SPEC prints a space and then returns to the PTAD subroutine, which, in turn, executes another return command to bring program execution back to the DISP subroutine at address C19A, which transfers the contents of B back to the accumulator. Now, the contents of the accumulator are compared with the character A so that the condition bits in the PSW can be set and then the contents of the memory location pointed to by the H and L registers is loaded into the accumulator. The condition bits control the next instruction—JZ command—which will cause the program to jump to the ASCD subroutine if there is a character match for the A monitor command. Otherwise, the program will continue and call subroutine PT2 to print out the ASCII representation of the two 4-bit halves of the byte held in the accumulator.

Since we started with the assumption that a D key was pressed, there was no match for A and subroutine PT2 is called, the characters are printed, and then the PT2 routine goes back to the DISP routine at address C1A4 where it executes a call command to the print space subroutine and then returns where another call is executed to subroutine BMP. This subroutine loads the accumulator with the contents of register E and subtracts from that the value held in the L register and if the result is not zero the program jumps to GOON. GOON increments the H register and then returns to the DISP subroutine at address C1AA. If the zero bit is set the program is finished and returns to the monitor program. However, if the zero bit isn't set the program decrements the contents of the C register and if the zero bit is then set the program jumps to subroutine ENT1, otherwise the program jumps to LP2.

Subroutine ENT1 loads register C with 10 and then calls subroutine PTAD to provide a carriage return and line feed as well as printing the address held in the H and L registers. Next, the program loops through the memory contents again in the new set of locations and continues looping until all the specified memory locations are printed on the screen.

As mentioned earlier, the DISP program checks for an A key since both the A and D command starts with the same. However, when the command key is the A key, the DISP subroutine diverts to the ASCD subroutine, which starts at address C1B2. When the program jumps, the accumulator contains the first byte of data as addressed by the starting address held in the H and L registers and the final address is held in the D and E registers. The first instruction of the ASCD subroutine masks the byte in the accumulator to eliminate any control codes so they will not be output. If the result is nonzero the character is not a control character and can be handled by the program starting at NCON. This part of the program transfers a data byte at the memory address specified by the H and L registers into the accumulator, masks the data for a 7-bit ASCII code, and then calls another subroutine PTCN to print the character on the terminal and then return.

After returning, the subroutine loops to another subroutine BMP, that checks to see if the pointer address in the H and L registers is the same as the ending address held in the D and E registers. When the addresses match the program returns to address C1AA and performs another return or carry bit set back to the main program. If the addresses don't match, register C is decremented and if it's nonzero the program loops again to C191 to load in the next byte and then output it. When the count in the C register reaches zero the program breaks out of one loop and goes to address C195 where C is reloaded with 16, the current contents of the H and L registers are printed out, and then the next byte of data is ready for outputting. This continues (the use of the two loops) until the count in H and L matches the value in D and E.

These are just three of the nine command choices the monitor program can handle. As you can see the program makes extensive use of conditional jumps, subroutine calls, and subroutines in general. In fact, the main program actually extends only from address C000 to C04B. All the other program lines are subroutines. This monitor program exemplifies many of the programming techniques used for microprocessor and even larger computer programming.

Once you have some sort of monitor program you're ready to write your own applications programs, or purchase already written programs to use on the computer. However, loading programs into the computer by typ-

ing the printed listing is quite tedious, and many software vendors offer programs in paper-tape, cassette, floppy-disc, and even ROM form. Often, monitor programs permit you to load in other, more complex programs that let you write programs using English-like statements.

Assembly-language programming is fine for many applications, but when programs of several hundred or several thousand lines are put together, assembly language restricts the use of the program to only a processor that uses the same instruction mnemonics. To make the programs more interchangeable, high-level languages such as BASIC, FORTRAN, COBOL, PASCAL, APL, and others are used. Programs written in these English-like languages can be "transported" from one computer to another since each computer responds to the same instructions through a special translator program written for the specific computer.

BASIC is by far the most popular language used on microcomputer systems. However, there are many different forms of BASIC available—from the so-called TINY BASIC that requires only several thousand bytes of memory to the Extended Disk BASIC, which requires about 20 kbytes. Extended Disk BASIC, developed by Pertec for their Altair 8800 microcomputer, provides you with an interpreter program that reads instructions and then directs the computer to execute the proper sequence of 8080 machine instructions for each BASIC command.

Altair BASIC includes many useful diagnostic and editing features to ease program development. The Extended version also contains features to handle large disk files and input/output routines. Before you can program in a language like BASIC, though, you must learn the command set and then how to take advantage of the various instructions. There are many fine books that provide helpful, detailed explanations of how BASIC instructions work and how to write programs in BASIC.

There are, of course, many fine points to all the instructions in the BASIC command set, and they cannot be covered in the space of this book. A summary of most of the instructions is provided in Table 8.6. For exact details of the programming of computers in BASIC, read one of the many books that deal with just programming; some of them are listed in Appendix A. However, there are some procedures that must be followed if Disk BASIC is to be used on the Altair computer system. These procedures include loading it into the computer and initializing it for the system configuration you have. To load BASIC into the computer, you must define the system and make sure it can support the language. The system configuration assumed to be available is a CPU with 32 kbytes of RAM, a cassette interface, a dual floppy-disk system, a serial I/O port and CRT terminal, and a parallel I/O port and printer.

The first step is to determine in which form the bootstrap program will be; PROM is the easiest since all that must be done is to put the programmed memory onto the PROM card, set up the front-panel address switches to access the program, and hit the RUN switch. The PROM contains a bootstrap program that tells the disk how to transfer data to the computer's memory and then the loaded program takes over and controls the computer and the disks so that more complex operations can be performed. Once the BASIC program is loaded it must be initialized in the system. The first thing that happens is an output from the computer once the program is loaded asking:

MEMORY SIZE?

You must respond by typing on the terminal the total amount of RAM available to the system; in the example system then, 32k would be typed, followed by a carriage return. Next, the computer will ask another question:

HIGHEST DISK NUMBER?

You must type back the highest physical disk address in the system, followed by a carriage return. If no number is typed in, the default number is 0. BASIC next asks how many files are to be OPEN at one time in the program. This number includes both random and sequential files. If a CR is typed, the default is zero. Each file allocated requires 130 bytes of RAM for buffer space.

HOW MANY FILES?

Finally, BASIC asks how many random files are to be OPEN at one time. The amount of memory allocated is the answer *257. This memory space is used to keep track of the location on the disk where the files reside.

The paper-tape or cassette versions of the disk bootstrap can be used to load the disk if the bootstrap PROM is not used. Depending on the medium selected and the type of interface available (serial, parallel, or ACR), the program necessary to load the bootstrap will change.

In the BASIC package supplied by Pertec is a utility program called PIP that can be used to perform such common functions as printing directories, copying data from one disk to another, and even initializing the disks before data are stored on them. Some of the commands the utility program has (LIS, DIR) require that one <file number> be configured during initialization dialog. This is done by answering the HOW MANY FILES? question with a number greater than zero. Once the BASIC disk has been loaded and initialized, PIP can be accessed by typing the following:

```
RUN "PIP" <carriage return>
(PIP will then type)
*
```

Table 8.6 Commands Available in Extended Disk BASIC

MITS BASIC includes a number of features that make it well-suited for business programming.

- Print Using: The PRINT USING command makes report generation easy by providing precise control of the formatting of each line. You may specify tabbing, spacing, rounding of numbers, insertion of commas and decimal points, fixed or floating currency signs, and minus signs.
- Numeric Storage: MITS BASIC allows numbers to be stored in any of three formats:
  —Integer, with a range of −32,768 to +32,767.
  —Single Precision, with six significant digits and an exponent of $\pm 10^{38}$.
  —Double precision, with sixteen significant digits and an exponent of $\pm 10^{38}$.
- String Operators: MITS BASIC features a complete set of character string operators to allow the entry and editing of alphanumeric information with a minimum of programming effort. These operators include:

LEFT$	Returns characters from the left side of a string.
RIGHT$	Returns characters from the right side of a string.
MID$	Returns characters from the middle of a string.
INSTR	Finds a string within another string.
LEN	Returns the length of a string.
STR$	Converts a number to a string.
SPACE$	Returns a string composed of spaces.
STRING$	Returns a string of repeated characters.
CHR$	Returns a one-character string of an ASCII code.
+	Used to concatenate strings.

In addition, MITS BASIC features:

—A comprehensive text editor for entering and modifying programs.
—Error trapping.
—Disk data files (sequential and random access).
—Read after write and head position verification on disk I/O

ARITHMETIC OPERATORS

↑	Exponentiation
*	Multiplication
/	Division
\	Integer Division
MOD	Modulus Arithmetic
+	Addition
-	Subtraction

RELATIONAL OPERATORS

=	Equal
< >	Not equal
<	Less than
< =	Less than or equal to
>	Greater than
> =	Greater than or equal to

LOGICAL OPERATORS

NOT	Negation
AND	Disjunction
OR	Conjunction
XOR	Exclusive OR
EQV	Equivalence
IMP	Implication

DISK COMMANDS

MOUNT	Brings a disk on line.
UNLOAD	Takes a disk off line.
FILES	Lists disk directory.
SAVE	Saves a program in disk.
LOAD	Loads a program from disk to memory.
MERGE	Merges a program on disk with a program in memory.
RUN	Loads and executes a program.
KILL	Deletes a file from disk.
NAME	Renames a disk file.
DSKF	Returns the amount of free space on a disk.

PROGRAM EDIT COMMANDS

LIST	List a program on the CRT.
LLIST	List a program on the printer.
RENUM	Renumber a program.
EDIT	Edit and change lines in a program.
DELETE	Delete lines in a program.

FILE COMMANDS

OPEN	Opens a data file.
INPUT	Reads from the CRT keyboard or sequential file.
PRINT	Writes to the CRT or sequential file.
EOF	Set to TRUE at end of file.
GET	Reads from a random file.
PUT	Writes to a random file.

INTRINSIC FUNCTIONS

ABS	Absolute value.
ASC	ASCII code.
ATN	Arctangent.
LOG	Natural log.
RND	Random number.
TAN	Tangent.
INT	Returns integer
HEX$	Decimal to hexadecimal conversion.
VAL	Returns the value of a numeric string.
CINT	Precision conversion—integer.
CSNG	Precision conversion—single.
CDBL	Precision conversion—double.
COS	Cosine.
EXP	"e" to the power.
FIX	Truncated integer.
SIN	Sine of a number.
SQR	Square root.
SGN	Returns sign of a number.
OCT$	Decimal to octal conversion.

PROGRAMMING COMMANDS

AUTO	DEFUSR
CLEAR	DIM
CONSOLE	END
CONT	ERASE
DATA	ERL
DEF	ERR
DEFDBL	ERROR
DEFINT	FOR-NEXT
DEFSNG	FRE
DEFSTR	GOSUB

Table 8.6 (cont'd) Commands Available in Extended Disk BASIC

GOTO	POKE	TRON	WAIT
IF-THEN-ELSE	POS	USR	WIDTH
INP	PRINT	VARPTR	
INPUT	PRINT USING		
LET	READ	**CONTROL CHARACTERS**	
LINEINPUT	REM	Control/A	Edit the current input buffer.
LIST	RESTORE	Control/C	Stop program execution.
LLIST	RESUME	Control/H	Blank previous character typed (backspace).
LPOS	RESUME NEXT		
LPRINT	RETURN	Control/I	Tab.
ON ERROR GOTO	SPC	Control/O	Toggle print output on/off.
ON-GOSUB	STOP	Control/S	Suspend program execution.
ON-GOTO	SWAP	Control/Q	Resume program execution.
OUT	TAB	Control/U	Erase the current input buffer.
PEEK	TROFF	Control/X	Same as Control/U

At this point PIP is ready to accept commands. To exit PIP, type a carriage return in response to the prompting asterisk. To initialize a disk on drive zero, just type INI0 immediately following the asterisk. PIP will type DONE when it is finished. Any disk number can, of course, be substituted in this example and PIP will reformat the disk in that drive. Remember, though, any files on that disk will be lost.

Giving PIP the command *DIR <disk number>, prints out a directory of the files stored on the specific disk. The name of each file is printed along with the file's "mode" (S for sequential, R for random) and the starting track and sector number of the first block in that file. The command SRT disk number prints out a sorted director of file names. An LIS command can be used to list the contents of a sequential data file on a specified disk. The format of the LIS instruction is

LIS <disk number>, <file name>.

A copy instruction is available to duplicate data held on disk, thus providing a back-up copy in case of damage or a copy for someone else to have. This instruction is set up as follows:

COP <old disk number>, <new disk number>.

Before the actual copying is done, PIP prints out the following message:

FROM <disk number> TO <disk number>.

Typing Y followed by a CR causes execution to proceed. Any other response aborts the command, thus preventing accidental loss of data. Another command, DAT, causes a particular sector of the disk to be dumped out in octal. When the DAT <disk number> command is issued, PIP asks for the numbers of the track and sector to be dumped. One other instruction on the disk's PIP program, CNV, is used to update older versions of the MITS BASIC to the latest version. Also provided on the system disk containing BASIC is a game program for playing STARTREK, a game based on the science fiction television series of the same name that originated about 10 years ago.

Now That You Have BASIC, What Do You Do With It?

Assuming that you've been able to get your system together and running, you're ready to start writing programs to solve mathematical problems, do stock market analysis, perform general accounting routines, play games, or process data for almost any other purpose. To perform actual physical control of motors, lamps, and other high power devices, special interface circuits must be developed to handle the power demanded by these loads and to prevent damage to the delicate computer circuits if some of that power should try to feed back. The realm of input/output interfaces to the world is so large, another book would barely scratch the surface of the topic. Several books that attempt to show ways to connect the computer to external devices are listed in Appendix A. Pertec also offers a family of interface cards that contain relays and special conversation circuits to change the digital data words into analog signals or vice versa. These process control interfaces and control cards can be programmed either via assembly language or BASIC to do the desired jobs.

CHAPTER 9

Interfacing the Microcomputer to Real-World Applications

The basic computer system described in its various sections consists of the cabinet with control panel, the central processor, an assortment of memory boards (ROM and RAM cards), some serial and parallel I/O cards, a bulk-memory interface (cassette or floppy-disk controller and deck or drive), a CRT terminal or printing terminal, and possibly a printer. For most applications the system described is a complete computer system and probably has a cost of about $6000. However, the computer can just do what it was built for, and that is compute. It has no ability to control anything external to it such as a motor, lamp, alarm, or factory—all it can do is accept data in digital form, manipulate the data (process them), and print out or display the answer.

For the computer to really do some other types of operations, it must be given the "muscle" to handle high-power loads such as the motors or lamps. The muscle consists of special-purpose I/O boards that contain power-boosting circuitry to amplify or translate the computer's logic signals to the powerful levels needed to open and close relay contacts, or convert the computer words into signals such as synthesized speech. There are special circuits that permit computers to convert analog signals into digital form, store the digital coded equivalent, manipulate it, and then output the code to another circuit that does the reverse conversion—from digital to analog.

These special interface boards are made by many manufacturers and it is rare that two boards perform exactly the same function. Several boards that will be examined to illustrate what manufacturers can do to get information into or out of the computer include the 88-AD/DA analog I/O card and the 88-PCI process control interface, both offered by Pertec (Fig. 9.1).

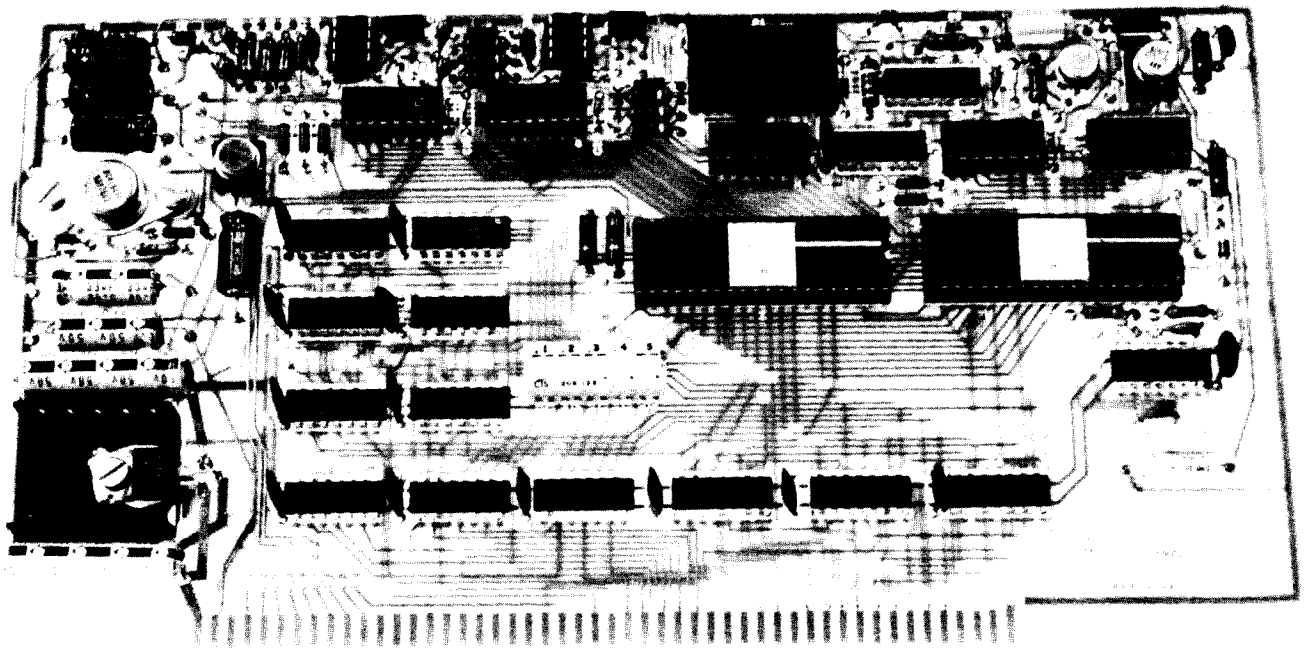
Bring Analog Signals into the Digital World

The 88-AD/DA board for the S-100 bus is a computer controlled subsystem that can accept analog signals from up to eight sources simultaneously, convert the signals into 8-bit digital equivalents, and then feed the equivalents to the computer's CPU, memory, or another I/O port (Fig. 9.2). The board can also work in reverse. It contains two circuits that can accept an 8-bit word from the computer bus and change that word into an analog signal. The specialized circuits used to perform the conversions are appropriately called analog-to-digital converters (a/d's) and digital-to-analog converters (d/a's).

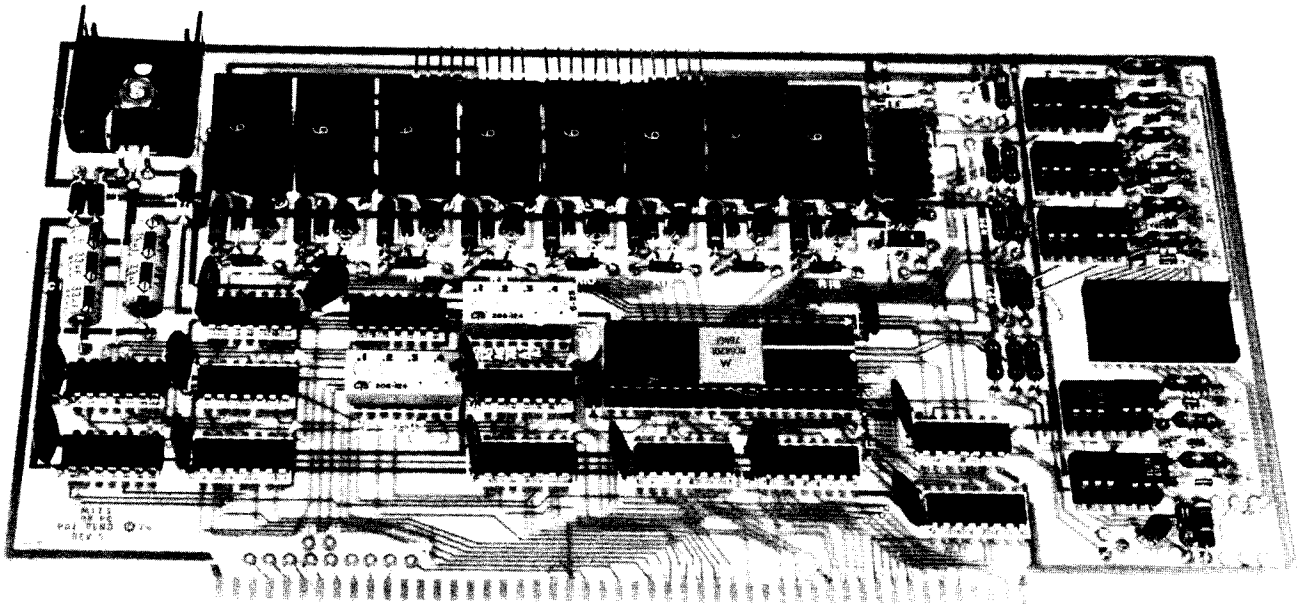
Signals fed into the board from outside sources are analog signals that usually vary in amplitude with time. For the a/d to handle them properly, the signals may have to be adjusted (conditioned) to meet the input requirements of the conversation system and minimize any distortion. Once conditioned, the analog signals are first fed into a circuit called a multiplexer, which is like a rapidly rotating eight-position switch, similar to the channel selector on a television set. As the multiplexer steps through its eight positions, each analog input, in turn, appears at the input to the next part of the card—the sample-and-hold amplifier (s/h).

The s/h is like a temporary camera—each time a new input appears and a special signal is received by the s/h control line, the circuit takes a "snapshot" of the value of the signal. This temporarily "freezes" the input to the a/d converter, thus preventing the converter from seeing more than one value at a time. After the signal is held at a fixed value, the a/d converter receives a start signal to begin the actual conversion of the signal value into digital form. When the converter finishes its job it sends a signal back to the computer, telling the CPU that the conversion is finished and the digital word is ready for transmission over the computer's bus to the CPU, memory, or I/O device. The entire process, from the time the multiplexer selects a channel to the time the converter says it finishes a conversion, takes approximately 10 to 12 μs—a mere wink of an eye.

The reverse process, performed by the two d/a's, is simpler. All eight bits of data are first fed from the computer bus to a 6820 PIA. When given a signal from the CPU the PIA will accept the data and feed them through to the d/a. The d/a immediately converts the digital word into analog form (a voltage) and feeds it out to



(A)



(B)

Fig. 9.1 Performing analog input and output, the 88-AD/DA board (a) handles up to eight inputs and delivers two outputs. Capable of controlling eight separate high-power devices, the 88-PCI handles loads of up to 1 A at 120 V ac (b). (*Courtesy* Pertec)

the "real" world. This process is even faster than the previous sequence—the analog equivalent is available a microsecond or so after the 8-bit digital word appears on the converter's input.

To access the board once it is plugged into the bus, address lines A3 to A7 provide 32 possible address selections in increments of eight. Address line A2 provides the selection signals for the particular PIA, thus de-

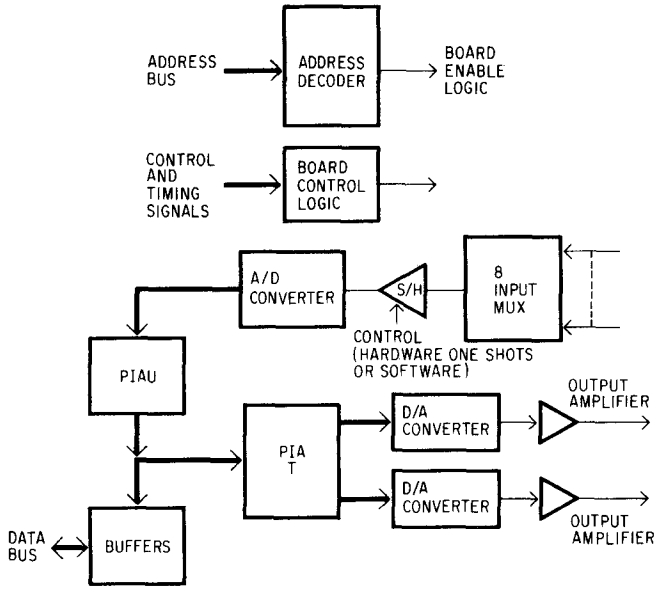


Fig. 9.2 Block diagram of the 88-AD/DA board.

termining whether an a/d or d/a operation will take place. One PIA is used solely for a/d operations and the other PIA services both d/a converters.

Address lines A0 and A1 select which of the three internal registers of the PIA gets accessed by the computer once A2 selects the desired PIA (Table 9.1).

For each PIA, four addresses are used to select the desired register (see Table 9.2). The board address is referred to as the base address and then an offset of Ø, 1, 2, or 3 is used to select one PIA's registers, and an offset of 4, 5, 6, or 7 is used for the other PIA. When the PIA is selected, the data bus interface lines (DOØ to DO7 and DIØ to DI7) are set up as inputs or outputs. Inputs are used for an a/d operation and outputs for a d/a operation.

There are many jumper options on the board that have to be selected before the board is installed:

1. The desired output voltage swing of the d/a converters can be set for 0 to 10 V, −10 to 0 V, or −10 to 10 V.
2. On-board timers for the a/d converter and sample-and-hold amplifier must be set if software is not used to control all a/d operations.
3. The comparator output of the a/d converter must be coupled to the rest of the subsystem, with or without any feedback.
4. The board address must be selected by switch settings, not jumpers.

Table 9.1 Internal Register Selection Chart for the 88-AD/DA

		Control/Status Registers		
A1	A0	A-bit 2	B-bit 2	Location Selected
0	1	1	X	Data channel register A
0	1	0	X	Data direction register A
0	0	X	X	Control/status register A
1	1	X	1	Data channel register B
1	1	X	0	Data direction register B
1	0	X	X	Control/status register B

X = Don't care.

Table 9.2 Register Address Definition Chart for the 88-AD/DA

	Address Required	Location Selected
PIA-U	Base address + 0	Control status register A
	Base address + 1	Data direction register A
	Base address + 1	Data channel register A
	Base address + 2	Control/status register B
	Base address + 3	Data direction register B
	Base address + 3	Data channel register B
PIA-T	Base address + 4	Control/status register A
	Base address + 5	Data direction register A
	Base address + 5	Data channel register A
	Base address + 6	Control/status register B
	Base address + 7	Data direction register B
	Base address + 7	Data channel register B

For an input to the computer to take place, address lines A0 to A7 should be set so that the board is selected by the desired address sent over the address lines of the main bus. At the same time both the SINP and SOUT lines of the bus must be HIGH along with the A2 line to select the a/d converter. This sequence also sets the data bus drivers so that they will deliver a digital word to the bus, starts the a/d conversion process, and begins a WAIT state (this tells the computer to idle for one instruction cycle). At the end of the WAIT state the converter signals that it has finished and delivers its digital output to the PIA, which, in turn, sends the word to the computer's data-in bus. The process for a d/a operation is similar—the address must be selected; however, only SOUT must be HIGH, thus telling the selected PIA that data will be sent from the data bus to the PIA and on to the selected d/a converter.

Before the 88-AD/DA board is installed, switches and jumpers should be set for the desired configuration of the application. The board address, the analog output range, the sequencing of the sample-and-hold amplifier (hardware or software controlled), and the sequencing of the a/d converter must all be set. Hardware control is normally used to keep the processor free for other duties. For highest speed operation, the sample-and-hold amplifier can be bypassed; however, accuracy might suffer for some input signals. Unused multiplexer inputs should be grounded due to the high input impedance of the sample-and-hold amplifier. This will prevent erroneous voltage readings. Cable layout is very critical since noise and other types of electrical interference can cause incorrect readings. The longer the cables, the more prone to interference the system will be. A full schematic of the board is in Appendix C.

After the board is physically set up and plugged into the bus, the next step is to develop the software that will

control the board and transfer data back and forth. Registers inside the PIAs must be set up to control the flow of data back and forth. The control status and data direction registers must be initialized. The DDRs tell the PIA whether to set up the I/O lines as inputs or outputs. The C/S registers control the function of the CA1, CB1, CA2, and CB2 signals and determine whether to write to the DDR or the data channel register.

To write the DDR, bit 2 of the C/S register must be set to zero; this is accomplished by writing a zero to the C/S register of the selected PIA section (A or B) in the selected PIA. When writing to section A of PIA-U, zero's are written to the base address (C/S register A); or when writing to section B, zero's are written to the base address + 2 (C/S register B). For writing to PIA-T section A, zero's are written to the base address + 4(C/S register A); or for section B, zero's are written to the base address + 6 (C/S register B).

Once the zeros are entered, the associated DDR can be accessed and modified. When writing to PIA-U, base address + 1 selects DDR-A and base address + 3 selects DDR-B. For PIA-T, base address + 5 selects DDR-A and base address + 7 selects DDR-B. Section A of PIA-U accepts the digital output of the a/d converter and must thus be set up as an input port; section B outputs address information to the multiplexer and must be set up as an output port. Sections A and B of PIA-T deliver the digital words to the d/a converters and must therefore be set up as output ports.

To actually do the initialization of PIA-U, zeros must be written to section A and ones to section B. For PIA-T, ones must be written to both DDR registers. After both PIAs have their DDRs initialized, the C/S registers can be set up for their final operating form. Bit 2 of the C/S registers must be set to one; this will move the addressing from the DDRs to the data channel registers. The program shown in Fig. 9.3 is a sample initialization program and assumes the board is addressed at location 100_8. The two lowest order bits, 0 and 1 of the C/S register, are used to control the interrupt input lines CA1 and CB1 (Table 9.3). In control/status registers A and B, bit 0, when HIGH, is used to enable the interrupt request ($\overline{IRQA}$ or $\overline{IROB}$). If interrupts are not required, bit 0 should be set LOW. Bit 1

Machine Language			
Location	Contents	Comments	Basic
000	076	Load Accumulator	
1	000	with 0s	10 OUT 64, 0
2	323	output	20 OUT 65, 0
3	100		30 OUT 66, 0
4	323	output	40 OUT 68, 0
5	101		50 OUT 70, 0
6	323	output	60 OUT 67, 255
7	102		70 OUT 69, 255
010	323	output	80 OUT 71, 255
1	104		90 OUT 64, 44
2	323	output	100 OUT 66, 44
3	106		110 OUT 68, 44
4	076	Load Accumulator	120 OUT 70, 44
5	377	with 1s	
6	323	output	
7	103		
020	323	output	
1	105		
2	323	output	
3	107		
4	076	Load Accumulator	
5	054	with 054_8	
6	323	output	
7	100		
030	323	output	
1	102		
2	323	output	
3	104		
4	323	output	
5	106		

Fig. 9.3 Program showing a sample initialization of the 88-AD/DA board.

Table 9.3 Interrupt Input Control Definition for the 88-AD/DA

Control/Status Register		Interrupt Input	Interrupt Flag Control/Status Register	MPU Interrupt Request
A (B)-bit 1	A (B)-bit 0	CA1 (CB1)	A (B)-bit 7	$\overline{IRQA}$ ($\overline{IRQB}$)
0	0	Active negative transition	Set HIGH on negative transition of CA1 (CB1)	Disabled—$\overline{IRQ}$ remains HIGH
0	1	Active negative transition	Set HIGH on negative transition of CA1 (CB1)	Goes LOW when the interrupt flag bit, control/status register A (B) bit 7, goes HIGH
1	0	Active positive transition	Set HIGH on positive transition of CA1 (CB1)	Disabled—$\overline{IRQ}$ remains HIGH
1	1	Active positive transition	Set HIGH on positive transition of CA1 (CB1)	Goes LOW when the interrupt flag bit control/status register A (B) bit 7, goes HIGH

Note 1: The interactions of control/status register A, CA1 and $\overline{IRQA}$ are identical to the interactions of control/status register B, CB1, and $\overline{IRQB}$.

Note 2: The interrupf flag bit for the appropriate control/status register, bit 7, is cleared by an MPU read data operation.

Note 3: If control/status register A (B), bit 0, is LOW when the interrupt occurs (interrupt disabled) and later goes HIGH, $\overline{IRQA}$ ($\overline{IRQB}$) occurs after control/status register A (B), bit 0, is written to a 1.

Table 9.4 Control Line Definition for the 88-AD/DA

Control/Status Register A (B)-bit 4	Control/Status Register A (B)-bit 3	Interrupt Input CA2 (CB2)	Interrupt Flag Control/Status Register A (B)-bit 6	MPU Interrupt Request $\overline{IRQA}$ ($\overline{IRQB}$)
0	0	Active negative transition	Set HIGH on negative transition of CA2 (CB2).	Disabled—$\overline{IRQ}$ remains HIGH
0	1	Active negative transition	Set HIGH on negative transition of CA2	Goes LOW when the interrupt flag bit control/status register A (B) bit 6, goes HIGH
1	0	Active positive transition	Set HIGH on positive transition of CA2 (CB2)	Disabled—$\overline{IRQ}$ remains HIGH
1	1	Active positive transition	Set HIGH on positive transition of CA2 (CB2)	Goes LOW when the interrupt flag bit control/status register A (B) bit 6, goes HIGH

Note 1: The interaction of control/status register A, CA2 and $\overline{IRQA}$ is identical to the interaction of control/status register B, CB2 and $\overline{IRQB}$.

Note 2: Control/status register A (B)—bit 5 is LOW.

Note 3: The interrupt flag bit for the appropriate contol/status register, bit 6, is cleared by an MPU read data operation.

Note 4: If control/status register A (B), bit 3, is LOW when an interrupt occurs (interrupt disabled) and later goes HIGH, $\overline{IRQA}$ ($\overline{IRQB}$) occurs after control/status register A (B) is written to a 1.

of the C/S register determines the active transition of the interrupt input signals. When bit 1 is LOW, the interrupt flag (bit 7) is set HIGH on a HIGH-to-LOW (negative-going) signal transition; and if bit 1 is HIGH, bit 7 is set HIGH on a LOW-to-HIGH (positive-going) signal transition.

The a/d converter output consists of the eight data bits as well as a signal line that signals when the converter has completed a conversion. This signal is fed to the CA1 line of the PIA. Thus, bit 1 of C/S register A (base address + 0) should be set LOW to make CA1 LOW active. Since there are no interrupts in this example, bit 0 should be set LOW.

Bits 3, 4, and 5 of the C/S registers are used to control CA2 or CB2. If bit 5 is LOW, CA2 and CB2 function as interrupt input lines and are similar in operation to CA1 and CB1 (see Table 9.4). When bit 5 is HIGH, CA2 and CB2 serve as peripheral control output lines. As output control lines, C/S register bits 3 and 4 determine the exact characteristics of the lines (see Tables 9.5 and 9.6). CB2 of PIA-U enables the timing sequence on the board to control the sequencing of the s/h amplifier and the a/d converter. To use CB2 as a control line, the C/S register B (base address + 2) bit 5 is set HIGH, bit 4 is set LOW, and bit 3 is set HIGH.

Some sample programs, shown in Figs. 9.4 and 9.5, provide both hardware and software control examples of timing sequences. Normally, hardware control is used. However, software control does permit adjustments without turning off the system to modify the circuit. The program of Fig. 9.4 uses hardware control via the on-board timing circuits to provide the data settling time. The program in Fig. 9.5 includes the extra steps for software control of the sequencing. Output programs, which are generally simple, merely require the computation and the output. Sometimes a minimum delay time between new outputs from the d/a converters is desirable. To do this, the program must be designed to compute the first value and store it in an available register in memory or in the processor. Then the other value is computed and stored in another location. Since data cannot be output to both d/a converters simultaneously, the first value is initially retrieved and output to one channel. Then the second value is retrieved and then sent out to the other channel. This results in a minimum time delay between channel outputs.

Applications for a multiple function board such as the 88-AD/DA are unlimited, but just for an example, let's examine how the board can be used to create video displays on either a monitor that has X and Y inputs or an oscilloscope (Fig. 9.6). By using both d/a converters to feed the display inputs, pictures can be drawn on the screen by feeding the proper digital information to the d/a converter inputs. The general scheme used to generate graphics in this manner starts with a serial table of the successive points in the figure to be drawn on the screen. The table, of course, is stored in the computer's memory and the area of memory holding the table is often referred to as the position table. Under program control these points are fed to the d/a converter's inputs and then converted into analog voltage for the display.

However, most displays cannot hold the information unless it is constantly refreshed, much in the same way dynamic memories must be refreshed. To do this, the values held in the table must be repeatedly output to the display. Otherwise, only a brief image will flash on the screen. The image produced by the converter outputs is composed of points of light; however, by speeding up the rate at which the converter's output amplifier can change its value (the slew rate), the display can be made to appear as a continuous line.

One simple way to actually draw a picture on the display screen is to use a joystick to enter the points to be displayed into the computer memory. (A joystick is often a lever connected to several potentiometers

Table 9.5 Bit Settings for I/O Control Lines for the 88-AD/DA

Control/Status Register		CB2	
B-bit 4	B-bit 3	Cleared	Set
0	0	LOW on the positive transition of the first "E" pulse following an MPU write section B data operation, occurring after control/status register, bit 7, is cleared by a read section B data operation.	HIGH when the interrupt flag bit, control/status register B, bit 7, is set by an active transition of the CB1 signal.
0	1	LOW on the positive transition of the first "E" pulse after an MPU write section B Data operation.	HIGH on the positive edge of the first "E" pulse following an "E" pulse which occurred while the part was deselected.
1	0	LOW when Control/status register B, bit 3, goes LOW as a result of an MPU write in control status register B.	Always LOW as long as control/status register B, bit 3, is LOW. Will go HIGH on an MPU write in control/status register, section B, that changes control/status B, bit 3, to 1
1	1	Always HIGH as long as control/status register B, bit 3, is HIGH. Will be cleared when an MPU write control/status register B results in clearing control status register B, bit 3, to 0.	HIGH when control/status register B, bit 3, goes HIGH as a result of an MPU write into control/status register B.

Note: Control/status register B—bit 5 is HIGH.

Table 9.6 CA and CB Line Function Selection for the 88-AD/DA

Control/Status Register		CA2	
A-bit 4	A-bit 3	Cleared	Set
0	0	LOW on negative transition of "E" after an MPU read section A data operation.	HIGH when the interrupt flag bit, control/status register A, bit 7, is set by an active transition of the CA1 signal.
0	1	LOW on the negative transition of "E" after an MPU read section A data operation.	HIGH on the negative edge of the first "E" pulse which occurs during a deselect.
1	0	LOW when control/status register A, bit 3, goes LOW as a result of an MPU write to control/status register A.	Always LOW as long as control/status register A, bit 3, is LOW. Will go HIGH on an MPU write in control/status register A, that changes control/status register A, bit 3, to 1.
1	1	Always HIGH as long as control/status register A, bit 3, is HIGH. Will be cleared on an MPU write to control/status register A that clears control/status register A, bit 3, to a 0.	HIGH when control/status register A, bit 3, goes HIGH as a result of an MPU write into control/status register A.

Note: Control/status register A—bit 5 is HIGH.

and power supplies.) By moving the lever, the voltage outputs of the potentiometers change and this information can then be converted into digital form by the a/d converter and then stored. The computer can then read this information from the memory and display it on the screen.

To view the joystick data as they are entered, the two steps of acquiring the data and displaying the data must be multiplexed by the software. This is accomplished by outputting the table information each time a new point is entered. To avoid filling the memory with redundant information when the joystick is stationary, the program should test each input location to see if it is different from the last byte stored. Further reduction of memory storage requirements can be accomplished by masking off bits at the least significant end of the position word (the new position data must be different from the old by some amount before they are stored). Masking off two bits still provides six bits of resolution. Since two coordinates are involved, the amount of memory used for the position table is directly proportional to the square of the resolution. Thus, an 8-bit system uses 16 times as much memory as a 6-bit system. The flowchart for the joystick data entry and display routine is shown in Fig. 9.7.

Provide Control for Large Loads

Most computer circuits are limited when it comes to the amount of power they can control—just mere milliamps in most cases. When larger power requirements must be handled, special drive circuits must be

NOTE:
X = Don't Care
CBA = Binary Multiplexer Channel Number
CBA = 000 = CH#0
CBA = 001 = CH#1
CBA = 010 = CH#2
etc.
etc.
CBA = 111 = CH#7

Location	Contents	Comments	Bits
n + 000	076	LXI Acc	7 6 5 4 3 2 1 0
001		→	X X X X C B A X
002	323	Output	
003	103		
004	333	Input ←	
005	100		
006	346	ANI	
007	200	DATA	
n + 010	312	JZ	
011	(n +		
012	004)		
013	333	INPUT	
014	101		

Fig. 9.4 Sample program for hardware control of the 88-AD/DA board.

NOTE:
X = Don't Care
CBA = Binary Multiplexer Channel Number
CBA = 000 = CH#0
CBA = 001 = CH#1
CBA = 010 = CH#2
etc.
etc.
CBA = 111 = CH#7

Location	Contents	Comments	Bits
n + 000	076	LXI Acc	7 6 5 4 3 2 1 0
001		→	X X X 0 C B A 1
002	323	Output	
003	103		
004	000	NOP } data settle	
005	000	NOP } time delay	
006	076	LXI Acc	
007		→	X X X 0 C B A 0
n + 010	323	Output	
11	103		
12	076	LXI Acc	
13		→	X X X 1 C B A 0
14	323	Output	
15	103		
16	333	Input ←	
17	100		
n + 020	346	ANI	
21	200	100 00000	
22	312	JZ	
23	n +		
24	016		
25	333	Input	
26	101		

Fig. 9.5 Sample program for software control of the 88-AD/DA board.

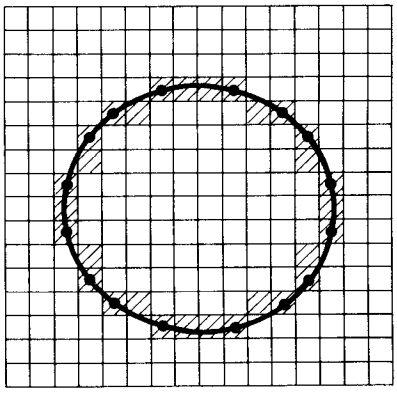


Fig. 9.6 By using both d/a converter outputs from the 88-AD/DA board, full X-Y drawings such as circles can be done on video displays or recorders.

used to boost the current and voltage handling capabilities of the computer circuits. The 88-PCI made by Pertec is one possible circuit card that can plug into the S-100 bus and control motors, lamps, alarm systems, and much more (Fig. 9.8).

The 88-PCI provides eight individual control outputs, each capable of handling 1 A at 120 V ac (resistive loads). All control is performed via a 6820 PIA. This control circuit has 16 programmable lines (eight of which are used as output lines and eight as input lines) for sensing and response acknowledgment. There are also control lines for each set of eight lines. All the control lines are programmable via the instructions as inputs or outputs, as described earlier in this chapter for the 6820's used in the 88-AD/DA board. For the 88-PCI board, section A of the PIA acts as the input lines to the computer while section B acts as the eight output lines that control the high-power loads. A full schematic of the board is supplied in Appendix C.

The board is set up so that it can be located at any one of 64 possible locations—four addresses are required for the board. The actual address selection is done via a multiple-position switch mounted on the board. When the set address is input to the board from the bus, the logic on the board gets enabled and the board is ready for its first command. Address lines A2 to A7 are used to select the main board address and lines A0 and A1 are also fed into the board. A0 selects the PIA register C/S when LOW and DDR when HIGH, and A1 determines which section of the PIA—section A when LOW in conjunction with RS1, and section B when both lines are HIGH.

Section A, the input channel, reads data from the optoisolators, and section B, the output channel, controls the relays that in turn control the high-power loads.

When a valid address is present, the circuits are enabled and are looking for the SOUT or the SINP signals from the bus to tell the PIA whether to perform an input or an output operation. After the computer

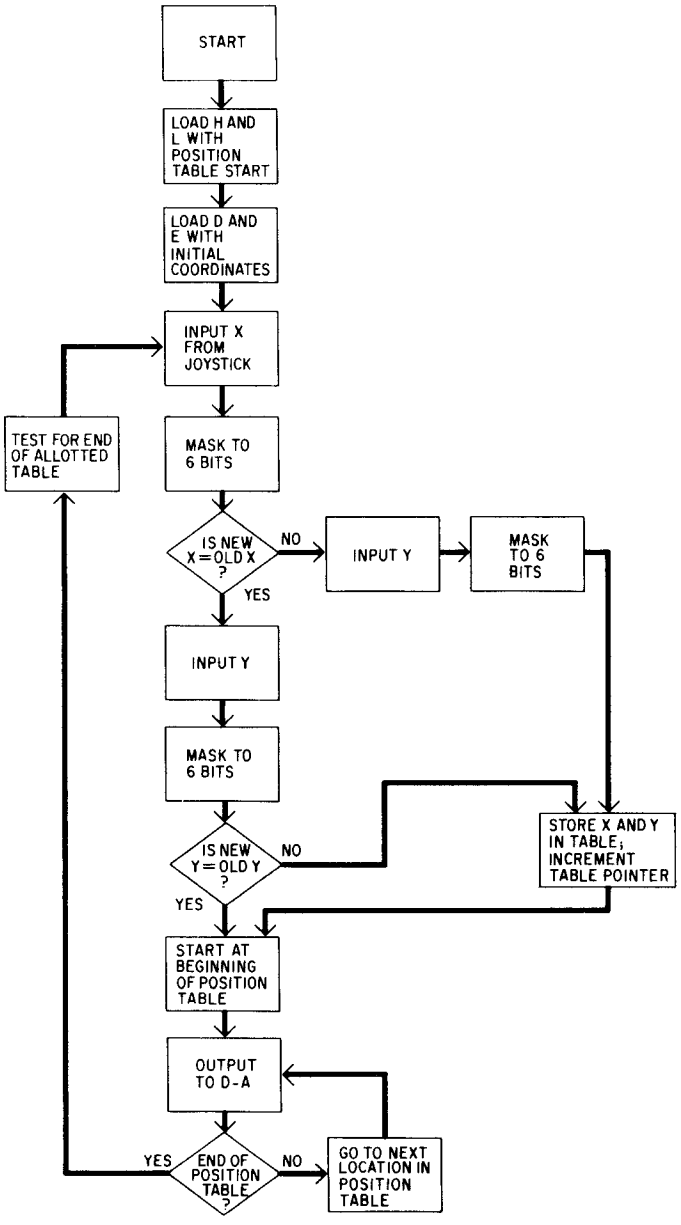


Fig. 9.7 To enter data from joysticks, this simple flowchart can be converted to a full control program that monitors the a/d converter outputs on the 88-AD/DA board for a change in value before storing the new value in the computer's memory.

requests an input from the PCI board, a wait-state generator circuit on the board causes a short processor delay to help synchronize the processor to the PIA chip. A WAIT cycle can be caused by the PWAIT signal from the bus too. And, the bus $\overline{POC}$ signal can be used to initialize the PIA so when power comes on data can be loaded into the PIA control registers.

The only timing signal required by the PIA to lock onto or output data is the enable signal (E pulse). This signal is generated by a combination of the $\overline{PWR}$, PDBIN, PINTE, and phase 1 clock signals. The PWR and PDBIN signals can normally control the E line of the PIA. And, the PINTE and $\phi 1$ clock can generate an interrupt to stop the PIA operation when the processor has halted for any reason.

Each output of the PIA port B drives a combination transistor/relay circuit. The transistor is used to boost the current output of the PIA lines enough to drive the low-power coil of the relay, which can drive the larger power loads. The mechanical contacts of the relay are routed to a bare area on the circuit board where you can customize the output circuitry and include any contact protection circuits that can help prolong relay contact life. Each type of load the relay drives will require a different type of protection circuit, and we'll look at some of these schemes shortly.

The PIA input lines, protected from the dangers of uncontrolled electrical signals by optical isolators, can handle just about any kind of electrical input. The optical isolator typically consists of an LED and a photodiode or phototransistor that are very tightly coupled together. When the LED is turned on by the input control signal from some external device, the light from the LED turns on the electrically isolated detector, causing a predefined logic level to appear at the input to the PIA. However, each application has a different set of voltages that must be scaled or boosted so that the LED will generate enough light to cause the isolator output to be at the proper level for the PIA.

Several isolators are used to provide isolation for output control signals to the peripheral I/O bus. For these circuits there are various options that must be considered before the board can be used—for instance, the isolator output can be made to match the PIA outputs or they can be made to complement the PIA outputs. A simple jumper connection determines whether the isolator output is the same as the PIA's output, or the inverse. Also, the phototransistor output of the isolators is completely unconnected, and must be connected to the appropriate power supplies, grounds, or other circuit interfaces. Isolators are provided for the CA1 and CB1 inputs and for the CA2 and CB2 outputs.

The 88-PCI board performs a total of three types of operations—input, output, and interrupt. An input operation starts with the board address being sent out on the address bus to enable the rest of the board circuitry. The SINP line in conjunction with the ADDRESS VALID signal enables the PIA and the data bus drivers, thus giving control of the data input bus to the 88-PCI card. These signals will also cause the board to initiate a WAIT state. When the WAIT state is finished, the PDBIN signal causes an ENABLE signal which clocks the data at the PIA input lines through the 6820 and onto the data input bus.

For an output operation, the address should again be input to the board, thus enabling the circuitry. The SOUT line should also be HIGH, thus causing the PIA to go into a READ mode from the data out bus. When the DATA VALID signal appears in conjunction with the SOUT signal, the bus receivers get enabled and

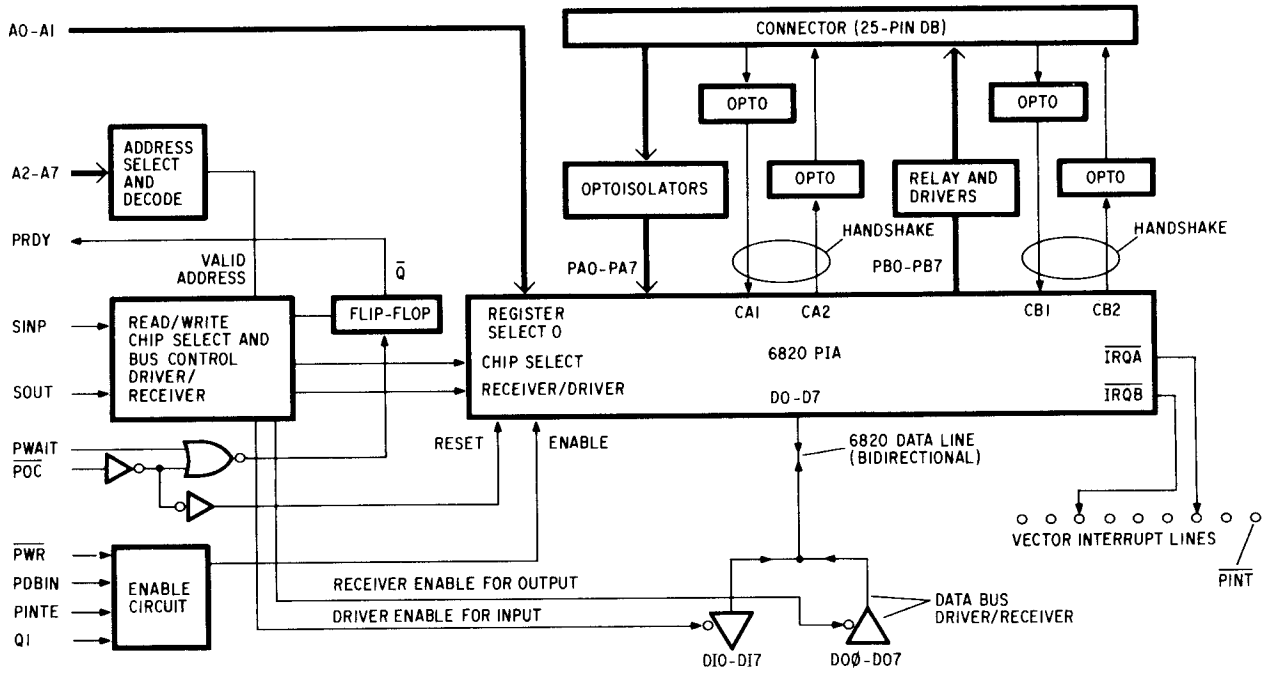


Fig. 9.8 Block diagram of the 88-PCI card.

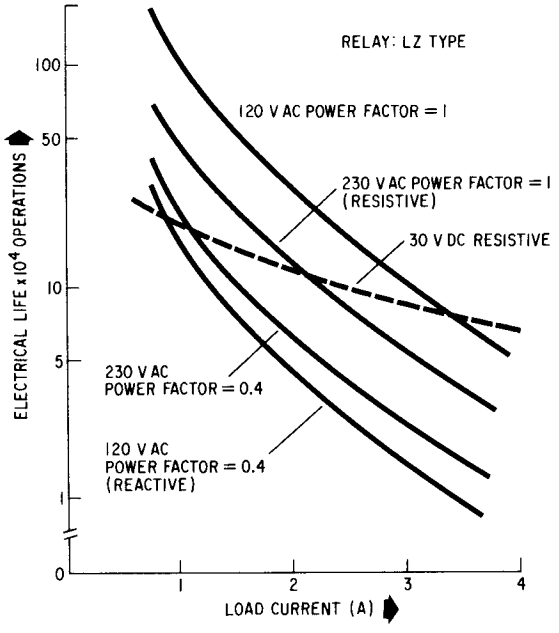


Fig. 9.9 Contact life curve for the relays on the 88-PCI card.

data from the bus reach the PIA data bus inputs. The bus $\overline{\text{PWR}}$ signal then causes the data to be fed through the PIA and appear on the PB output data lines. A logic one will turn on a relay and a logic zero will leave the relay de-energized.

The board also performs an interrupt operation via the CA1 or CB1 input lines. During board initialization, the C/S registers determine whether this change will be from HIGH to LOW or from LOW to HIGH. Assuming the interrupt signals are enabled, either IRQA for section A or IRQB for section B will be LOW on the ENABLE pulse when this transition takes place. Thus, the interrupt request is clocked out of the 6820 by the ENABLE pulse. The type of interrupt requested depends on whether the interrupt request lines are tied to the PINTE bus line (which is normal), or to one of the vectored interrupt lines of the bus (if a vectored interrupt card is used).

Get to Know the 88-PCI Board Options

To get a board such as the 88-PCI set up for your application, there are many factors to consider—relay contact life, optoisolator input and output conditioning, software initialization, and developing the specialized applications software.

The first step should be to determine the number of relays that will be needed, the power each must handle, and the type of load to be driven (resistive, inductive, or capacitive). The application determines the number of relays by the control functions that must be performed. The voltage and current that the relay contacts can safely handle without damage must be calculated. For the particular relays on the 88-PCI board, the chart of Fig. 9.9 will help determine the number of operations the relay can handle for a particular voltage, current, and load. For example, a 120 V ac, 1 A resistive load (power factor = 1) permits a contact life of about 500,000 operations. If this number of operations is too low, the relay must be used to drive a smaller load, such as a higher-rated relay or solid-state switch that can handle the actual power requirements over the desired lifetime.

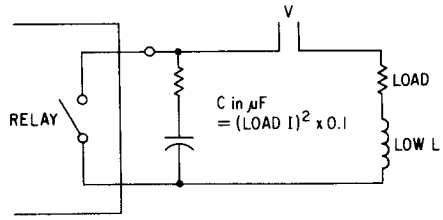


Fig. 9.10 For dc loads, this snubber circuit can help protect the relay contacts.

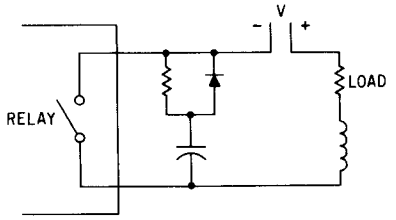


Fig. 9.11 Another version of a snubber circuit for dc loads on the 88-PCI.

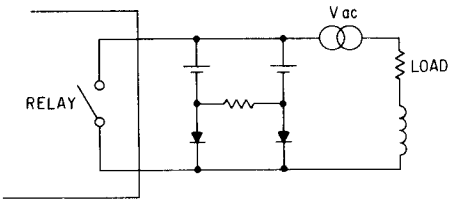


Fig. 9.12 For ac inductive loads, this snubber circuit can help protect the relay contacts on the 88-PCI board.

The type of load the relay drives has a lot to do with the contact lifetime. Simple resistive loads are the easiest to handle since they don't cause any reactive voltages or currents that cause arcs and thus damage to the contacts as the contacts open and close. Special circuits, though, can extend the contact life to sometimes double or triple the calculated time by protecting the relays from arcing as the contacts open and close. The arcing causes the contacts to rapidly wear and also causes undesirable electromagnetic and radio-frequency interference.

Protection circuits should be assembled in a chassis suitable to handle the power they must dissipate. Ideally, they should be located close to the relays to minimize the effects of the wire inductance. Inductance from a lamp or coil of any sort is the most dangerous type of load since as the magnetic field created by the current collapses (when the power is cut) it causes a large back voltage that can generate arcs between the relay contacts.

For dc loads, a series resistor and capacitor connected across the contacts can be used to minimize arcing (Fig. 9.10). Component values should be selected to suppress arcing but not affect load or relay performance. Unfortunately, the best way to select values is through empirical testing. The capacitance should be large enough for worst-case conditions, and the resistance should be large enough to limit the capacitance charge/discharge current. However, if the resistor is too small, the contacts will weld shut, and if too large, the capacitor's effect will be negated. A good starting point, though, can be determined with the capacitance value computed with this formula:

$$C = I^2/10$$

where C is in microfarads, and I is in amps.

Next, determine the time constant you would like for the RC network, and then determine the resistance from the formula

$$R = \tau/C$$

where τ is in microseconds, C is in microfarads, and R is in ohms.

Another alternative circuit for dc loads is shown in Fig. 9.11. When contact separation occurs, it gives a near-zero voltage drop across the relay contacts, even for highly inductive loads. The values of the capacitor and the diode are selected so that at the instant of separation, the peak voltage to which the capacitor charges will not cause breakdown of the diode, the contact gap, or the capactor itself.

When ac voltages are used to power loads, the arc caused by current continuing to flow extinguishes when the current passes through zero during the cycle. Thus, for ac resistive loads, contact protection for the relays is not as critical as for dc loads. An arc can last no longer than 8.3 milliseconds on a 60 Hz power line, since current reversals occur 120 times a second. And, the higher the power frequency, the shorter the duration of the arc. For ac inductive loads, the protection circuit of Fig. 9.12 shows how to minimize contact damage. This network allows the arc to extinguish naturally by making an inductive load appear resistive to the contacts.

If complete electrical isolation is needed between the load and the relay, the on-board optoisolators can be used. To interface the isolators to the load, there are several factors to consider—the voltage applied to the inputs, the switching time and propagation delays, and the method of transmitting the signals to the board. Pads on the 88-PCI board provide a circuit that can be adapted for a wide range of voltages and pulse widths. The optoisolator configuration is shown in Fig. 9.13. The isolators are dedicated to either input or output isolation functions.

To adapt input voltages for PIA sensing, the circuit of Fig. 9.14 can be used. However, several calculations must be performed to set the external component values that limit the LED voltage and current and the output current of the phototransistor. For the CA1 and CB1 PIA inputs, the input leakage current is 2.5 μA,

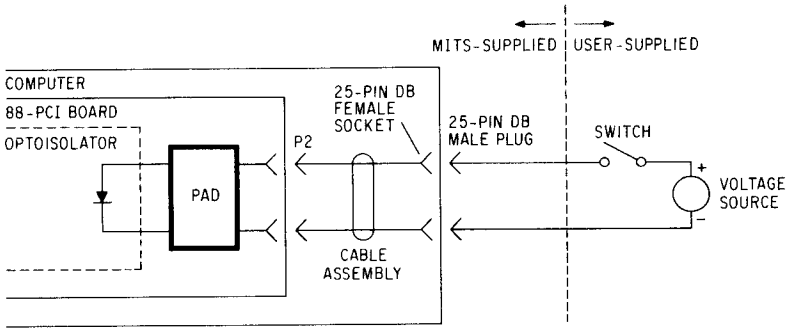


Fig. 9.13 If the 88-PCI board is purchased fully assembled, the optoisolators on it are set up in this configuration.

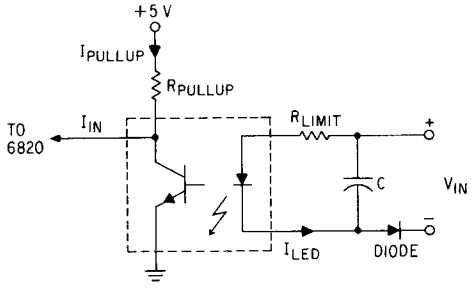


Fig. 9.14 This circuit can adapt the optoisolator inputs to handle any voltage level that a signal external to the 88-PCI board might present.

which is negligible, and the current into the PIA is virtually zero. Thus, the current through the pullup resistor (I_{pullup}) is equal to the transistor collector current (I_C). The input high voltage (V_{IH}) is 2 V (min), which makes the voltage across the pull-up resistor equal to the supply voltage (V_{CC}) minus V_{IH}, or 3 V. Next, the current in the pullup resistor can be calculated by dividing the voltage across the resistor by the resistance, and that turns out to be 3 V/2.2 kΩ = 1.36 mA (max). The particular optoisolators that are used have a current transfer ratio of 35%, which means that the current through the transistor collector represents approximately 35% of the current that flows through the LED. Thus, the current through the LED can now be found to be 3.9 mA (max).

Therefore, the maximum input current through the LED for a logic HIGH input is 3.9 mA. Now, the calculations for a logic LOW input must be performed. These calculations start with the determination of the logic LOW level, which for the PIA is 0.8 V. Thus, the voltage across the pullup resistor can be found by subtracting the PIA input voltage from the supply voltage (5 V − 0.8 V = 4.2 V). The minimum pullup resistor current can then be found by dividing the voltage by the resistance (4.2 V/2.2 kΩ = 1.91 mA). With the CTR of 35%, the LED current then becomes 5.45 mA.

Next, the isolators for the PIA inputs PA0 to PA7 can be set up by first calculating the current flow and voltage drops to compute the LED current. To start the calculations, begin by examining the PIA input parameters:

Input high current (I_{IH}) = −100 μA (min)
Input low current (I_{IL}) = −1.6 mA (max)
Input high voltage (V_{IH}) = 2 V (min)
Input low voltage (V_{IL}) = 0.8 V (max)

For a logic HIGH at the PIA input the pull-up current is the supply voltage minus V_{IH} with the result divided by the pull-up resistance, or

$$(5 - 2)/2.2\ k\Omega = 1.36\ mA$$

Now the phototransistor current can be found by subtracting the input HIGH current from the pull-up current, or

$$1.36\ mA - (-0.1\ mA) = 1.46\ mA$$

Finally, the LED current can be determined by dividing the transistor collector current I_C by the CTR

$$I_{LED} = 1.46/35\% = 4.17\ mA$$

When the PIA has a logic LOW on its input, the minimum pull-up current can be found by subtracting V_{IL} from the supply voltage and dividing the result by $R_{pull\text{-}up}$:

$$I_{pull\text{-}up} = (5 - 0.8)/2.2\ k\Omega = 1.91\ mA$$

Next, the I_C can be found by subtracting I_{IL} from $I_{pull\text{-}up}$:

$$I_C = 1.91\ mA - (-1.6\ mA) = 3.51\ mA$$

Now the LED current can be found by dividing I_C by the CTR:

$$I_{LED} = 3.51\ mA/35\% = 10.03\ mA$$

Table 9.7 summarizes the LED currents for active and inactive levels.

Table 9.7 Active and Inactive LED Voltage Levels for the 88-PCI

	For CA1, CB1			For PA0-PA7		
All in mA	min.	nominal	max.	min.	nominal	max.
I_{LED} for input LOW	5.45	10	100	10.03	15	10
I_{LED} for input HIGH	−0.1 μA	0	3.90	−0.1 μA	0	4.1

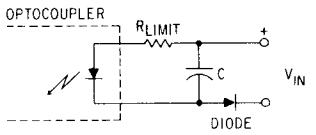


Fig. 9.15 When input signal pulses are too narrow to cause the LED to stay on long enough for the phototransistor, this circuit can stretch the pulse widths so the LED will cause the transistor to turn on.

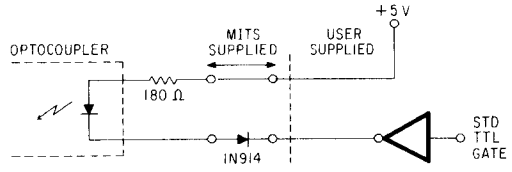


Fig. 9.16 To couple the optoisolators on the 88-PCI to standard TTL loads, this circuit adjusts all the voltages to the proper values.

Other set-up problems that can occur include signal pulses that are too narrow to cause the LED to light long enough to turn on the phototransistor. For the isolators on the 88-PCI to operate properly, input pulses must be greater than 20 μs in width. When narrower pulses are input, the circuit shown in Fig. 9.15 can help widen the pulses. Pads are available on the board for the necessary components to be added. The values of R and C that reshape the pulse are best determined by cut-and-try methods, but for a basic procedure start with some values that result in a time constant (RC product) equal to the pulse width.

First, the capacitor must charge to the maximum input voltage less the forward voltage of the diode ($V_{in} - V_D$) when the pulse occurs. As the input voltage returns LOW, the following formula can help determine the LED current:

$$I_{LED} = [(V_{CC} - 1.3)/R]\ e^{(-\tau/RC)}$$

where τ is the pulse duration in seconds, V_{CC} the pulse height, R the limiting resistance, C the capacitance, and e the natural log bsae (2.71838). Because the LED has a maximum reverse current of 0.1 μA, the external diode should be chosen for low reverse leakage current.

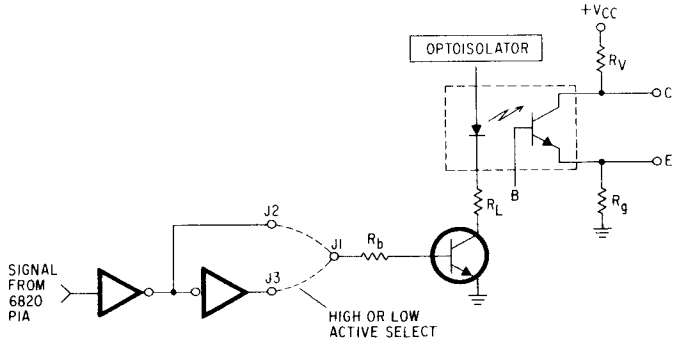


Fig. 9.17 For the PIA control lines, the output optocouplers can be set up like this to provide a load drive capability of about 12 mA at 20 V.

On the 88-PCI board, the inputs are set up by the factory for standard TTL-level interfaces, as shown in Fig. 9.16. In this form, the TTL gate outputs will sink approximately 15 mA in the LOW state and 5 to 10 μA in the HIGH state.

The output side of the PIA has some similar requirements for set up, except that there are more limited input conditions for the optoisolators and more flexible requirements for the phototransistor collector outputs. Figure 9.17 shows the optoisolator connected in a typical output configuration for the CA2 and CB2 outputs. When driven in this manner, the phototransistor can handle up to 12.6 mA when the LED is driven at its maximum current. The phototransistor should have no more than 20 V placed across the collector-emitter junctions (BV_{CEO}) to prevent burnout. Also, current levels should not exceed the overall safety limits of the isolator—a total power dissipation of 200 mW (LED plus phototransistor). When the LED is handling a 36 mA maximum current, it dissipates about 47 mW and thus the transistor dissipation should be limited to about 150 mW to avoid overheating.

Although the optoisolators can provide isolation of 1000 V or more, they are limited in their frequency response depending upon the type of load they must drive. The switching time of the phototransistor is directly proportional to the load resistance R_L and the open-circuit base capacitance of the transistor C_{ob}. In most phototransistors, relatively long time constants occur in the output circuit because the typical value of C_{ob} is approximately 25 pF. This time constant will be proportional to the transistor's current gain β. For example, for a 500 Ω load, a phototransistor with a typical minimum β of 100 and a C_{ob} of 25 pF produces a time constant of 1.25 μs. Switching time requires five time constants for a total of 6.25 μs, which corresponds to a bandwidth of just 130 kHz. With most optocouplers in use today, load resistances of 1000 Ω or more will severely limit the switching signals to frequencies of 500 kHz and less. For most applications, this shouldn't pose a problem; however, make sure it is adequate for the system being designed.

There are some things that can be done to speed up the optocouplers—removing the stored charge in the transistor's base circuit. To do this, a base-to-emitter bleed resistor can be added (see Fig. 9.18). This resistor will, however, reduce the coupler's sensitivity and current transfer ratio along with cutting the switching time. The limiting resistor value is zero ohms, which shorts the base of the phototransistor to its emitter, thus creating a photodiode. At this point the speed is the fastest possible, but the sensitivity has reached its minimum value.

Connecting the phototransistor to a load means that the collector and emitter leads must be wired to power supplies, ground terminals, resistors, or other components to provide the desired signal levels for the

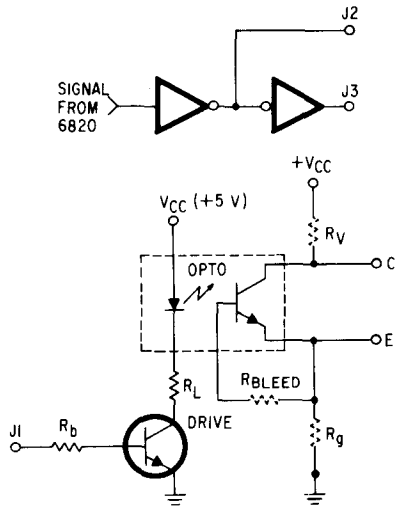


Fig. 9.18 To speed up the phototransistor response, a bleed resistor can be added between the base and emitter terminals of the optoisolators on the 88-PCI board.

Table 9.8 Address and Control Line Status Definition for the 88-PCI

Register Selection						
RS0	A0	RS1	A1	CRA-2	CRB-2	Register Selected
1	0	0	0	X	X	Control/status register (section A)
1	0	1	1	0	X	Data direction register (section A)
1	0	1	1	1	X	Data register (section A)
0	1	0	0	X	X	Control/status register (section B)
0	1	1	1	X	0	Data direction register (section B)
0	1	1	1	X	1	Data register (section B)

application. Pads on the board are provided for both a collector resistor and an emitter resistor to simplify the connection of the transistor to both the power supply and ground buses. Factory recommendations advise that the pads only be used where short cable runs are used and where supply isolation is unimportant. Generally, the power supply of the system connected to the transistor should be used to supply the bias levels to permit total isolation from the computer system. Some examples of the various interface connections are shown in Fig. 9.19.

After the hardware is set up for the application, the next step is to prepare the software necessary for the computer to control the board. To do the software initialization, the registers in the PIA must be loaded with the set-up information. Table 9.8 defines the board address and control-line status necessary to set up the data direction registers and the control/status registers. The DDRs tell the PIA whether to set up the I/O lines as inputs or outputs and the C/S registers control the functions of the CA1, CB1, CA2, and CB2 signal lines.

The C/S registers also determine whether to write to the data register or the DDR. Before data can be written to the DDR, bit 2 of the C/S register must be set to 0. This is done by writing a 0 to the C/S register of the PIA section (A or B) desired. The section A DDR is enabled by writing 0s to the board base address ("A" C/S register); and the B section by writing 0s to the base address + 2 ("B" C/S register).

Thus, with bit 2 set to 0, the next time the data channels (base address + 1, and base address + 3) are written to, the data are loaded into the DDR. Section A connects to the optoisolator inputs and section B to the relays. Therefore, section A must be set as input lines

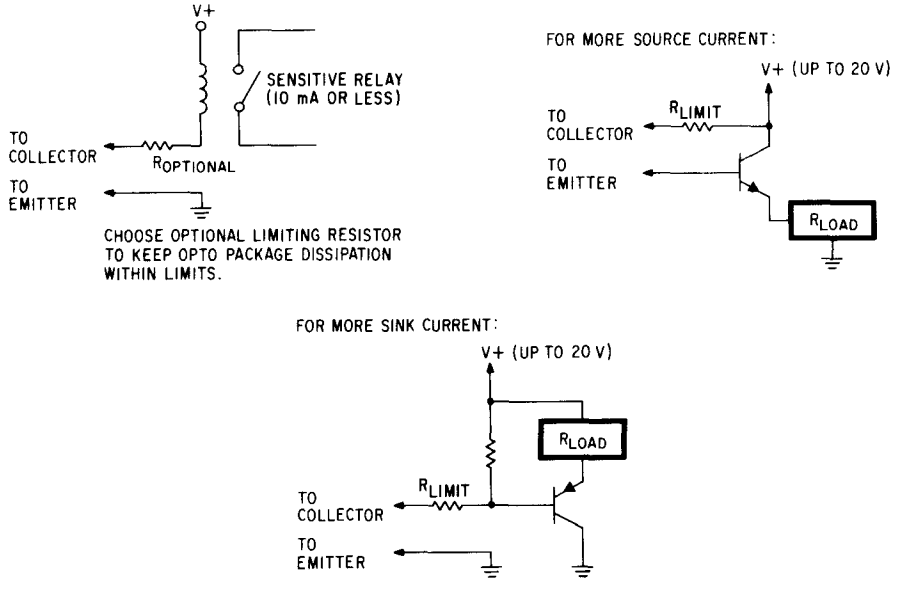


Fig. 9.19 Some typical optoisolator interface examples.

Table 9.9a Control Line Definitions for the 88-PCI

Bit No.	7	6	5 4 3	2	1 0
C/S section A	IRQA 1 flag	IRQA 2 flag	CA2 control	DDR-A access	CA1 control
C/S section B	IRQB 1 flag	IRQB 1 flag	CB2 control	DDR-B access	CB1 control

Table 9.9b Control Line Definitions for the 88-PCI

C/S bit 1	C/S bit 2	CA1 or CB1	Intr. flag C/S bit 7	MPU $\overline{IRQ}$
0	0	↓ active	Set HIGH	Disabled
0	1	↓ active	Set HIGH	Goes LOW when C/S bit 7 goes HIGH
1	0	↑ active	Set HIGH	Disabled
1	1	↑ active	Set HIGH	Goes LOW when C/S bit 2 goes HIGH

↓ = from HIGH to LOW signal transition
↑ = from LOW to HIGH signal transition
1 = Logic HIGH
0 = Logic LOW

Section A: C/S bit 7 (Int. flag) is reset (LOW) by a read of A section data channel register.

Section B: C/S bit 7 (Int. flag) is reset (LOW) by a read of B section data channel register.

Table 9.9c Control Line Definitions for the 88-PCI

CB2:

C/S bit 5	C/S bit 4	C/S bit 3	Cleared	Set
1	0	0	Low on ↑ of first E pulse after write of B data channel register after C/S bit 7 is reset by a read of B data channel register.	HIGH when Int. flag (C/S bit 7) is SET.
1	0	1	LOW on ↑ of first E pulse after a write of B section data channel register.	HIGH on ↑ of next E pulse.
1	1	0	Always LOW when bit 3 LOW.	
1	1	1		Always HIGH when bit 3 is HIGH.

CA2 and CB2 differ slightly in function.

and B as output lines. To initialize section A (at base address + 1) as an input, 0s should be written to the DDR, and to initialize section B as outputs (at base address + 3), 1s should be written into the respective DDR. After DDR initialization, the C/S register must be set to 1, thus preventing the data in the DDRs from being altered. Tables 9.9a, b, c, and d show the various options for the control lines and their interaction with the interrupt-request lines and the status bit in the C/S register.

Table 9.9d Control Line Definitions for the 88-PCI

For CA2:

C/S bit 5	C/S bit 4	C/S bit 3	Cleared	Set
1	0	0	LOW on ↓ of E pulse after read of A data channel register.	HIGH when Int. flag (C/S bit 7) is set.
1	0	1	LOW on ↓ of E pulse after read of A data channel register.	HIGH on ↓ of first E pulse which occurs while device is deselected.
1	1	0	Same as CB2.	
1	1	1		Same as CB2.

Now that all the aspects of electrical set-up for the 88-PCI have been examined, it's time to look at how the computer actually performs control functions. To to that, let's examine some simple examples that illustrate some of the things the board can do. However, before going through any example, some programming assumptions must be made. For the sake of consistency from program to program, the board base address will be assumed to be at octal address 100 (optoisolator input control status), the base address + 1 is location 101 (optoisolator input data + DDR), the base address + 2 is location 102 (relay control control/status), and base address + 3 is location 103 (relay control data + DDR).

The programs in Table 9.10a and b contain just the initialization procedures to set up the PIA so that the A section lines are inputs, all B section lines are outputs, and the C/S lines of sections A and B are set up as in Tables 9.9a, b, c, and d. With the PIA initially set up, the next step is to move data into or out of the computer through the PIA. To bring data into the PIA, the program in Table 9.11 brings all eight isolated lines into the computer's accumulator, masks the desired bits, and then performs a conditional jump or subroutine call operation by testing the result.

Output routines that turn a specific bit on or off without affecting any others are shown in Tables 9.12a and b. And to set up the state of all relays with a single byte, the program segment of Table 9.13 can be used.

Depending on the application, there are an almost unlimited number of functions possible for the 88-PCI board. For household applications, input uses include thermostats, alarm sensors, clocks, manual switches, and remote control transducers. By using the computer to monitor these different input signals, output items such as heaters, air conditioners, alarms, appliances, and stereo equipment can readily be controlled. In many industrial applications, inputs such as strain gauges, limit switches, pressure transducers, thermostats, position sensors, and many other sensors can be used by the computer to control various applications such as

Table 9.10a Assembly Language Initialization Procedure for the 88-PCI

Address	Contents	Mnemonic	Description
000	076	MVI → A	Load all 0's to Accumulator A
1	000	data	
2	323	OUT	Zero control/status register (making bit 2 = 0, giving access to DDR)
3	100	address	
4	323	OUT	Write zeros to DDR of Section A (making PA lines inputs)
5	101	address	
6	323	OUT	Zero B control/status register (make bit 2 = 0, giving access to B DDR)
7	102	address	
010	076	MVI → A	Load all 1s to Accumulator
1	377	data	
2	323	OUT	Write 1s to DDR of Section B (make PB lines outputs)
3	103	address	
4	076	MVI → A	Load Accumulator with 00100100 bit pattern
5	044	data	
6	323	OUT	Write 044 (octal) to A C/S set function
7	100	address	
020	323	OUT	Write 044 (octal) to B C/S set function
1	102	address	

Table 9.10b BASIC Initialization Procedure for the 88-PCI

```
Out 64, 0
Out 65, 0
Out 66, 0
Out 67, 255 (377 octal = 255 decimal)
Out 64, 36 (044 octal = 36 decimal)
Out 66, 36
```

Table 9.11 Simple Input Subroutine for the PIA on the 88-PCI

Address	Contents	Mnemonic	Description
n	333	IN	Input from A data (opto inputs)
n + 1	101	address	
n + 2	346	ANI	Logical AND with immediate data (mask for bits of interest)
n + 3	XXX	data	
n + 4	302 or 312	JNZ or JZ	Conditional jump or subroutine call
n + 5	XXX	low address	
n + 6	XXX	high address	

Table 9.12a Routine to Turn On a Specific Output Bit on the 88-PCI

Address	Contents	Mnemonic	Description
n	333	IN	Read output register for existing condition
n + 1	103	address	
n + 2	366	ORI	OR with data of bits to turn on (to turn on bit 0, data would be 001 octal)
n + 3	XXX	data	
n + 4	323	OUT	Output new control word
n + 5	103	address	

Table 9.12b Routine to Turn Off a Specific Output Bit on the 88-PCI

Address	Contents	Mnemonic	Description
n	333	IN	Read existing output register
n + 1	103	address	
n + 2	346	ANI	AND with complement of bits to be turned off (to turn off bit 0, data would be 376 octal)
n + 3	XXX	data	
n + 4	323	OUT	Output new control word
n + 5	103	address	

Table 9.13 Routine to Output a Full Byte on the 88-PCI
(When you know what state you want the relays in, the output is simple):

Address	Contents	Mnemonic	Description
n	076	MVI → A	Load on and off bit pattern to accumulator
n + 1	XXX	data	
n + 2	323	OUT	Output to relay control channel
n + 3	103	address	

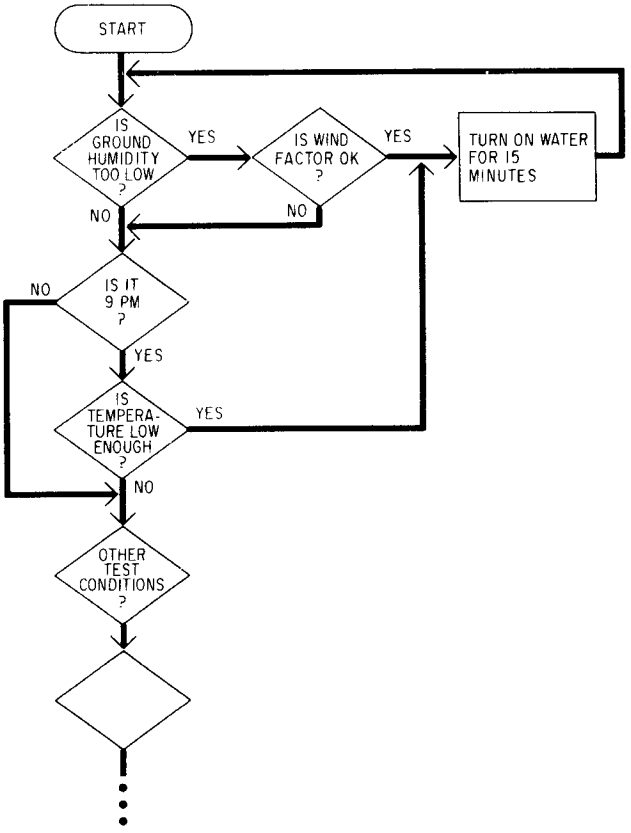


Fig. 9.20 Flowchart of a lawn sprinkler system example using the 88-PCI.

tool motor power, positioning motors, solenoid valves, heaters, solenoid driver rejectors, and solenoid stampers.

One example of a specific application is a lawn sprinkler control system. In this application, the computer will monitor the moisture level of the lawn and when the level drops below a specified value, the computer will turn on the water until the moisture level returns to the desired level. However, just an on/off control system won't do. Other factors such as the amount of water absorption versus the level of evaporation should be taken into account. Moisture can be measured by use of a humidistat buried in the lawn. A measurement of the wind speed can be used as a simple parameter to gauge the water absorption factor, and a simple go/no-go switch can be used to inhibit watering if the wind speed is too high to permit efficient use of the water (water is evaporating as fast or faster than it is being absorbed).

Other factors can further add to system complexity—temperature and relative humidity also have an effect on evaporation. And these parameters can be directly sensed and then fed into the computer through the 88-PCI. In any given 24-hour period, optimum temperature and humidity usually occur concurrently at night; thus a photocell or other light sensor can be used to tell the computer whether it is day or night outside.

When the lawn is too dry to wait for optimum watering conditions, a more complex humidistat or additional humidistats set for lower moisture levels can be used to signal these conditions. Also, separately controllable valves can be used if different sections of the lawn are to be watered at different times. Often, the cost for water might differ depending upon the time of the day it is used. So, to use the water only during the lower cost periods, a clock can be connected as an input to tell the computer the time of day. Of course, there are many other options that can be added to the system, and many features that can be taken away, depending on the complexity of the software controlling the system. The more features, the more complex the program. A very simple flowchart that shows some of the conditions and control functions mentioned is shown in Fig. 9.20.

There are, of course, many other types of interface boards that can be used with the Altair or Imsai or any other S-100 bus computer, but they are too numerous to list or cover in the space of a chapter or two. In fact, many of the boards have instruction manuals that could almost be books in themselves. These two boards—the 88-AD/DA and the 88-PCI—were selected to illustrate how the computer can be used to perform various monitoring and control functions because of their general-purpose capabilities.

CHAPTER 10

Troubleshooting the Microcomputer System

With a complex system such as the Altair 8800b or the Imsai 8080 microcomputer, there are many different things that can go wrong. Electronic circuits are prone to failure if overloaded, operated at too high a temperature, or from a variety of other causes. Troubleshooting a system like the Altair is a nightmare, especially if you don't know where to begin. You can, of course, return the equipment to the manufacturer and wait several weeks to get it repaired. To minimize the amount of time the system remains down (disabled), several spare boards and even ICs might be kept on hand. But which boards and which ICs?

Picking the right components is like trying to predict the future. However, there are some essential boards that, if they fail, would stop the entire system from operating. For example, the CPU card should have a spare—it's not that expensive a component and without the CPU the entire system is useless. Similarly for at least one of the serial and one of the parallel interfaces. Also, memory boards should be selected of a size such that one section of the memory can be removed and there is still enough memory capacity for the computer to perform even minimally. (This last spare can usually be accomplished by keeping an 8- or 16-kbyte memory board as a spare.) Although the temptation to put all the memory on as few boards as possible is great, until prices for the large memory chips come down the smaller 8- and 16-kbyte boards are still very economical. The only reason to really compact the memory is if you need all the other card slots for peripheral I/O control, data acquisition, or output.

Depending upon what ails the system, you might be able to readily discern the problem; then again, you may not. If whatever went wrong with the system doesn't permit you to operate any of the equipment, then troubleshooting the system will be like a game of blindman's buff. Sometimes, the entire system may not work, but you'll get a telltale sign of trouble—smoke. It's not unusual for a component to actually burn up from either too much stress (overloads), or possibly just due to faulty manufacturing. In either case, when you get such an indication, it may not just direct you to that component, but also to any of the components that connect directly to it.

A chain reaction effect occurs quite often in the power control sections of the computer. If a component being supplied power by a regulator should short out, the regulator could, in fact, be overloaded and also burn out. Thus, not only is there a faulty component, but a broken regulator as well. This type of chain reaction is very dangerous since many different circuits can be affected. A reaction in reverse can be even more damaging—if a voltage regulator fails and the unregulated voltage reaches the ICs, many of the circuits could be rapidly destroyed since the voltage that powers them goes above the maximum recommended value. Depending on the circuit being powered by the regulator it could be a matter of repairing several ICs or throwing out what was a $700 memory board.

Decide on the Level of Repair You'll Handle

Depending on your technical background, you may or may not want to tackle the repair of a system that is down. Several aspects of the technical troubleshooting will be examined later in this chapter, but first, let's take a look at the simplest level of system diagnosis possible beyond the phrase "it doesn't work." Assuming that when the power is turned on, the system doesn't give off any smoke or look like it will explode, let's start analyzing what problems may exist. First, let's backtrack for a moment and define a typical system. A typical system configuration suitable for small business and large personal applications would consist of the following:

1. A system mainframe with control front panel, power supply, and motherboard large enough to hold at least 10 S-100 compatible cards.
2. A central processor card (the examples given in this chapter assume an 8080A-based CPU card, although there are many Z-80 and other types of processors available.
3. Enough random-access read/write memory for the operating software (let's assume 32 kbytes, all lo-

Hexadecimal version		Octal version	
00	3E	000	076
01	03	001	003
02	D3	002	323
03	10	003	020
04	3E	004	076
05	11	005	021
06	D3	006	323
07	10	007	020
08	DB	010	333
09	10	011	020
0A	0F	012	017
0B	D2	013	322
0C	08	014	010
0D	00	015	000
0E	DB	016	333
0F	11	017	021
10	D3	020	323
11	11	021	021
12	C3	022	303
13	08	023	010
14	00	024	000

Fig. 10.1 Echo routine for the Altair 8800b computer to communicate with a serial terminal via the 88-2SIO board.

cated at addresses 0000 through 7FFF). There should also be some PROM with the bootstrap programs located at the top of the memory (F000 to FFFF).

4. At least one serial I/O port set up to operate in conjunction with a CRT terminal.

5. At least one parallel port set up to deliver its output to a printer.

6. A floppy-disk controller and at least one floppy-disk drive.

7. Optionally, a cassette recorder interface and tape recorder instead of the floppy-disk drive.

8. Finally, the operating system software. This software should permit the user to develop programs in a high-level language such as BASIC, and control all the interfaces to the printer, disk drive (or tape recorders), and the CRT terminal.

If the system doesn't work when power is turned on and the appropriate initialization procedures are performed, it's time to start tracking down the problem. Where to start, you may ask. Well, the best starting point is the front panel of the computer. Both the Altair 8800b and Imsai 8080 have switches on the panel that can be used to check out several functions of the computer—memory locations can be examined, their contents altered, and then even sent out to a specific port. If the switches on the computer permit you to perform these minimal functions, then the CPU is working, the basic memory array is functional, and the control panel is operating. However, if you can't even do these simple operations, you've already narrowed the problem to the CPU, memory, front panel, power supplies, or motherboard (card cage).

The next step is to check out the peripheral interfaces. There are some very simple machine-language programs that permit you to type in a character on a CRT terminal and get the same character outputted by the computer back to the terminal (Fig. 10.1). This "echo" procedure lets you check out not only the serial interface, part of the memory, and the CPU, but the terminal as well. Assuming that you could enter the routine via the front panel and the computer goes into a loop waiting for the character typed into the terminal, you know the CPU, the front panel, and the memory are all functional. The only part of the system in question is then the terminal or the serial interface. If they check out, then the problem might be either in the section of the system that loads the operating system from either the disk drive or the tape drive into the main memory, or it could be the bootstrap program, or it could be that part of the memory that the operating system gets loaded into is faulty.

Assuming the serial interface and the terminal check out, the most likely suspects for the fault are the disk (tape) memory or the bank's read/write memory. Checking out the disk drive is a very complicated procedure since special small subroutines must be loaded into the computer to check out the operation of the stepping motor that moves the read/write head inside the drive, the read and write circuits themselves, and the control of the other drive operations. Checkout of the memory is another matter.

Check the Memory with Software

In order to check the memory, either the cards can be pulled out of the system and plugged into another operating system to run a special diagnostic program, or a special diagnostic program can be loaded into a known good section of the computer memory and then run on the section of the memory in question. Of course, if the test program can't be loaded into the computer to test the memory then you have partially narrowed down the problem to a specific memory card or section of memory. Diagnostic test programs for memory boards, such as the one shown in Fig. 10.2, usually can pinpoint the problem as a specific bit or word that is bad within the entire array. Then, the defective memory chips can be replaced and the board can be put back into full operation after a complete diagnostic has been run with no faults being found.

If the system can operate to the point of permitting BASIC to run, then the short program shown in Fig. 10.3 can be used to test an 8192 byte memory card by using the RND instruction. The board to be tested by this program must be set so that its address is in some range above the existing memory. To run the program, the starting and final address of the memory board to be tested must be entered in decimal format. Several minutes are required to test the memory board. After

```
0000                    0010 CONC   EQU   0              CONSOLE STAT PORT
0000                    0020 COND   EQU   1              CONSOLE DATA PORT
0000                    0030 SPTR   EQU   0100H          STACK POINTER
0000 31 00 01           0040 START  LXI   SP,SPTR
0003 CD 37 00           0050        CALL  CRLF
0006 3E 2A              0060        MVI   A,'*'          PRINT "*"
0008 CD 2B 00           0070        CALL  PTCN
000B C3 4F 00           0080        JMP   TMEM
000E                    0090 *
000E                    0100 *** CONVERT UP TO 4 HEX DIGITS TO BIN
000E                    0110 *
000E 21 00 00           0120 AHEX   LXI   H,0            GET 16 BIT ZERO
0011 0E 04              0130        MVI   C,4            COUNT OF 4 DIGITS
0013 CD 41 00           0140 AHE1   CALL  RDCN           READ A BYTE
0016 29                 0150        DAD   H              SHIFT 4 LEFT
0017 29                 0160        DAD   H
0018 29                 0170        DAD   H
0019 29                 0180        DAD   H
001A D6 30              0190        SUI   48             ASCII BIAS
001C FE 0A              0200        CPI   10             DIGIT 0-10
001E DA 23 00           0210        JC    ALF
0021 D6 07              0220        SUI   7              ALPHA BIAS
0023 85                 0230 ALF    ADD   L
0024 6F                 0240        MOV   L,A
0025 0D                 0250        DCR   C              4 DIGITS?
0026 C2 13 00           0260        JNZ   AHE1           KEEP READING
0029 3E 20              0270 SPCE   MVI   A,20H          PRINT SPACE
002B F5                 0280 PTCN   PUSH  PSW            SAVE REG A
002C DB 00              0290 PTLOP  IN    CONC           READ PRTR STATUS
002E E6 80              0300        ANI   80H            IF BIT 7 NOT 0,
0030 C2 2C 00           0310        JNZ   PTLOP          WAIT TILL TIS
0033 F1                 0320        POP   PSW            THEN RECOVER A
0034 D3 01              0330        OUT   COND           AND PRINT IT
0036 C9                 0340        RET   RETURN         FROM PTCN
0037 3E 0D              0350 CRLF   MVI   A,0DH          PRINT CR
0039 CD 2B 00           0360        CALL  PTCN
003C 3E 0A              0370        MVI   A,0AH
003E C3 2B 00           0380        JMP   PTCN
0041                    0390 *
0041                    0400 *** READ FROM CONSOLE TO REG A ***
0041                    0410 *
0041 DB 00              0420 RDCN   IN    CONC           READ KB STATUS
0043 E6 01              0430        ANI   1              IF BIT 1 NOT 0
0045 C2 41 00           0440        JNZ   RDCN           REPEAT UNTIL IT IS
0048 DB 01              0450        IN    COND           READ FROM KB
004A E6 7F              0460        ANI   7FH            STRIP OFF MSB
004C C3 2B 00           0470        JMP   PTCN           ECHO ONTO PRINTER
004F                    0480 *
004F                    0490 *** MEMORY TEST ROUTINE ***
004F                    0500 *
004F CD 0E 00           0510 TMEM   CALL  AHEX           READ BLK LEN
0052 EB                 0520        XCHG                 PUT IN D,E
0053 CD 0E 00           0530        CALL  AHEX           READ ST ADD
0056 01 5A 5A           0540        LXI   B,5A5AH        INI B,C
0059 CD 83 00           0550 CYCL   CALL  RNDM
005C C5                 0560        PUSH  B              KEEP ALL REGS
005D E5                 0570        PUSH  H
005E D5                 0580        PUSH  D
005F CD 83 00           0590 TLOP   CALL  RNDM
0062 70                 0600        MOV   M,B            WRITE IN MEM
0063 23                 0610        INX   H              INC POINTER
0064 1B                 0620        DCX   D              DECR COUNTER
```

Fig. 10.2 Memory test program for location 0000H to 00FFH used by Vector Graphic Corp. It is excerpted from their monitor program and is designed to run in the lowest 256 bytes of the computer's memory.

```
0065 7A          0630          MOV   A,D          CHECK D,E
0066 B3          0640          ORA   E            FOR ZERO
0067 C2 5F 00    0650          JNZ   TLOP         REPEAT LOOP
006A D1          0660          POP   D
006B E1          0670          POP   H            RESTORE ORIG
006C C1          0680          POP   B            VALUES OF
006D E5          0690          PUSH  H
006E D5          0700          PUSH  D
006F CD 83 00    0710 RLOP     CALL  RNDM         GEN NEW SEQ
0072 7E          0720          MOV   A,M          READ MEM
0073 B8          0730          CMP   B            COMP MEM
0074 C4 A4 00    0740          CNZ   ERR          CALL ERROR ROUT
0077 23          0750          INX   H
0078 1B          0760          DCX   D
0079 7A          0770          MOV   A,D
007A B3          0780          ORA   E
007B C2 6F 00    0790          JNZ   RLOP
007E D1          0800          POP   D
007F E1          0810          POP   H
0080 C3 59 00    0820          JMP   CYCL
0083             0830 *** THIS ROUTINE GENERATES RANDOM NOS ***
0083 78          0840 RNDM     MOV   A,B          LOOK AT B
0084 E6 B4       0850          ANI   0B4H         MASK BITS
0086 A7          0860          ANA   A            CLEAR CY
0087 EA 8B 00    0870          JPE   PEVE         JUMP IF EVEN
008A 37          0880          STC
008B 79          0890 PEVE     MOV   A,C          LOOK AT C
008C 17          0900          RAL                ROTATE CY IN
008D 4F          0910          MOV   C,A          RESTORE C
008E 78          0920          MOV   A,B          LOOK AT B
008F 17          0930          RAL                ROTATE CY IN
0090 47          0940          MOV   B,A          RESTORE B
0091 C9          0950          RET                RETURN W NEW B,C
0092             0960 *
0092             0970 *** ERROR PRINT OUT ROUTINE
0092             0980 *
0092 CD 37 00    0990 PTAD     CALL  CRLF         PRINT CR,LF
0095 7C          1000          MOV   A,H          PRINT
0096 CD B3 00    1010          CALL  PT2          ASCII
0099 7D          1020          MOV   A,L          CODES
009A CD B3 00    1030          CALL  PT2          FOR
009D CD 29 00    1040          CALL  SPCE         ADDRESS
00A0 CD 29 00    1050          CALL  SPCE
00A3 C9          1060          RET
00A4 F5          1070 ERR      PUSH  PSW          SAVE ACC
00A5 CD 92 00    1080          CALL  PTAD         PRINT ADD.
00A8 78          1090          MOV   A,B          DATA
00A9 CD B3 00    1100          CALL  PT2          WRITTEN
00AC CD 29 00    1110          CALL  SPCE
00AF CD 29 00    1120          CALL  SPCE
00B2 F1          1130          POP   PSW          DATA READ
00B3 F5          1140 PT2      PUSH  PSW
00B4 CD BB 00    1150          CALL  BINH
00B7 F1          1160          POP   PSW
00B8 C3 BF 00    1170          JMP   BINL
00BB 1F          1180 BINH     RAR
00BC 1F          1190          RAR
00BD 1F          1200          RAR
00BE 1F          1210          RAR
00BF E6 0F       1220 BINL     ANI   0FH          LOW 4 BITS
00C1 C6 30       1230          ADI   48           ASCII BIAS
00C3 FE 3A       1240          CPI   58           DIGIT 0-9
00C5 DA 2B 00    1250          JC    PTCN
00C8 C6 07       1260          ADI   7            DIGIT A-F
00CA C3 2B 00    1270          JMP   PTCN
```

Fig. 10.2 (cont'd) Memory test program for location 0000H to 00FFH used by Vector Graphic Corp. It is excerpted from their monitor program and is designed to run in the lowest 256 bytes of the computer's memory.

```
SYMBOL TABLE

AHE1  0013   AHEX  000E   ALF   0023   BINH  00BB   BINL  00BF   CONC  0000
COND  0001   CRLF  0037   CYCL  0059   ERR   00A4   PEVE  008B   PT2   00B3
PTAD  0092   PTCN  002B   PTLOP 002C   RDCN  0041   RLOP  006F   RNDM  0083
SPCE  0029   SPTR  0100   START 0000   TLOP  005F   TMEM  004F

D 0000 00CF
0000  31 00 01 CD 37 00 3E 2A CD 2B 00 C3 4F 00 21 00
0010  00 0E 04 CD 41 00 29 29 29 29 D6 30 FE 0A DA 23
0020  00 D6 07 85 6F 0D C2 13 00 3E 20 F5 DB 00 E6 80
0030  C2 2C 00 F1 D3 01 C9 3E 0D CD 2B 00 3E 0A C3 2B
0040  00 DB 00 E6 01 C2 41 00 DB 01 E6 7F C3 2B 00 CD
0050  0E 00 EB CD 0E 00 01 5A 5A CD 83 00 C5 E5 D5 CD
0060  83 00 70 23 1B 7A B3 C2 5F 00 D1 E1 C1 E5 D5 CD
0070  83 00 7E B8 C4 A4 00 23 1B 7A B3 C2 6F 00 D1 E1
0080  C3 59 00 78 E6 B4 A7 EA 8B 00 37 79 17 4F 78 17
0090  47 C9 CD 37 00 7C CD B3 00 7D CD B3 00 CD 29 00
00A0  CD 29 00 C9 F5 CD 92 00 78 CD B3 00 CD 29 00 CD
00B0  29 00 F1 F5 CD BB 00 F1 C3 BF 00 1F 1F 1F 1F E6
00C0  0F C6 30 FE 3A DA 2B 00 C6 07 C3 2B 00 2B 00 C6
```

Fig. 10.2 (cont'd) Memory test program for location 0000H to 00FFH used by Vector Graphic Corp. It is excerpted from their monitor program and is designed to run in the lowest 256 bytes of the computer's memory.

```
 30  INPUT "HIGH MEMORY ADD.";H
 40  INPUT "LOW MEMORY ADD.";L
 50  PRINT "LOCATION", "WROTE", "READ"
 60  A=RND (1)
 70  B=RND (-A)
 80  FOR N=L TO H
 90  POKE N, INT (256 * RND (1))
100  NEXT
110  B=RND (-A)
120  FOR N=L TO H
130  IF PEEK (N) = INT (256*RND (1)) GO TO 150
140  PRINT N, INT (256*RND (0)), PEEK (N)
150  NEXT
160  PRINT "CHECK OK"
170  GO TO 60
RUN
HIGH MEMORY ADD.? 20479
LOW MEMORY ADD.? 8192
LOCATION        WROTE        READ
CHECK OK
CHECK OK
CHECK OK
```

Fig. 10.3 A simple memory test program in BASIC that can be used if the system's operating system can still run with the terminal.

the test, the program outputs the phrase CHECK OK if all memory locations appear good and then continues testing the board. A thorough test of the board requires about 10 passes. If an error occurs, the location is printed out along with the number written into the memory and the number read out from the memory.

In many cases, however, your system will probably not be able to run BASIC. But, you might be able to load a machine-language program, either from a serial port or via the front panel, that can perform a similar test function. The program shown in Fig. 10.2 is designed to be entered and operate via a terminal with a minimal amount of memory and effort. It is excerpted from the monitor program developed by Vector Graphic Corp. and is assembled to run in the lowest 256 bytes of memory. Execution should start at location 0000_{16}. An asterisk will be output if the routines are entered properly. PTCN is the output routine for a Processor Technology 3P + S I/O board (with the status inverted) that is used in the computer system to communicate with the terminal. The same routine can be used for the Pertec rev 1 SIO. RDCN is the input routine. If you are using a board with a programmable USART, you must also initialize it in addition to changing the mask, jump condition, and port bits.

After the * is output, enter four hex characters for the length of the memory block to be tested (2000 for an 8k board) and four digits for the starting address of the board. Spacing is automatic. Typing any characters other than 0 to 9 or A to F will cause the program to do strange things. A reset will terminate the execution of the test. The program itself generates a $2^{16} - 1$ byte pseudorandom number sequence, writes a portion of it in the block of memory, and then regenerates the sequence from the same point to compare with what is read from memory. If the pass is correct, a new portion of the sequence is written into memory and checked. Errors are printed out with the address, what was written into memory, and what was read out. Possible memory problems include solder bridges between data or address lines, chips mounted on the board backward, poor seating in the socket, broken printed-circuit patterns, a defective chip, or just a poor solder connection. If you cannot locate the defective memory chip visually or by using the program on the entire board, try removing half the memory chips (assuming they're all mounted

in sockets) and running the test program on the reduced block size. Keep reducing the block size until the defect can be found.

Replacing the defective component can be a relatively simple job of pulling it out of a socket and plugging in a replacement. However, before installing a new component, make sure there are no other defects on the board that will cause the new part to fail.

If you've a mind to do all your own circuit troubleshooting, you'll have to make a pretty heavy investment in test equipment. In addition to the workhorse multimeter, you'll need a dual-channel 20-MHz scope, a logic probe, a timing generator, an extender card, some tiny test clips, several logic clips, and some power supplies. Unless you have a pretty thorough knowledge of digital electronic circuits and test techniques, you should not attempt to service your own equipment aside from possibly replacing a board. Most equipment purchased comes with a factory guarantee that covers just about every fault that can occur. Even local computer stores usually offer repair services that can do rush repair jobs when you can't wait for normal factory repair turnarounds.

Software Bugs: Gremlins That Can Drive You Crazy

If, after all the testing of the hardware, it seems to be operating properly, but you just can't get that new program you bought or wrote to run properly, you've got software bugs. Depending on the language you're working in, there are different levels of problems that might be in the software. For instance, if a program runs on a friend's system but not on yours, there are two possible problems. First, you might be loading it into an area of memory that will throw off all the addresses referred to in the program. For instance, a fixed program instruction that tells the processor to fetch the data from address XXXX will not find the information in that location if the loading procedure actually put the data in location YYYY.

Another possibility is that the system is configured differently than the one on which the program runs—the ports used for terminals, printers, or auxiliary devices could be assigned differently. All this makes the use of someone else's program quite chancy. Adjusting the port addresses is a necessity when you are trying to use a high-level language such as BASIC. Pertec, for instance, assumes a specific system configuration—if the system configuration is different, changes in the assembly-level routines must be made so that when the program outputs a command to a specific peripheral, that peripheral will receive the instruction and data.

Whether you work in machine, assembly, or high-level languages, there are three possible problems that can occur when you try to run a program and it doesn't work:

```
10    LET X=1
20    PRIMT X
30    END
RUN
SYNTAX ERROR
```

Fig. 10.4 A simple program that contains an error.

1. Errors in the program prevent the program from running at all.
2. Errors in the program permit it to start running then abruptly terminate at an unexpected point.
3. Errors that permit the program to run completely but cause the computer to produce an incorrect result.

The process of correcting a program that does not work is called debugging and often requires much patience since many of the problems are not immediately apparent. Since techniques of troubleshooting software are similar for machine, assembly, and high-level languages, let's look at some of the problems you could face when trying to debug programs that are written in BASIC.

Some of the simplest errors that prevent a program from running are typing errors that can occur when programs are entered on a terminal. For instance, the simple program shown in Fig. 10.4 will cause an error message from most BASIC operating systems since there is a spelling error in one of the lines (line 20). Altair Disk BASIC provides various error messages for many of the common operating errors that can occur. After an error does occur, BASIC returns to the command level and types OK. Variable values and the program text remain intact, but the program cannot be continued by the CONT command. In the 4k and 8k versions of BASIC, all GOSUB and FOR context is lost. However, the program can be continued by direct mode GOTO commands. When an error occurs in a direct statement, no line number is printed. Error messages have the following format:

Direct statement error:	?XX ERROR
Indirect Statement:	?XX ERROR IN YYYYY

where XX is the error code and YYYYY is the line number where the error occurred. There are about 45 error messages available in BASIC from Pertec to help you diagnose a faulty program. Each error message consists of an error code, an extended error message, and a number. For the extended Disk BASIC, all error codes apply. Here are the available messages that can help you track down a bug in your program:

BS	Subscript out of range	9

The indicator means that an attempt was made to reference an array element that is outside the dimensions of the array. This message will also appear if the wrong number of dimensions is used in an array reference.

DD Redimensioned array 10

After an array is dimensioned another dimension statement for the same array will cause this error indicator to appear. This error often occurs if an array has been given the default dimension of 10 and later in the program another statement tries to dimension the array.

FC Illegal function call 5

The parameter passed to a math or string function was out of range. FC errors can occur due to:

1. a negative array subscript
2. an unreasonably large array subscript (greater than 32,767)
3. LOG with a negative or zero argument
4. SQR with a negative argument
5. A**B with A negative and B not an integer
6. a call to USR before the address of a machine language subroutine has been entered
7. calls to MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, or ON . . . GOTO with an improper argument

ID Illegal direct 12

INPUT and DEF are illegal in the direct mode except for INPUT in the extended version of BASIC.

NF Next without FOR 1

The variable in a NEXT statement corresponds to no previously executed FOR statement.

OD Out of data 4

This code indicates that a READ statement was executed but all of the data statements in the program have already been read. Either the program tried to read too much data or insufficient data were included in the program.

OM Out of memory 7

This indicator appears when the program is too large or has too many variables, too many FOR loops, too many GOSUBs, or too complicated expressions.

OV Overflow 6

This indicates that the result of a calculation was too large to be represented in the format available to the computer. If an underflow occurs, zero is given as the result and execution continues without any error message being printed.

SN Syntax error 3

Missing parenthesis in an expression, illegal character in a line, incorrect punctuation, etc., will cause this error code to be printed out.

RG Return without GOSUB 3

A RETURN statement was encountered before a previous GOSUB statement was executed; the program has nowhere to return to.

UL Undefined line 8

The line reference in a GOTO, GOSUB, IF . . . THEN . . . ELSE, or DELETE was to a line that does not exist.

/0 Division by zero 11

This error indicator can occur with integer division and MOD as well as floating point division. Zero to a negative power also causes a /0 error.

CN Can't continue 17

Attempt to continue a program when no continuation exists, an error occurred, or after a modification was made to the program.

LS String too long 15

An attempt was made to create a string more than 255 characters long.

OS Out of string space 14

String variables exceed amount of string space allocated for them. Use the CLEAR command to allocate more string space or use smaller strings or fewer string variables.

ST String formula too complex 16

A string expression was too long or too complex. Break it into two or more shorter expressions.

TM Type mismatch 13

The left-hand side of an assignment statement was a numeric variable and the right-hand side was a string, or vice versa. Or, a function that expected a string argument was given a numeric one or vice versa.

UF Undefined user function 18

Reference was made to a user-defined function that had never been defined.

Missing operand 20

During evaluation of an expression, an operator was found with no operand following it.

No resume 19

BASIC entered an error-trapping routine, but the program ended before a RESUME statement was encountered.

Resume without error 21

A RESUME statement was encountered, but no error-trapping routine had been entered.

Unprintable error 22

An error condition exists for which there is no error

message available. Probably there is an ERROR statement with an undefined error code.

Line buffer overflow 23

An attempt was made to input a program or data line that has too many characters to be held in the line buffer. Shorten the line or divide it into two or more parts.

Field overflow 50

An attempt was made to allocate more than 128 characters of string variables in a single FIELD statement.

Internal error 51

Internal error in Disk BASIC. Report conditions under which errors occurred and all relevant data to Pertec software department. This error can also be caused by certain kinds of disk I/O errors.

Bad file 52

An attempt was made to use a file number that specifies a file that is not open or that is greater than the number of files entered during the initialization dialog.

File not found 53

Reference was made in a LOAD, KILL, or OPEN statement to a file that did not exist in the disk specified.

Bad file mode 54

An attempt was made to perform a PRINT to a random file, to open a random file for sequential output, to perform a PUT or GET on a sequential file, to load a random file, or to execute an OPEN statement where the file mode is not I, O, or R.

File already open 55

A sequential output mode OPEN for a file was issued for a file that was already open and had never been closed, or a KILL statement was given for an open file.

Disk not mounted 56

An I/O operation was issued for a file that was not on a disk already in place.

Disk I/O error 57

An I/O error occurred in disk X; a sector read (checksum) error occurred 18 consecutive times.

Set to non-disk string 58

An LSET or RSET was given for a string variable that had not been previously mentioned in a FIELD statement.

Disk already mounted 59

A MOUNT was issued for a disk that was already mounted but never unloaded.

Disk full 60

All disk storage is exhausted on the current disk; delete some old files and try again.

Input past end 61

An INPUT statement was executed after all the data on a file had been input. This will happen immediately if an INPUT is executed for a null (empty) file. Use of the EOF function to detect end of file will avoid this error.

Bad record number 62

In a PUT or GET statement, the record number is either greater than the allowable maximum (2046) or equal to zero.

Bad file name 63

A file name of 0 characters (null) or a file name whose first byte was 0 or 277_8 (255_{10}), or a file name with more than eight characters was used as an argument to LOAD, SAVE, KILL, or OPEN.

Mode mismatch 64

Sequential OPEN for output was executed for a file that already existed on the disk as a random (R) mode file, or vice versa.

Direct statement in file 65

A direct statement was encountered during a LOAD of a program in ASCII format; the LOAD is terminated.

Too many files 66

A SAVE or OPEN (O or R) was executed that would create a new file on the disk, but all 255 directory entries were already full; delete some files and try again.

Out of random blocks 67

An attempt was made to have more random files open at once than the number of random blocks that were allocated during initialization by the response to the "NUMBER OF RANDOM FILES?" question.

File already exists 68

The new file name specified in the NAME statement has the same name as another file that already exists on the disk; try using a different name.

File link error 69

During the reading of a file, a sector was read that did not belong to the file.

Now, armed with all the error codes, you can diagnose the problems that come up. However, knowing what the problem is and being able to do something about it are two totally different things. For instance, to correct the problem in the program shown in Fig. 10.4, all you must do is replace line 20 with the correctly spelled command—PRINT X. When the program is run the first time with the error, the computer will

```
10    READ A AND B
20    PRINT A,B
30    DATA 4,5,7,-11
40    GO TO 10
50    END
RUN
SYNTAX ERROR
```

(A)

```
10    READ A,B
RUN
4          5
7          -11
```

(B)

Fig. 10.5 Sample program with a syntax error (a). Corrected program and resultant output (b).

```
10    LET A=10
20    LET B=5
30    LET C=AB
40    PRINT C
50    END
RUN
```

Fig. 10.6 Multiplication program that contains an error.

probably come back with a syntax error in line 20 error statement. Similarly for the program shown in Fig. 10.5a. There is an error in line 10; the READ statement cannot have the structure shown. To correct the statement simply type: 10 READ A,B and hit the carriage return and give the RUN command again; the program will now function properly. The READ statement as originally defined is read by BASIC as just a READ A and the characters after the A are ignored. When corrected, the printout as shown in Fig. 10.5b results.

Another common omission in mathematical expressions are the various symbols necessary for the computer to perform the desired operations. For instance, the program shown in Fig. 10.6 has the computer multiply two numbers and print out the result. However, line 30 contains the product expression, but does not have the asterisk between the A and B, thus not signifying the desired multiplication. By replacing line 30 with LET C = A*B the program will run properly. If you are not sure that the program has totally correct statements that conform to BASIC format, use the LIST command—any non-BASIC statement will automatically be eliminated and your listing will consist of only totally BASIC-compatible statements.

To help troubleshoot programs that are under development, it could be quite useful to include breakpoints—instructions in the program that cause it to stop execution so that you can check some intermediate values. If you decide against breakpoints, another possibility is for the program to print out some comments as it gets further down the line and give you some indication, aside from the error statements, as to how far the program was executed. For instance, the program in Fig. 10.7 will start to run and then terminate after line 70 since it runs out of data to read for the READ X statement. Not all computers will print out the error statement after the last data statement has been read. But, they will automatically terminate execution. Illegal operations such as trying to raise 0 to the zero power will result in execution termination and there are many such cases that only experience will help you avoid.

```
10    PRINT "PROGRAM EXECUTION HAS BEGUN"
20    LET X=2
30    READ A,B,C
40    PRINT A*B↑X
50    PRINT "C=";C
60    PRINT "PROGRAM HAS REACHED LINE 60"
70    READ Y
80    PRINT Y
90    DATA 3,17,11
100   END
RUN
PROGRAM EXECUTION HAS BEGUN
867
C=11
PROGRAM HAS REACHED LINE 60

OUT OF DATA IN LINE 70
```

Fig. 10.7 Simple program with breakpoints that help monitor program operation.

There are other programming errors that will permit a program to run, but provide erroneous results. Some simple examples of such programs are shown in Fig. 10.8. Here are two examples of small things that can go wrong in programs. In Fig. 10.8a, there are four LET statements but nowhere is there a PRINT statement so that the computer will output the results of the calculations in the LET statements. To correct this problem all that must be done is the addition of a PRINT statement after the calculations, say in line 45, that might read PRINT A; B; C; D. The second program, shown in Fig. 10.8b, is a bit more complex. There is nothing really wrong with it except that every time it loops through the READ and PRINT statements it repeats the column headings—a waste of time and paper. So, to amend this program, just change the address of the GOTO statement of line 60 so that it loops back to the line just after the PRINT statement. Thus, line 60 should read GOTO 20 and the column headings will only be printed out the first time the program goes through its routine. But there's still another problem in the program—every time two numbers are read and used, the pairs indicator should increment so the first time the printout should read 1 PAIRS SO FAR, 2 PAIRS SO FAR the next time, and so forth. The problem stems

```
10    LET A=10
20    LET B=5
30    LET C=A↑2
40    LET D=B↑2
50    END
RUN

DONE
```

```
10    PRINT "X    Y    SUM PROD"
20    LET Z = 0
30    READ X,Y
40    LET Z=Z+1
50    PRINT X;Y;X-Y:X*Y;Z;" PAIRS SO FAR"
60    GO TO 10
70    DATA 10,20,11,9
80    DATA 1,2,-45,18
90    END
RUN

A          B          SUM        PROD
10         20         30         200        1 PAIRS SO FAR

A          B          SUM        PROD
11         4          20         99         1 PAIRS SO FAR

A          B          SUM        PROD
1          2          3          2          1 PAIRS SO FAR

A          B          SUM        PROD
-45        18         -27        -81        1 PAIRS SO FAR

OUT OF DATA IN LINE 30
```

Fig. 10.8 Some simple program errors are in these programs. Although these programs will run on the computer, they will produce incorrect results.

from the LET statement in line 20 which is looped back to each time the program finishes its PRINT routine. Again, line 60 must be changed so that the program loops back to line 30, thus avoiding the reprinting of the column headings and the resetting of the Z counter on every loop. The revised line 60 should then read GOTO 30.

Whenever you write programs, try to make sure that there are always lines for the program to loop back to, that there are always END statements, and that there are always enough data to be read in. Arrays should always be dimensioned and never redimensioned in the same program. The best teacher, of course, is experience. But unless you make the mistakes and go through the trouble of debugging your own programs, you'll never really understand the problems of writing and running your own programs.

There are several tricks you can use to help minimize the number of programming errors you might make. For instance, you can use a scratch pad and pencil to keep track of the GOSUB, RETURN, GOTO, CALL, and other branching instructions that are often the places where errors occur. By keeping this separate list, you have, at a glance, all the locations where program flow can divert to. Similarly, you can keep a list of variable names, thus making sure you don't duplicate the names you are already using.

There are dozens of books and courses available to you that can provide much more detail about the ins and outs of programming in BASIC and other languages. In fact, by the time this book is in your hands, more powerful languages, such as FORTRAN, PASCAL, COBOL, and FORTH will readily be available for the microcomputer user.

APPENDIX A

Reference Books and Publications

Basic Electronics and Integrated Circuits

Herrick, Clyde. *Survey of Electronics.* New York: The MacMillan Co., 1973.

Jackson, Herbert. *Introduction to Electric Circuits.* Englewood Cliffs, NJ: Prentice-Hall, 1976.

Lenk, John. *Manual for Integrated Circuit Users.* Reston, VA: Reston Publishing Co., 1973.

____. *Manual for MOS Users.* Reston, VA: Reston Publishing Co., 1975.

Lewis, Rhys. *Solid State Devices and Applications.* London: Butterworth & Co., 1971.

Luecke, Gerald; Mize, Jack; and Carr, William. *Semiconductor Memory Design and Application.* New York: Texas Instruments and McGraw-Hill, 1973.

The MacMillan Co. *Understanding Solid State Electronics.* New York: The Macmillan Co., 1973.

Mandl, Matthew. *Solid-State Circuit Designer's Manual.* Reston, VA: Reston Publishing Co., 1977.

Mooris, Robert, and Miller, John. *Designing with TTL Integrated Circuits.* New York: Texas Instruments and McGraw-Hill, 1971.

Roney, Peter, and Larsen, David. *The Bugbooks Volumes I and II.* Derby, CT: E & L Instruments, 1974.

Ryder, J., and Thomson, C. *Electronic Circuits and Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1976.

Schure, A. *Basic Transistors.* Rochelle Park, NJ: Hayden Book Co., 1976.

Stanley, George. *Transistor Basics.* Rochelle Park, NJ: Hayden Book Co., 1967.

Toners, T. *Practical Solid State Power Supplies.* Blue Ridge Summit, PA: Tab Books, 1977.

Programming

Albrecht, Bob. *BASIC.* New York: Wiley, 1978.

Brown, Jerald. *Instant BASIC.* Menlo Park, CA: Dymax, 1977.

Cassel, Don. *BASIC Programming in Real Time.* Reston, VA: Reston Publishing Co., 1975.

Coan, James. *Advanced BASIC.* Rochelle Park, NJ: Hayden Book Co., 1976.

Coan, James. *Basic BASIC.* Rochelle Park, NJ: Hayden Book Co., 1970.

Cook, G.; Wade, B.; and Upton, C. *Computer Accounting Methods.* New York: Petrocelle Books, 1976.

Elliot, Ronald. *Problem Solving and Flow Charting.* Reston, VA: Reston Publishing Co., 1972.

Findley, Robert. *8080 Software Gourmet Guide and Cookbook.* Milford, CT: Scelbi Computing.

Graham, Neil. *Microprocessor Programming for Computer Hobbyists.* Blue Ridge Summit, PA: Tab Books, 1977.

Larsen, D.; Roney, P.; and Titus, J. *The Bugbooks Volumes VI and VII.* Derby, CT: E & L Instruments, 1977.

McCabe, Dwight, ed. *PCC's Reference Book of Personal & Home Computing.* Menlo Park, CA: Peoples Computer Co., 1977.

McMurran, M. *Programming Microprocessors.* Blue Ridge Summit, PA: Tab Books, 1977.

Peoples Computer Company. *What to Do After You Hit Return, or PCC's First Book of Computer Games.* Menlo Park, CA: Peoples Computer Co., 1975.

Poole, Lon, and Borchers, Mary. *Some Common BASIC Programs.* Berkeley: Adam Osborne & Associates, 1977.

Scelbi. *Galaxy Game for the 8080/8080.* Milford, CT: Scelbi Computing, 1977.

Scientific Research Instruments. *BASIC Software Library. My Computer Likes Me . . . When I Speak BASIC.*

Schoman, Kenneth, Jr. *The BASIC Workbook: Creative Techniques for Beginning Programmers.* Rochelle Park, NJ: Hayden Book Co., 1977.

Smith, Robert. *Discovering BASIC.* Rochelle Park, NJ: Hayden Book Co., 1970.

Spencer, Donald. *A Quick Look at BASIC.* Ormond Beach, FL: Camelot Publishers, 1977.

____. *Game Playing with BASIC.* Rochelle Park, NJ: Hayden Book Co., 1977.

____. *Sixty Challenging Problems with BASIC Solutions.* Ormond Beach, FL: Camelot Publishers, 1977.

Van Tessel, Dannie. *Program Style, Design, Efficiency, Debugging and Testing.* Englewood Cliffs, NJ: Prentice-Hall, 1974.

Walter, Russ. *The Secret Guide to Computers.* Boston: Walter Publishing Co., 1978.

Microprocessors

Barden, William, Jr. *How to Buy and Use Minicomputers and Microcomputers.* Indianapolis: Howard Sams & Co., 1976.

Bishop, Ron. *Basic Microprocessors and the 6800.* Rochelle Park, NJ: Hayden Book Co., 1979.

Lancaster, Don. *TV Typewriter Cookbook.* Indianapolis: Howard Sams & Co., 1976.

Martin, Donald. *Microcomputer Design.* Northbrook, IL: Martin Research, 1976.

Osborne, Adam. *An Introduction to Microcomputers.* Vol. I. Basic Concepts. Berkeley: Adam Osborne & Associates, 1976.

———. *6800 Programming for Logic Design.* Berkeley: Adam Osborne & Associates, 1976.

———. *8080 Programming for Logic Design.* Berkeley: Adam Osborne & Associates, 1976.

———. *An Introduction to Microcomputers.* Vol. II. Some Real Products. Berkeley: Adam Osborne & Associates, 1977.

Rockwell, Charles. *An Introduction to Microprocessors.* Microlog, 1977.

Solomon, L., and Veit, S. *Getting Involved with Your Own Computer: A Guide for Beginners.* Short Hills, NJ: Enslow Publishers, 1977.

Waite, Mitchell, and Pardee, Michael. *Microcomputer Primer.* Indianapolis: Howard Sams & Co., 1976.

Zaks, Rodnay, and Lesea, Austin. *An Introduction to Personal Computing.* Berkeley: Sybex, 1977.

———. *Bit Slices.* Berkeley: Sybex, 1977.

———. *Industrial Microprocessor Systems.* Berkeley: Sybex, 1977.

———. *Interfacing Techniques.* Berkeley: Sybex, 1977.

———. *International Microprocessor Dictionary.* Berkeley: Sybex, 1977.

———. *Microprocessor Interfacing Techniques.* Berkeley: Sybex, 1977.

———. *Microprocessors.* Berkeley: Sybex, 1977.

———. *Microprocessors, From Chips to Systems.* Berkeley, Sybex, 1977.

———. *Military Microprocessor Systems.* Berkeley: Sybex, 1977.

———. *Programming and Microprogramming.* Berkeley: Sybex, 1977.

Cassette Sources

Introduction to Microprocessors. Berkeley: Sybex, 1977.

Programming Microprocessors. Berkeley: Sybex, 1977.

Advanced Digital Theory and Computer Architecture

Chirlian, Paul. *Analysis and Design of Digital Circuits and Computer Systems.* Champaign, IL: Matrix Publishers, 1976.

Abd-Elfallah, M., Abd Alla; and Arnold, Meltzer. *Principles of Digital Computer Design.* Englewood Cliffs, NJ: Prentice-Hall, 1976.

Favert, Andrew. *Digital Computer Principles and Applications.* New York: Van Nostrand Reinhold, 1972.

Fuori, William. *Introduction to the Computer.* Englewood Cliffs, NJ: Prentice-Hall, 1977.

Gothmann, William. *Digital Electronics.* Englewood Cliffs, NJ: Prentice-Hall, 1977.

Gschwind, Hans, and McCluskey, Edward. *Design of Digital Computers.* New York: Springer-Verlag, 1975.

Lenk, John. *Handbook of Logic Circuits.* Reston, VA: Reston Publishing Co., 1972.

Libes, Sol. *Fundamentals and Applications of Digital Logic Circuits.* Rochelle Park, NJ: Hayden Book Co., 1975.

Mowle, Fredric. *A Systematic Approach to Digital Logic Design.* Reading, MA: Addison-Wesley, 1976.

Peatman, John. *The Design of Digital Systems.* New York: McGraw-Hill, 1972.

Roney, P.; Larsen, D.; and Titus, J. *The Bugbooks Volumes III and V.* Derby, CT: E & L Instruments, 1976.

Sloan, M. *Computer Hardware and Organization.* Chicago: Science Research Associates, 1976.

Stone, Harold. *Introduction to Computer Architecture.* Chicago: Science Research Associates, 1975.

Tocci, Ronald. *Digital Systems Principles and Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1977.

Troubleshooting

Gilmore, Charles. *Understanding and Using Modern Electronic Servicing Test Equipment.* Blue Ridge Summit, PA: Tab Books, 1976.

Horowitz, Mannie. *How to Troubleshoot and Repair Electronic Test Equipment.* Blue Ridge Summit, PA: Tab Books, 1974.

Lenk, John. *Handbook of Practical Electronic Tests and Measurements.* Englewood Cliffs, NJ: Prentice-Hall, 1969.

Mandl, Matthew. *Handbook of Electronic Testing, Measurement and Troubleshooting.* Reston, VA: Reston Publishing, 1976.

Risse, Joe. *Electronic Test Equipment and How to Use It.* Blue Ridge Summit, PA: Tab Books, 1974.

Publications

Byte, Byte, Inc., Division of McGraw-Hill, Peterborough, NH

Creative Computing, Ideametrics, Morristown, NJ

Dr. Dobb's Journal of Computer Calisthenics and Orthodontia, Menlo Park, CA

Interface Age, McPheters, Wolfe & Jones, Cerritos, CA

Kilobaud, 1001001, Inc., Peterborough, NH

On Computing, Byte, Inc., Division of McGraw-Hill, Peterborough, NH
On-Line, Los Gatos, CA
Periodical Guide for Computerists, E. Berg Publications, Aloha, OR
Personal Computer World, Intra Press, London, England
Personal Computing, Benhill Publishing, Boston, MA
Popular Electronics, Ziff-Davis, New York, NY
Radio-Electronics, Gernsback Publications, New York, NY
Recreational Computing, People's Computer Co., Menlo Park, CA
73 Magazine, 73, Inc., Peterborough, NH

APPENDIX B

Commonly Used Components and Suppliers of S-100 Bus Systems

Low-Power Schottky TTL

74LS00	Quad 2-input NAND gate
74LS01	Quad 2-input NAND gate (open collector output)
74LS02	Quad 2-input NOR gate
74LS03	Quad 2-input NAND gate (open collector output)
74LS04	Hex inverter
74LS05	Hex inverter (open collector output)
74LS08	Quad 2-input AND gate
74LS09	Quad 2-input AND gate (open collector output)
74LS10	Triple 3-input NAND gate
74LS11	Triple 3-input AND gate
74LS12	Triple 3-input NAND gate (open collector output)
74LS13	Dual 4-input NAND Schmitt triggers
74LS14	Hex Schmitt trigger
74LS15	Triple 3-input AND gate (open collector output)
74LS20	Dual 4-input NAND gate
74LS21	Dual 4-input AND gate
74LS22	Dual 4-input NAND gate (open collector output)
74LS26	Quad 2-input high-voltage NAND gate
74LS27	Triple 3-input NOR gate
74LS28	Quad 2-input NOR buffer
74LS30	8-input NAND gate
74LS32	Quad 2-input OR gate
74LS33	Quad 2-input NOR gate (open-collector output)
74LS37	Quad 2-input NAND buffers
74LS38	Quad 2-input NAND buffers (open collector outputs)
74LS40	Dual 4-input NAND buffers
74LS42	1-of-10 decoder
74LS47	BCD to 7-segment decoder/driver (open collector output)
74LS48	BCD to 7-segment decoder/driver
74LS51	Dual 2-wide, 2-input AND-OR-INVERT gate
74LS54	4-wide AND-OR-INVERT gate
74LS55	2-wide, 4-input AND-OR-INVERT gate
74LS73	Dual J-K flip-flop with clear
74LS74	Dual D positive edge-triggered flip-flop with preset and clear
74LS75	Quad D latch
74LS76	Dual J-K flip-flop with preset and clear
74LS78	Dual J-K flip-flop with preset and common clear and clock
74LS83	4-bit full adder
74LS85	4-bit magnitude comparator
74LS86	Quad Exclusive-OR gate
74LS90	BCD or decade counter
74LS92	Divide by 12 counter
74LS93	Binary counter
74LS95	4-bit shift register
74LS96	5-bit shift register
74LS107	Dual J-K master slave flip-flop with clear
74LS109	Dual J-$\overline{K}$ positive edge triggered flip-flop with preset and clear
74LS112	Dual J-K negative-edge-triggered flip-flop with preset and clear
74LS113	Dual J-K negative-edge-triggered flip-flop with preset
74LS114	Dual J-K negative-edge-triggered flip-flop with preset and common clear and clock
74LS122	Retriggerable one-shots with clear
74LS123	Dual retriggerable one-shots with clear
74LS124	Dual voltage-controlled oscillator
74LS125	Three-state quad buffers
74LS126	Three-state quad buffers
74LS132	Quad 2-input NAND Schmitt triggers
74LS133	13-input NAND gate
74LS136	Quad Exclusive-OR gate (open-collector outputs)
74LS138	1-of-8 decoder/demultiplexer
74LS139	Dual 1-of-4 decoder
74LS145	1-of-10 decoder
74LS148	8 line to 3 line encoder
74LS151	8-input multiplexer
74LS153	Dual 4-input multiplexer
74LS154	1-of-16 decoder

74LS155 Dual 1-of-4 decoder
74LS156 Dual 1-of-4 decoder (open collector output)
74LS157 Quad 2-input multiplexer, noninverting
74LS158 Quad 2-input multiplexer, inverting
74LS160 BCD counter with asynchronous reset
74LS161 4-bit binary counter with asynchronous reset
74LS162 BCD counter with synchronous reset
74LS163 4-bit binary counter with synchronous reset
74LS164 8-bit serial-in/parallel-out shift register
74LS168 Synchronous BCD decade up/down counter
74LS169 Synchronous 4-bit binary up/down counter
74LS170 4-bit by 4-bit register file (open collector output)
74LS173 Quad D register (three-state output)
74LS174 Hex D flip-flop with clear
74LS175 Quad D flip-flop with clear
74LS181 4-bit arithmetic and logic unit
74LS190 Decade up/down counter
74LS191 Binary up/down counter
74LS192 Up/down decade counter
74LS193 Up/down binary counter
74LS194 4-bit R/L shift register
74LS195 4-bit shift register
74LS196 Decade counter
74LS197 4-bit binary counter
74LS221 Dual one-shots with Schmitt trigger inputs
74LS240 Octal inverting driver (three-state output)
74LS241 Octal noninverting driver (three-state output)
74LS242 Quad inverting bus transceiver
74LS243 Quad noninverting bus transceiver
74LS244 Octal noninverting driver (three-state output)
74LS247 BCD to 7-segment decoder/driver (open collector output)
74LS248 BCD to 7-segment decoder/driver
74LS249 BCD to 7-segment decoder/driver (open collector output)
74LS251 8-input multiplexer (three-state output)
74LS253 Dual 4-input multiplexer (three-state output)
74LS257 Quad 2-input multiplexer (three-state output)
74LS258 Quad 2-input multiplexer (three-state output)
74LS259 8-bit addressable latch
74LS260 Dual 5-input NOR gates
74LS261 2-bit by 4-bit parallel binary multiplier
74LS266 Quad 2-input Exclusive-NOR gate (open collector output)
74LS273 Octal D latch with clear
74LS279 Quad S-R latch
74LS281 4-bit parallel binary accumulator
74LS283 4-bit full adder
74LS290 Decade counter
74LS293 4-bit binary counter
74LS295 4-bit shift register (three-state output)
74LS298 Quad 2-input multiplexer (open collector output)
74LS299 8-bit universal parallel-in/parallel-out shift register
74LS365 Hex buffer with three-state output and common enable
74LS366 Hex inverter with three-state output and common enable
74LS367 Hex buffer with three-state output (set up as 4-bit and 2-bit groups)
74LS368 Hex inverter with three-state output (set up as 4-bit and 2-bit groups)
74LS374 Octal D flip-flop (low-voltage output)
74LS375 4-bit bistable latch
74LS377 Octal D flip-flop with enable
74LS378 Hex D flip-flop with enable
74LS381 4-bit ALU/function generator
74LS386 Quad Exclusive-OR gate
74LS390 Dual decade counter
74LS393 Dual 4-bit binary counter
74LS395 4-bit cascadable shift register (three-state output)
74LS399 Quad 2-input multiplexer with storage
74LS490 Dual decade counter
74LS670 4-bit by 4-bit register file (three-state output)

4000 Series CMOS

4000 Dual 3-input NOR gate plus inverter
4001 Quad 2-input NOR gate
4002 Dual 4-input NOR gate
4006 18-bit static shift register
4007 Dual complementary pair plus inverter
4008 4-bit full adder
4011 Quad 2-input NAND gate
4012 Dual 4-input NAND gate
4013 Dual D flip-flop
4014 8-bit static shift register
4015 Dual 4-bit static shift register
4016 Quad analog switch/quad multiplexer
4017 Decade counter/divider
4018 Presettable divide-by-N counter
4020 14-bit binary counter
4021 8-bit static shift register
4022 Octal counter/divider
4023 Triple 3-input NAND gate
4024 Seven-stage ripple counter
4025 Triple 3-input NOR gate
4027 Dual J-K flip-flop
4028 BCD-to-decimal decoder
4032 Triple serial adder (positive logic)
4034 8-bit universal bus register
4035 4-bit shift register
4038 Triple serial adder (negative logic)
4040 12-bit binary counter
4042 Quad latch
4043 Quad NOR R-S latch
4044 Quad NAND R-S latch
4046 Phase-locked loop
4049 Hex inverter/buffer
4050 Hex buffer

4051 8-channel analog multiplexer
4052 Dual 4-channel analog multiplexer
4053 Triple 2-channel analog multiplexer
4066 Quad analog switch
4068 8-input NAND gate
4069 Hex inverter
4070 Quad Exclusive-OR gate
4071 Quad 2-input OR gate
4072 Dual 4-input OR gate
4073 Triple 3-input AND gate
4075 Triple 3-input OR gate
4076 Quad D-type register
4077 Quad Exclusive-NOR gate
4078 8-input NOR gate
4081 Quad 2-input AND gate
4082 Dual 4-input AND gate
4093 Quad 2-input NAND Schmitt trigger
4160 Decade counter (asynchronous clear)
4161 Binary counter (asynchronous clear)
4162 Decade counter (synchronous clear)
4163 Binary counter (synchronous clear)
4174 Hex D flip-flop
4175 Quad D flip-flop
4194 4-bit universal shift register
4408 Binary-to-phone pulse converter
4409 Binary-to-phone pulse converter
4410 2-of-8 tone encoder
4411 Bit-rate frequency generator
4412 Universal low-speed modem
4415 Quad precision timer/divider
4419 2-of-8 keypad-to-binary encoder
4422 Remote control transmitter
4431 12-bit a/d converter
4433 3-½ digit a/d converter
4435 3-½ digit a/d logic subsystem
4440 LCD watch/clock circuit
4450 Oscillator 2^{16} divider/buffer
4451 Oscillator/divider/buffer
4452 Digitally trimmed frequency divider
4490 Hex contact bounce eliminator
4501 Triple gate
4502 Strobed hex inverter/buffer
4503 Hex three-state buffer
4505 64 × 1-bit static RAM
4506 Dual expandable AOI gate
4507 Quad Exclusive-OR gate
4508 Dual 4-bit latch
4510 BCD up/down counter
4511 BCD-to-7-segment latch/decoder/driver
4512 8-channel data selector
4514 4-bit latch/4-to-16 line decoder (high)
4515 4-bit latch/4-to-16 line decoder (low)
4516 Binary up/down counter
4517 Dual 64-bit static shift register
4518 Dual BCD up counter
4519 4-bit AND/OR selector
4520 Dual binary up counter
4521 24-stage frequency divider
4522 Programmable BCD divide-by-N counter
4524 256 × 4-bit read-only memory
4526 Programmable binary divide-by-N counter
4527 BCD rate multiplier
4528 Dual monostable multivibrator
4529 Dual 4-channel analog data selector
4530 Dual 5-input majority logic gate
4531 12-bit parity tree
4532 8-bit priority encoder
4534 Real-time 5-decade counter
4536 Programmable timer
4537 256 × 1-bit static random access memory
4538 Dual precision monostable multivibrator
4539 Dual 4-channel data selector/multiplexer
4541 Programmable oscillator/timer
4543 BCD-to-7-segment latch/decoder/driver
4549 Successive approximation register
4552 64 × 4-bit static RAM
4553 3-digit BCD counter
4554 2 × 2 bit parallel binary multiplier
4555 Dual binary to 1-of-4 decoder
4556 Dual binary to 1-of-4 decoder (inverting)
4557 1 to 64-bit variable-length shift register
4558 BCD-to-7-segment decoder
4559 Successive approximation register
4560 NBCD adder
4561 9's complementer
4562 128-bit static shift register
4566 Industrial time base generator
4568 Phase comparator/programmable counter
4569 Dual programmable BCD/binary counter
4572 Hex gate
4580 4 × 4 multiport register
4581 4-bit arithmetic-logic unit
4582 Look-ahead carry block
4583 Dual Schmitt trigger
4584 Hex Schmitt trigger
4585 4-bit magnitude comparator

Microprocessor Manufacturers

Advanced Micro Devices
901 Thompson Place
Sunnyvale, CA 94086
(408) 732-2400

American Microsystems
3800 Homestead Road
Santa Clara, CA 95051
(408) 246-0330

Data General
Route 9
Southboro, MA 01772
(617) 485-9100

EMM Semiconductor
3883 N. 28th Avenue
Phoenix, AZ 85017
(602) 968-4431

Fairchild Semiconductor
464 Ellis Street
Mountain View, CA 94042
(415) 962-3816

Ferranti
Western Road, Bracknell
Berkshire, RG12 1RA, England

Fujitsu Ltd.
6-1, Marunouchi 2 Chome
Chiyoda-ku
Tokyo, Japan

General Instrument
600 West John Street
Hicksville, NY 11802
(516) 733-3130

Harris Semiconductor
P.O. Box 883
Melbourne, FL 32901
(305) 727-5400

Hitachi, Ltd.
Nippon Building
No. 6-2, 2 Chome,
Ohtemachie, Chiyoda-ku
Tokyo 100, Japan

Hughes Aircraft Co.
Microelectronic Products Div.
500G Superior Avenue
Newport Beach, CA 92663
(714) 548-0671

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051
(408) 987-8080

Intersil
10710 North Tantau Avenue
Cupertino, CA 95014
(408) 996-5000

ITT Semiconductors
Maidstone Road
Footscray Sidcup
Kent, England

Matsushita
1006 Kadoma, Osaka, Japan

Monolithic Memories
1165 East Arques Avenue
Sunnyvale, CA 94086
(408) 739-3535

MOS Technology
Valley Forge Corporate Center
950 Rittenhouse Road
Norristown, PA 19401
(215) 666-7950

Mostek
1215 West Crosby Road
Carrollton, TX 75005
(214) 242-0444

Motorola Semiconductor
3501 Ed Bluestein Boulevard
Austin, TX 78721
(512) 928-2600

Motorola Semiconductor
2002 West 10th Place
Tempe, AZ 85282
(602) 244-3466

National Semiconductor
2900 Semiconductor Drive
Santa Clara, CA 95050
(408) 737-5000

Nippon Electric Co.
33-1, Shiba Gochome
Minato-ku,
Tokyo 108, Japan

NEC Microcomputers
5 Militia Drive
Lexington, MA 02173
(617) 862-6410

RCA
Box 3200, Route 202
Somerville, NJ 08876
(201) 685-6423

Raytheon
350 Ellis Street
Mountain View, CA 94040
(415) 968-9211

Rockwell International
P.O. Box 3669
RCOI—Dept. 720
Anaheim, CA 92803
(714) 632-2321

Siemens AG
Central Information Dept.
Oskar-von-Miller Ring 18
D-8000 Munchen 2
Federal Republic of Germany

Signetics
811 East Arques Avenue
Sunnyvale, CA 94086
(408) 739-7700

Solid State Scientific
Montgomeryville Industrial Park
Montgomeryville, PA 18936
(215) 855-8400

Synertek
3050 Coronado Drive
Santa Clara, CA 95051
(408) 984-8900

Texas Instruments
13500 North Central Expressway
Dallas, TX 75222
(214) 238-2481

Thomson-CSF, Sescosem
101 Boulevard Murat
75781 Paris Cedex 16
France

Toko, Inc. (Toko America)
1-17 Higashiyukigaya
2 Chome, Ohta-ku
Tokyo 145, Japan

Toshiba Transistor Works
1 Komukai Toshiba-cho
Kawasaki-shi Kanagana-ken
Japan

Western Digital
3128 Red Hill Avenue
Newport Beach, CA 92663
(714) 557-3550

Zilog Microcomputers
10460 Bubb Road
Cupertino, CA 95014
(408) 446-4666

S-100 Computer and Board Manufacturers

Advanced Computer Products, P.O. Box 17329K, Irvine, CA 92713, (714) 558-8813 (memory cards)

Artec Electronics, 605 Old Country Rd., San Carlos, CA 94070, (415) 592-2740 (memory and breadboards)

Associated Electronics, 1885 W. Commonwealth, Unit G, Fullerton, CA 92633, (714) 879-7541 (memory boards)

Base 2, Inc., P.O. Box 9941, Marian DelRay, CA 90291, (213) 822-4499 (memory boards)

The Bit Stop, P.O. Box 973, Mobile, AL 36601

Canada Systems, P.O. Box 516, La Canada, CA 91011, (213) 790-7957 (clock and control boards)

CMC Marketing/TEI Corp., 7231 Fondren Rd., Houston, TX 77036, (713) 774-9526 (mainframe)

Central Data Corp., 1207 N. Hagan St., Champaign, IL 61820, (217) 359-8010 (memory boards)

Computalker Consultants, P.O. Box 1951, Dept. K, Santa Monica, CA 90406, (213) 392-5230 (speech synthesizer)

Compt/Time, P.O. Box 417, Huntington Beach, CA 92646, (714) 638-2094 (high-speed malt processor)

Cromemco, 2400 Charleston Road, Mt. View, CA 94043, (415) 964-7400 (full systems and boards)

Data Speed, Inc., 1302 Noe St., San Francisco, CA 94131 (disk controller)

Digital Group, P.O. Box 6528, Denver, CO 80206, (303) 777-7133 (cards and systems)

Digital Micro Systems, Box 1212, Orem, UT 84057, (800) 453-1444 (cards)

Digital Research Corp., P.O. Box 401247, Garland, TX 75040, (214) 271-2461 (memory cards)

Dynabtye, 4020 Fabian, Palo Alto, CA 94303, (415) 494-7817 (memory cards)

Electronic Control Technology, 763 Ramsey Avenue, Hillside, NJ 07205, (201) 686-8080 (card cages and boards)

Godbout Electronics, Box 2355 Oakland Airport, CA 94614, (415) 562-0636 (boards and parts)

D. C. Hayes Assoc., P.O. Box 9884, Atlanta, GA 30319, (404) 231-0574

Imsai, 14860 Wicks Blvd., San Leandro, CA 94577, (415) 483-2093 (full range of systems and cards)

Integrand, 8474 Ave. 296, Visalia, CA 93277, (209) 733-9288 (mainframe)

International Data Systems, 400 North Washington St., Suite 200, Falls Church, VA 22046, (703) 536-7373 (boards)

Intersystems, 1650 Hanshaw Rd., P.O. Box 91, Ithaca, NY 14850, (607) 257-0190 (full range of hardware) (formerly Ithaca Audio)

Jade Computer Products, 5351 West 144 St., Lawndale, CA 90260, (213) 679-3313 (memory and CPU boards and parts)

Matrox Electronic Systems, 5800 Andover Rd., Montreal Que. H4T1H4, (514) 481-6838 (display boards)

MITS (full range of systems and cards—see Pertec Computer Corp.)

MicroDesign, 679-I S. State College Blvd., Fullerton, CA 92631, (714) 870-9860 (memory boards)

Micronics, Box 3514, 123 West 3rd St., Suite 8, Greenville, NC 27834, (919) 758-7757 (software trap board)

Objective Design, P.O. Box 20325, Tallahassee, FL 32304, (904) 224-5545 (display card and cage)

Parasitic Engineering, Equinox Div., P.O. Box 6314, Albany, CA 94706 (800) 648-5311 (cards and mainframe)

Pertec Computer Corp., 20630 Nordhoff Street, Chatsworth, CA 91311 (213) 998-1800

Prime Radix, P.O. Box 11245, Denver, CO 80211, (303) 573-5942 (memory system)

Seals Electronics, 10728 Dutchtown Rd., Concord, TN 37922, (615) 966-8771 (memory boards, systems)

S. D. Sales, P.O. Box 28810-K, Dallas, TX 75228, (214) 271-0022 (memory, CPU boards, and parts).

S-100 Inc., 7 White Place, Clark, NJ 07066, (201) 382-1318 (boards)
Space Byte, 1720 Pontius Ave., Suite 201, Los Angeles, CA 90025, (213) 468-8080 (CPU boards)
Szerlip Enterprises, 1414 W. 259 St., Harbor City, CA 90710 (PROM program board)
Tarbell Electronics, 950 Dovlen Place, Suite B, Carson, CA 90746, (213) 538-2254 (cassette and disk interfaces, and breadboard)
Technical Design Labs., see Xitan.
Validity Corp., 4901 Morena Blvd., San Diego, CA 92117, (714) 272-7703 (interface boards)
Vanderberg Data Products, P.O. Box 2507, Santa Maria, CA 93454, (805) 937-7951 (memory boards)
Vector Graphic Inc., 31364 Via Colinas, Westlake Village, CA 91361, (213) 991-2302 (mainframe, boards, and disk system)
Wameco, 3107 Laneview Drive, San Jose, CA 95132 (boards)
Wasatch Semiconductor Products, 25 South 300 East, Suite 215, Salt Lake City, UT 84111 (memory board)
Xitan Corp., 1057 Main Street, Hanson, MA 02341 (systems and support boards—formerly known as Technical Design Labs)
Xybek, P.O. Box 4925, Stanford, CA 94305, (408) 296-8188 (programmer and memory cards)

Printers

Anadex, 9825 DeSoto Ave., Chatsworth, CA 91311, (213) 998-8010
Axiom, 5932 San Fernando Rd., Glendale, CA 91202, (213) 245-9244
Centronics Data Computer Corp., Hudson, NH 03051, (603) 883-0111
Comprint, 340 E. Middlefield Road, Mt. View, CA 94043, (415) 969-6161
Expandor Corp., 612 Beatty Rd., Dept. 101A, Monroeville, PA 15146, (412) 373-0300
MPI, Box 22161, Salt Lake City, UT 84122, (801) 566-0201
Practical Automation, Trap Falls Rd., Shelton, CT 06484, (203) 929-5381
TeleSpeed Communications, P.O. Box 647, Syosset, NY 11791
Texas Instruments Digital Systems Div., P.O. Box 1444, Houston, TX 77001, (713) 494-5115

CRT Terminals

CMC Marketing, 7231 Fondren Rd., Houston, TX 77036, (713) 774-9526
Computer Data Systems, 5460 Fairmont Dr., Wilmington, DE 19808, (302) 738-0933
Lear Siegler, 714 N. Brookhurst St., Anaheim, CA 92803, (714) 774-1010
Microterm, P.O. Box 9387, St. Louis, MO 63117, (314) 645-3856
Southwest Technical Products, 219 Rhapsody, San Antonio, TX 78216, (512) 344-0241
Xitex, P.O. Box 20887, Dallas, TX 75220, (214) 350-5291

Printing Terminals

Abacus Computer Systems, 6315 Eunice Ave., Los Angeles, CA 90042, (213) 666-1711
Anderson-Jacobson, 521 Charcot Ave., San Jose, CA 95131, (408) 263-8530
Center for the Study of the Future, 4110 N.E. Alameda, Portland, OR 97212
International Peripheral Systems, Inc., 1849 N. Helm, Fresno, CA 93727, (209) 252-3635
Sharp and Associates, Box 26045, Lakewood, CO 80226
Terminal Systems, 11300 Hartland St., North Hollywood, CA 91605, (213) 769-6772
Texas Instruments Digital Systems Div., P.O. Box 1444, Houston, TX 77001, (713) 494-5115

Breadboarding and Cabinets

A P Products, P.O. Box 110-4, Plainsville, OH 44077, (216) 354-2101
Bishop Graphics, 5388 Sterling Ctr., Box 5007, Westlake Village, CA 91359, (213) 991-2660
Continental Specialties, 44 Kendall Street, Box 1942, New Haven, CT 06509, (203) 624-3103
E & L Instruments, 61 First St., Derby, CT 06418, (203) 735-8774
Enclosure Dynamics, P. O. Box 6276, Bridgewater, NJ 08807, (201) 725-7982
Vector Electronic Products, 12460 Gladstone Ave., Sylmer, CA 91342, (213) 365-9661
Vero Electronics, 171 Bridge Rd., Hauppauge, NY 11787 (516) 234-0400

PROM Erasers

UltraViolet Products, 5100 Walnut Grove Ave., San Gabriel, CA 91778, (213) 285-3123

Floppy Disk Drive and Controller Manufacturers

Alpha Microsystems, 17875 Sky Park North, Irvine, CA, (714) 957-1404
Century Data, Div. of Xerox, 2411 West La Lalma Ave., Anaheim, CA 92801, (714) 821-2541
Icom Microperipherals, 6741 Variel Ave., Canoga Park, CA 91303, (213) 348-1391
MFE Corp., Keewaydin Drive, Salem, NH 03079, (603) 893-1921

Micromation, 524 Union St., San Francisco, CA 94133, (415) 398-0289
Micropolis, 7959 Deering Ave., Canoga Park, CA 91304, (213) 703-1121
MSD, Inc., 2765 S. Colorado Blvd., Denver, CO 80222, (303) 758-7411
North Star Computers, 2465 Fourth St., Berkeley, CA 94710
Peripheral Vision, P.O. Box 6267, Denver, CO 80206, (303) 777-4292
PerSci, 12210 Nebraska Ave., W. Los Angeles, CA 90025, (213) 820-3764
Pertec Computer Corp., 21111 Erwin St., Woodland Hills, CA 91367, (213) 999-2020
Sykes Datatronics, 375 Orchard St., Rochester, NY 14606, (716) 458-8000
Vista Computer, 2807 Oregon Court, Torrance, CA 90503, (213) 370-3880
Wango, 5404 Jandy Rd., Los Angeles, CA 90066, (213) 390-8081

Supplementary Products

Tape drives

Meca, 7026 O.W.S. Road, Yucca Valley, CA 92284, (714) 365-7686
Quantex Div., North Atlantic Industries, 200 Terminal Drive, Plainview, NY 11803, (516) 681-8350
Triple-I, 4605 North Stiles, Oklahoma City, OK 73118, (405) 521-9000

Tape controller

Ro-Che Systems, 7101 Mammoth Ave., Van Nuys, CA 91405

Paper-tape readers or punches

Addmaster Corp., 416 Junipero Serra Dr., San Gabriel, CA 91776, (213) 285-1121
Heathkit, Benton Harbor, MI 49022, (616) 982-3434
Decitek, 250 Chandler St., Worcester, MA 01602, (617) 798-8731
Oliver Advanced Engineering, Inc., 676 West Wilson Ave., Glendale, CA 91203, (213) 240-0080

I/O remote control

Mountain Hardware, P.O. Box 1133, Ben Lomand, CA 95005, (408) 336-2455

Video modulator

ATV Research, 13-K Broadway, Dakota City, NE 68731, (402) 987-3771

Logic analyzer

Databyte, P.O. Box 14, 7433 Hubbard Ave., Middleton, WI 53562, (608) 831-7666

Component and Surplus Dealers

Ace Electronic Parts, 5400 Mitchelldale B-8, Houston, TX 77092, (713) 688-8114
Active Electronic Sales, P.O. Box 1035, Framingham, MA 01701, (617) 879-0077
Adelco, 2281Q Babylon Turnpike, Merrick, NY 11566, (516) 378-4555
Adva Electronics, Box 4181 ER, Woodside, CA 94062, (415) 851-0455
Ancrona, P.O. Box 2208P, Culver City, CA 90230, (213) 641-4064
B & F Electronics, 119 Foster Street, Peabody, MA 01960, (617) 532-2323
Bullet Electronics, P.O. Box 19442E, Dallas, TX 75219, (214) 823-3240
California Industrial, P.O. Box 3097A, Torrance, CA 90503, (213) 772-0800
Delta Electronics, P.O. Box 2, 7 Oakland Street, Amesburg, MA 01913, (617) 388-4705
Digi-Key Corp., P.O. Box 677, Thief River Falls, MN 56701, (218) 681-6674
Digital Research Corp., P.O. Box 401247, Garland, TX 75040, (214) 271-2461
Etco Electronics, Dept. OV, 521 5th Avenue, New York, NY 10017
Formula International, 12603 Crenshaw Blvd., Hawthorne, CA 90250, (213) 679-5162
F. Reichert Sales, 1110 E. Garvey Ave., W. Covina, CA 91790
Godbout Electronics, Box 2355, Oakland Airport, CA 94614, (415) 562-0636
Integrated Circuits Unlimited, 7889 Clairemont Mesa Blvd., San Diego, CA 92111, (714) 278-4394
International Electronics Unlimited, Village Square, P.O. Box 449, Carmel Valley, CA 93924 (408) 659-3171
Intersystems (Ithaca Audio), P.O. Box 91, Ithaca, NY 14850, (607) 257-0190
Jade Computer Products, 5351 W. 144th Street, Lawndale, CA 90260, (213) 679-3313
JameCo Electronics, 1021-A Howard Ave., San Carlos, CA 94070, (415) 592-8097
J. B. Saunders, 3050 Valmont Road, Boulder, CO 80301, (303) 442-1212
Meshna Electronics, P.O. Box 62, East Lynn, MA 01904, (617) 595-2275
New Tone Electronics, P.O. Box 1738R, Bloomfield, NJ 07003, (201) 748-6171
Optoelectronics, Box 219, Hollywood, FL 33022, (305) 921-2056
Poly Paks, P.O. Box 942R, Lynnfield, MA 01940, (617) 245-3829
Quest Electronics, P.O. Box 4430 E., Santa Clara, CA 95054, (408) 988-1640

Radio Hut, P.O. Box 38323, Dallas, TX 75238, (214) 271-8423

Radio Shack, (check local city directories)

Ramsey Electronics, Box 4072B, Rochester, NY 14610, (716) 271-6487

S. D. Sales, P.O. Box 28810, Dallas, TX 75228, (214) 271-0022

Solid State Sales, P.O. Box 74D, Somerville, MA 02143, (617) 547-4005

APPENDIX C

Schematic Diagrams of Commonly Used S-100 Bus Boards

Contents

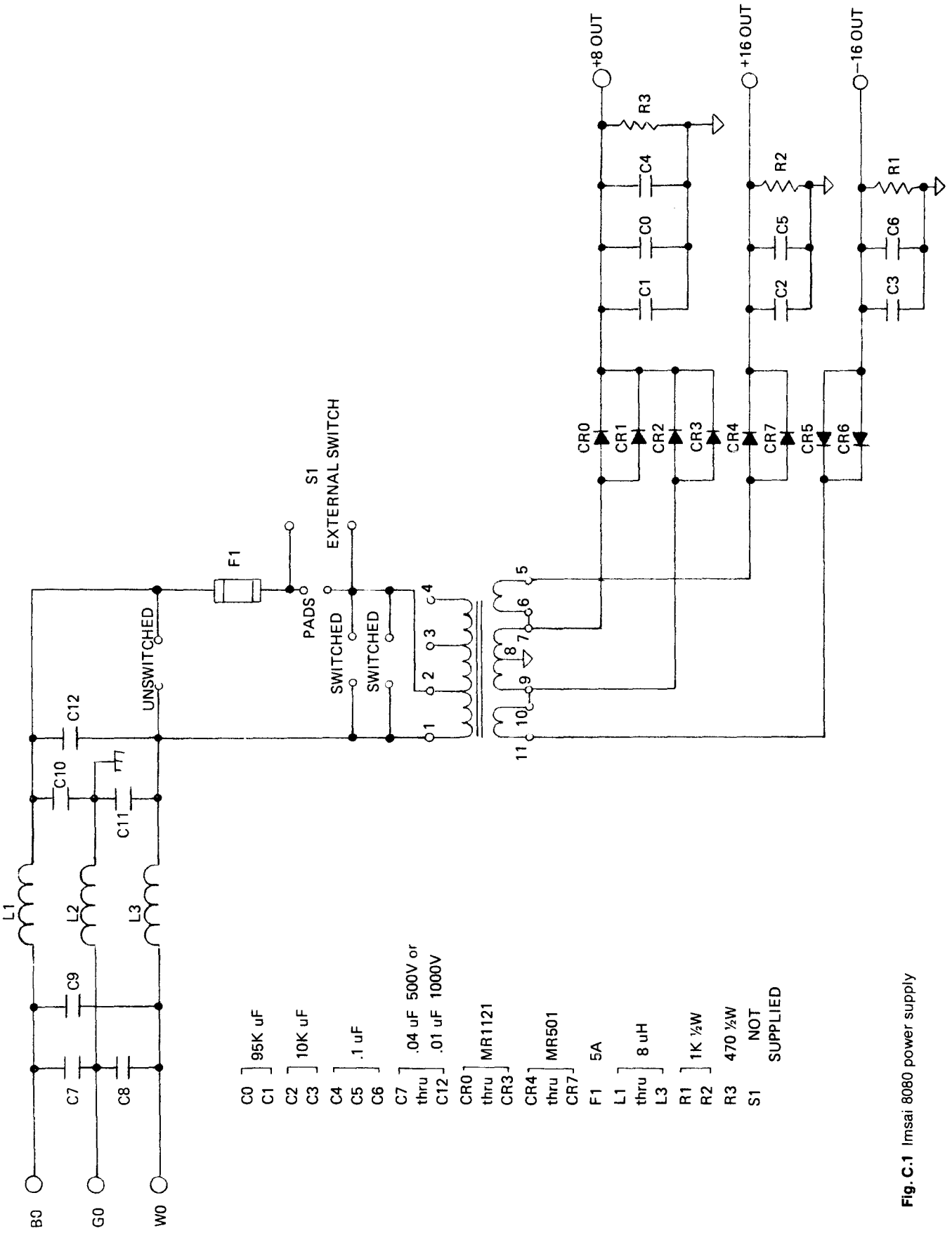


Fig. C.1 Imsai 8080 power supply

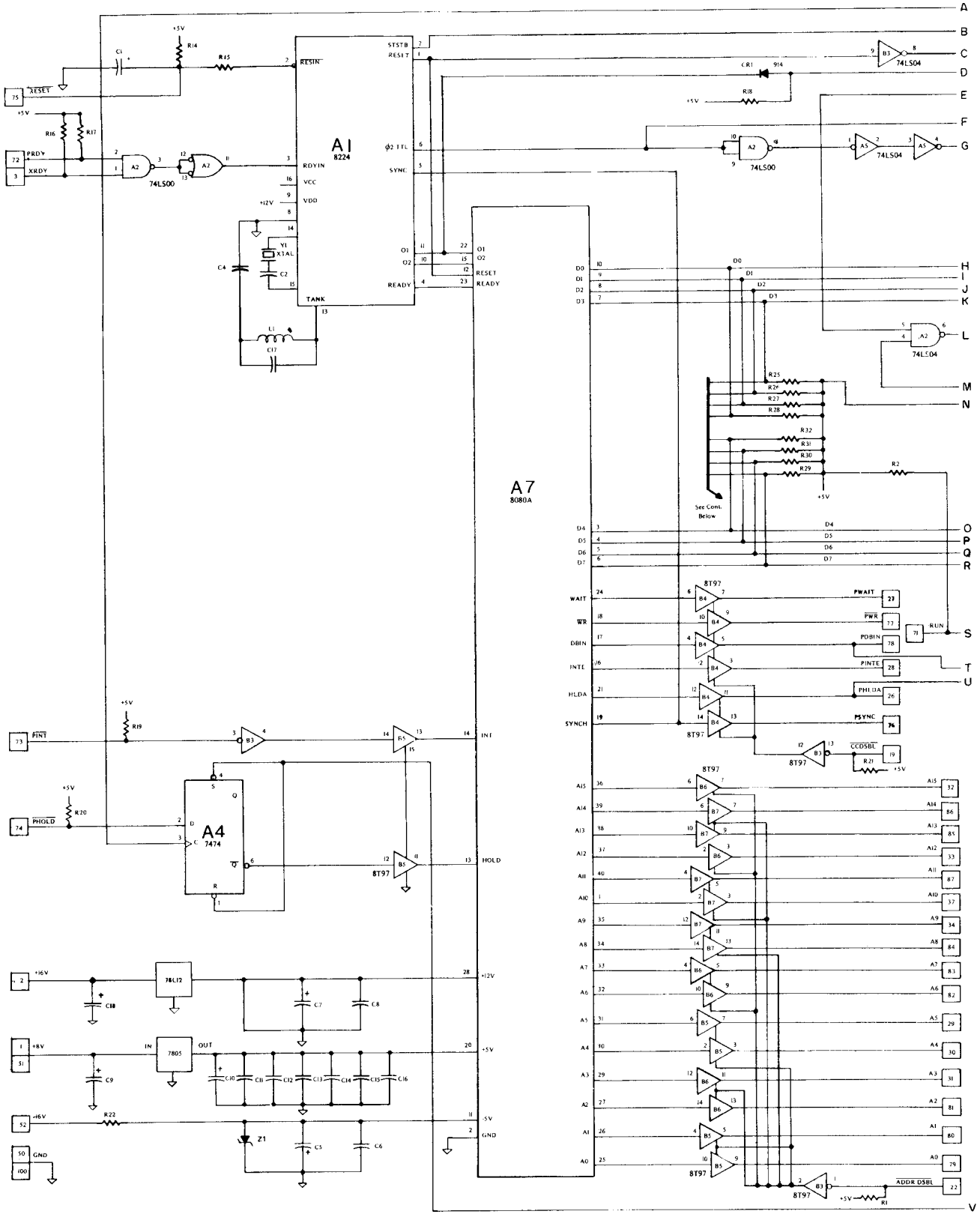


Fig. C.2 Imsai 8080 central processor board *(sheet 1 of 2)*

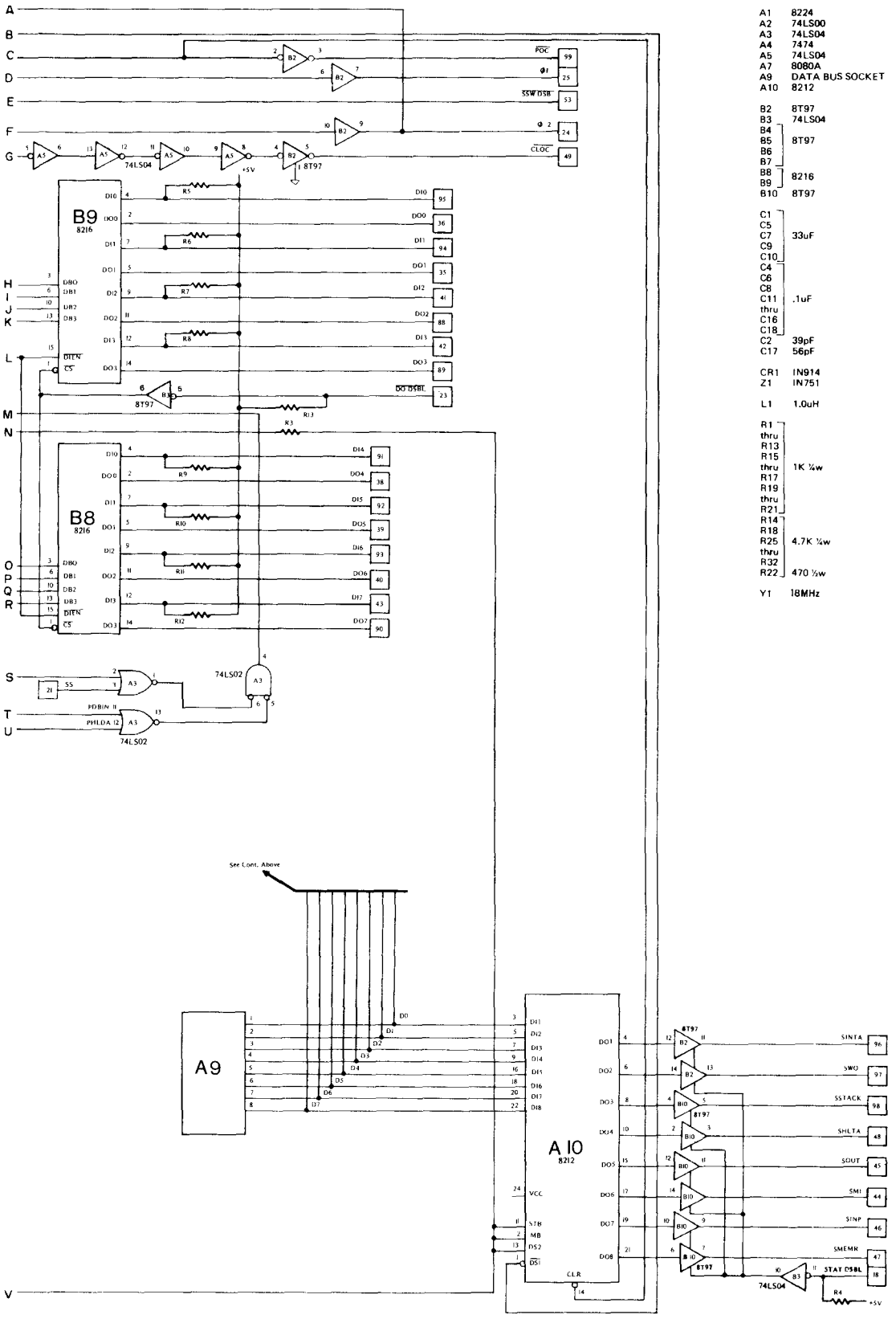


Fig. C.2 Imsai 8080 central processor board *(sheet 2 of 2)*

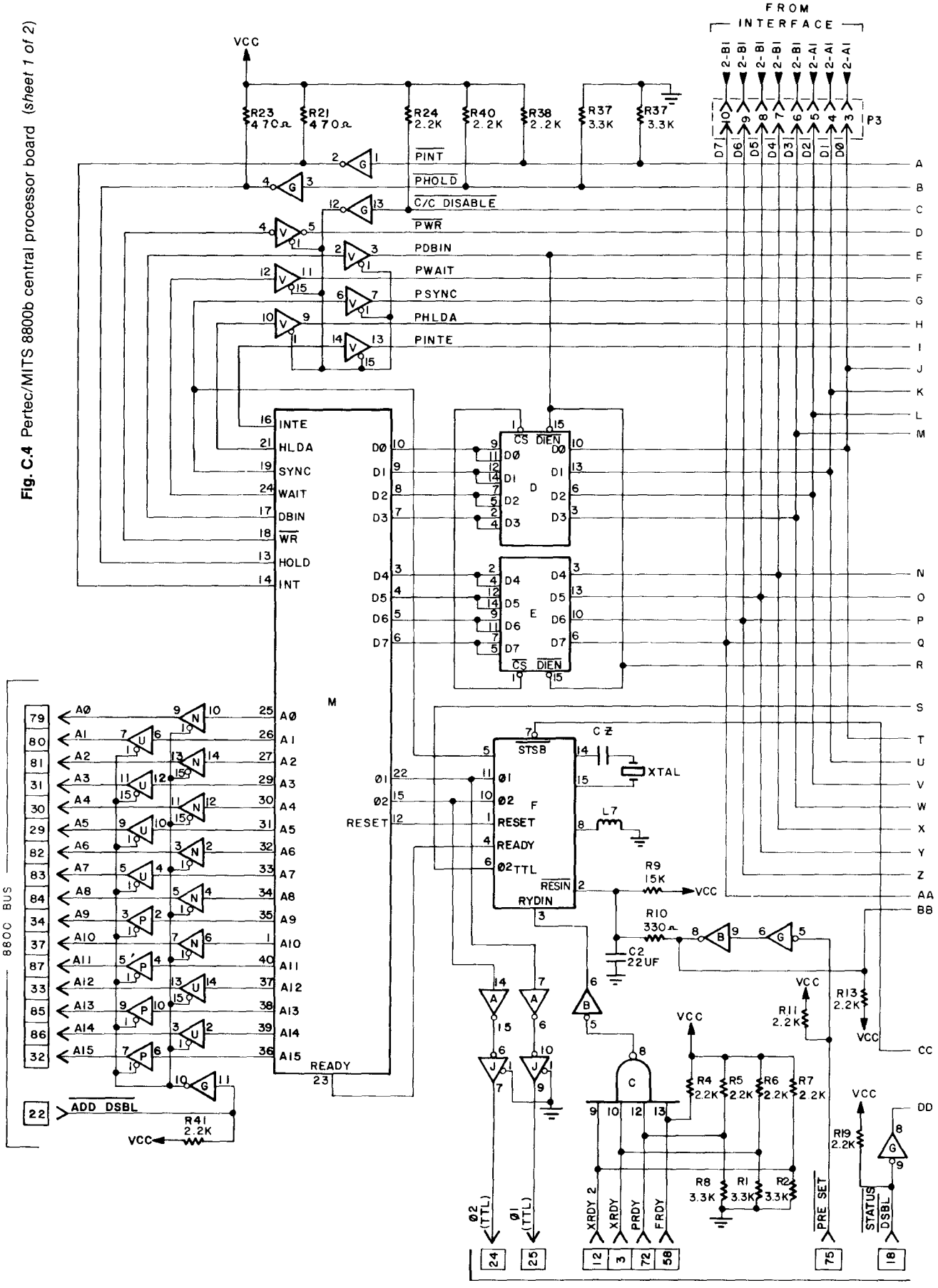
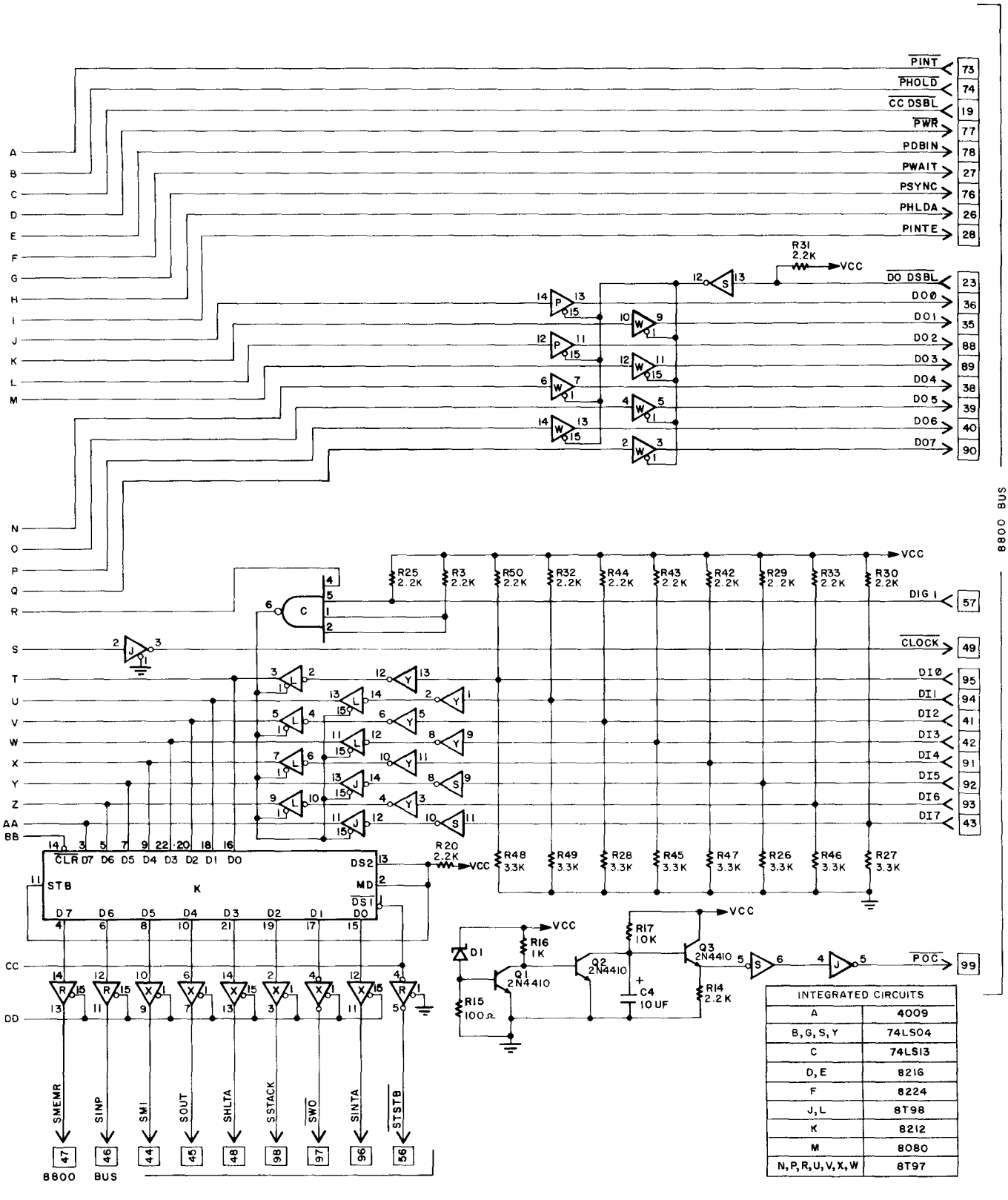


Fig. C.4 Pertec/MITS 8800b central processor board (*sheet 1 of 2*)



INTEGRATED CIRCUITS	
A	4009
B, G, S, Y	74LS04
C	74LS13
D, E	8216
F	8224
J, L	8T98
K	8212
M	8080
N, P, R, U, V, X, W	8T97

Fig. C.4 Pertec/MITS 8800b central processor board *(sheet 2 of 2)*

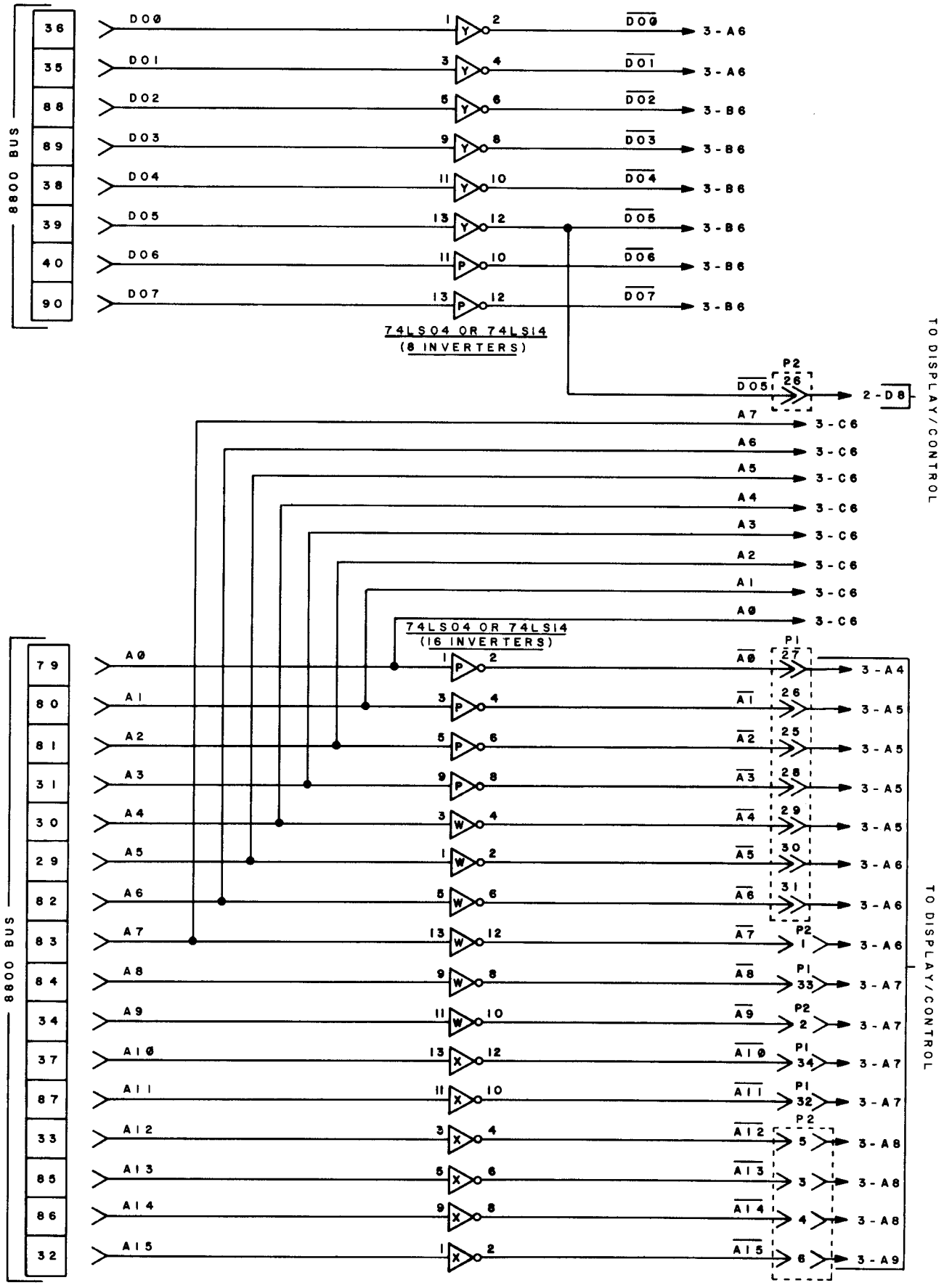


Fig. C.5 Pertec/MITS 8800b front panel control logic *(sheet 1 of 10)*

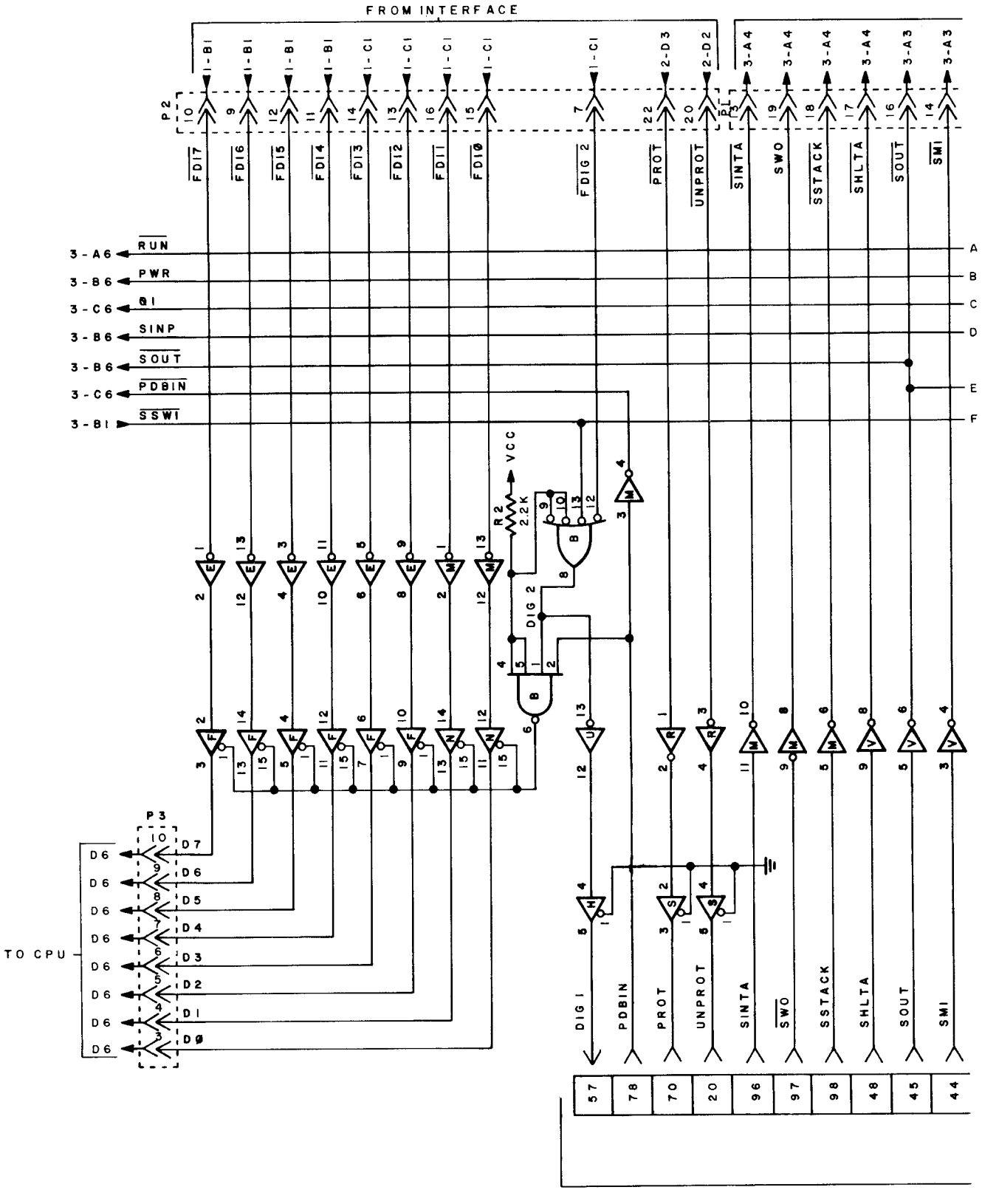


Fig. C.5 Pertec/MITS 8800b front panel control logic (*sheet 2 of 10*)

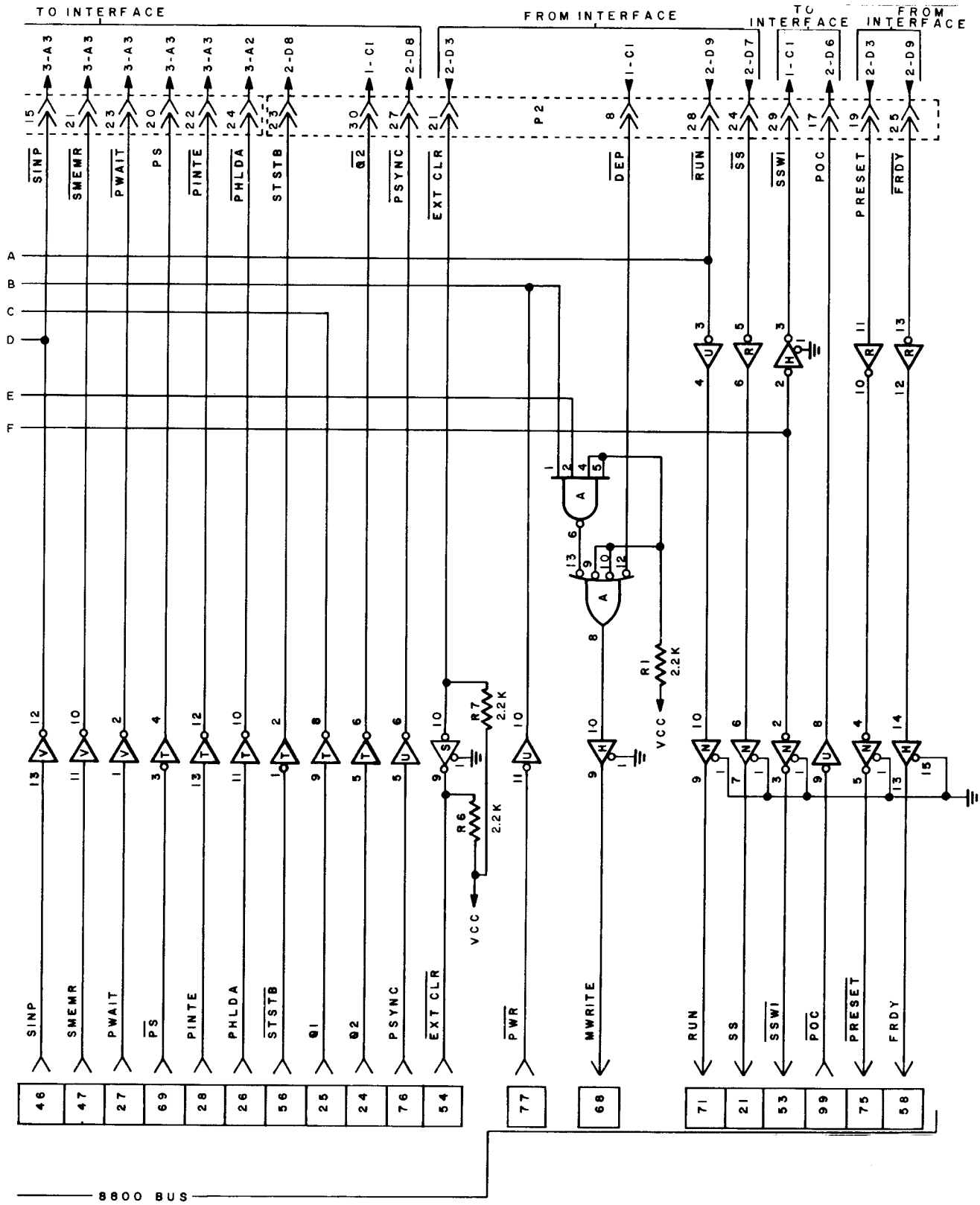


Fig. C.5 Pertec/MITS 8800b front panel control logic *(sheet 3 of 10)*

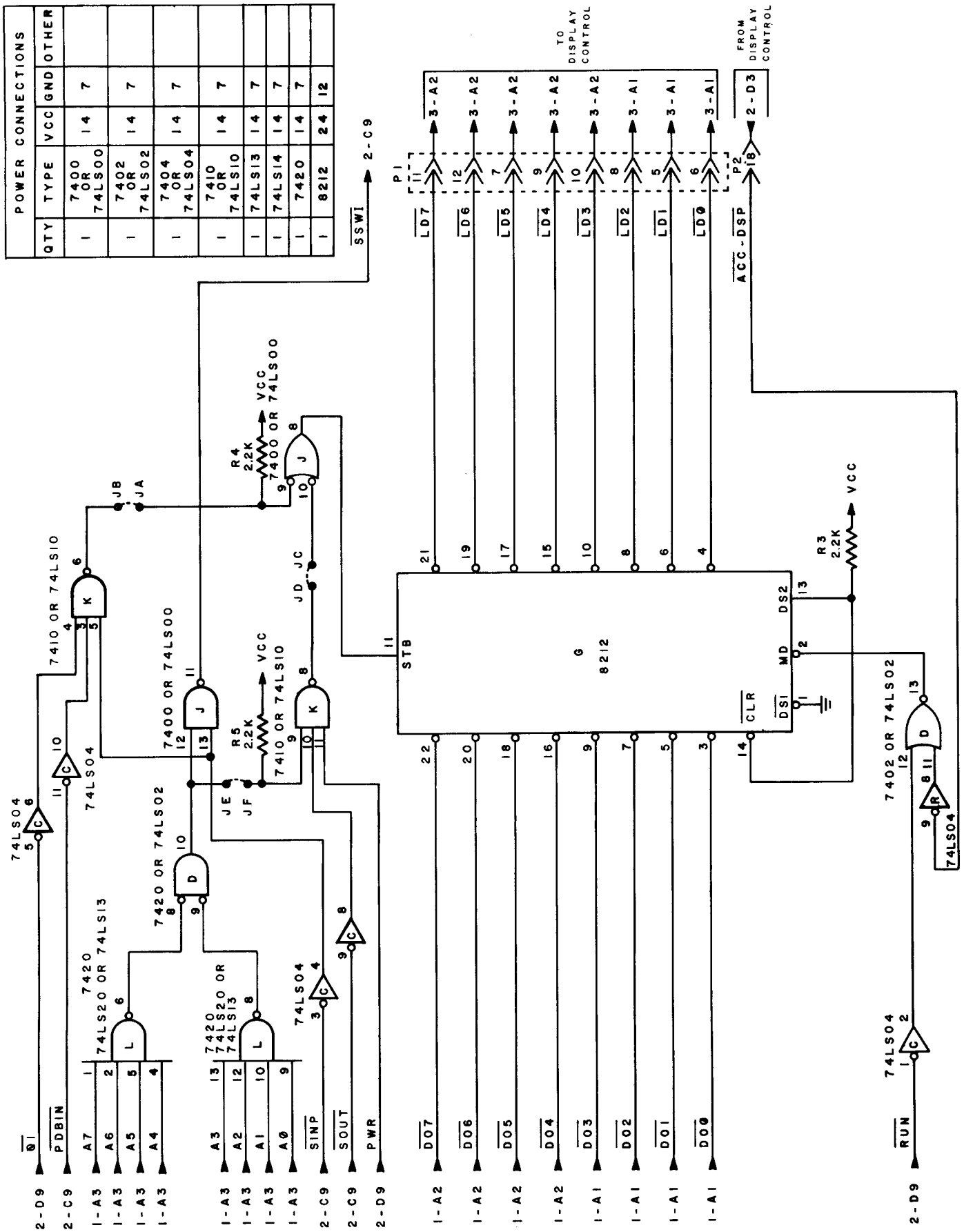
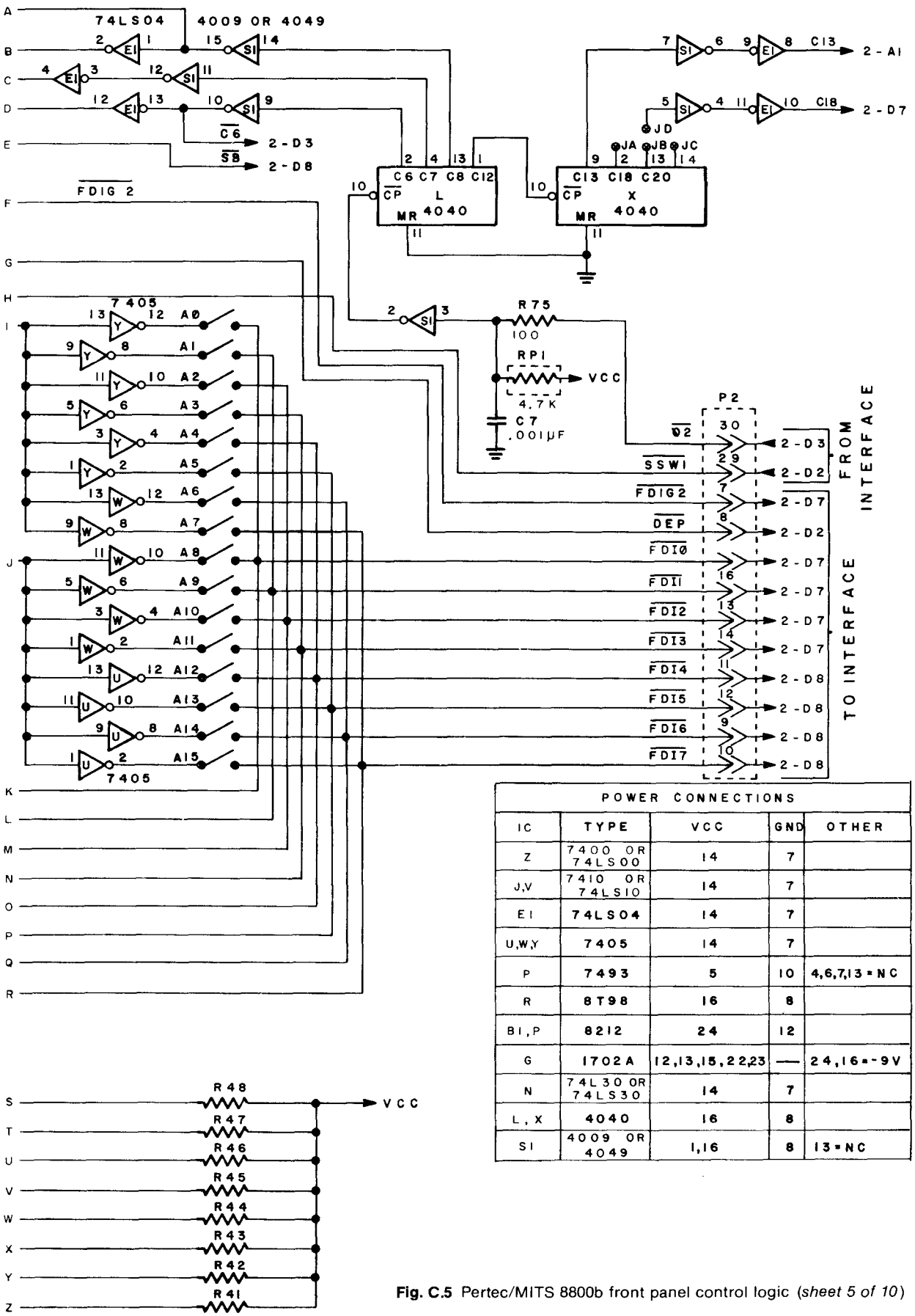


Fig. C.5 Pertec/MITS 8800b front panel control logic *(sheet 4 of 10)*



POWER CONNECTIONS				
IC	TYPE	VCC	GND	OTHER
Z	7400 OR 74LS00	14	7	
J,V	7410 OR 74LS10	14	7	
E1	74LS04	14	7	
U,W,Y	7405	14	7	
P	7493	5	10	4,6,7,13 = NC
R	8T98	16	8	
B1,P	8212	24	12	
G	1702A	12,13,15,22,23	—	24,16 = -9V
N	74L30 OR 74LS30	14	7	
L, X	4040	16	8	
S1	4009 OR 4049	1,16	8	13 = NC

Fig. C.5 Pertec/MITS 8800b front panel control logic *(sheet 5 of 10)*

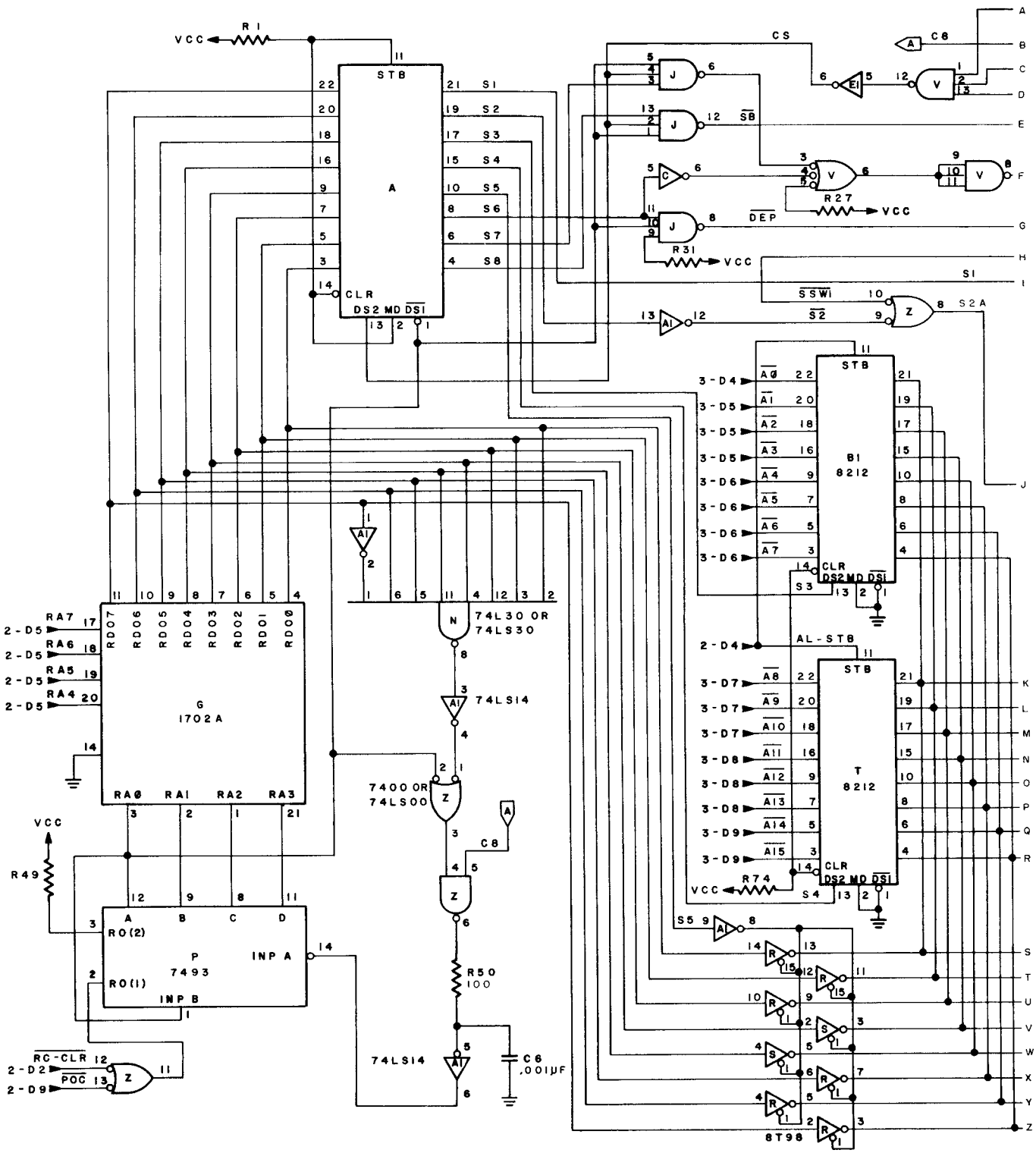


Fig. C.5 Pertec/MITS 8800b front panel control logic *(sheet 6 of 10)*

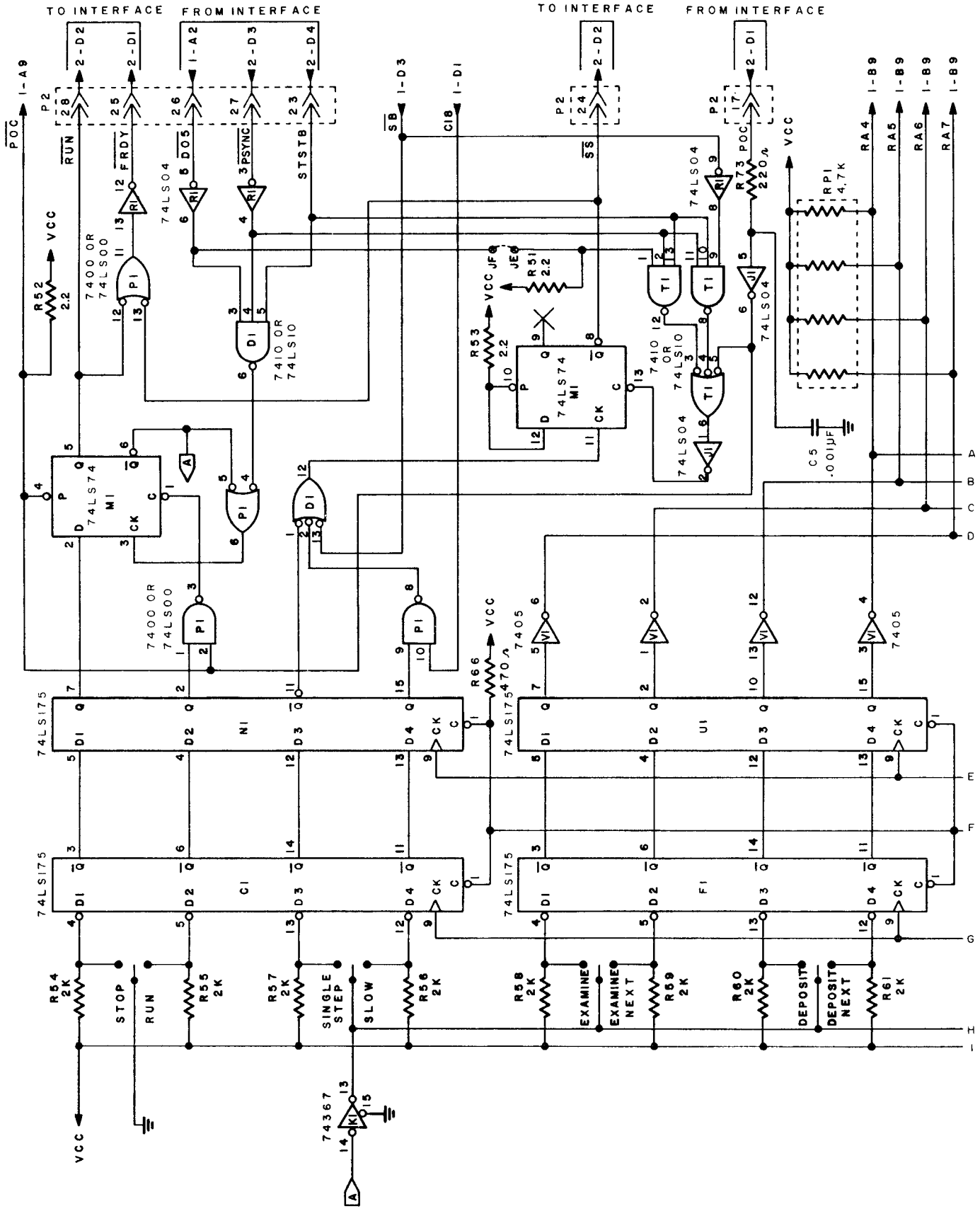


Fig. C.5 Pertec/MITS 8800b front panel control logic (*sheet 7 of 10*)

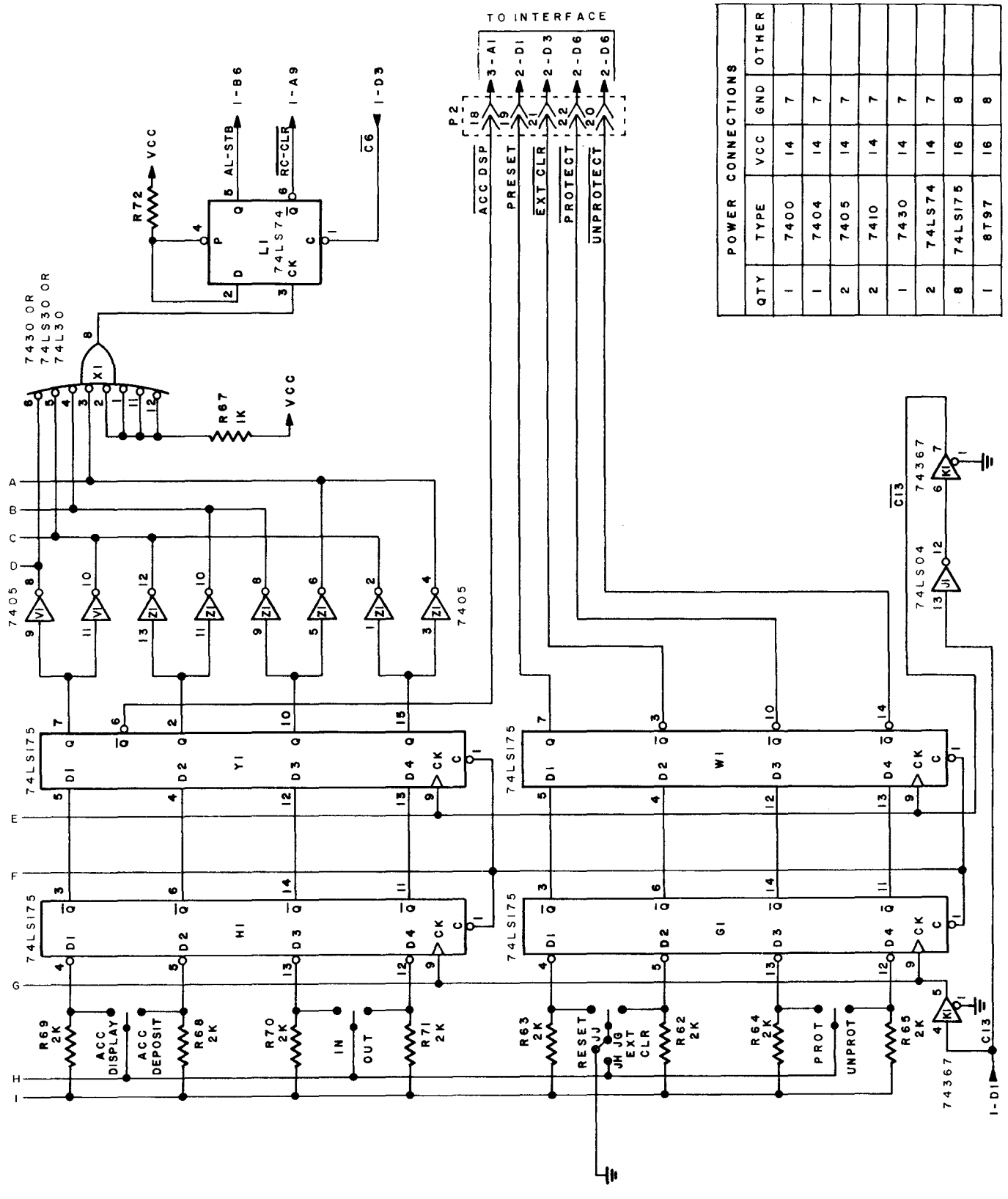


Fig. C.5 Pertec/MITS 8800b front panel control logic *(sheet 8 of 10)*

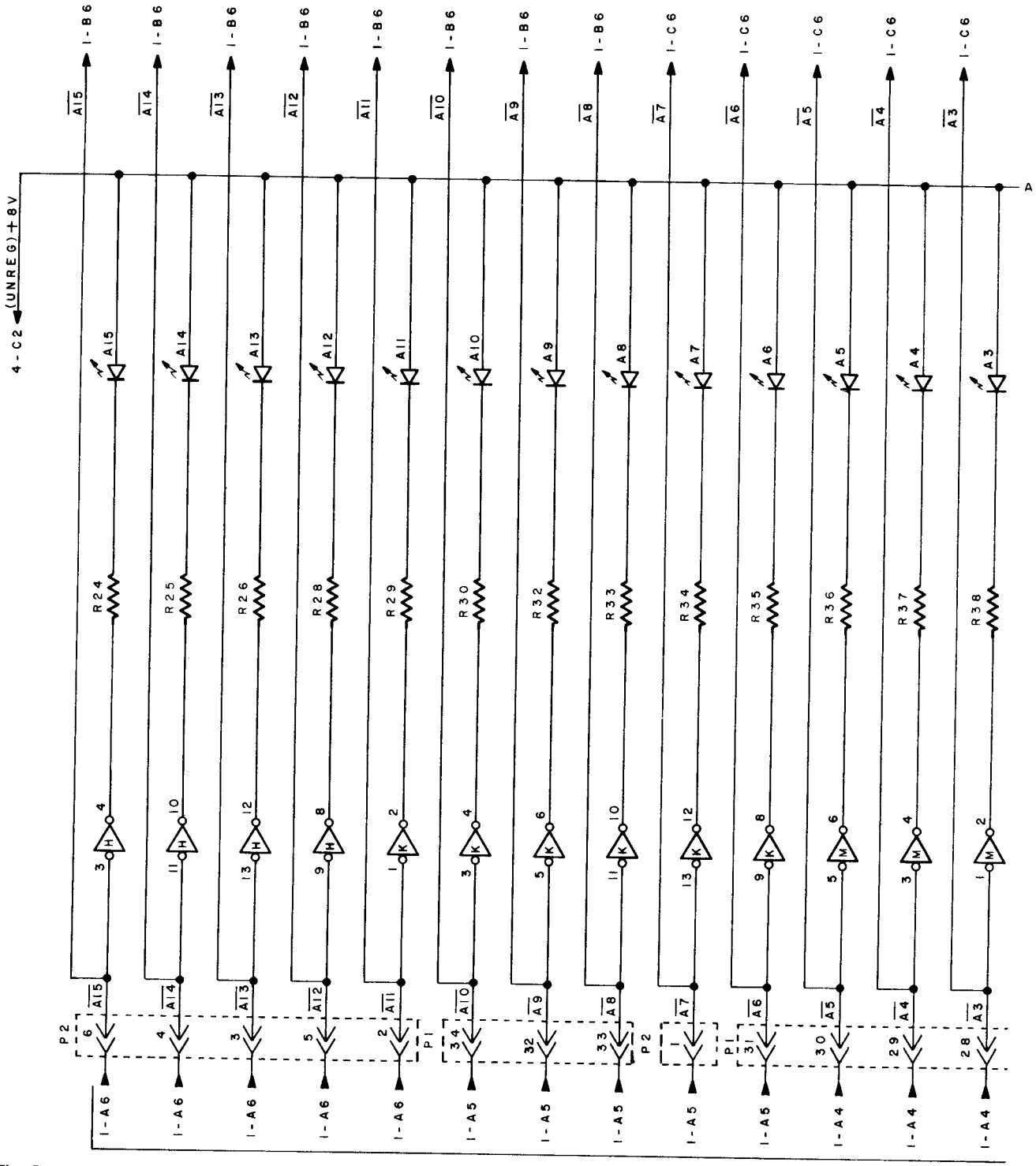
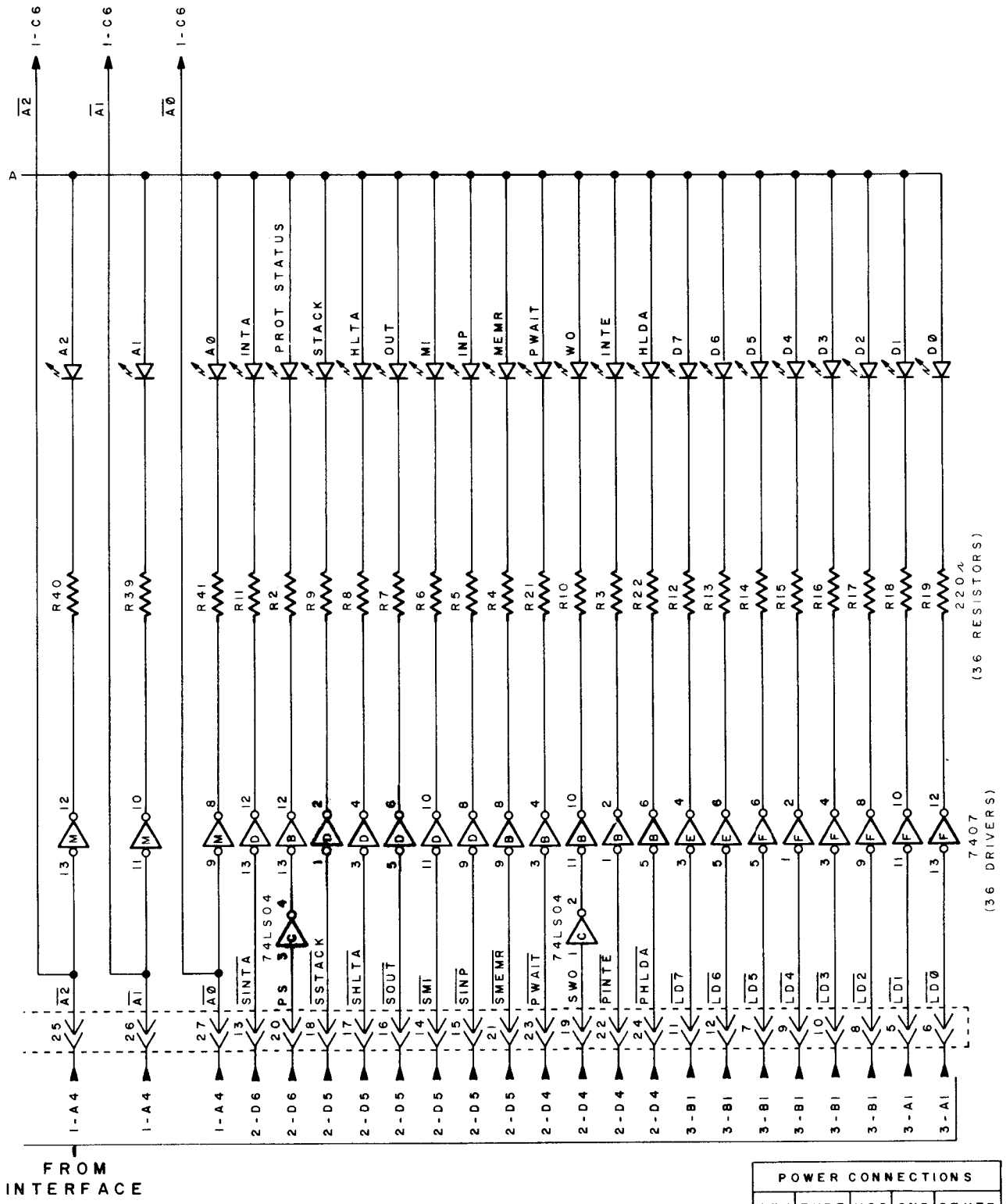


Fig. C.5 Pertec/MITS 8800b front panel control logic (*sheet 9 of 10*)



POWER CONNECTIONS				
QTY	TYPE	VCC	GND	OTHER
7	7407	14	7	
1	74LS04	14	7	

Fig. C.5 Pertec/MITS 8800b front panel control logic (*sheet 10 of 10*)

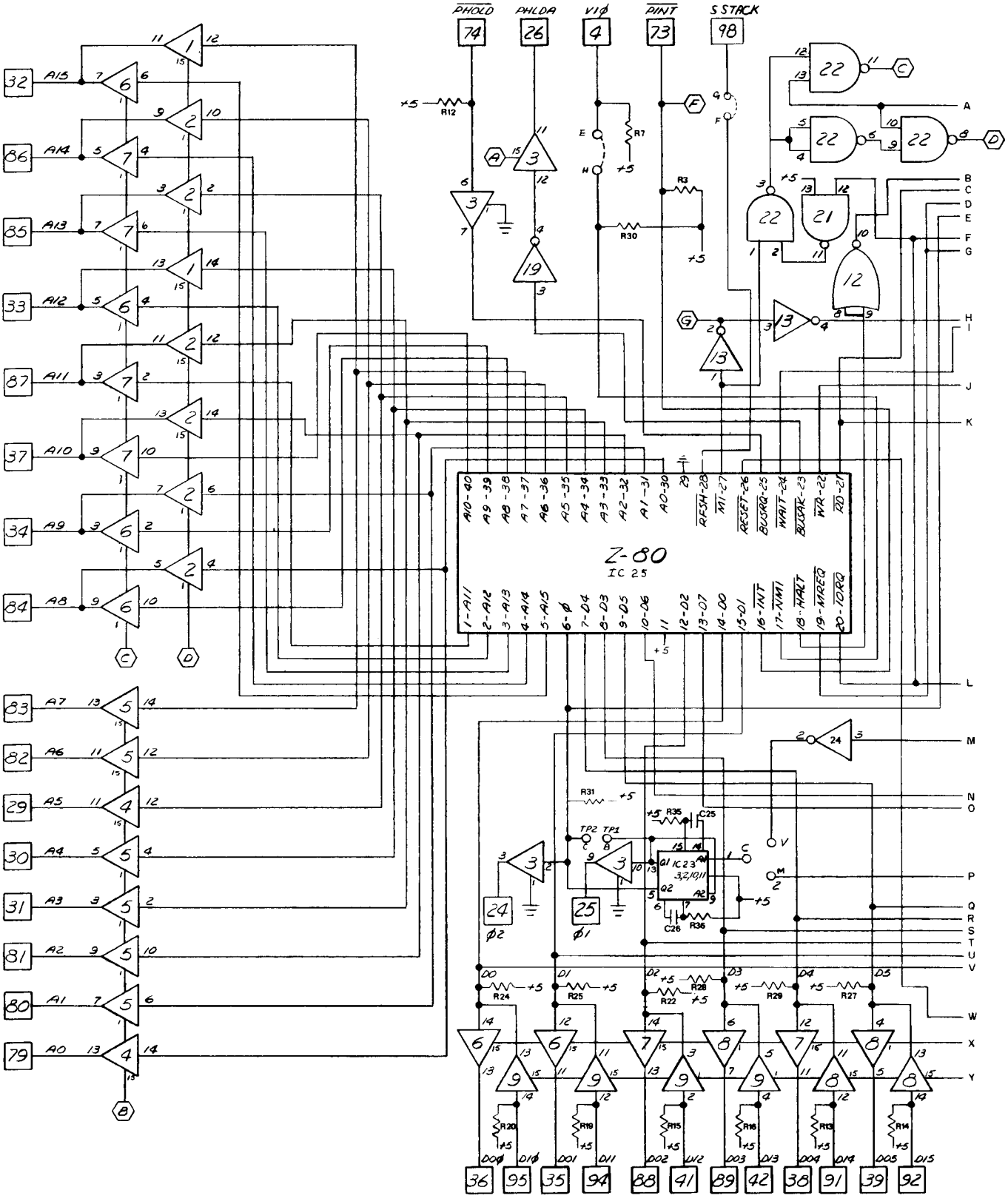
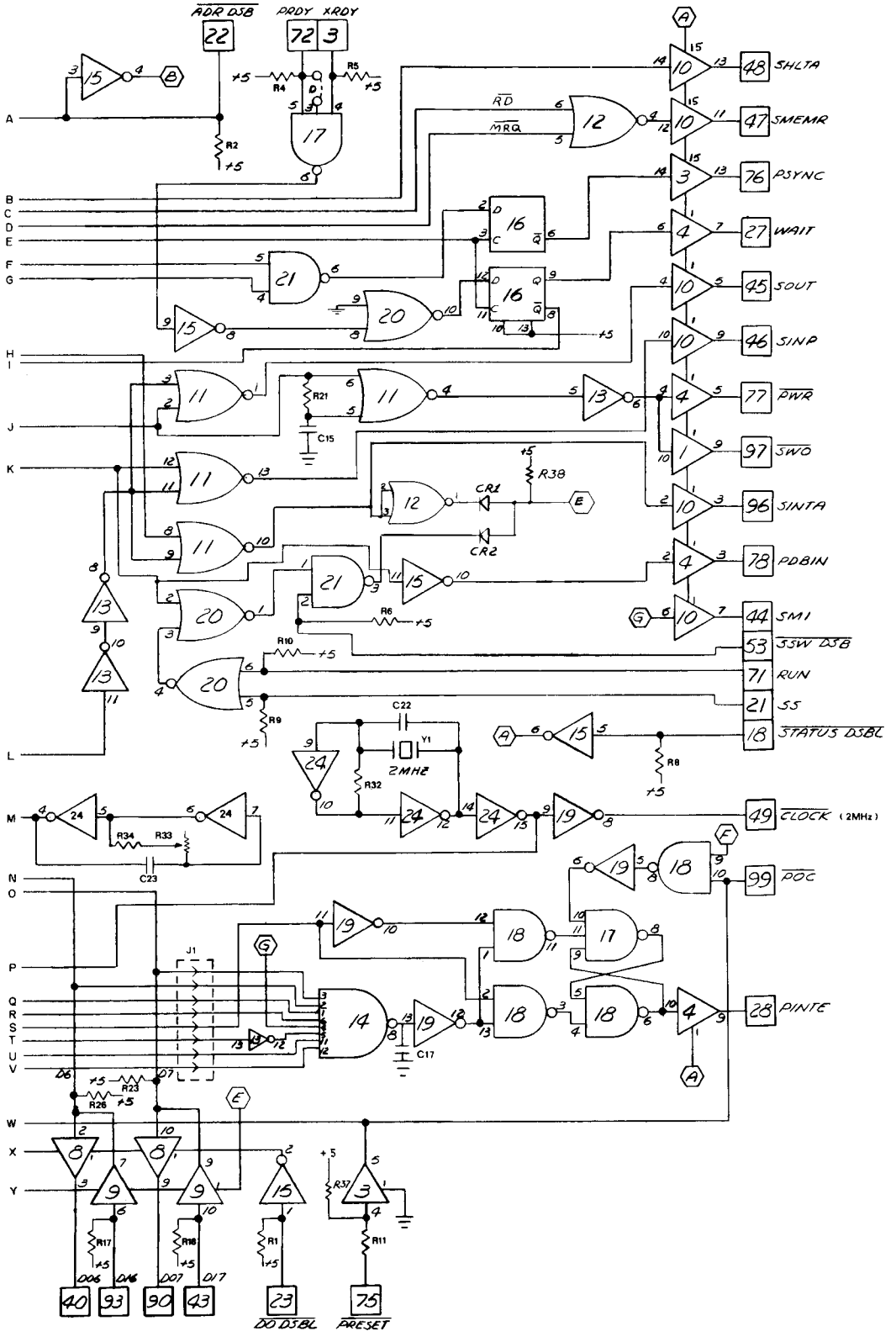


Fig. C.6 Xitan/TDL ZPU, Z80 central processor board (*sheet 1 of 2*)



PARTS LIST

IC	1 TO 10	8T97B
IC	11, 12, 20	74LS02
IC	13, 15, 19	74LS04
IC	14	74LS30
IC	16	74LS74
IC	17	74LS10
IC	18, 21, 22	74LS00
IC	23	74123
IC	24	4049
IC	25	Z80
CR	1,2	IN270
R	1 TO 10, 12 TO 20, 30	1K
R	11	100 ohm
R	21	47
R	22 TO 29, 32, 38	4.7K
R	33	20 K TRIMPOT
R	34	2.2K
R	35	10K
R	36	12K
R	37	1K
R	31	330 ohm
C	15	001 Mf
C	22	10 pf
C	23	6 pf
C	25	27 pf
C	26	47 pf
C	17	560 pf
J1 (A or B)		10 PIN MOLEX or 16 PIN SOCKET

Fig. C.6 Xitan/TDL ZPU, Z80 central processor board (*sheet 2 of 2*)

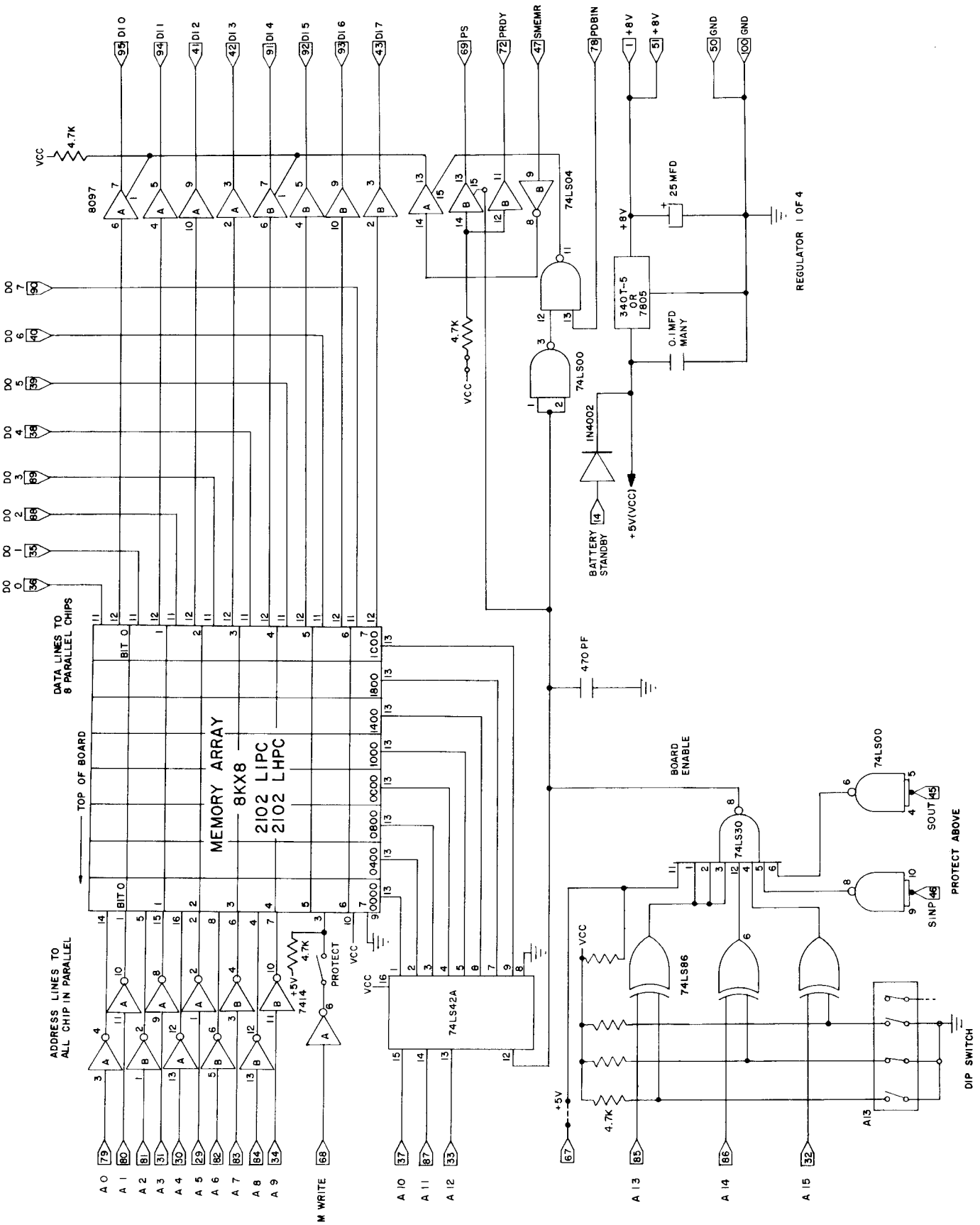


Fig. C.7 Vector Graphic 8 kbyte static memory card

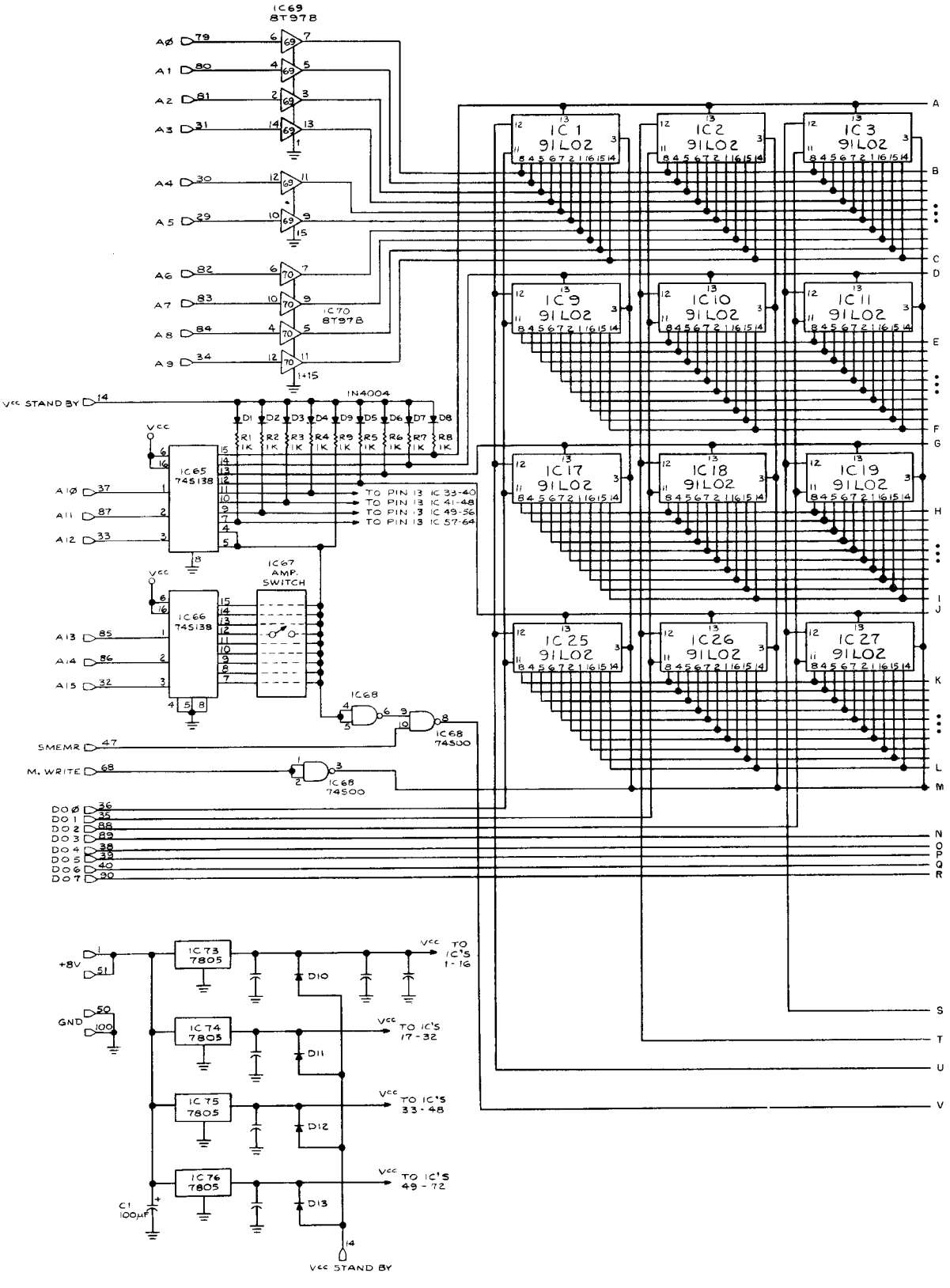


Fig. C.8 Seals Electronics 8 kbyte static memory card (*sheet 1 of 2*)

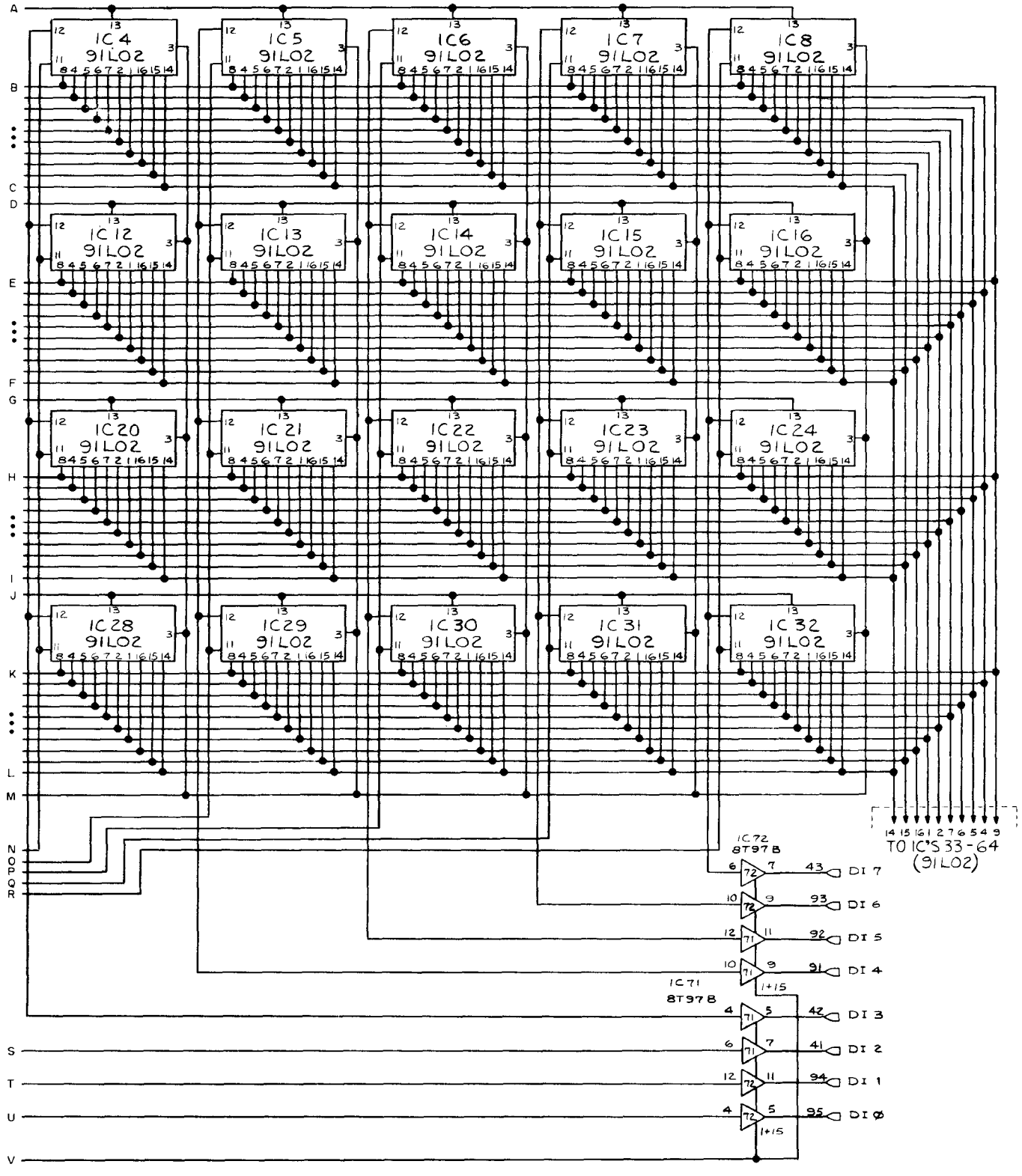


Fig. C.8 Seals Electronics 8 kbyte static memory card (*sheet 2 of 2*)

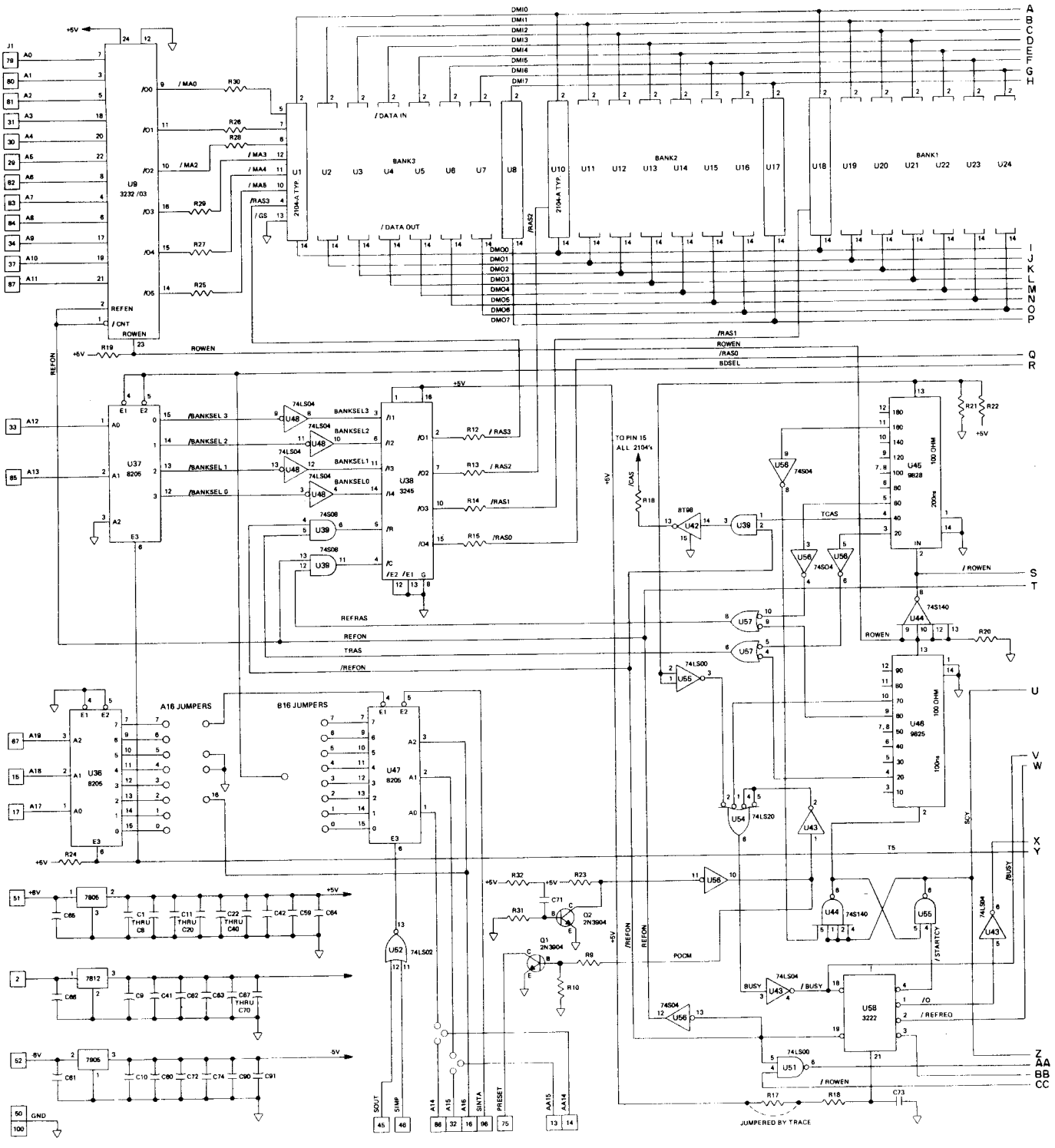
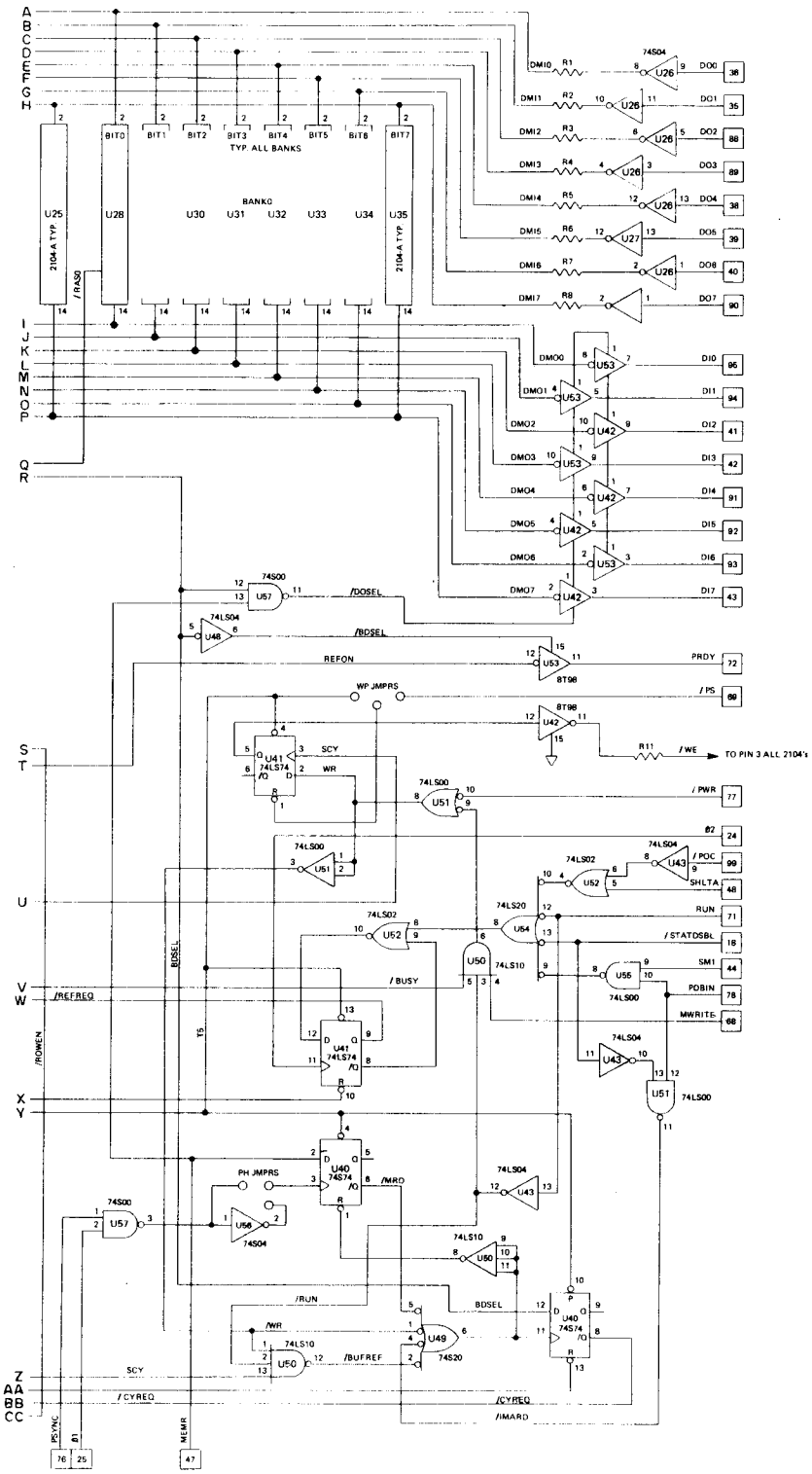


Fig. C.9 Imsai 8080 RAM-16, 16 kbyte dynamic memory card *(sheet 1 of 2)*



ALL I.C.'S IN SOCKETS

U1 THRU U8, U10 THRU U25, U28 THRU U35	2104A-4	C9, C10, C41, C60, C61, C64 THRU C67	33 uF
U9	3232	C21, C43 THRU C58, C75 THRU C89	.33 uF
U26, U27, U56	74S04	C71	2.2 uF
U36, U37, U47	8205	C73	5600 pF
U38	3245	Q1, Q2	2N3904
U39	74S08	R1 THRU R8, R11 THRU R16, R25 THRU R31	22 ¼w
U40	74S74	R9, R10, R24	1K ¼w
U41	74LS74	R17, R19, R22	NOT USED
U42, U53	8T98	R18	7.32K
U43, U48	74LS04	R20, R21	110 ¼w
U44	74S140	R23	4.7K ¼w
U45	9828	R32	10K ¼w
U46	9825	R33	47K ¼w
U49	74S20		
U50	74LS10		
U51, U55	74LS00		
U52	74LS02		
U54	74LS20		
U56	74S04		
U57	74S00		
U58	3222		
C1 THRU C8, C11 THRU C20, C22 THRU C40, C42, C59, C62, C63, C68 THRU C70, C72, C74, C90, C91	.1 uF		

Fig. C.9 Imsai 8080 RAM-16, 16 kbyte dynamic memory card *(sheet 2 of 2)*

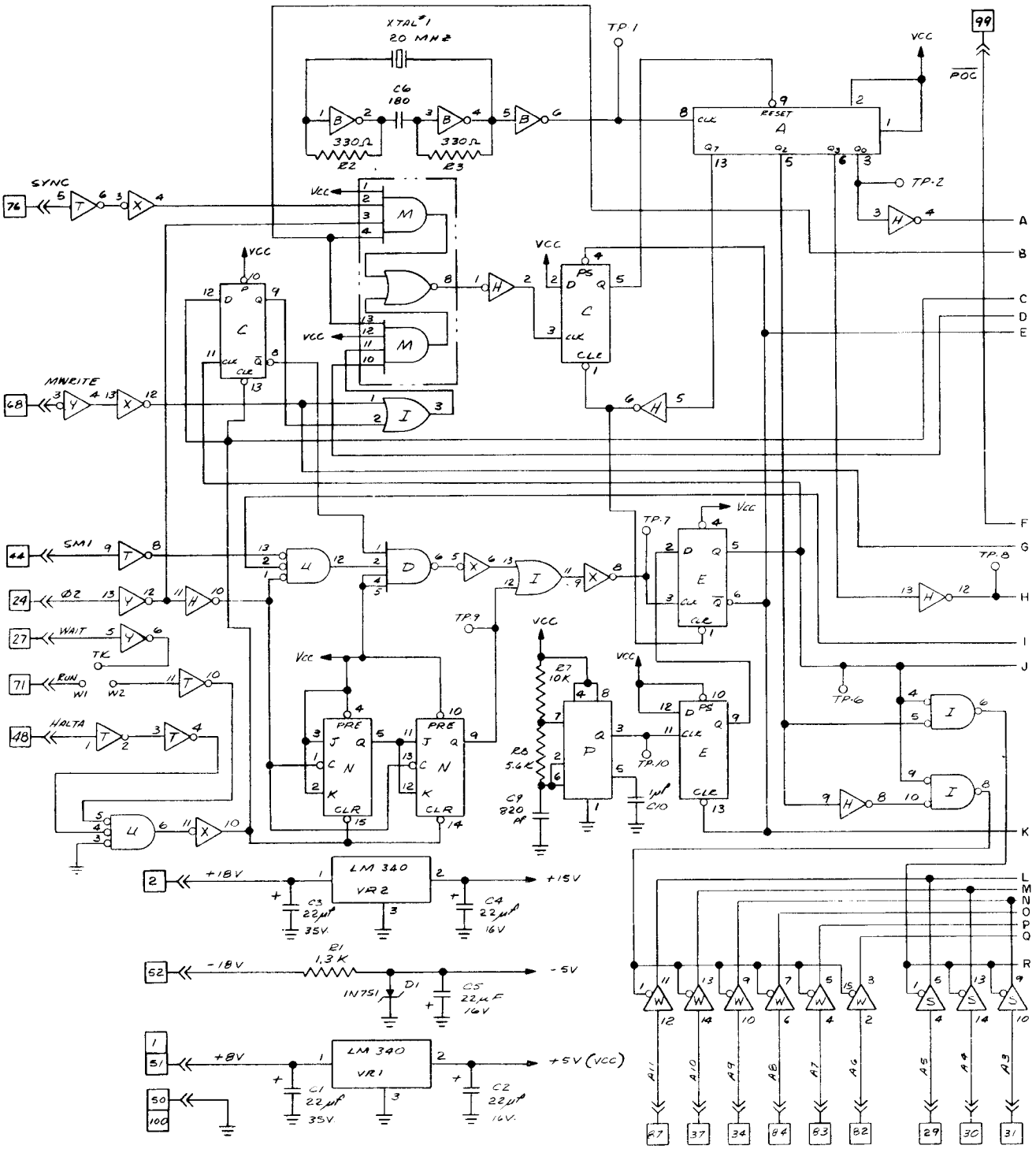


Fig. C.10 Pertec/MITS 88-MCD, 16 kbyte dynamic memory card (*sheet 1 of 4*)

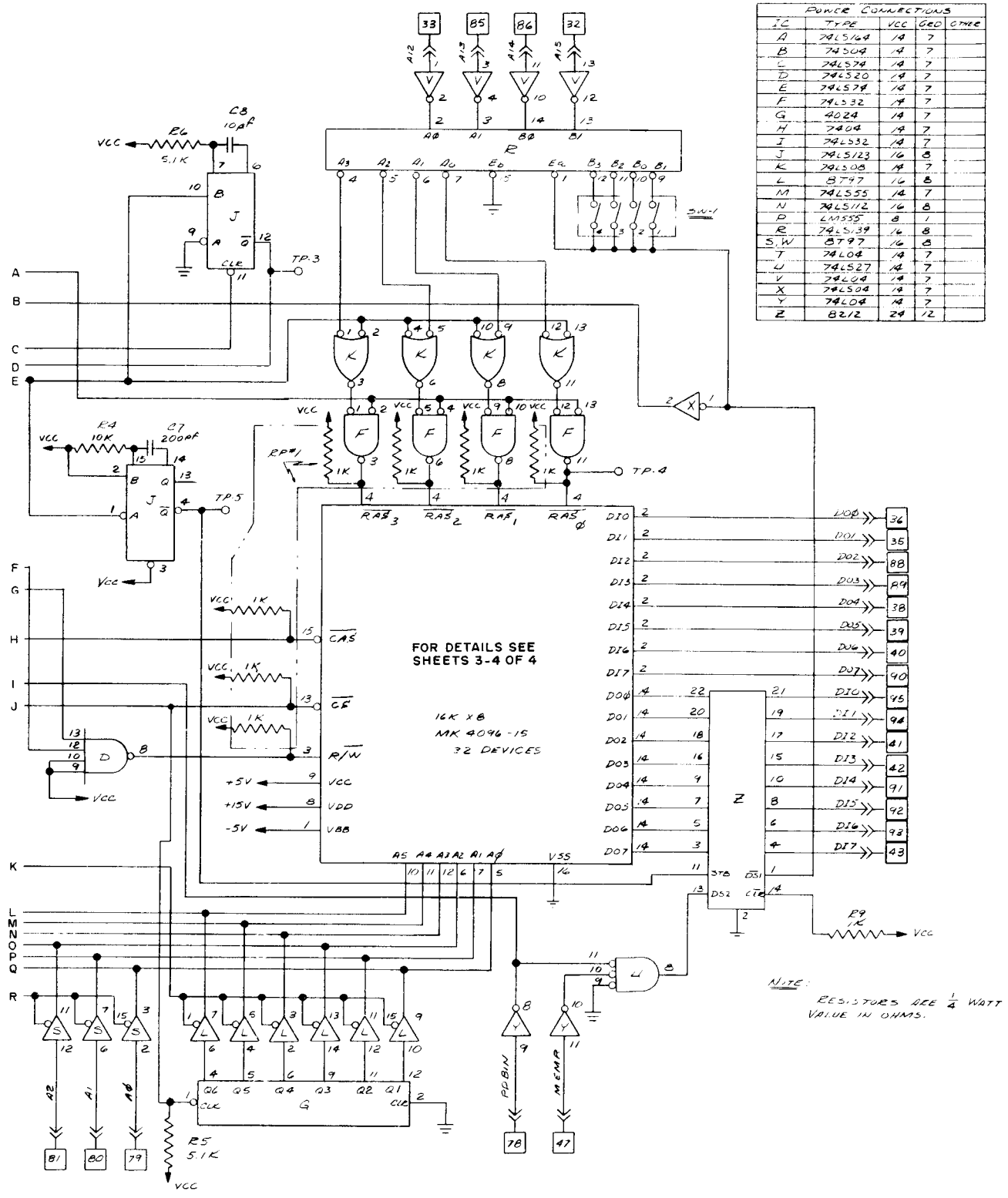


Fig. C.10 Pertec/MITS 88-MCD, 16 kbyte dynamic memory card (*sheet 2 of 4*)

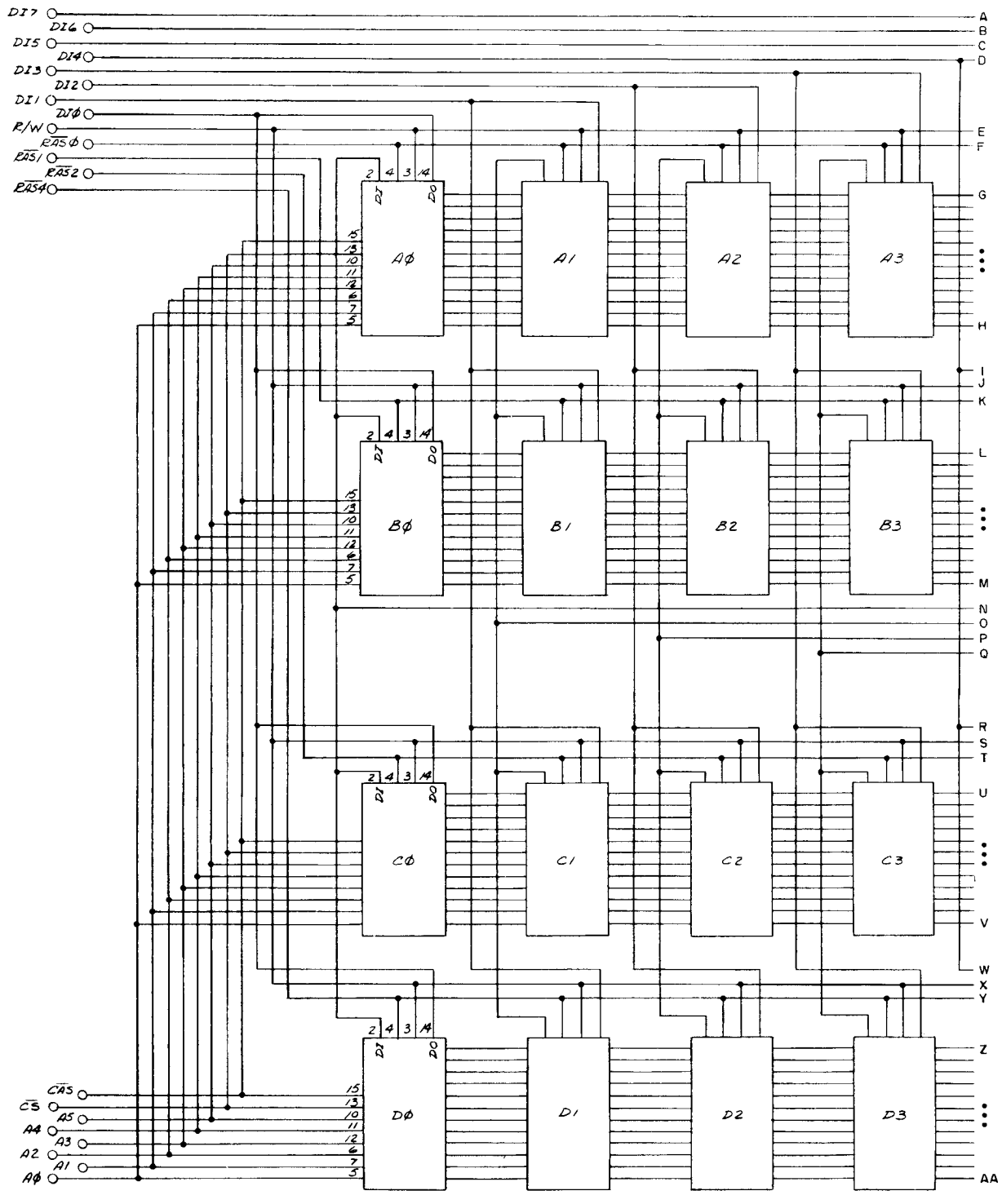


Fig. C.10 Pertec/MITS 88-MCD, 16 kbyte dynamic memory card (*sheet 3 of 4*)

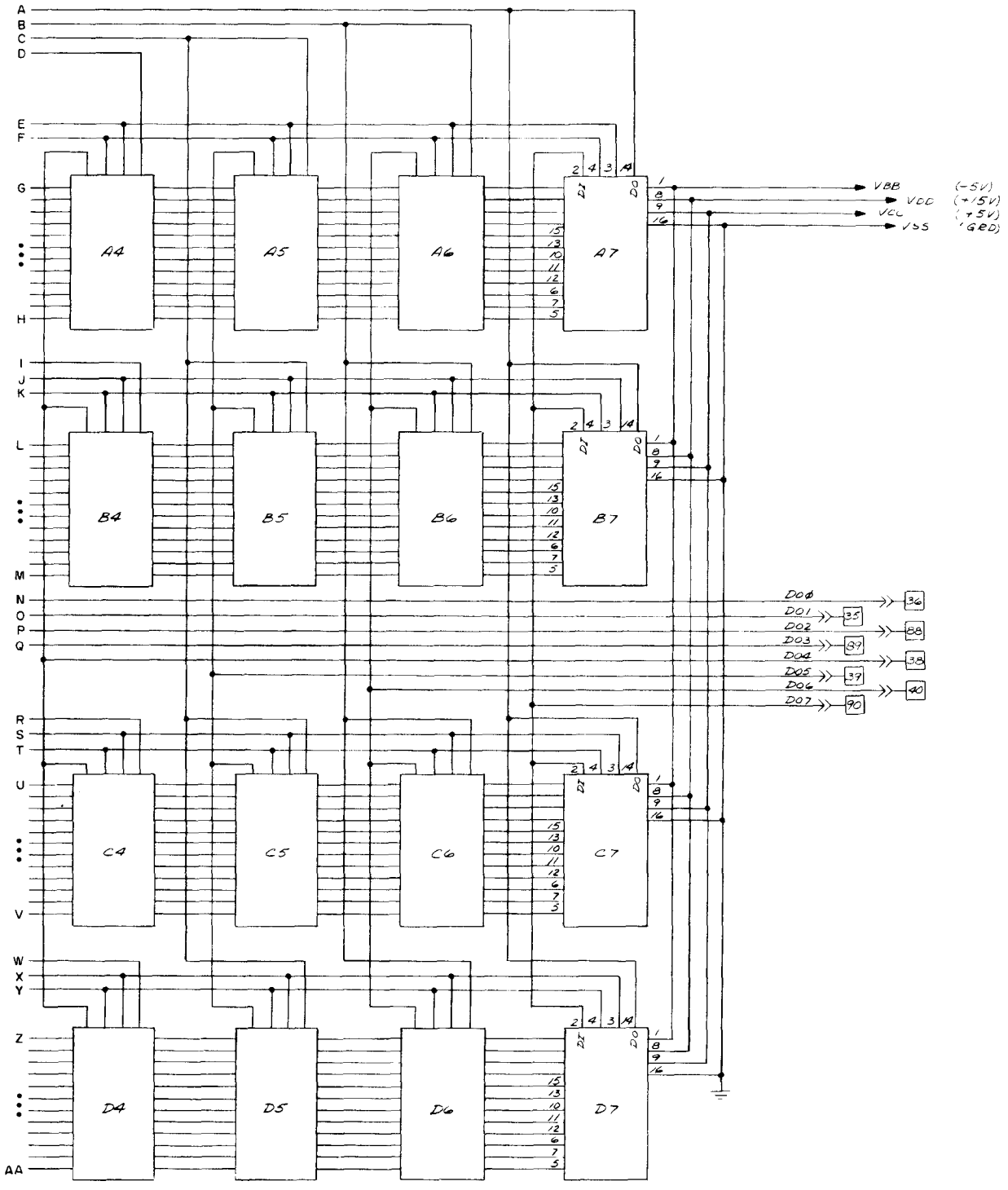


Fig. C.10 Pertec/MITS 88-MCD, 16 kbyte dynamic memory card (*sheet 4 of 4*)

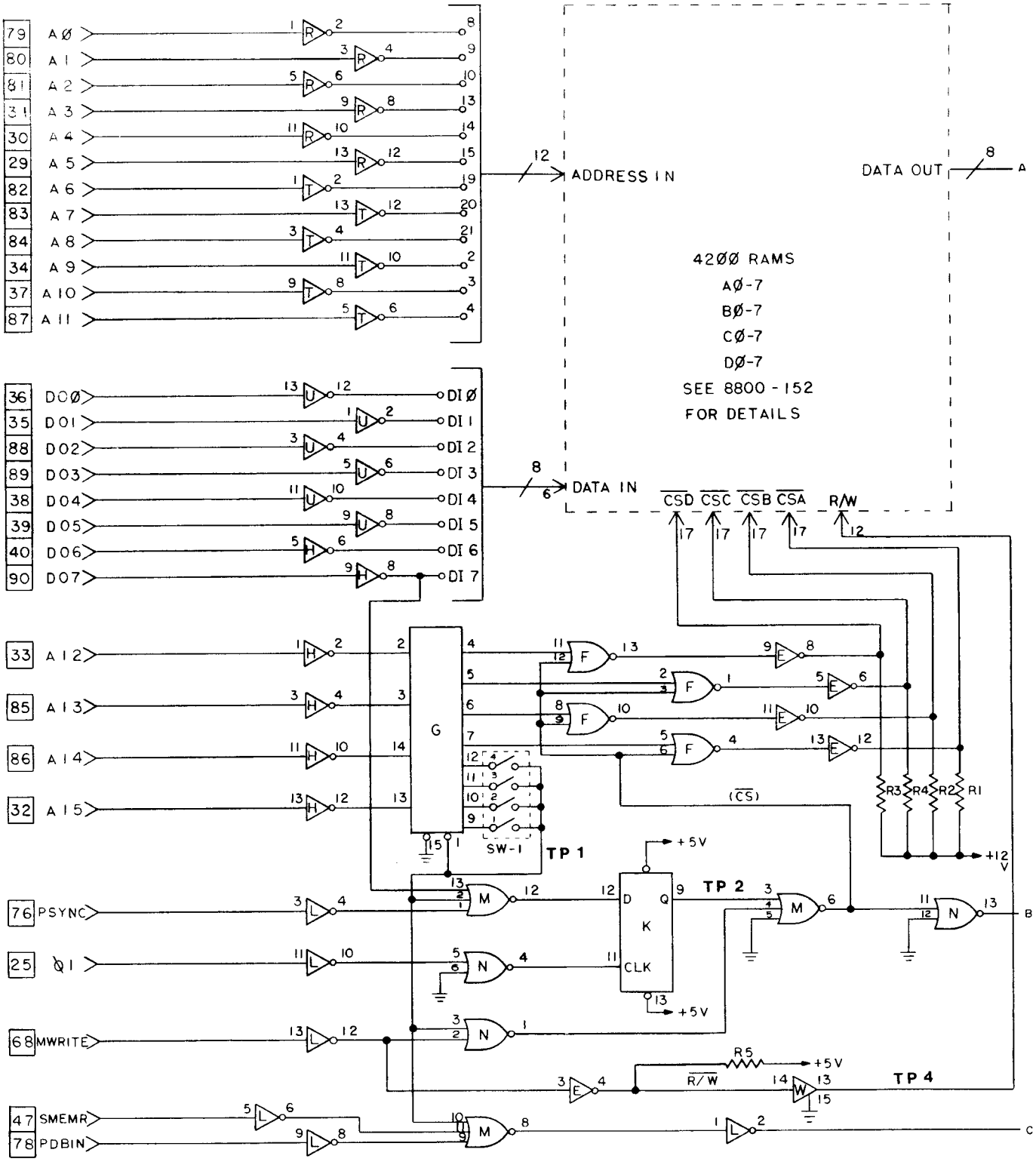
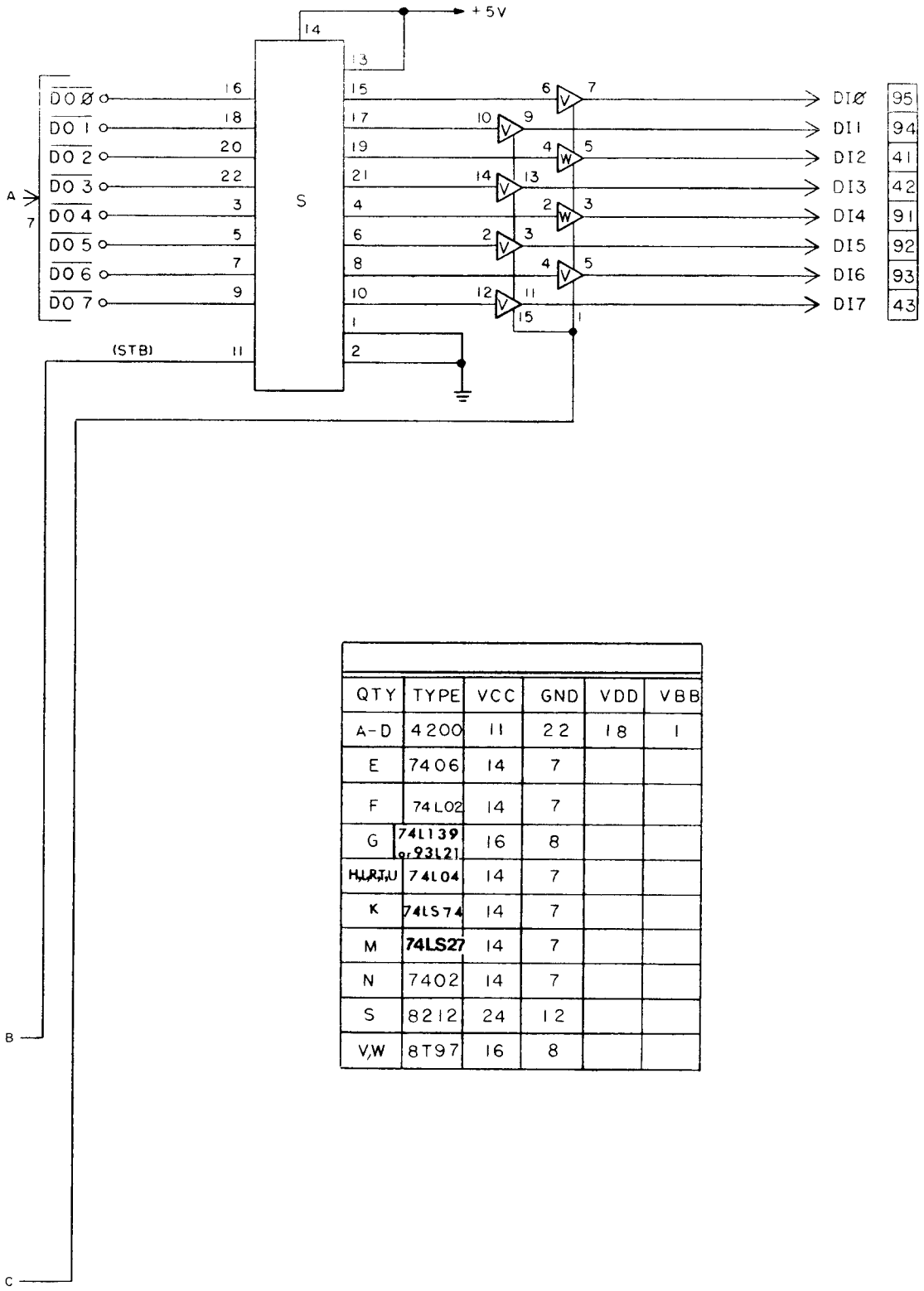


Fig. C.11 Pertec/MITS 88-MCS, 16 kbyte static memory card (*sheet 1 of 4*)



QTY	TYPE	VCC	GND	VDD	VBB
A-D	4200	11	22	18	1
E	7406	14	7		
F	74L02	14	7		
G	74L139 or 93L21	16	8		
H,J,R,T,U	74L04	14	7		
K	74LS74	14	7		
M	74LS27	14	7		
N	7402	14	7		
S	8212	24	12		
V,W	8T97	16	8		

Fig. C.11 Pertec/MITS 88-MCS, 16 kbyte static memory card *(sheet 2 of 4)*

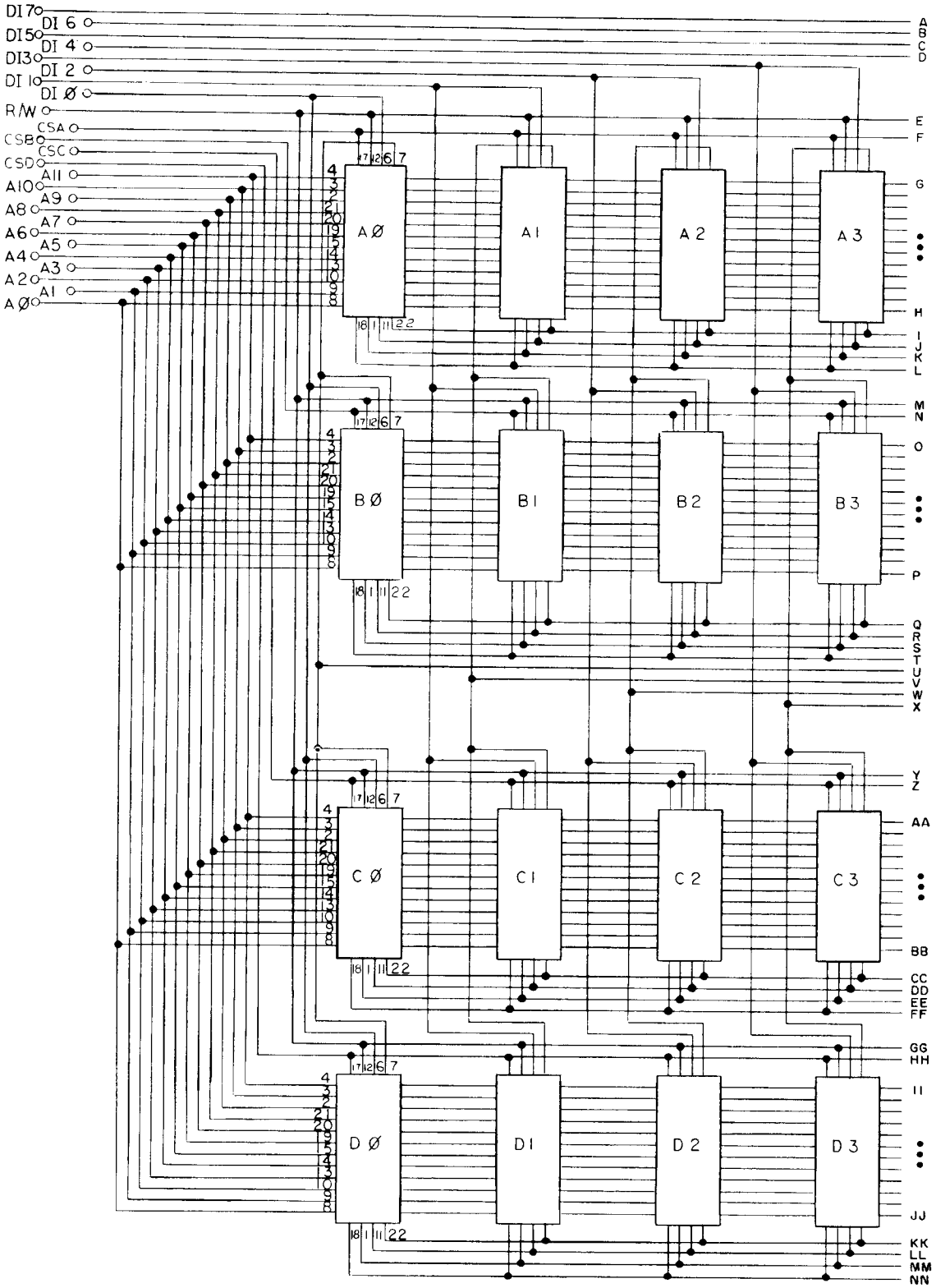


Fig. C.11 Pertec/MITS 88-MCS, 16 kbyte static memory card *(sheet 3 of 4)*

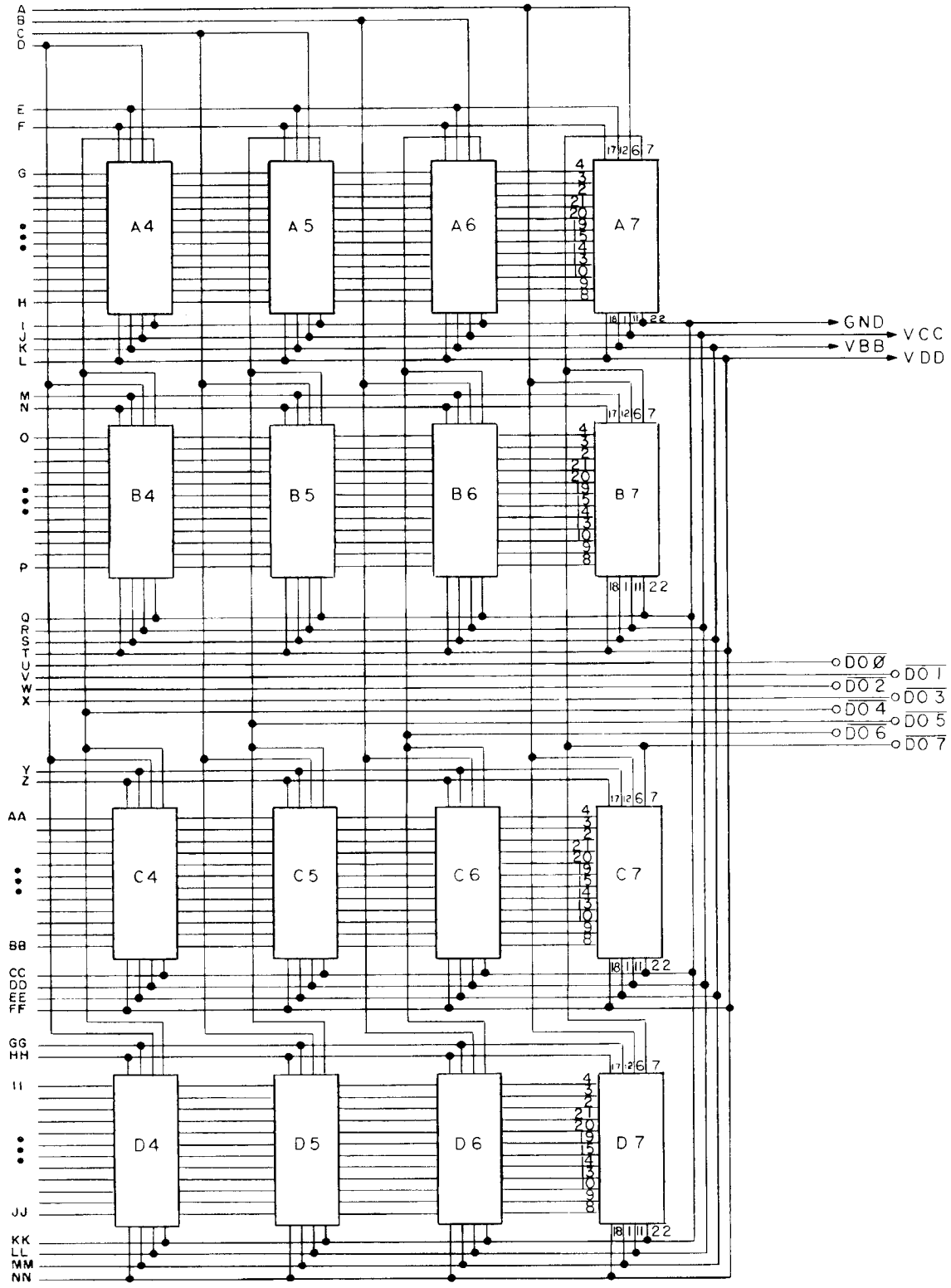


Fig. C.11 Pertec/MITS 88-MCS, 16 kbyte static memory card *(sheet 4 of 4)*

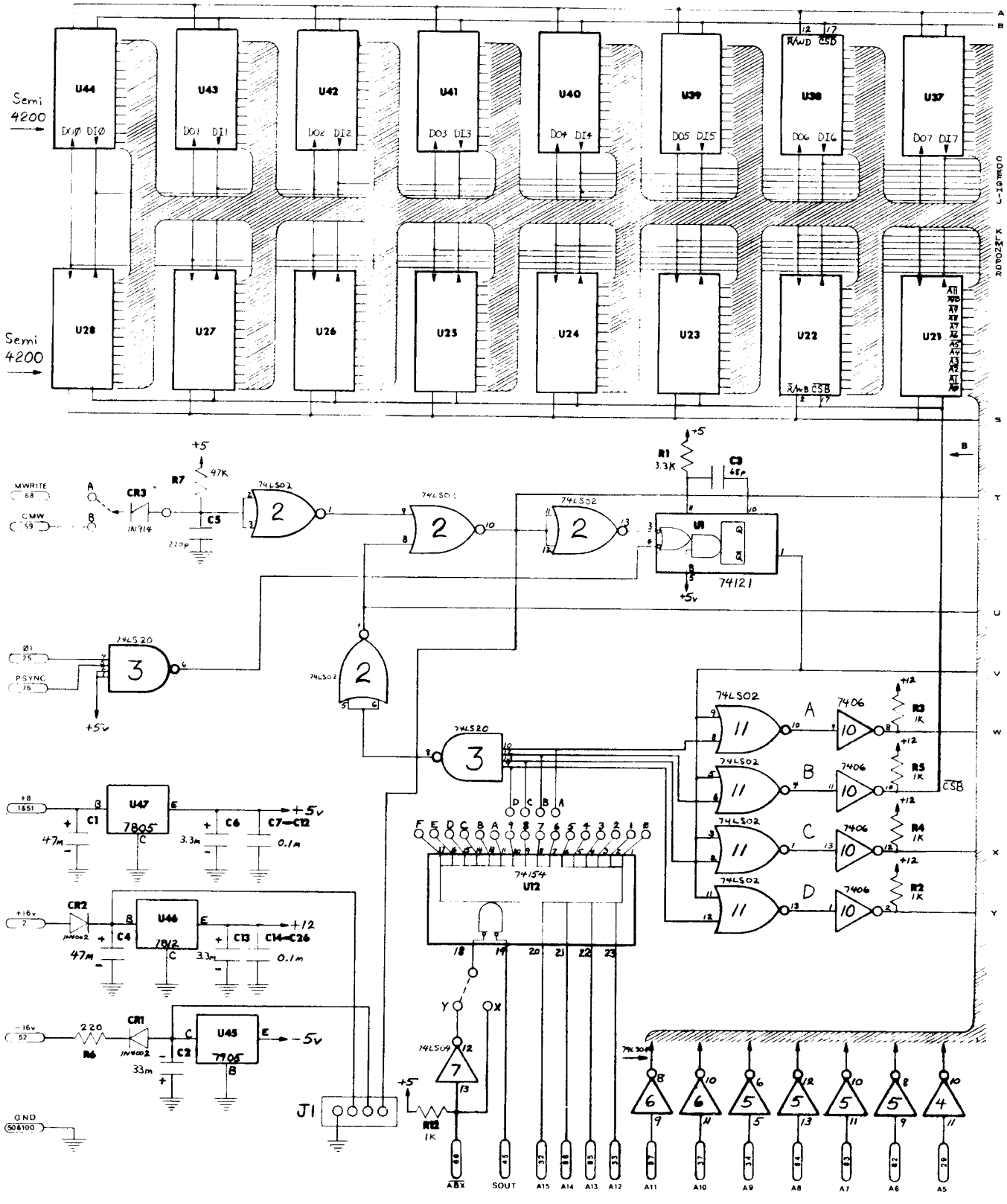
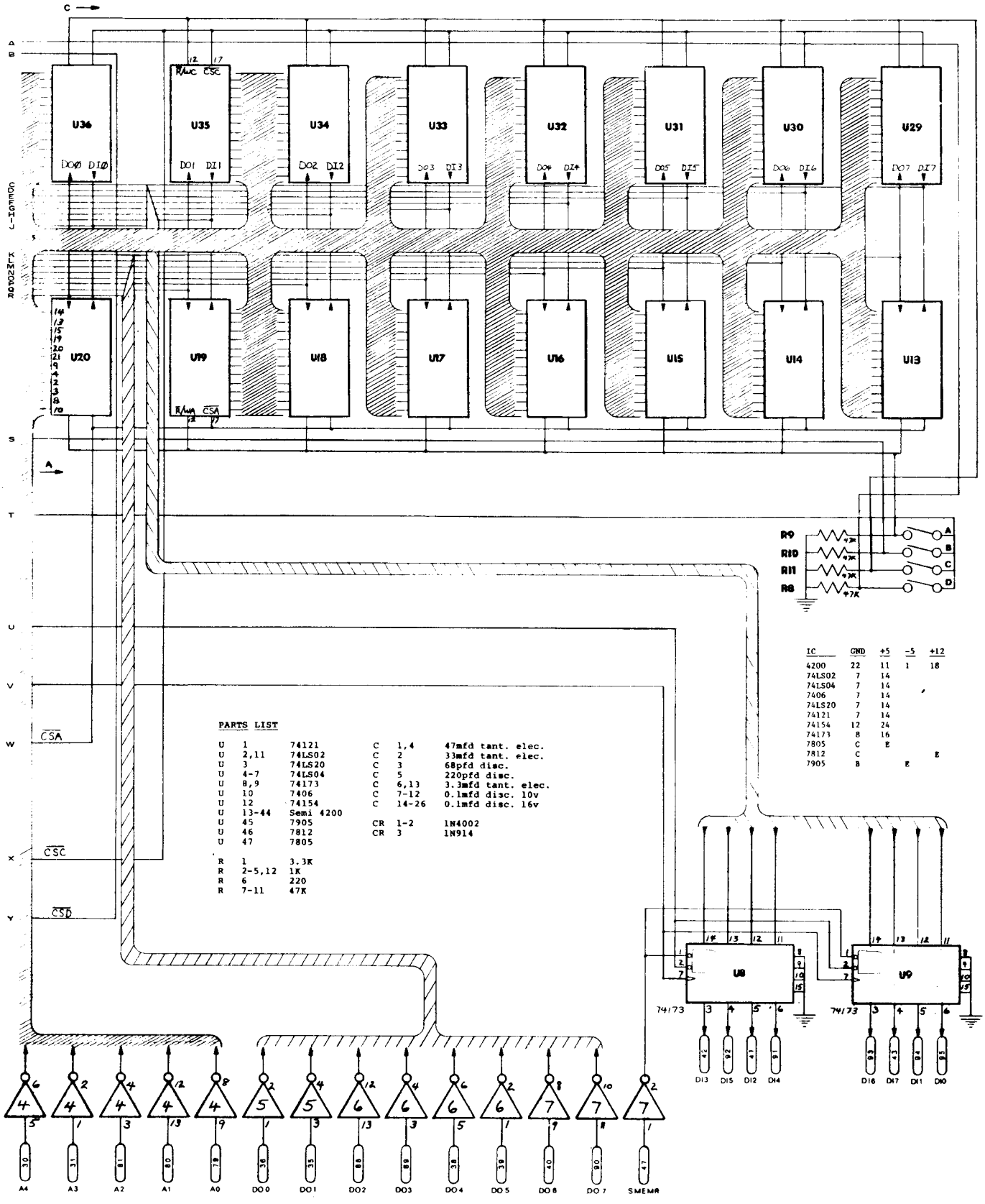


Fig. C.12 Xitan/TDL Z16, 16 kbyte static memory card (*sheet 1 of 2*)



PARTS LIST

U	1	74121	C	1,4	47mfd tant. elec.
U	2,11	74LS02	C	2	33mfd tant. elec.
U	3	74LS20	C	3	68pfd disc.
U	4-7	74LS04	C	5	220pfd disc.
U	8,9	74173	C	6,13	3.3mfd tant. elec.
U	10	7406	C	7-12	0.1mfd disc. 10v
U	12	74154	C	14-26	0.1mfd disc. 16v
U	13-44	Semi 4200			
U	45	7905	CR	1-2	1N4002
U	46	7812	CR	3	1N914
U	47	7805			
R	1	3.3K			
R	2-5,12	1K			
R	6	220			
R	7-11	47K			

IC	GND	+5	-5	+12
4200	22	11	1	18
74LS02	7	14		
74LS04	7	14		
7406	7	14		
74LS20	7	14		
74121	7	14		
74154	12	24		
74173	8	16		
7805	C	E		
7812	C			E
7905	B		E	

Fig. C.12 Xitan/TDL Z16, 16 kbyte static memory card (*sheet 2 of 2*)

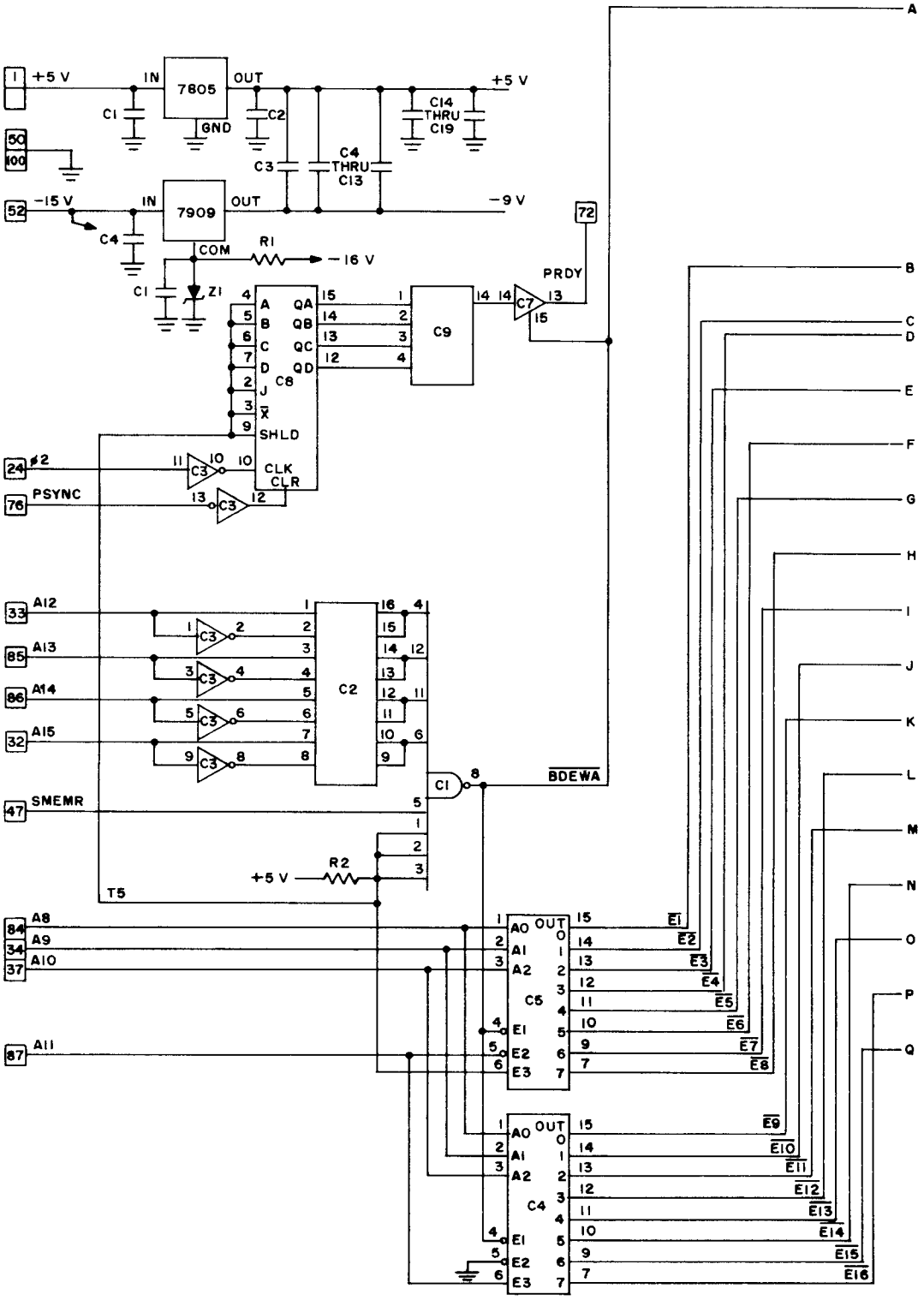
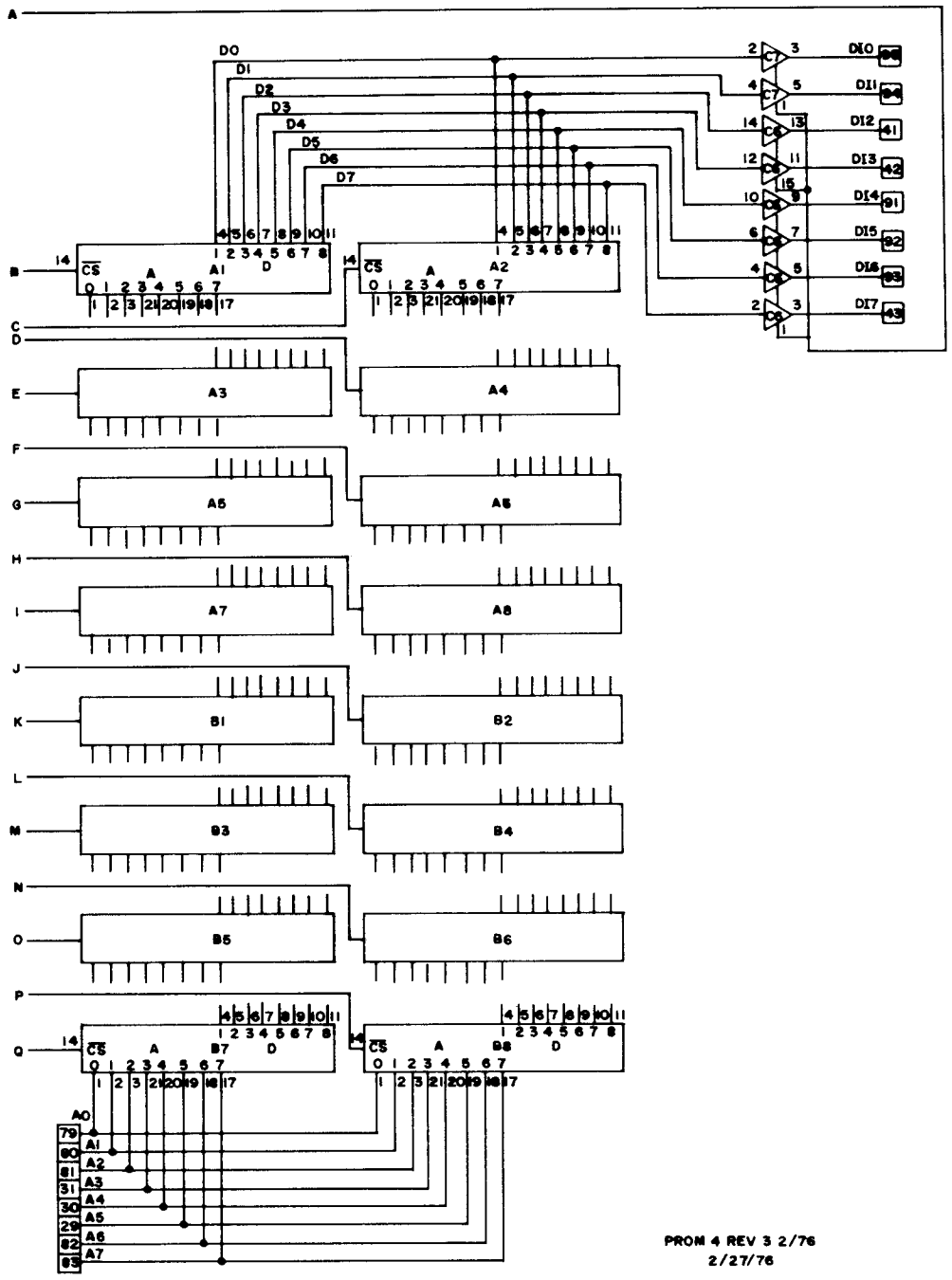


Fig. C.13 Imsai 8080 PROM 4, 1702A PROM memory card *(sheet 1 of 2)*



A1 THRU A8	1702A
B1 THRU B8	1702A
C1	74LS30
C2, C9	JUMPER SOCKET
C3	74LS04
C4, C5	8205
C6, C7	8T97
C8	74195
C2 THRU C5	33 mF
C1, C6 THRU C19	1 mF
Z1	IN748
R1, R2	1K, 1/4 W

Fig. C.13 Imsai 8080 PROM 4, 1702A PROM memory card (*sheet 2 of 2*)

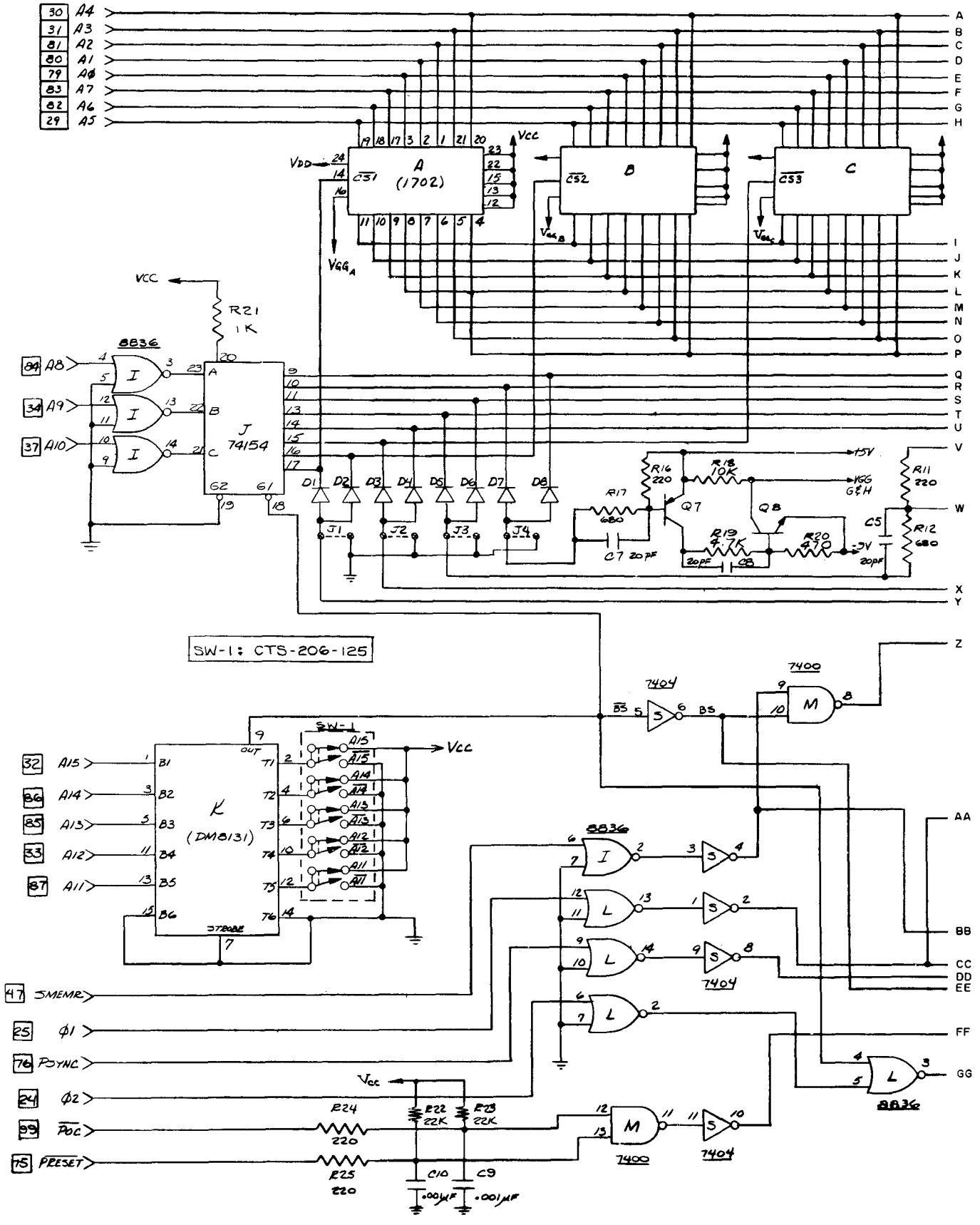


Fig. C.14 Pertec/MITS 88-PMC, 1702A PROM memory card (*sheet 1 of 2*)

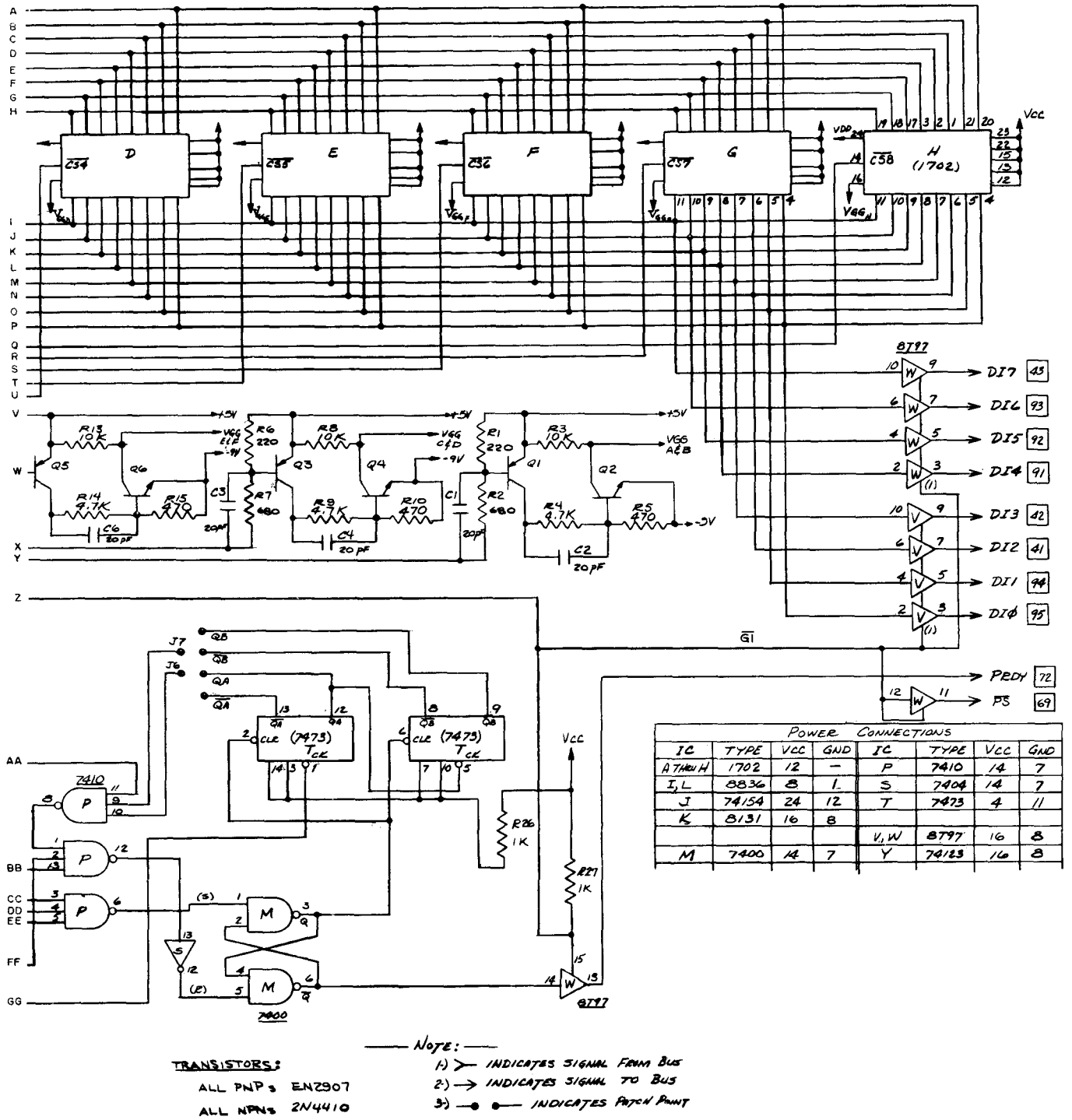


Fig. C.14 Pertec/MITS 88-PMC, 1702A PROM memory card (*sheet 2 of 2*)

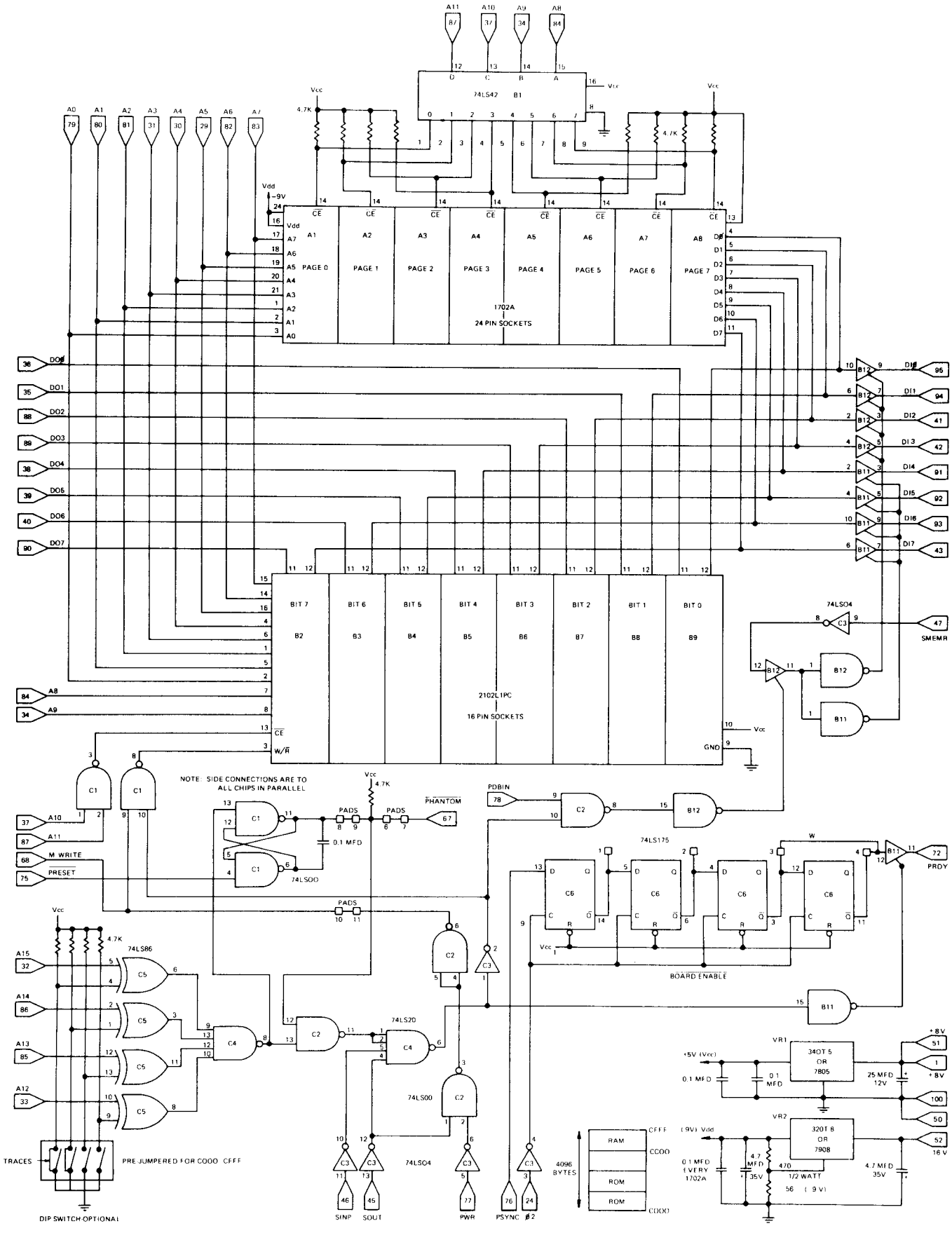


Fig. C.15 Vector Graphic PROM/RAM, combination PROM and RAM card

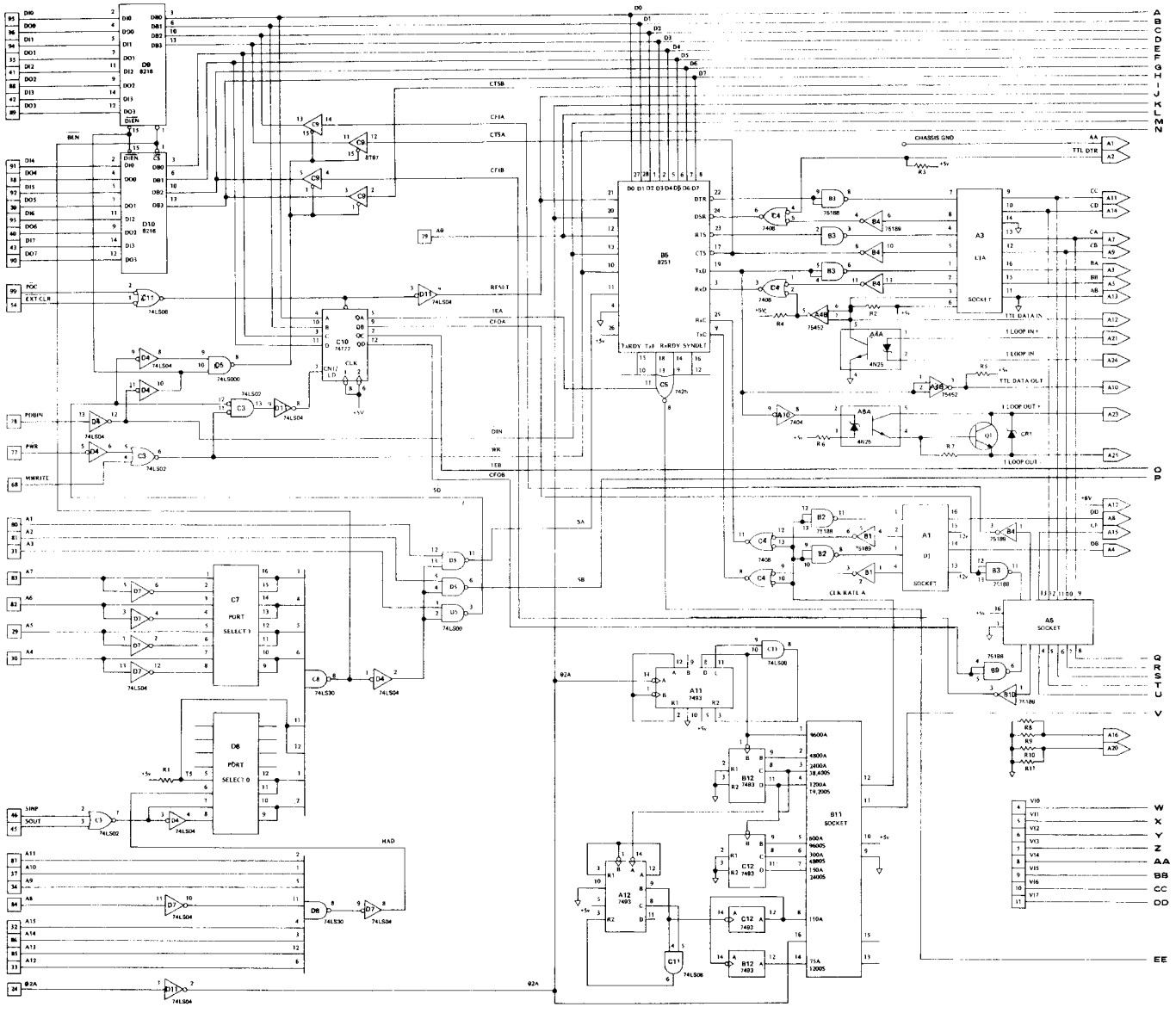
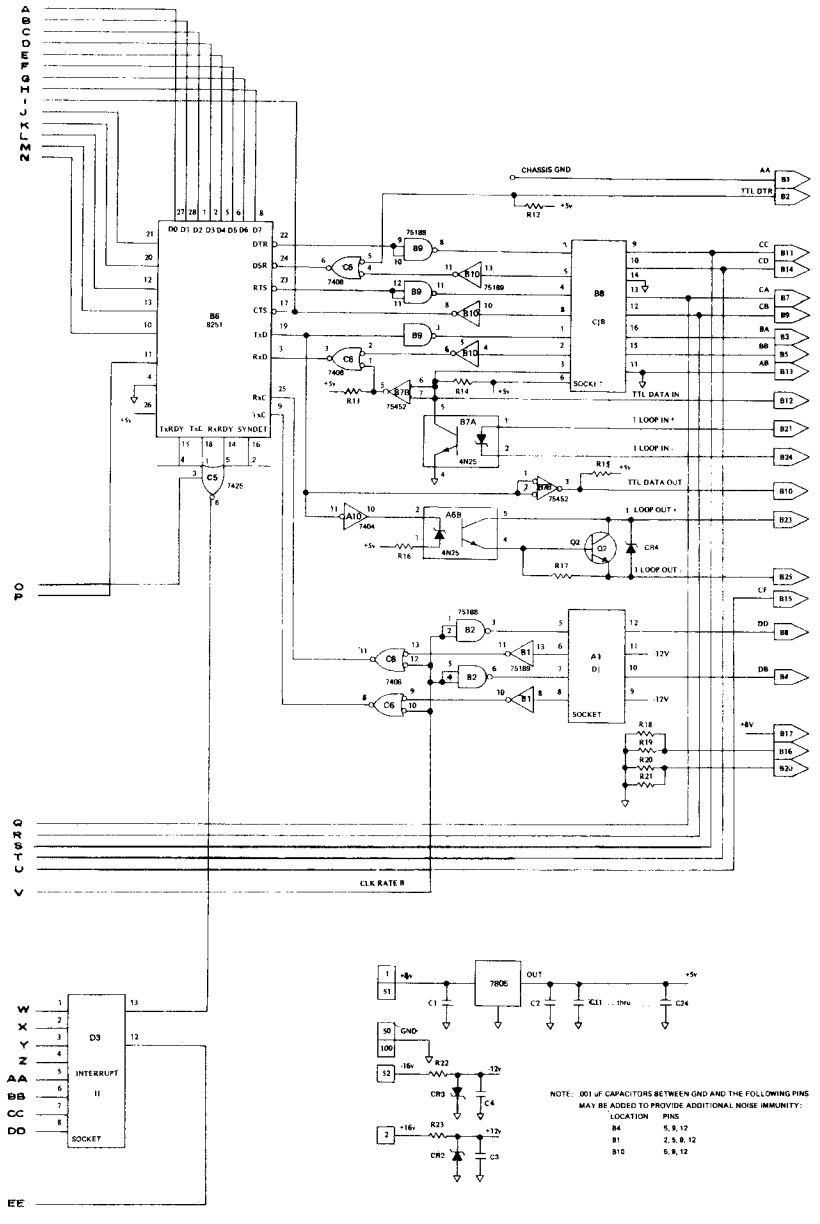


Fig. C.16 Imsai 8080 SIO 2-2, dual series interface card (*sheet 1 of 2*)



A1, A3, A5	OPTIONAL JUMPER SOCKET	R1, R2, R3, R13, R14	1K ¼W
A4B	75452		
A4A, A6A, A6B	4N25	R4, R5, R6, R12, R15, R16	220 ¼W
A10	7404		
A11, A12	7493		
B1, B4, B10	75189	R7, R17	4.7 ¼W
B2, B3, B9	74188	R8, R9, R10, R11, R18, R19, R20, R21	470 ¼W
B5, B6	8251		
B7A	4N25		
B7B	75452		
B8, B11	OPTIONAL JUMPER SOCKET	R22, R23	56 ½W
B12	7493		
C3	74LS02		
C4, C6	7408		
C5	7425		
C7	PORT SELECT 1		
C8	74LS30		
C9	8T97		
C10	74177		
C11	74LS08		
C12	7493		
D3, D6	OPTIONAL JUMPER SOCKET		
D4, D7, D11	74LS04		
D5	74LS00		
D8	74LS30		
D9, D10	8216		
C1 thru C4	33 uF		
C11 thru C24	.1 uF		
CR1, CR4	1N914		
CR2, CR3	1N4742		
Q1, Q2	2N3904		

Fig. C.16 Imsai 8080 SIO 2-2, dual series interface card (*sheet 2 of 2*)

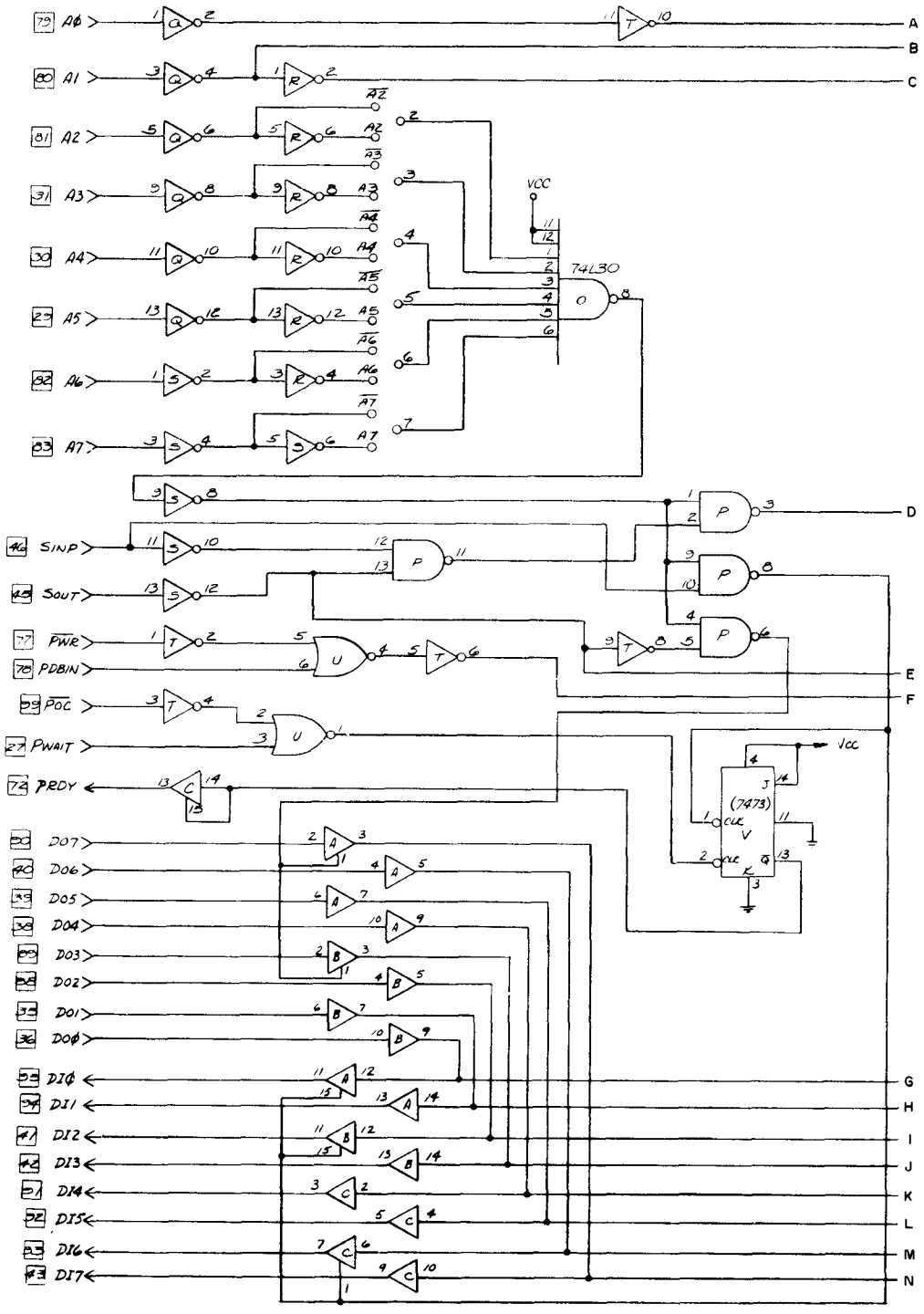


Fig. C.17 Pertec/MITS 88-2SIO, dual series interface card (*sheet 1 of 3*)

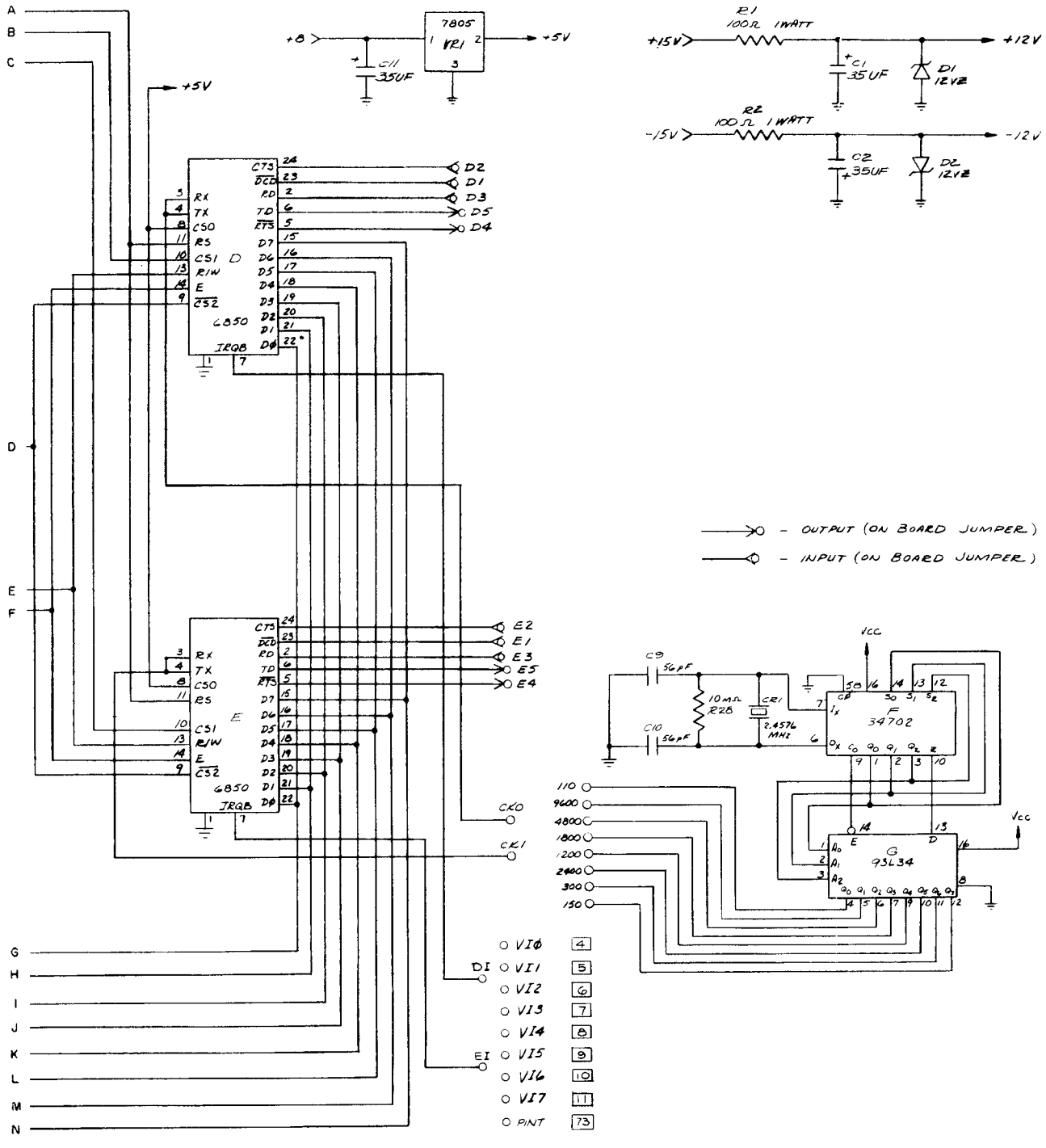


Fig. C.17 Pertec/MITS 88-2SIO, dual series interface card (*sheet 2 of 3*)

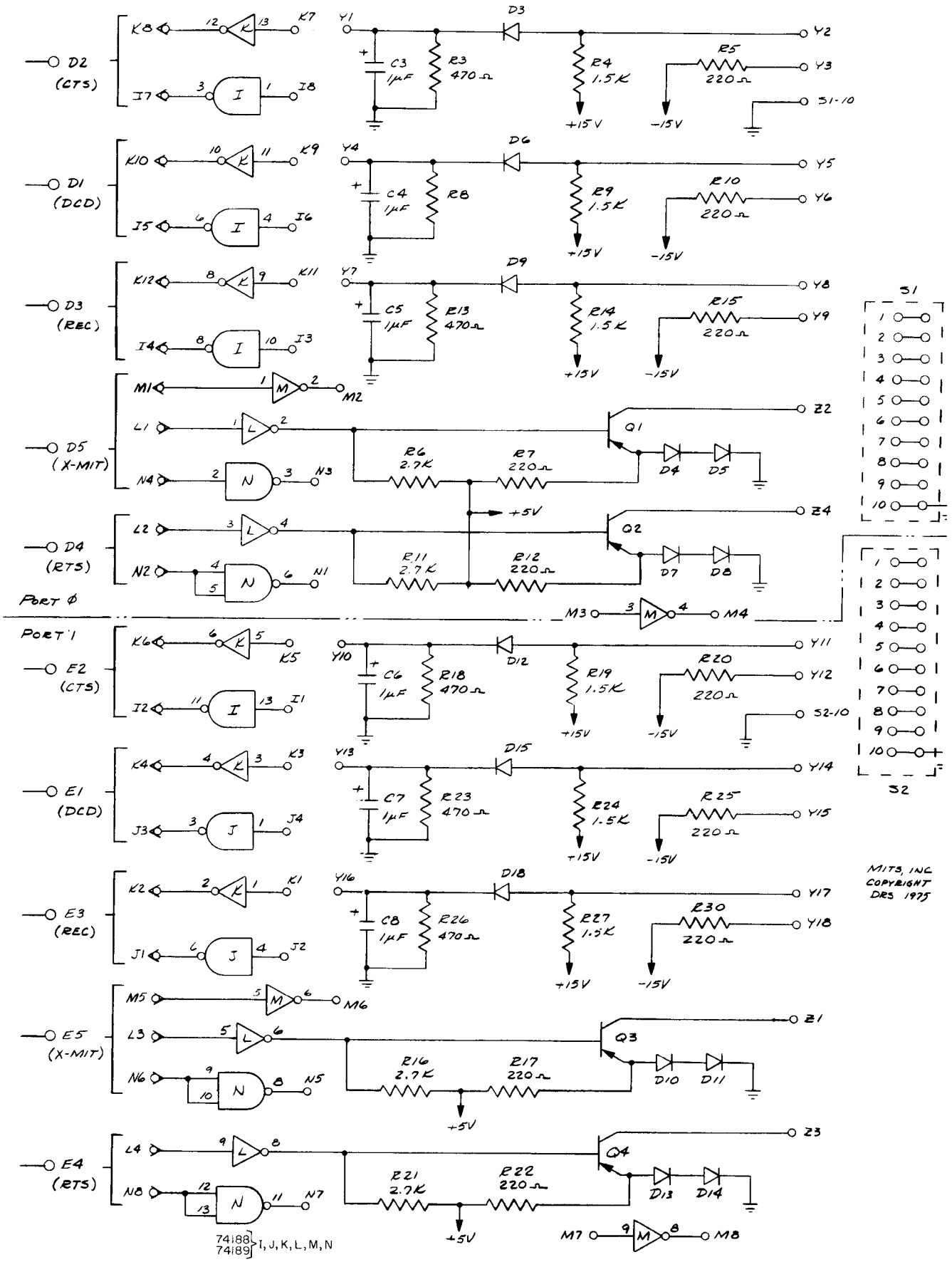


Fig. C.17 Pertec/MITS 88-2SIO, dual serial interface card (*sheet 3 of 3*)

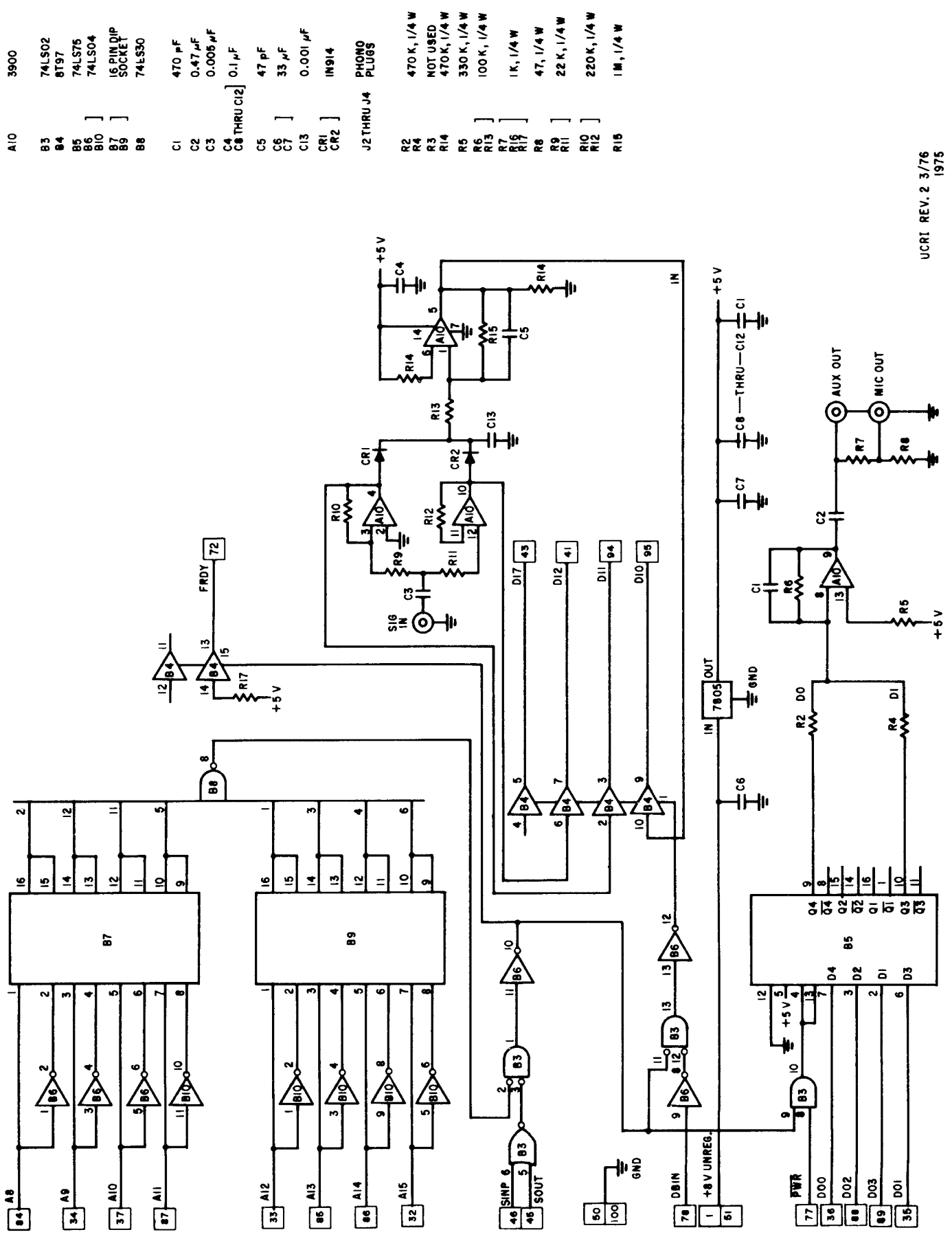


Fig. C.18 Imsai 8080 UCRI, cassette interface card

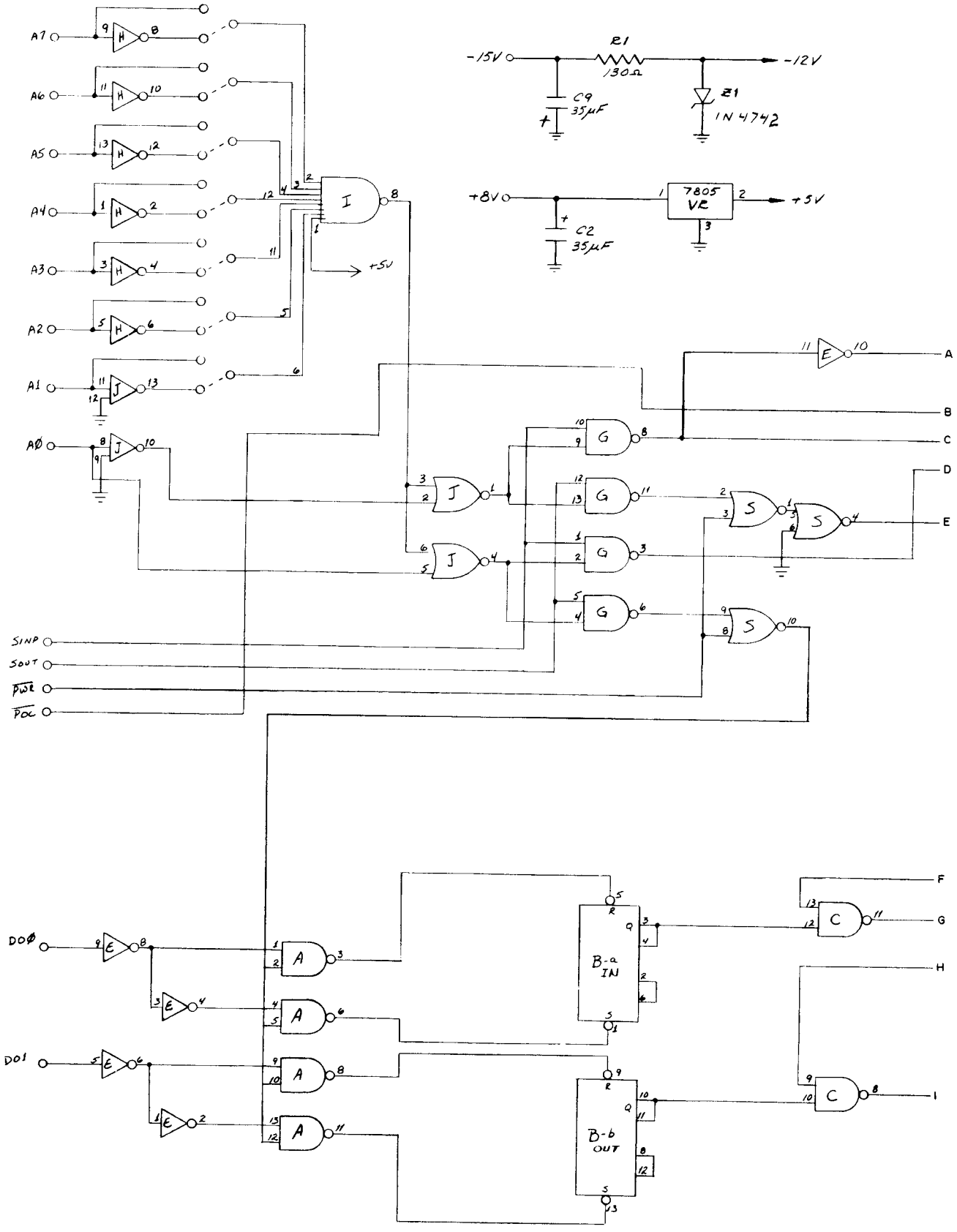


Fig. C.19 Pertec/MITS 88-ACR, cassette interface card (*sheet 1 of 5*)

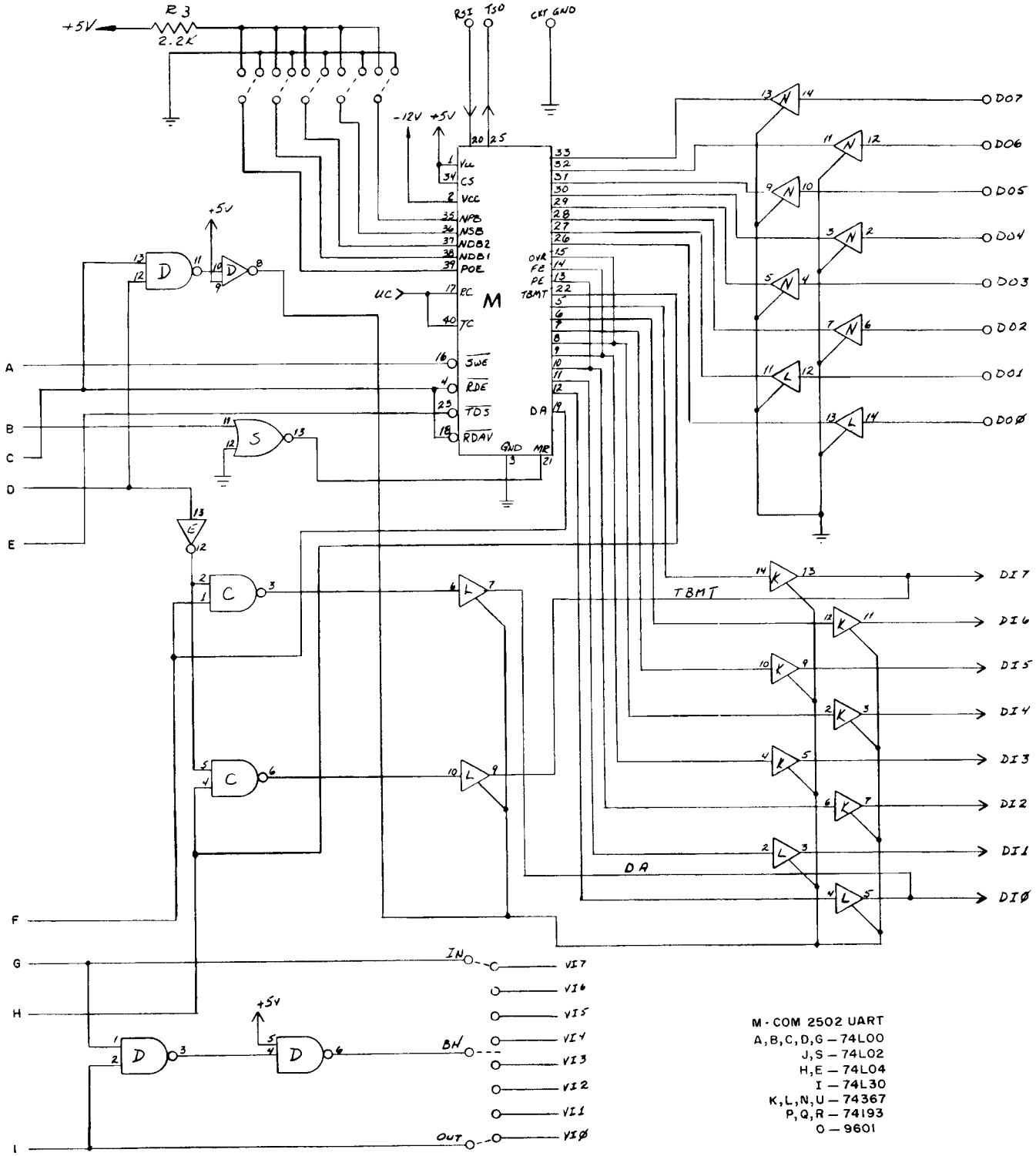


Fig. C.19 Pertec/MITS 88-ACR, cassette interface card (*sheet 2 of 5*)

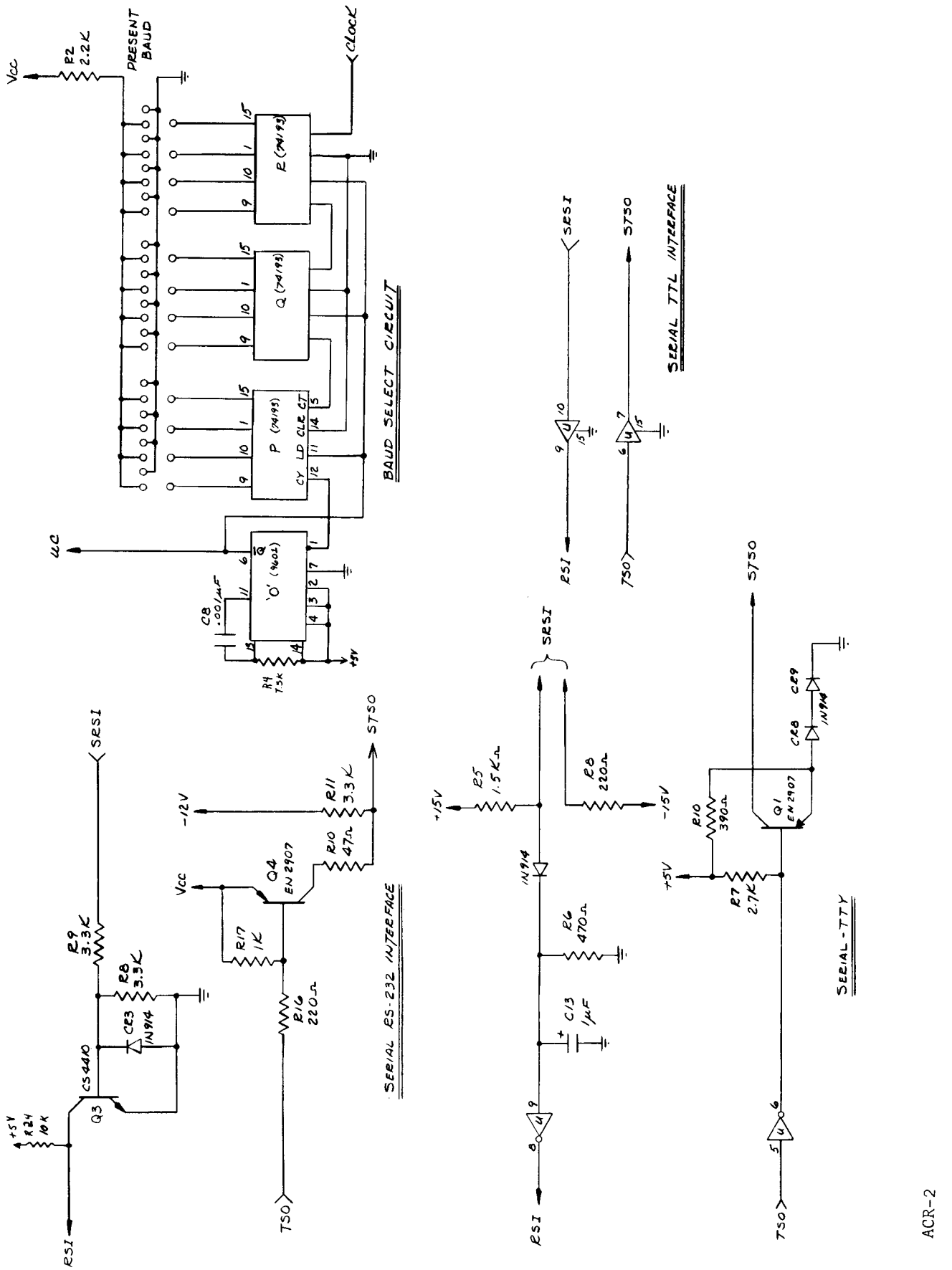
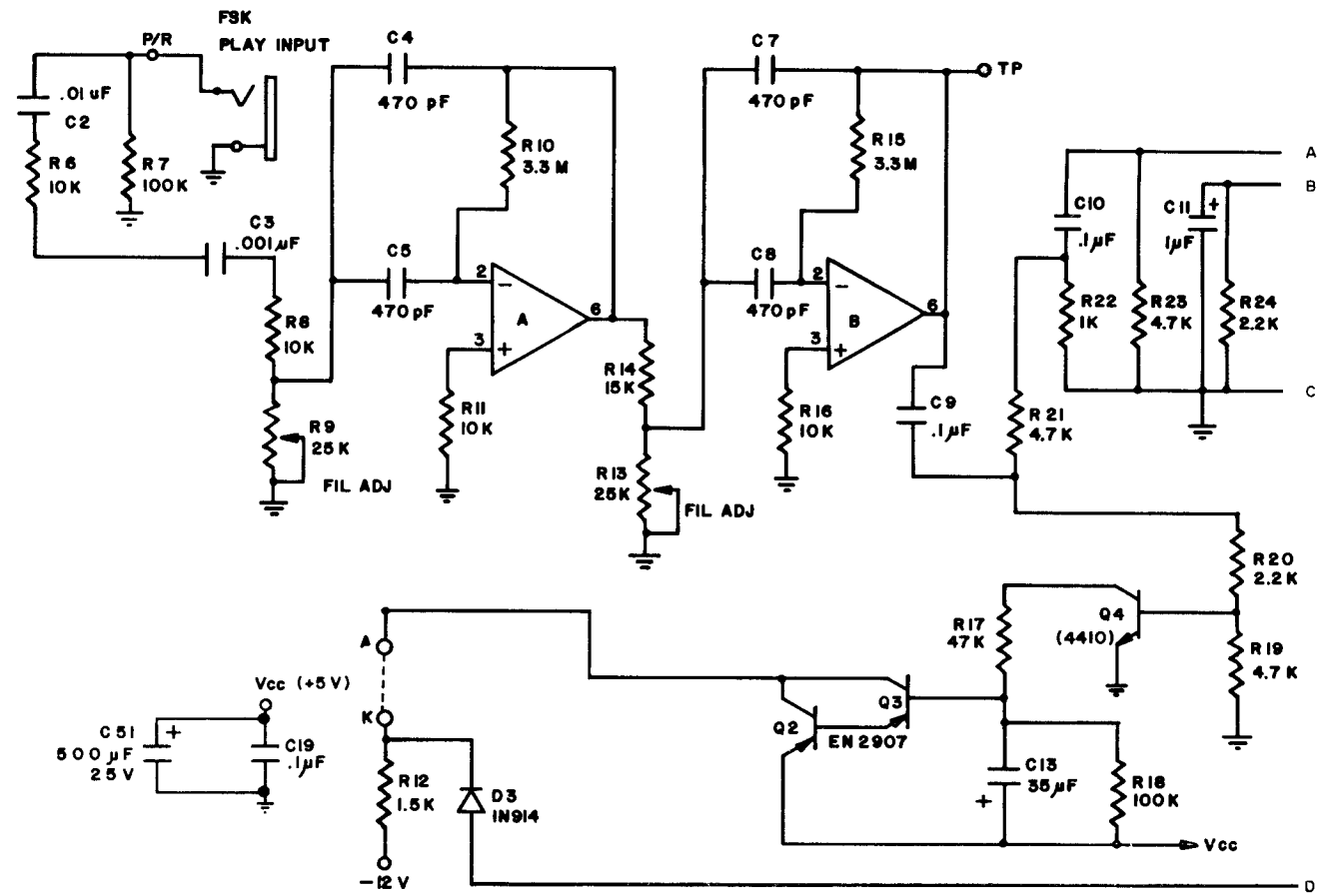


Fig. C.19 Pertec/MITS 88-ACR, cassette interface card *(sheet 3 of 5)*



POWER CONNECTIONS				
IC	TYPE	V_{CC}	GND	-12V
A	741	7		4
B	741	7		4
C	XR 210	16		7
E	7493	5	10	
G	7402	14	7	
H	7420	14	7	
J	9316	16	8	
K	9316	16	8	

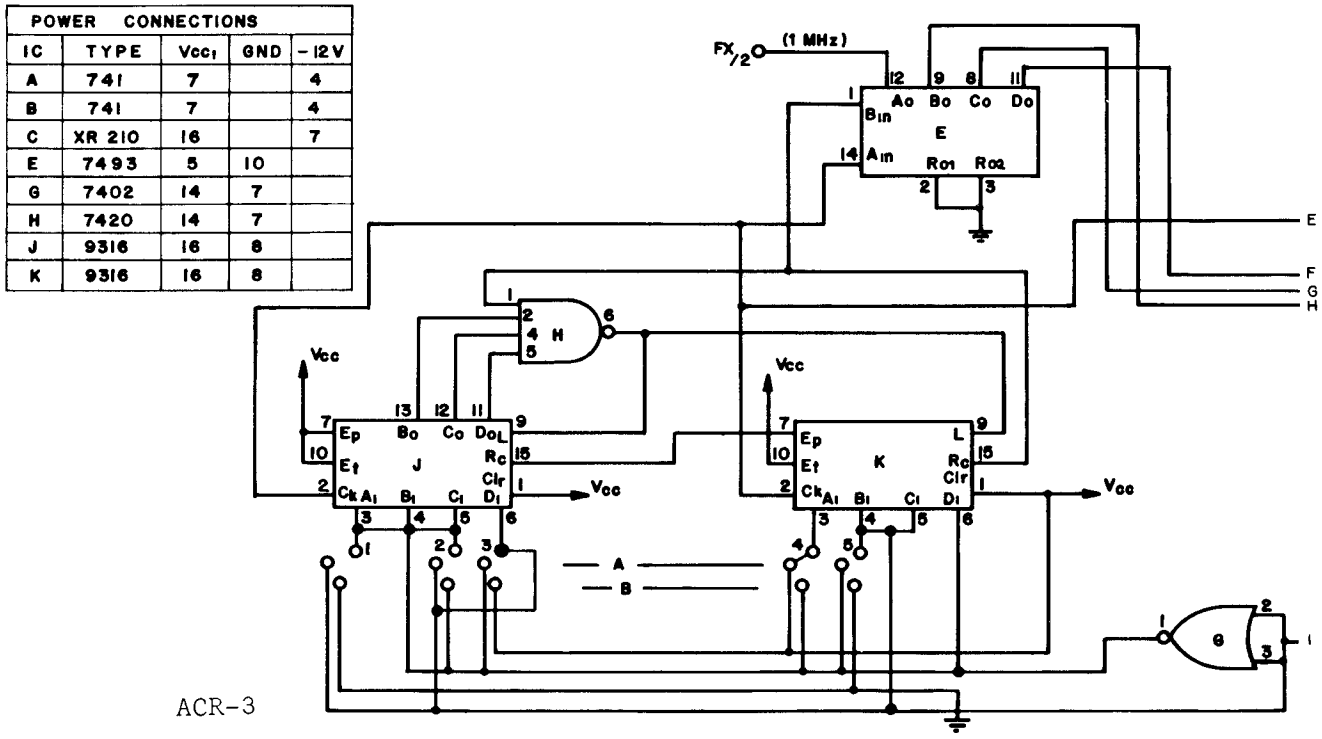


Fig. C.19 Pertec/MITS 88-ACR, cassette interface card *(sheet 4 of 5)*

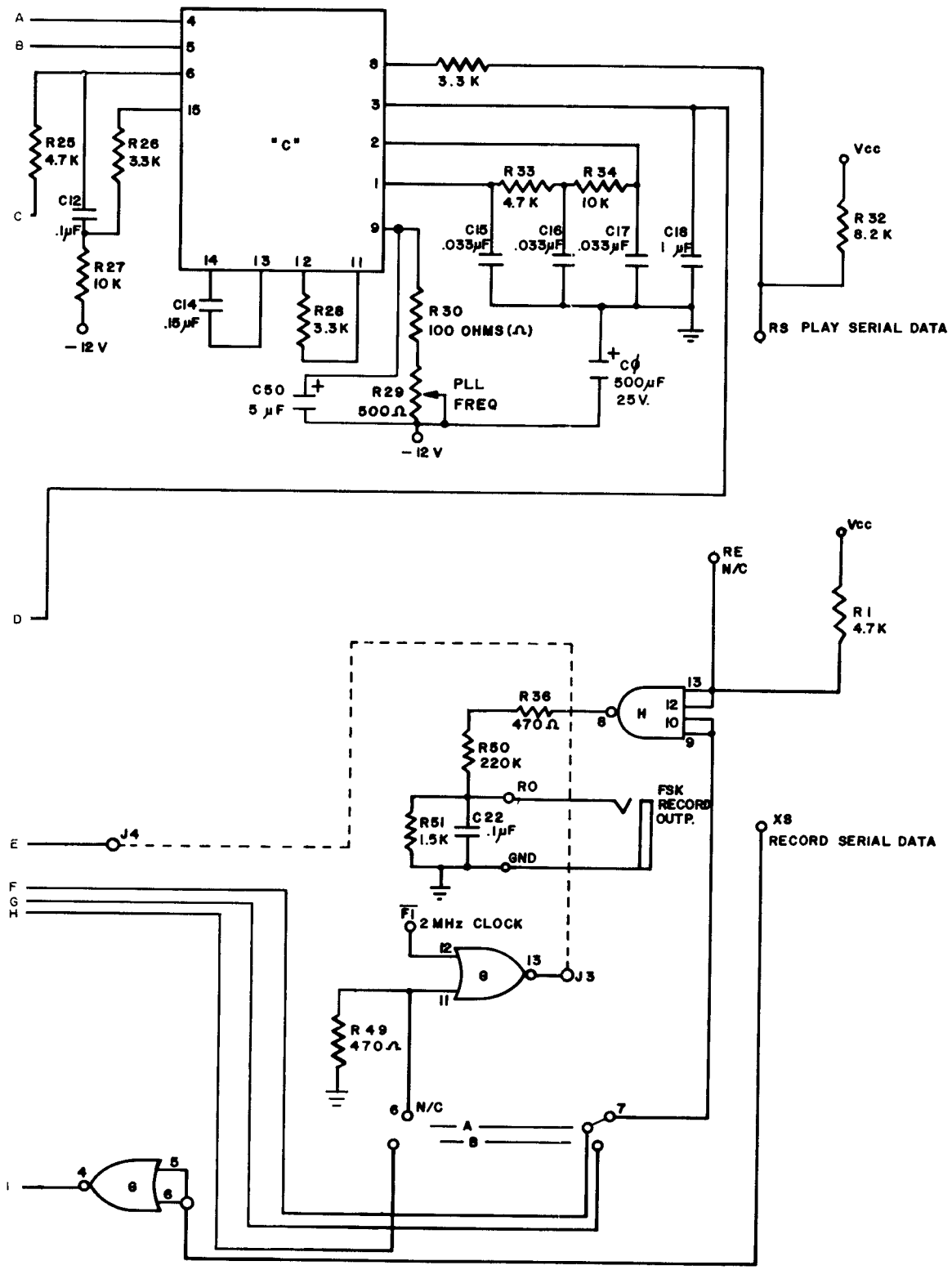


Fig. C.19 Pertec/MITS 88-ACR, cassette interface card *(sheet 5 of 5)*

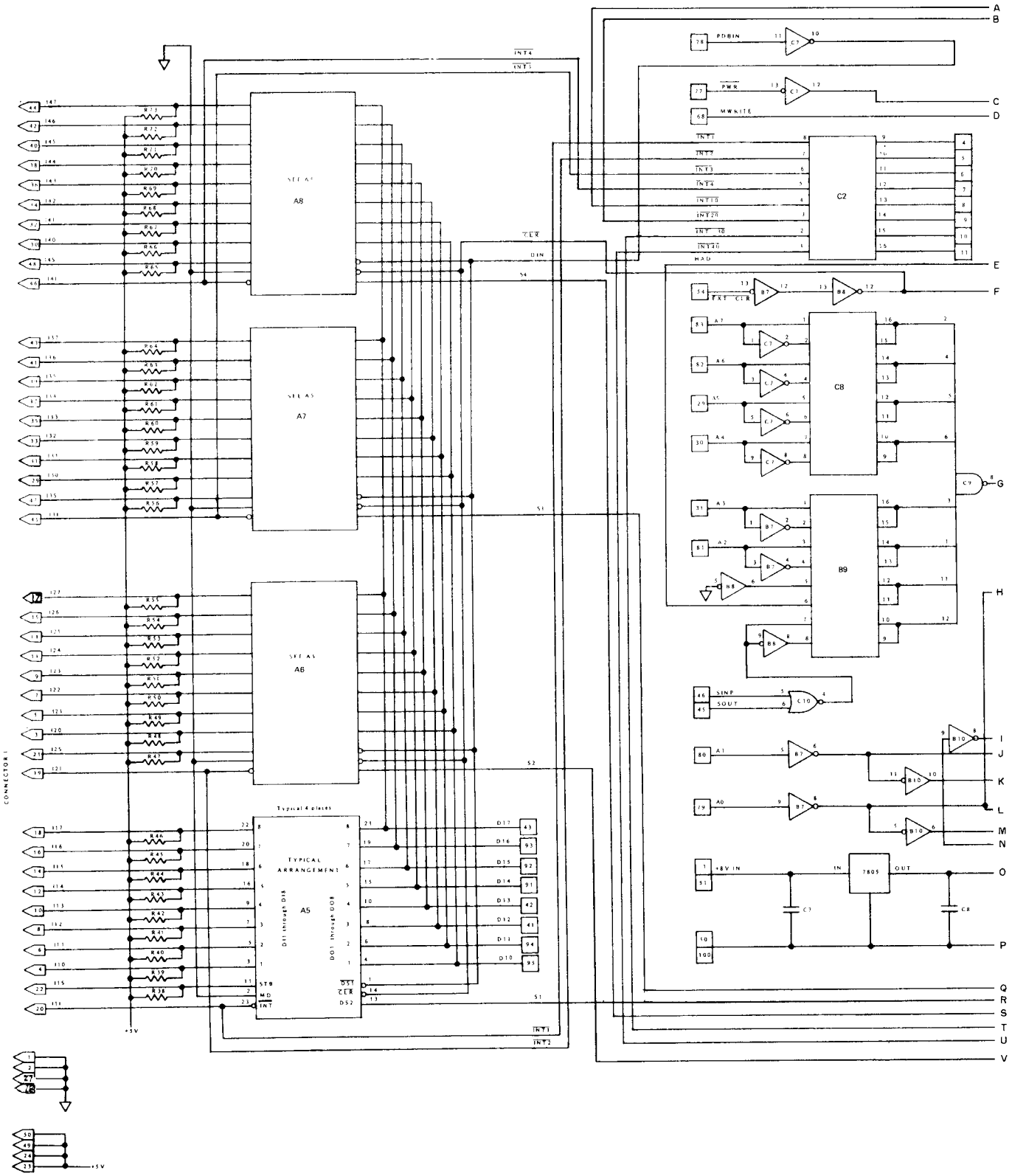


Fig. C.20 Imsai 8080 PIO, quad parallel 8-bit port card *(sheet 1 of 2)*

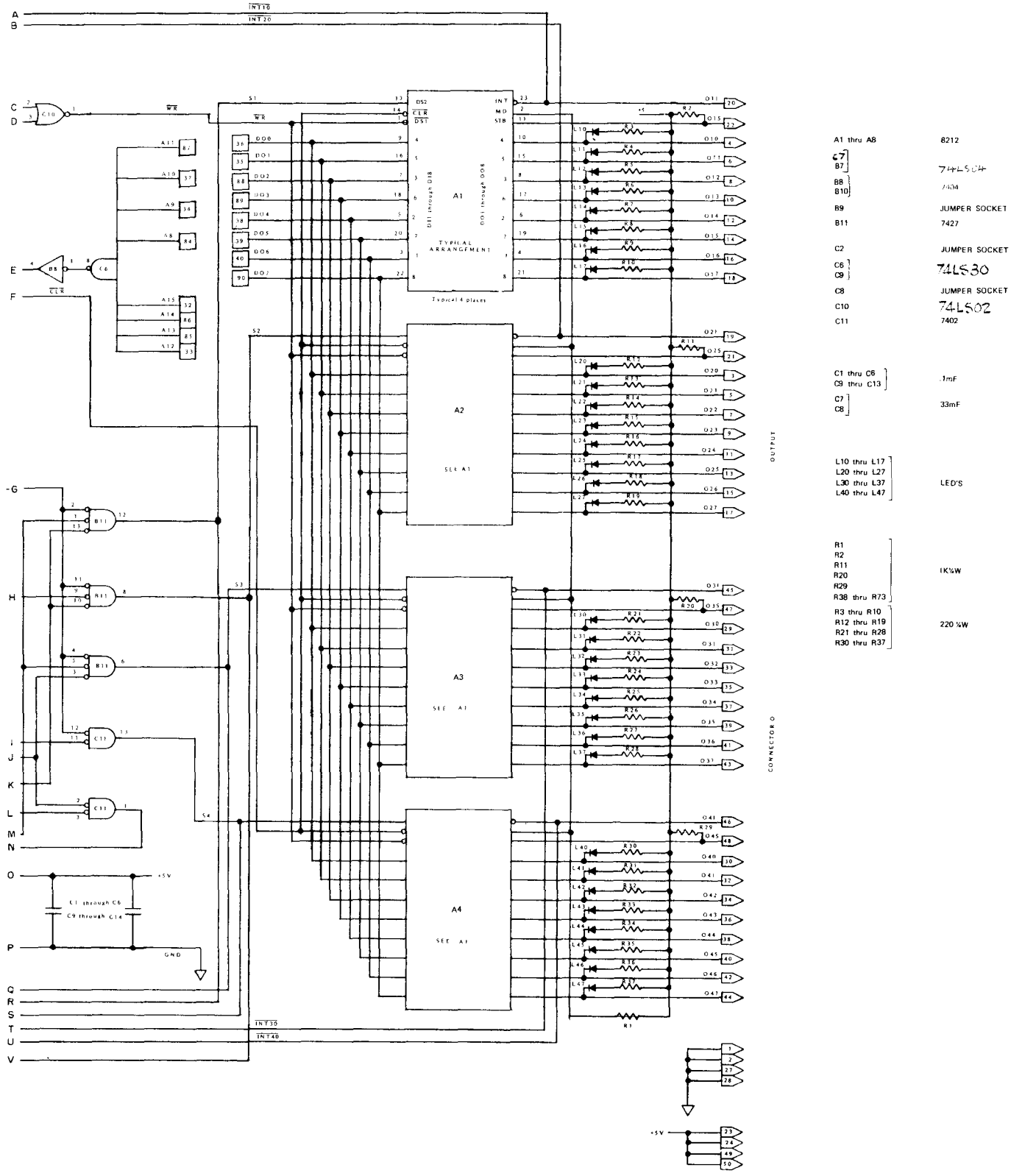


Fig. C.20 Imsai 8080 PIO, quad parallel 8-bit port card (*sheet 2 of 2*)

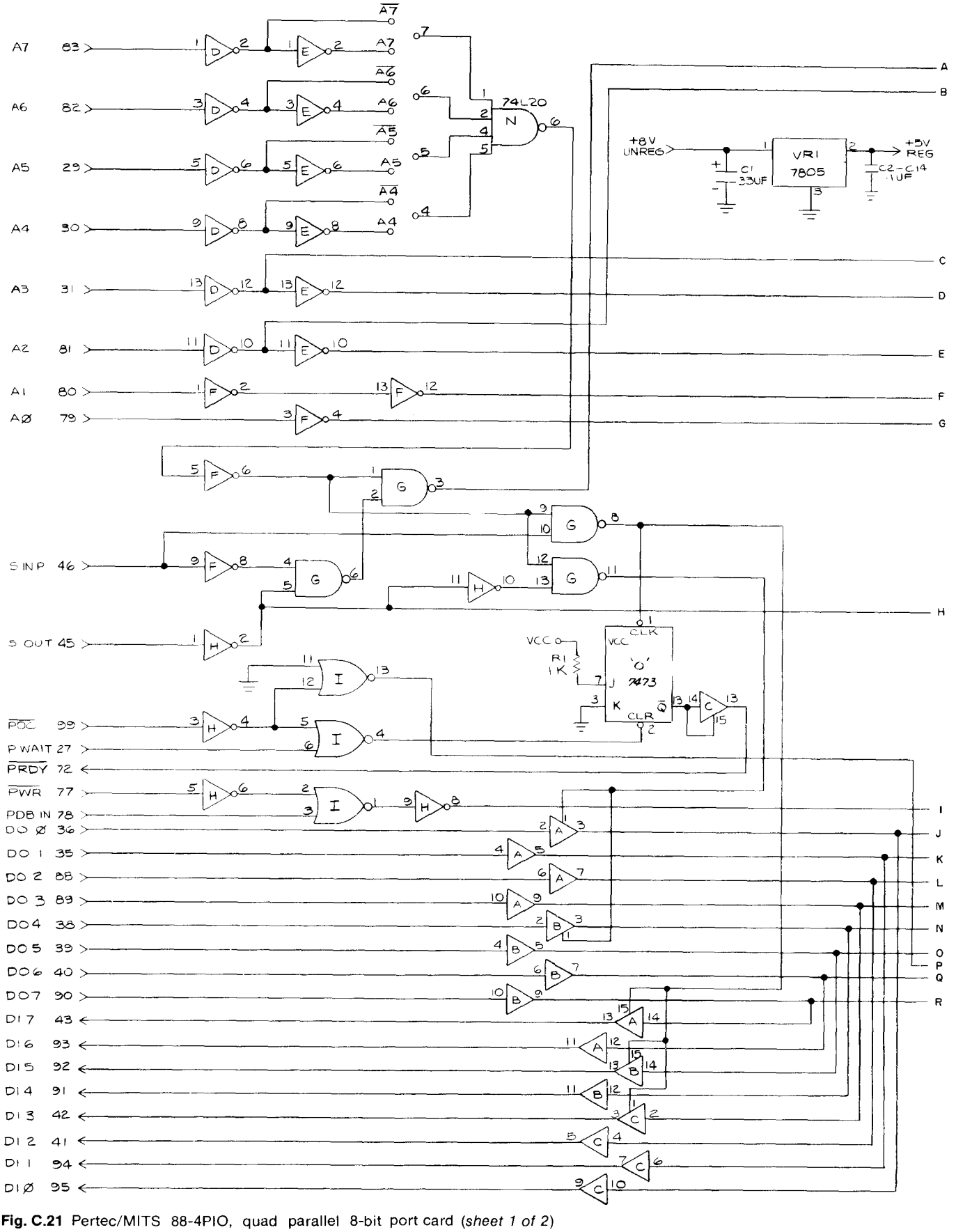
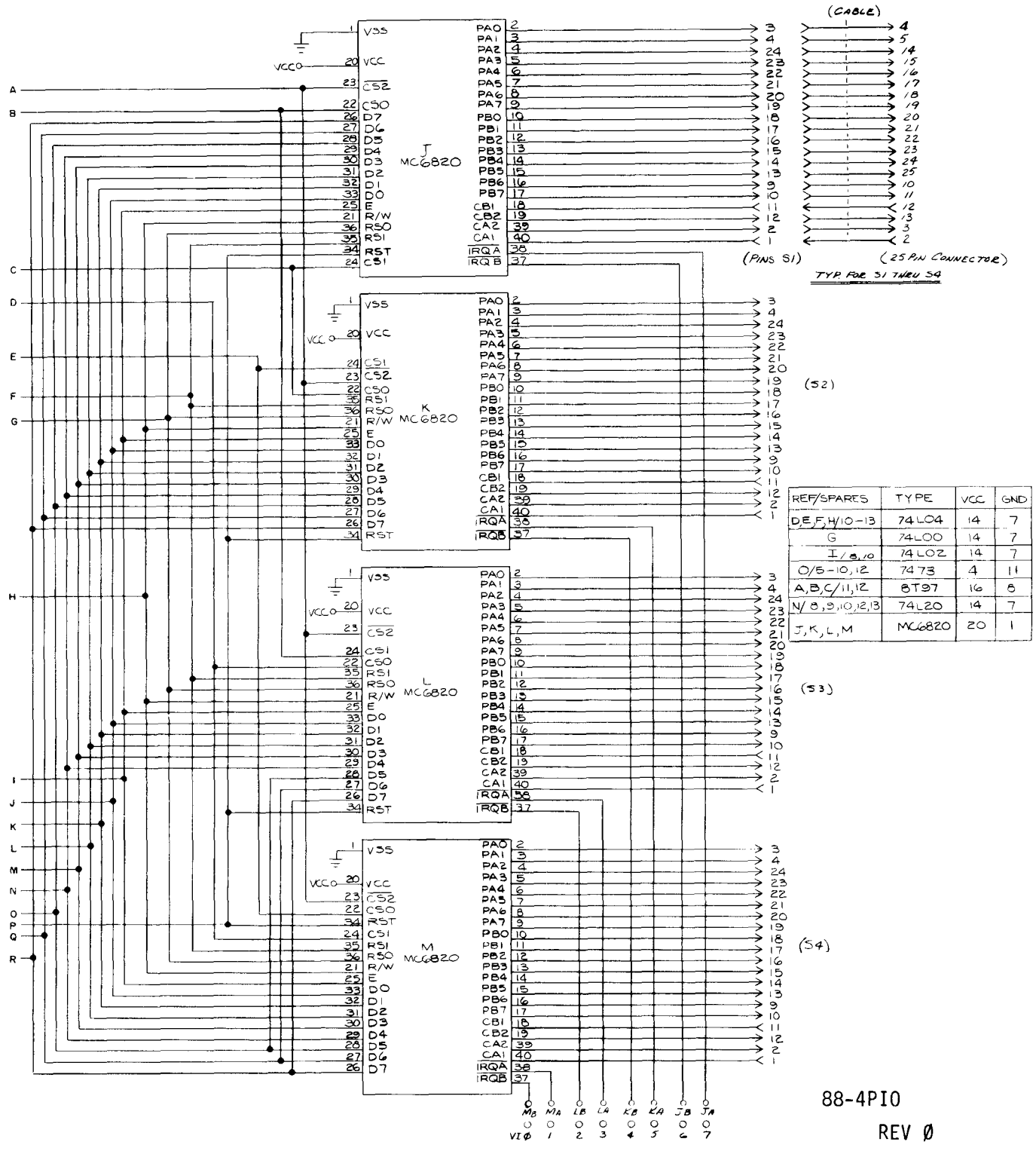


Fig. C.21 Pertec/MITS 88-4PIO, quad parallel 8-bit port card (*sheet 1 of 2*)



REF/SPARES	TYPE	VCC	GND
D,E,F,H/10-13	74L04	14	7
G	74L00	14	7
I/8,10	74L02	14	7
O/5-10,12	7473	4	11
A,B,C/11,12	8T97	16	8
N/8,9,10,12,13	74L20	14	7
J,K,L,M	MC6820	20	1

Fig. C.21 Pertec/MITS 88-4PIO, quad parallel 8-bit port card (*sheet 2 of 2*)

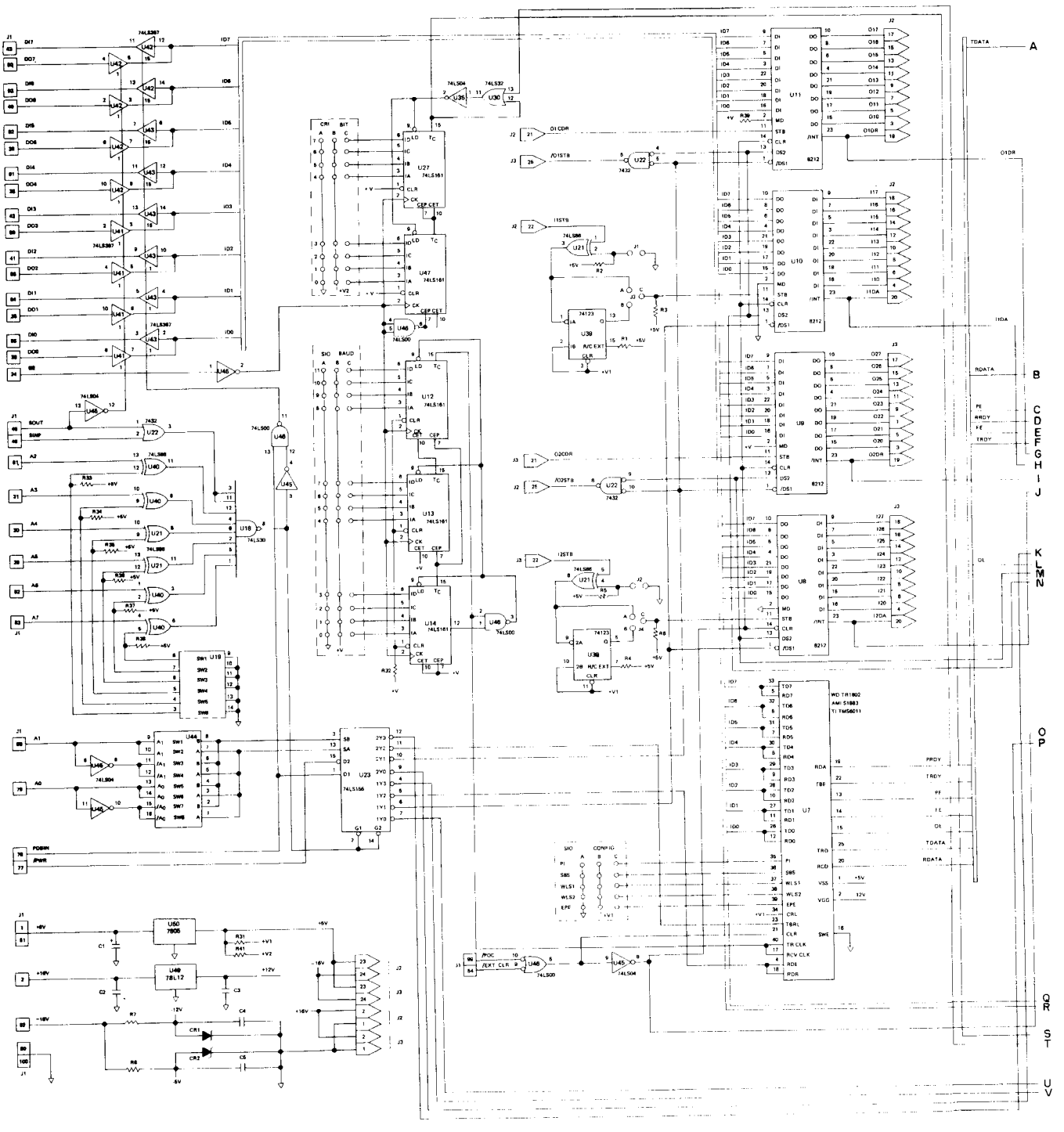


Fig. C.22 Imsai 8080 MIO, multiple parallel and serial I/O card (*sheet 1 of 3*)

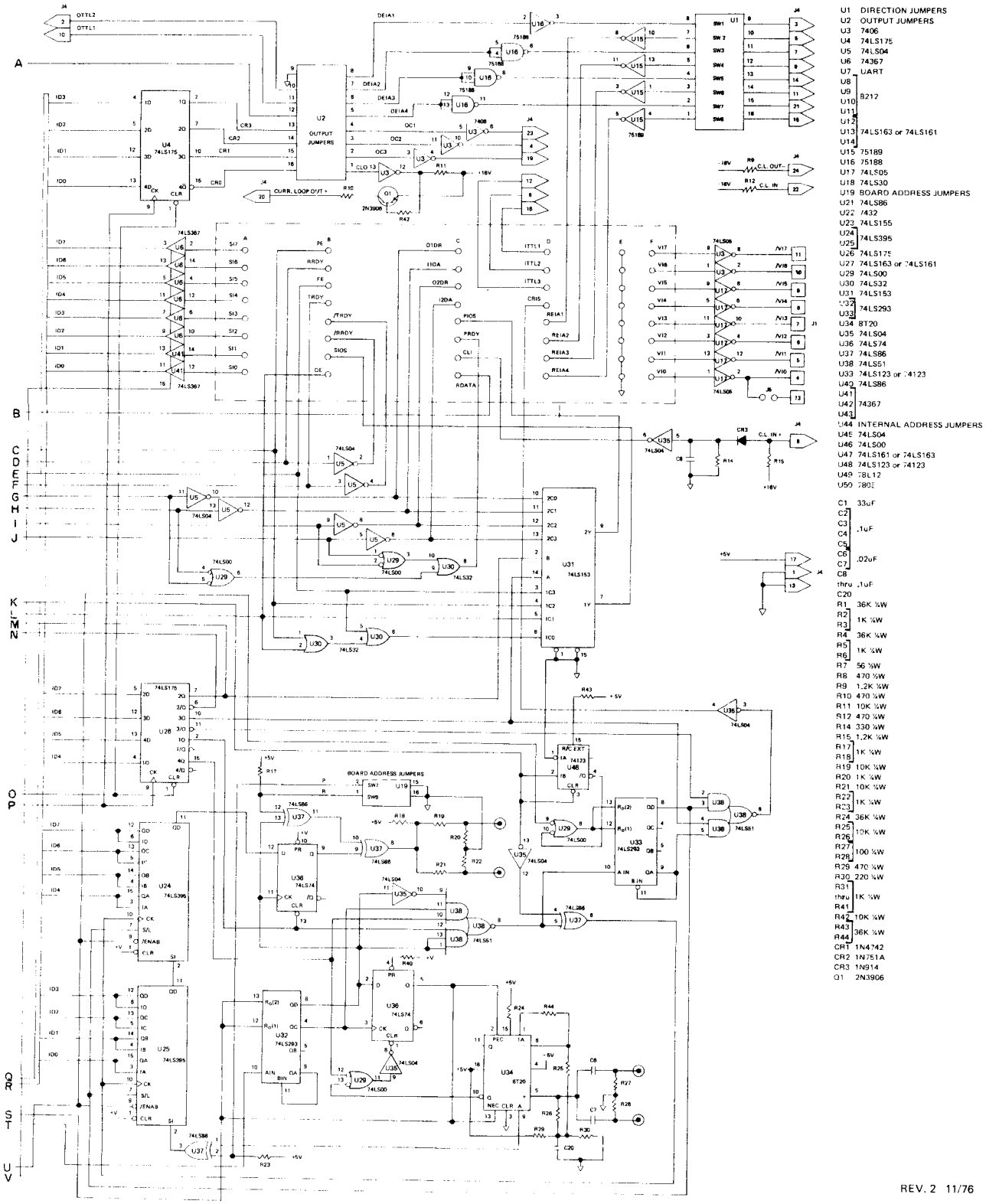


Fig. C.22 Imsai 8080 MIO, multiple parallel and serial I/O card *(sheet 2 of 3)*

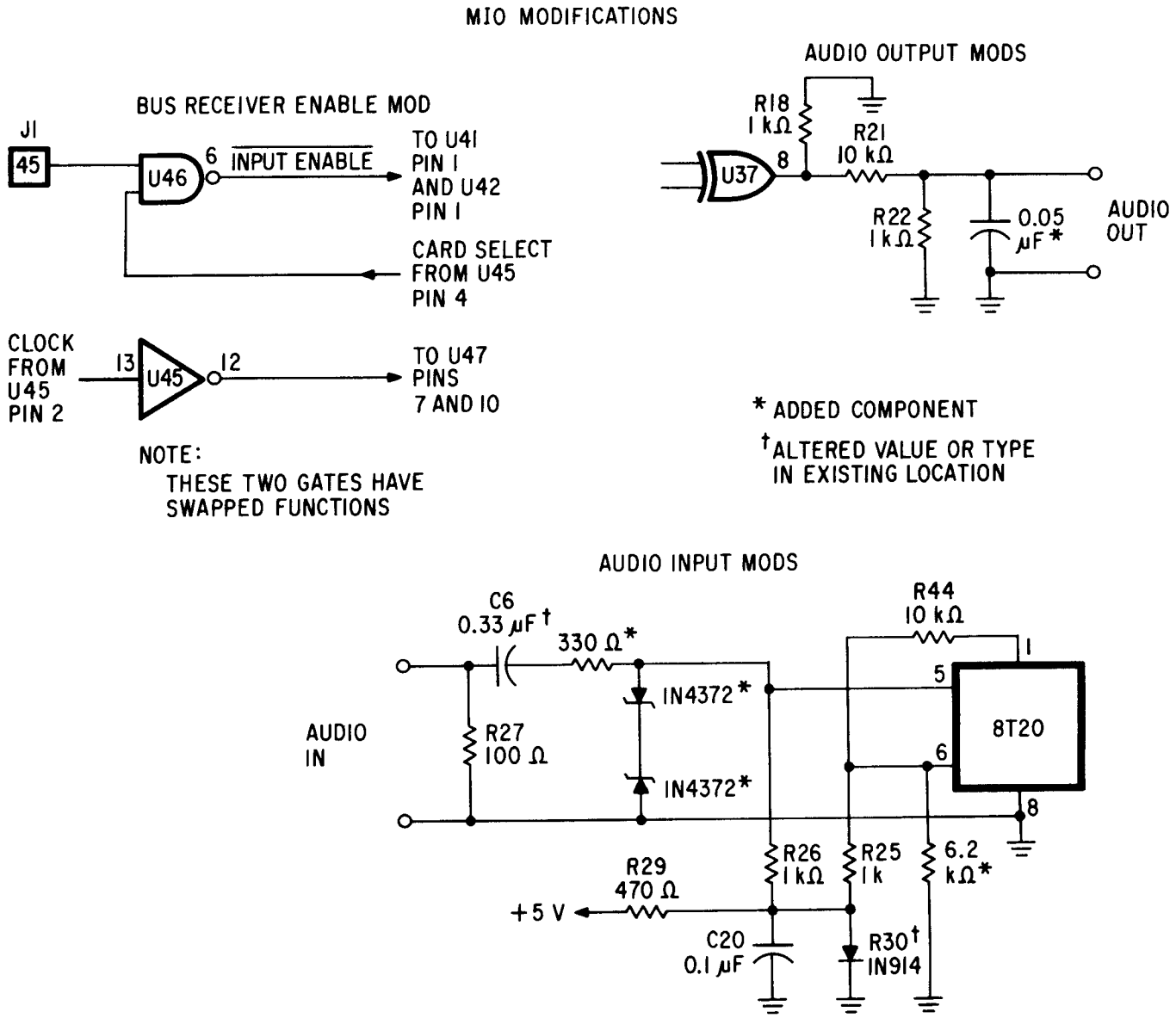


Fig. C.22 Imsai 8080 MIO, multiple parallel and serial I/O card (*sheet 3 of 3*)

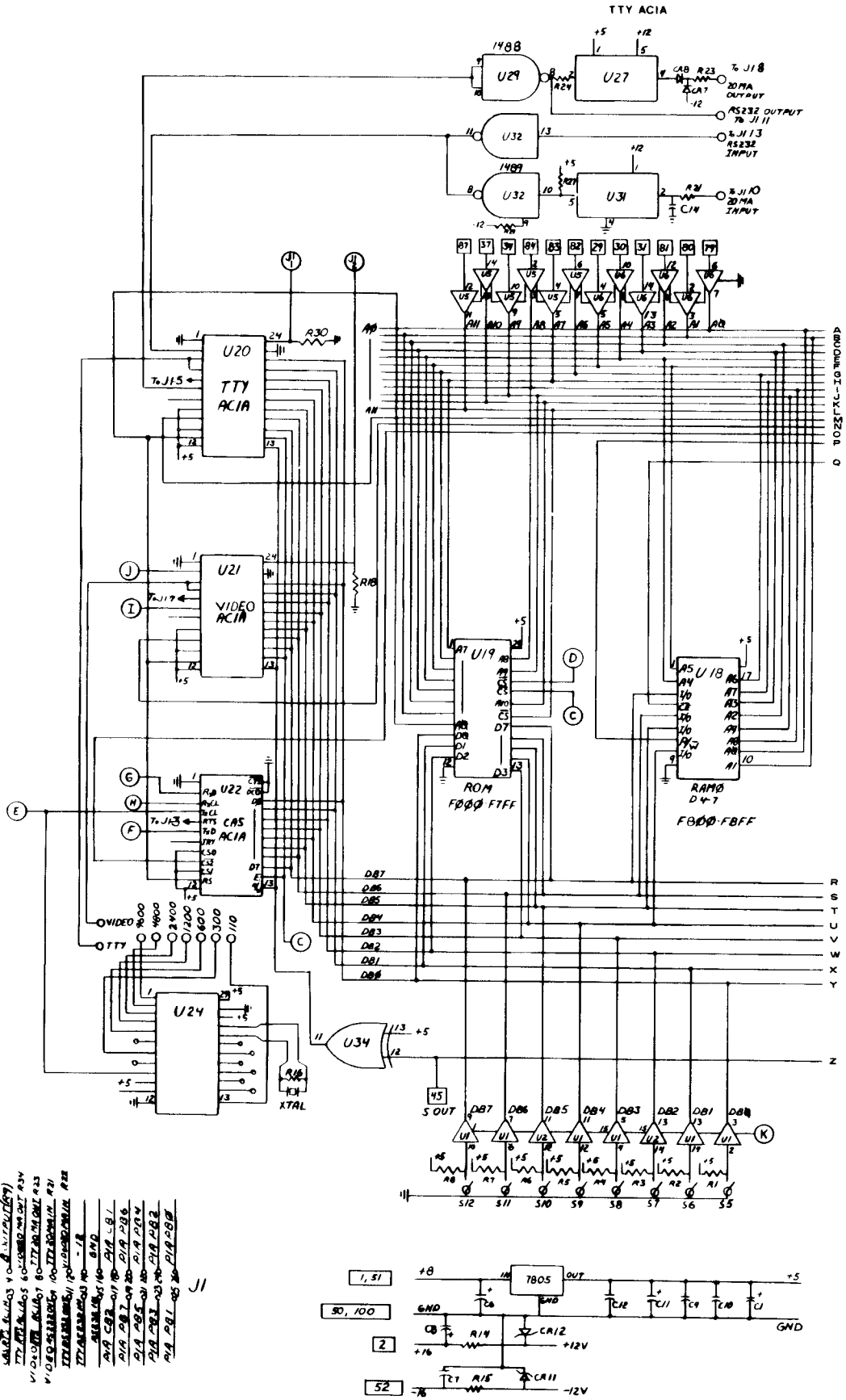
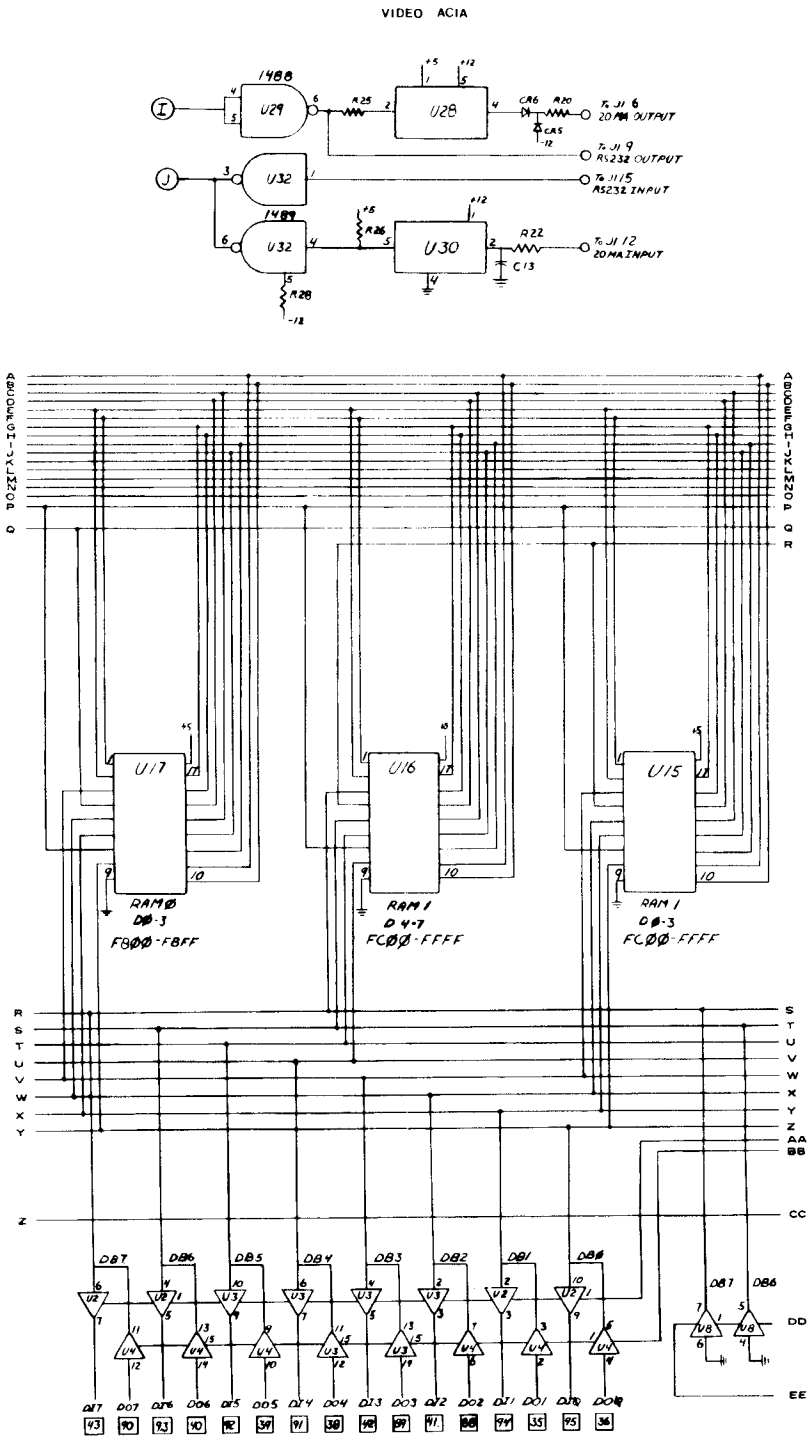


Fig. C.23 Xitan/TDL SMB, combination I/O and boot strap memory card (*sheet 1 of 3*)



CONNECTION POINT	LOCATION
A	B8
B	C8
C	D3, E9, F1, G10
D	D3, E8
E	A9, E0
F	A9, E0
G	A9, E0
H	B11, E0
I	A4, D0
J	B4, D0
K	C7, D9, G3

Fig. C.23 Xitan/TDL SMB, combination I/O and bootstrap memory card (*sheet 2 of 3*)

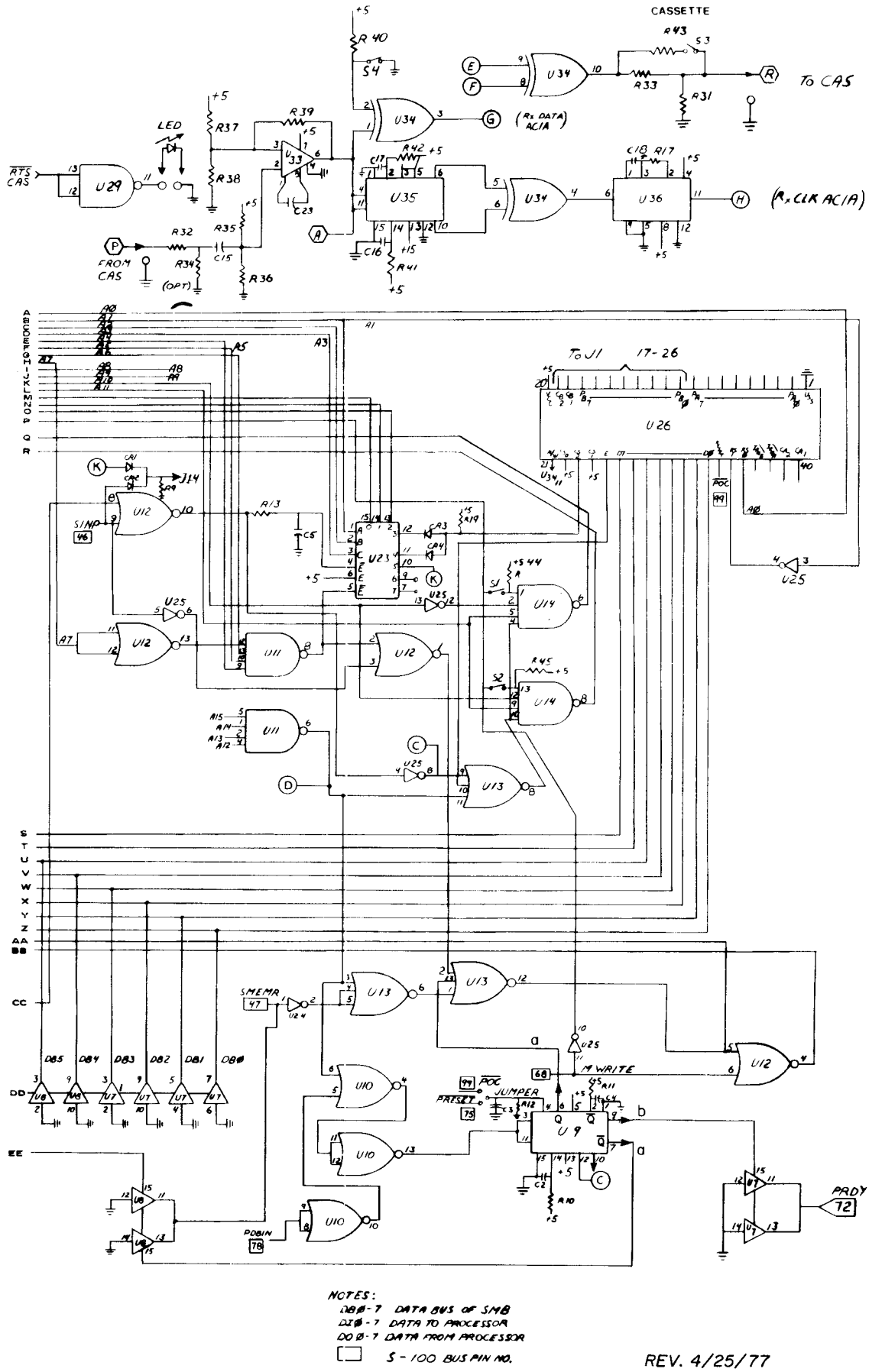


Fig. C.23 Xitan/TDL SMB, combination I/O and bootstrap memory card (*sheet 3 of 3*)

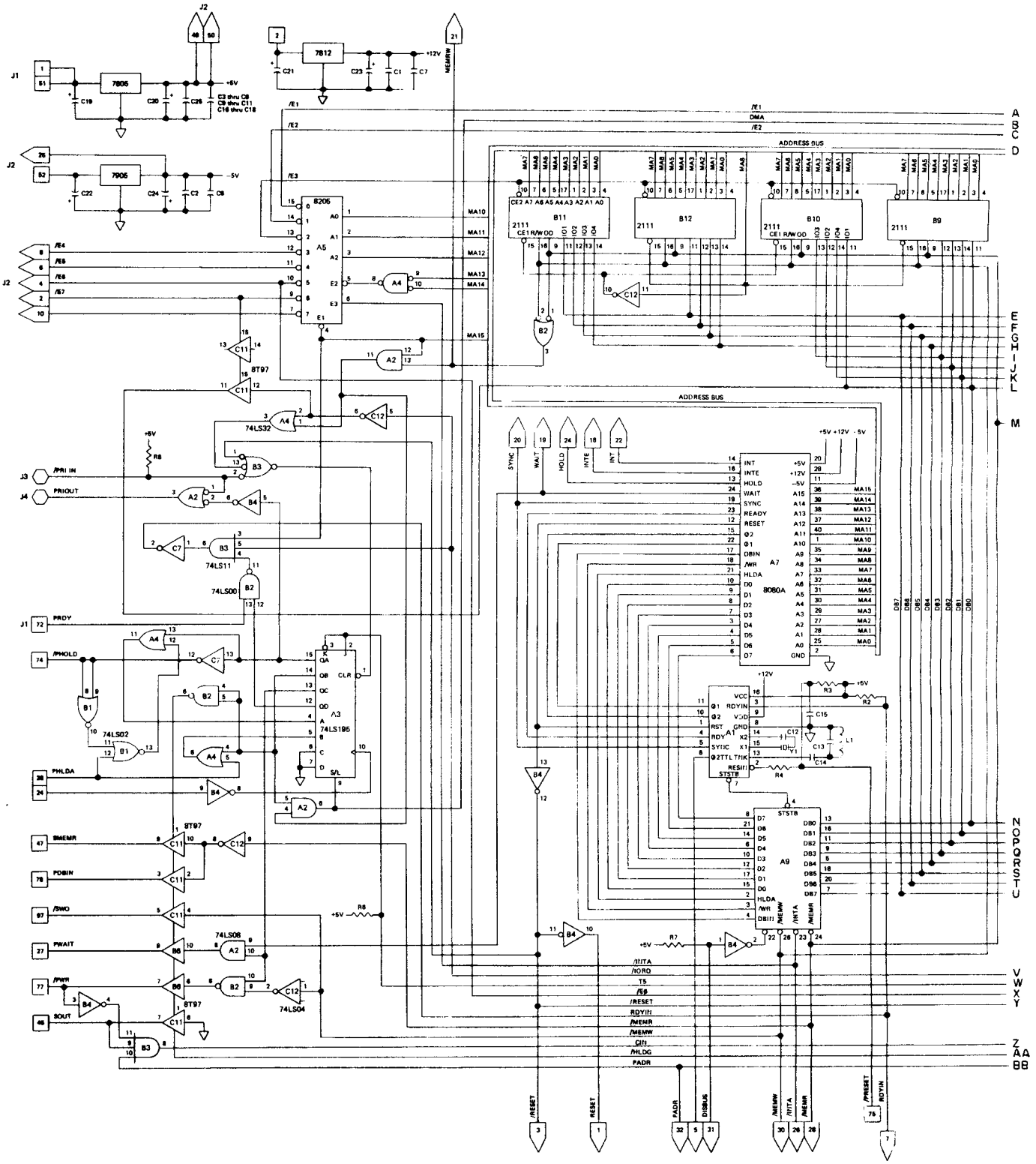


Fig. C.24 Imsai 8080 IFM, disk controller card set (*sheet 1 of 8*)

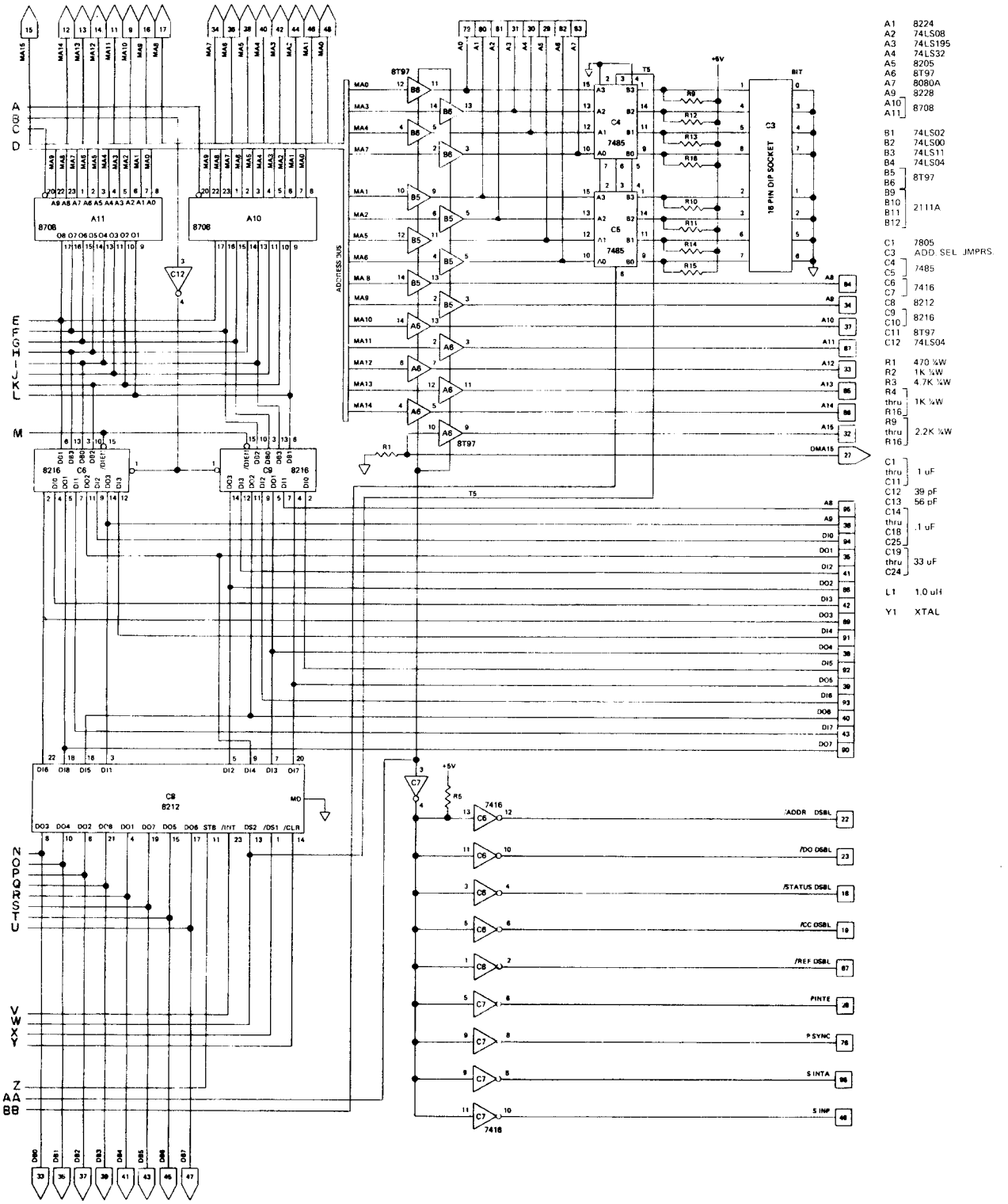


Fig. C.24 Imsai 8080 IFM, disk controller card set *(sheet 2 of 8)*

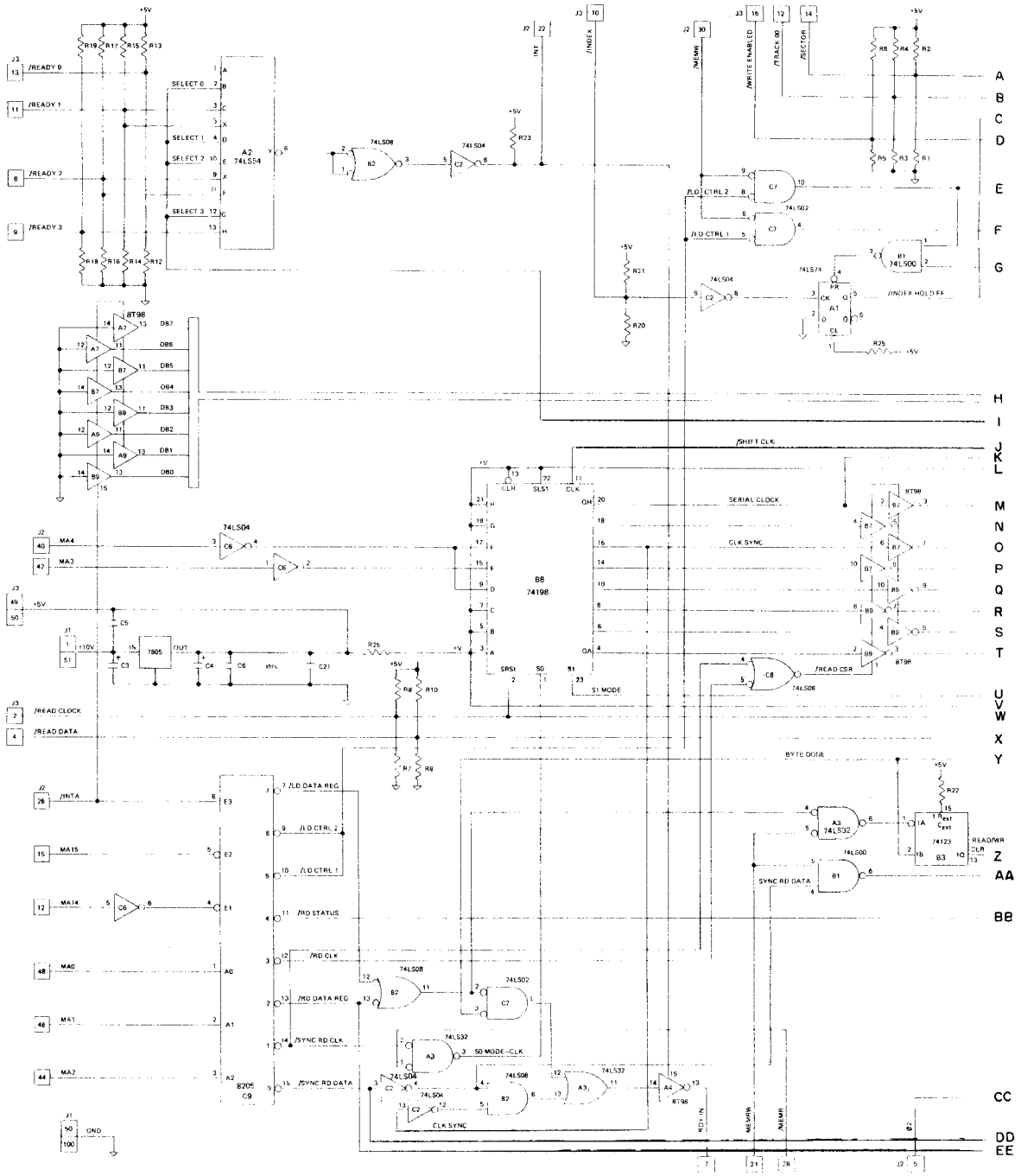


Fig. C.24 Imsai 8080 IFM, disk controller card set (*sheet 3 of 8*)

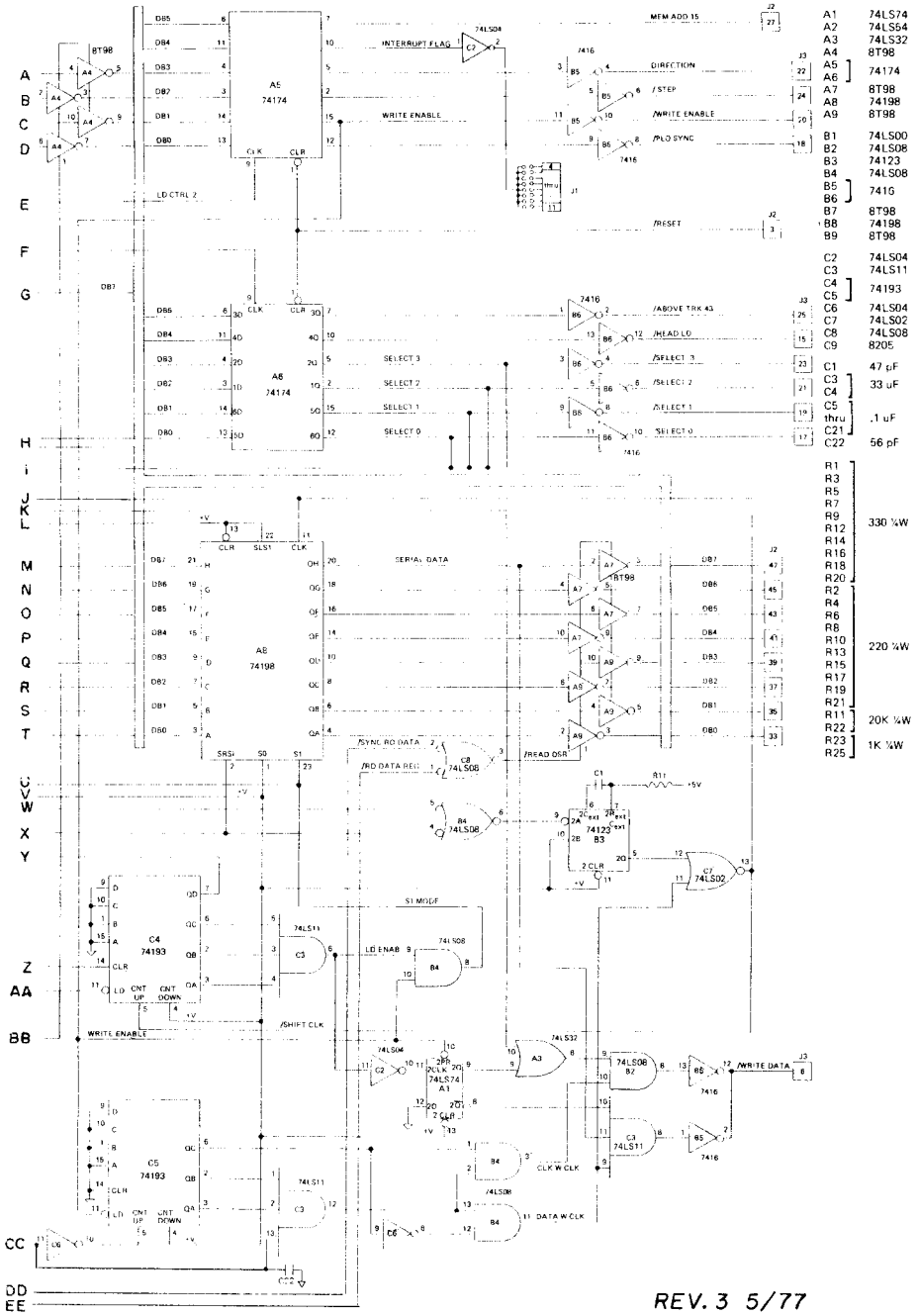


Fig. C.24 Imsai 8080 IFM, disk controller card set (*sheet 4 of 8*)

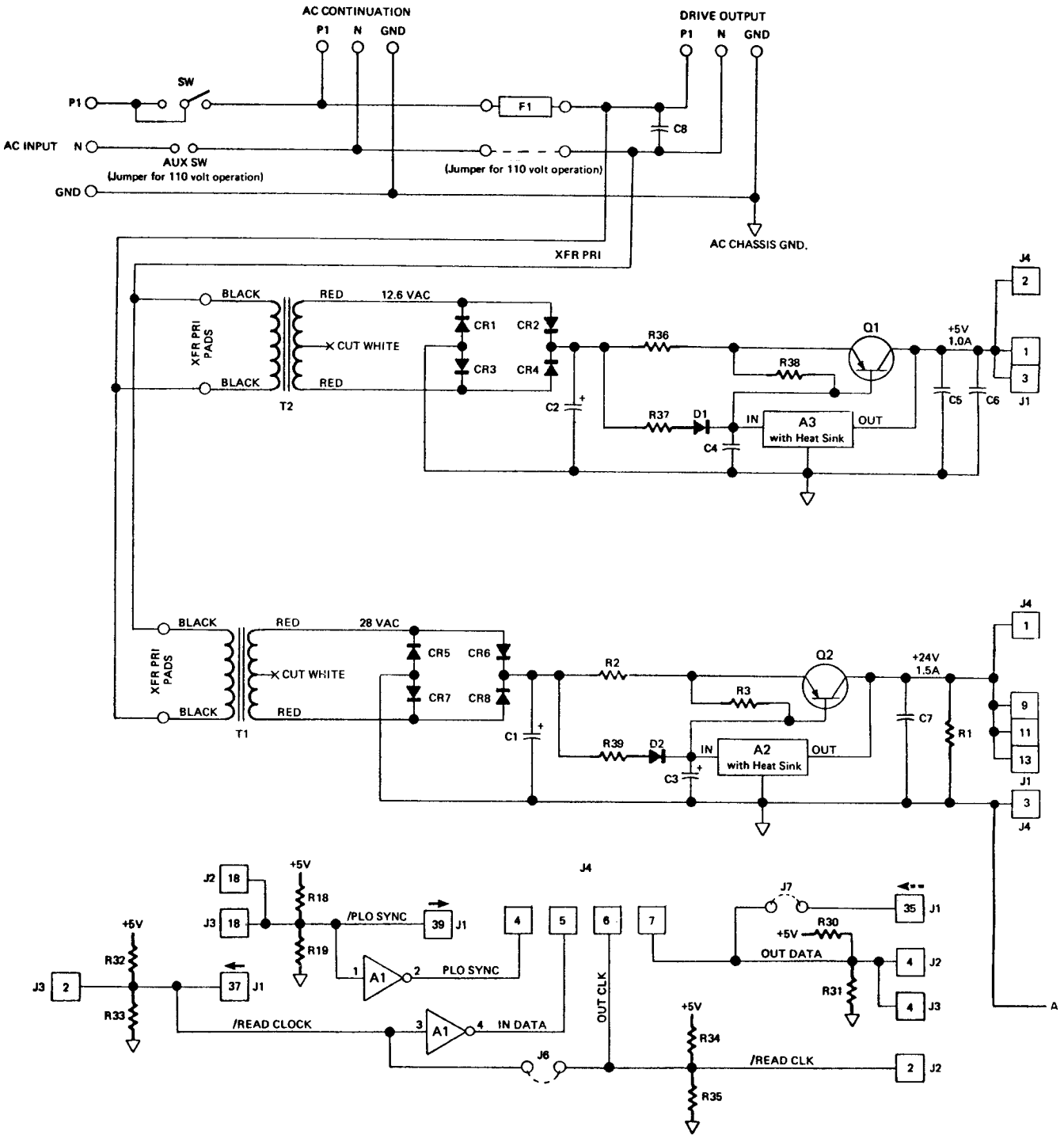
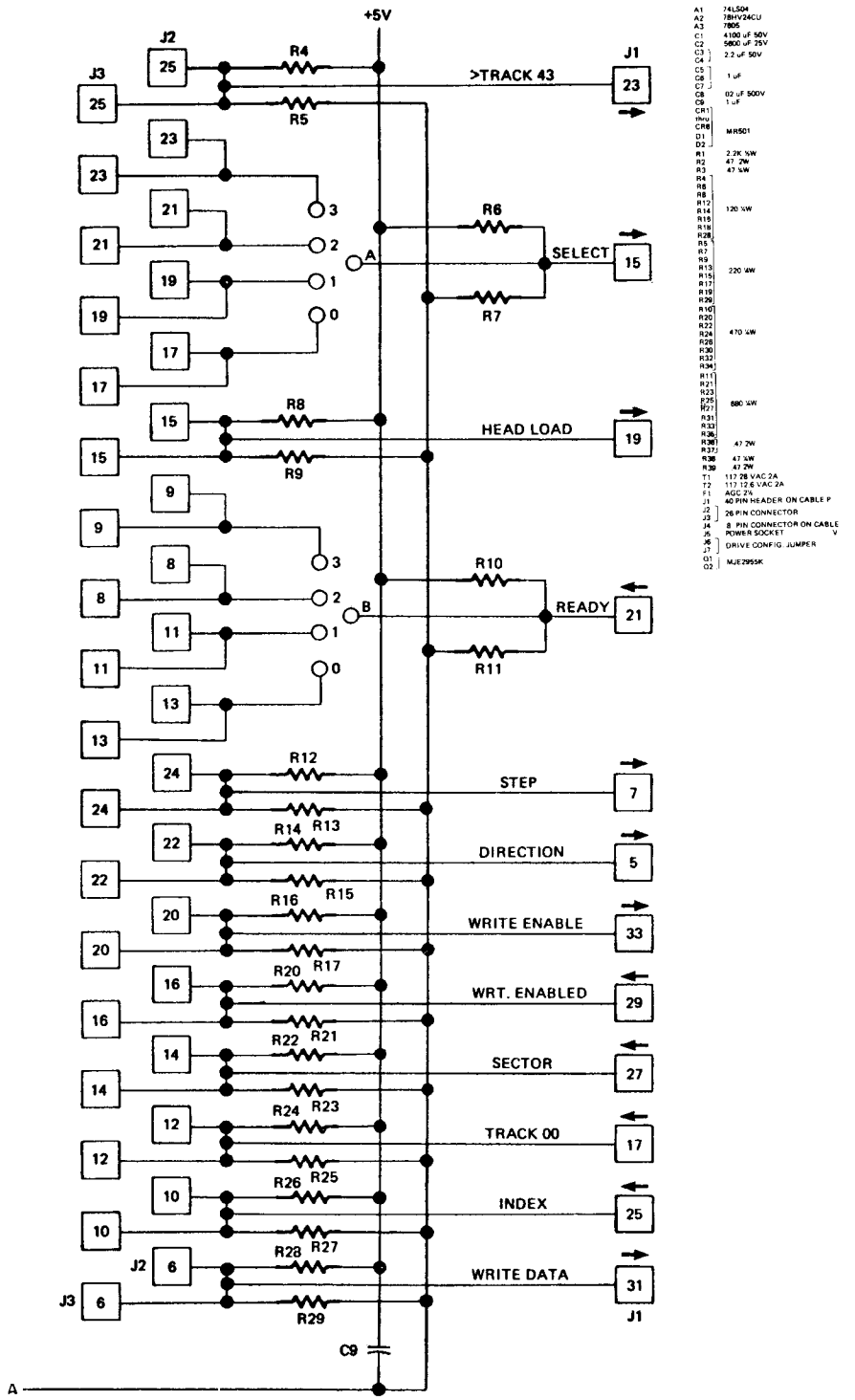


Fig. C.24 Imsai 8080 IFM, disk controller card set (*sheet 5 of 8*)



J1 – ALL EVEN NUMBERED PINS GROUNDED
J2 – PINS 1,3,5,7,26 GROUNDED
J3 – PINS 1,3,5,7,26 GROUNDED

REV. 3 12/76

Fig. C.24 Imsai 8080 IFM, disk controller card set (*sheet 6 of 8*)

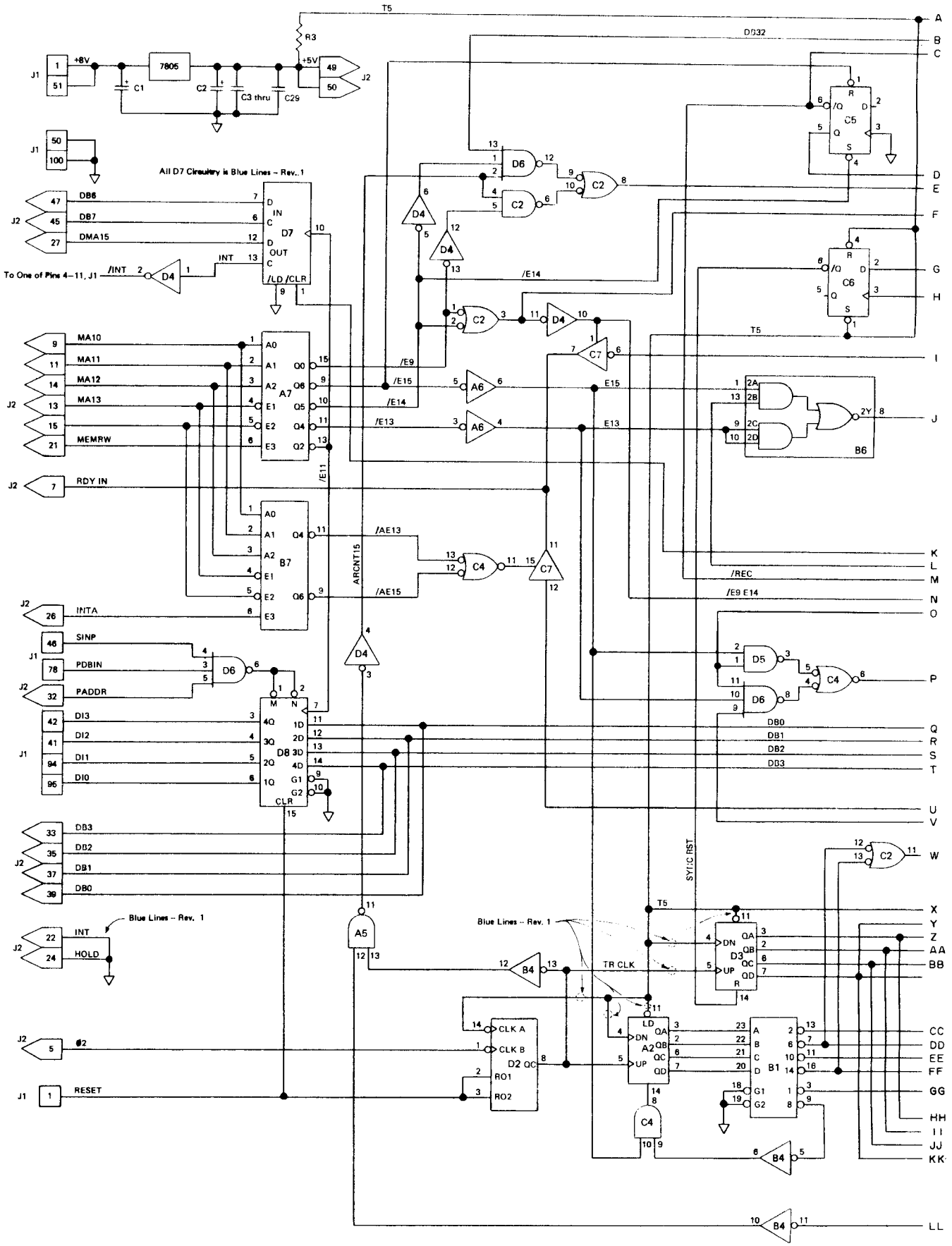


Fig. C.24 Imsai 8080 IFM, disk controller card set (*sheet 7 of 8*)

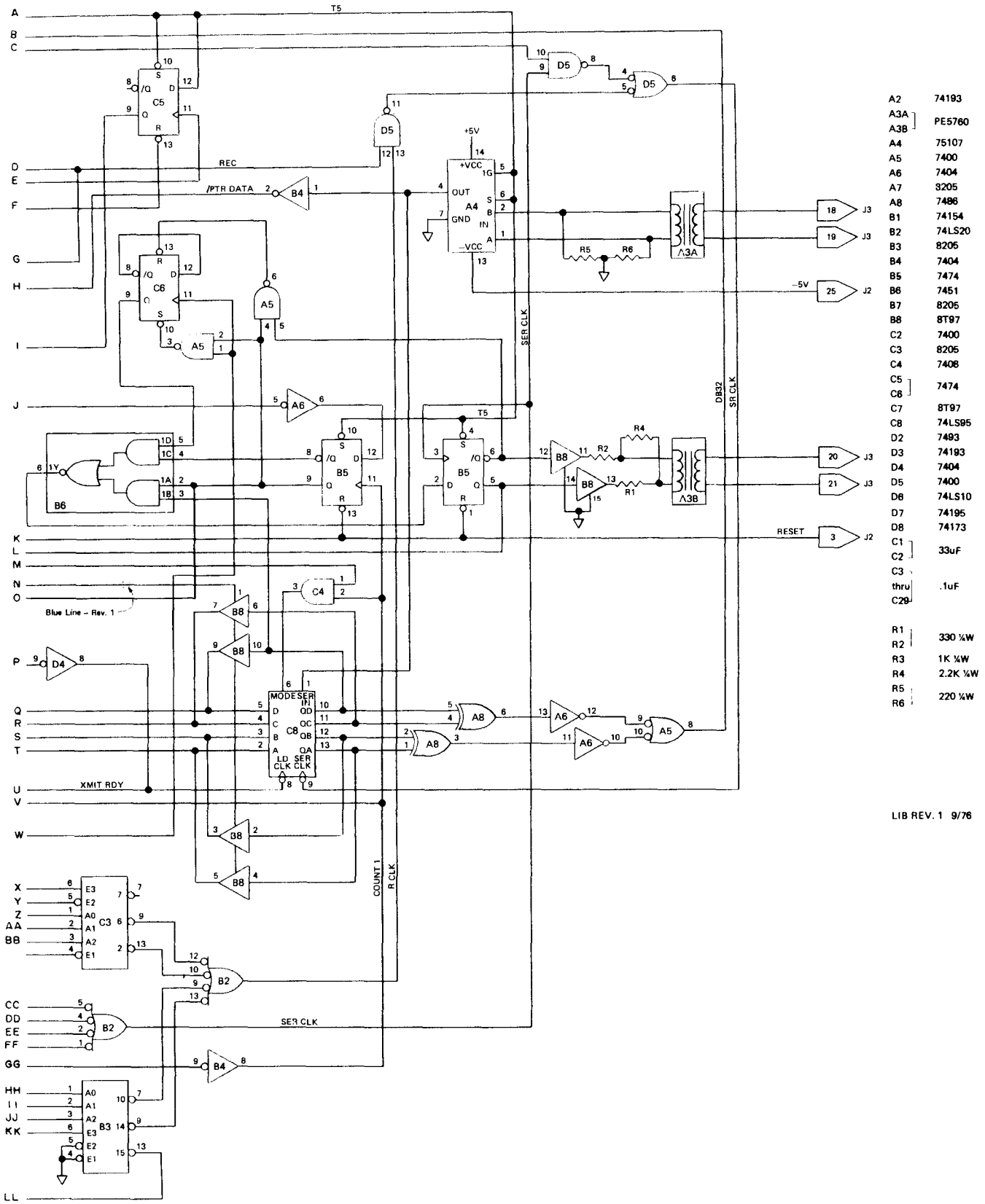


Fig. C.24 Imsai 8080 IFM, disk controller card set (*sheet 8 of 8*)

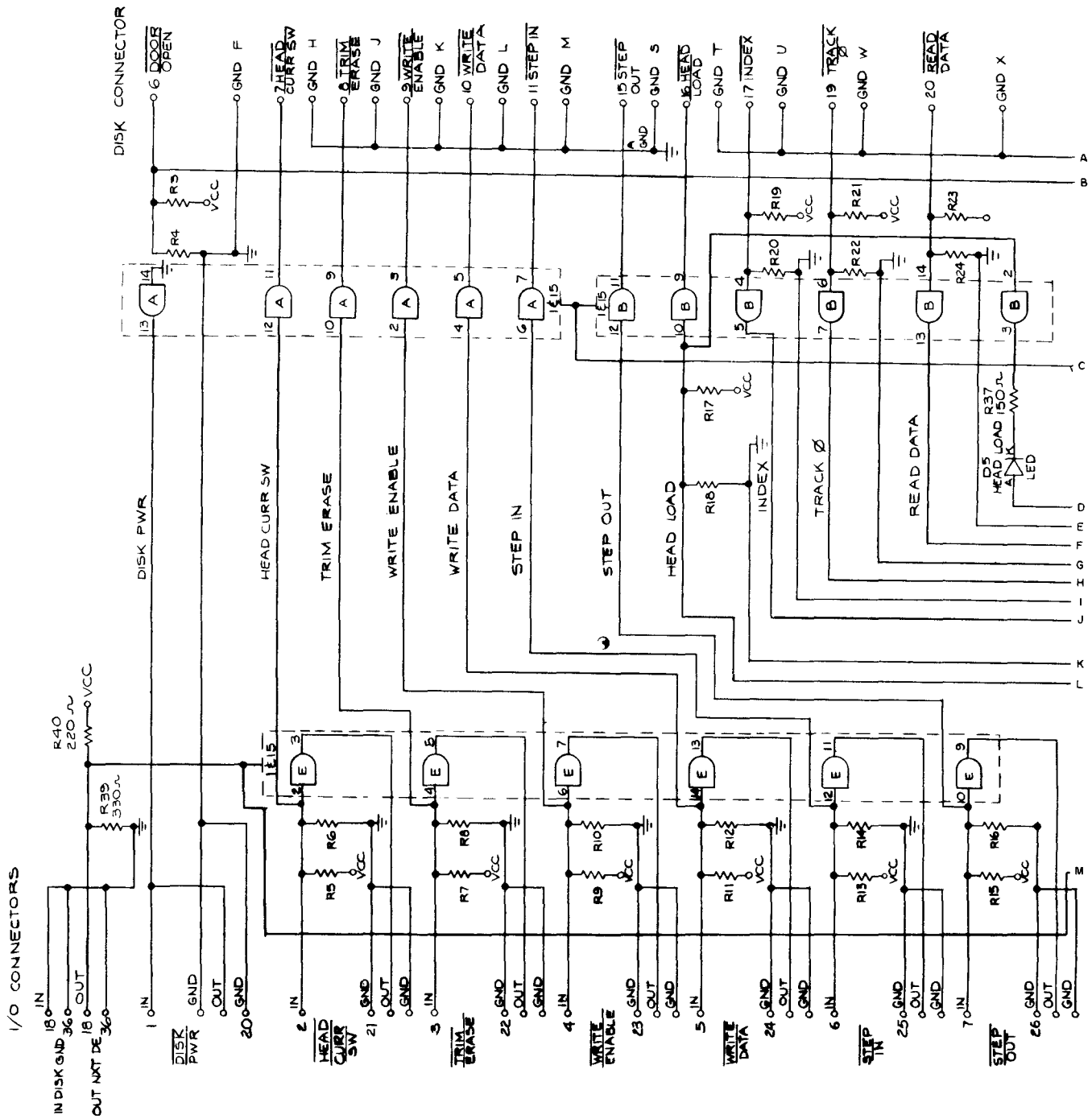


Fig. C.25 Pertec/MITS 88-DDC, disk controller card set (*sheet 1 of 12*)

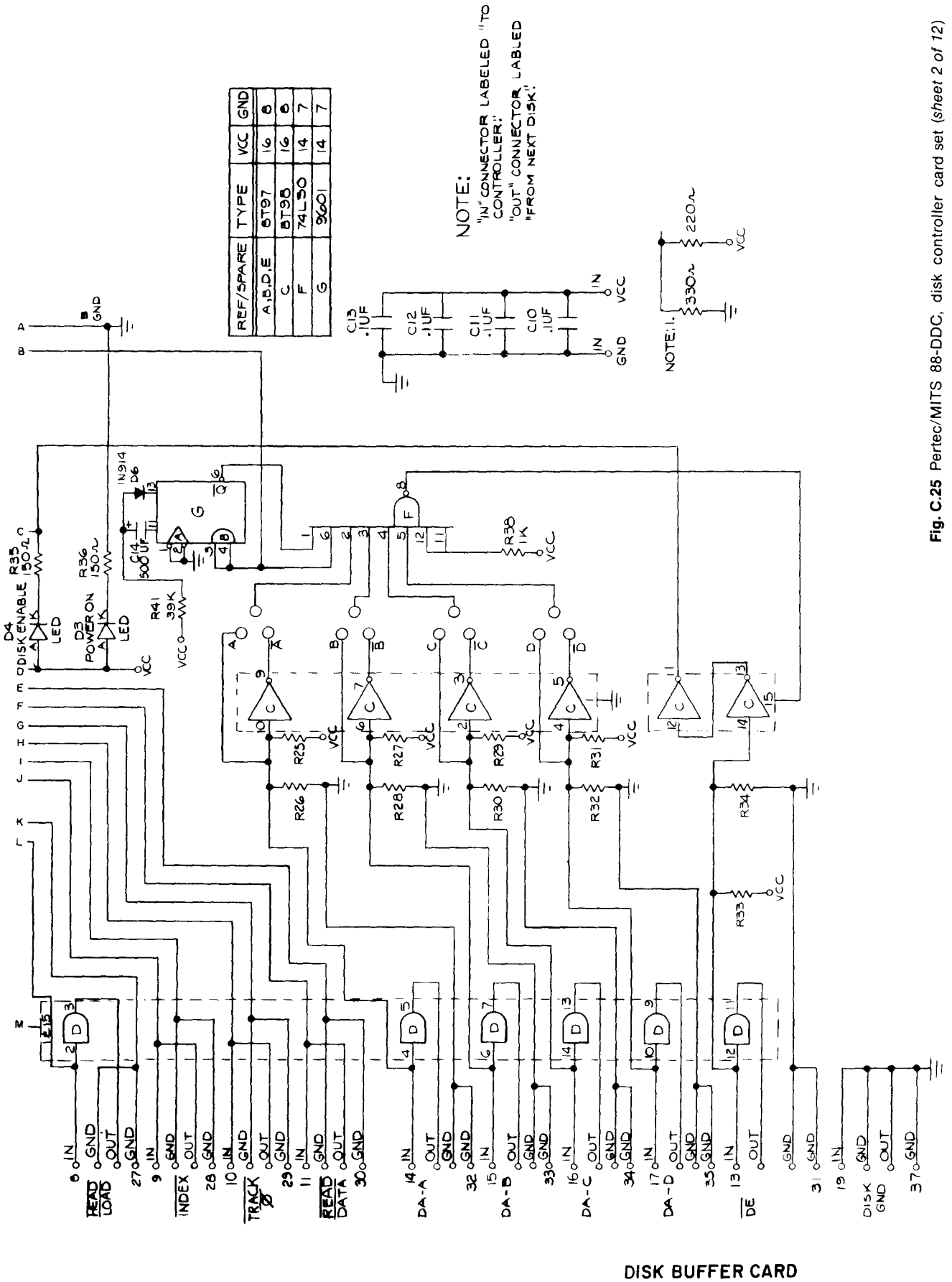


Fig. C.25 Pertec/MITS 88-DDC, disk controller card set (*sheet 2 of 12*)

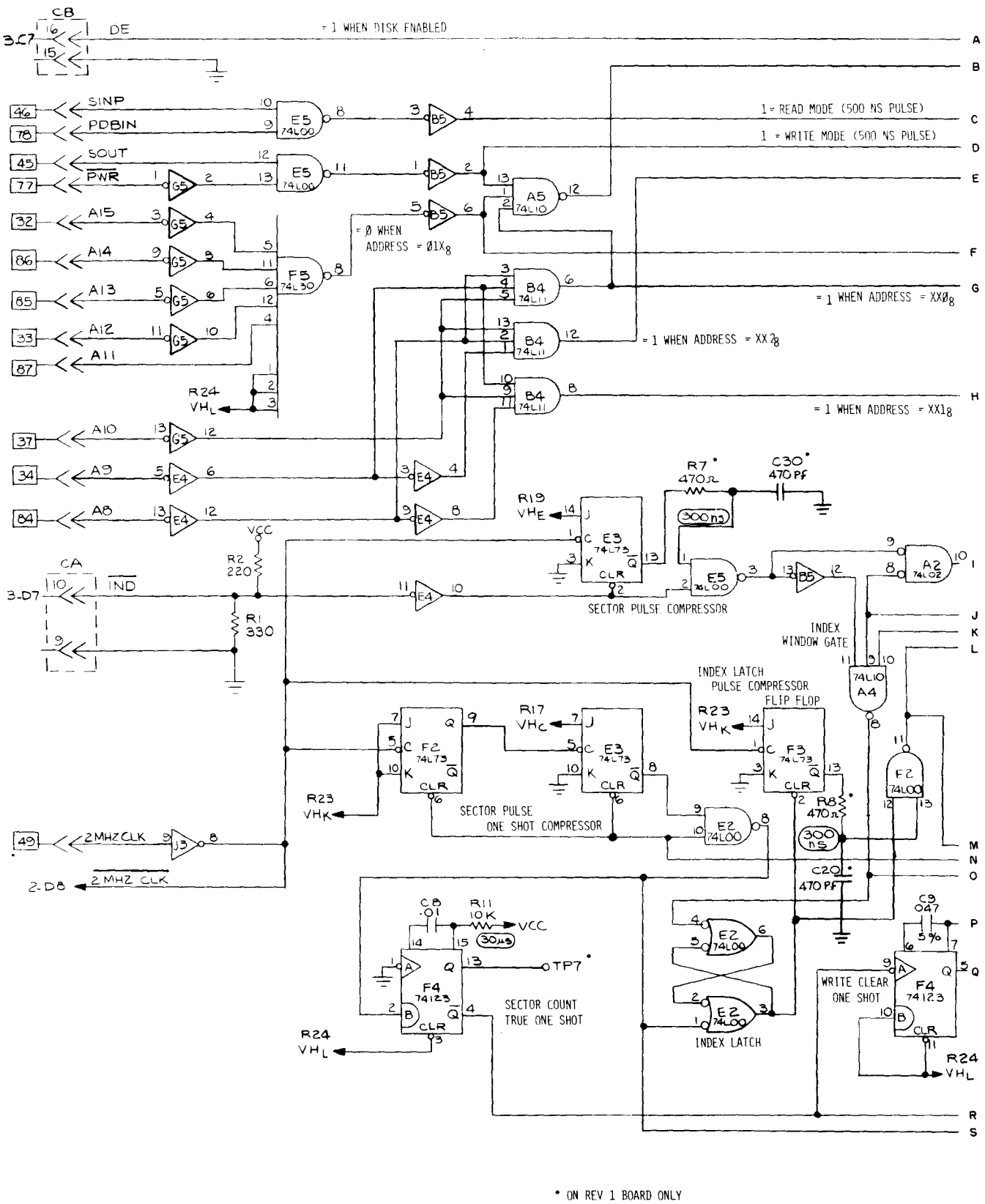


Fig. C.25 Pertec/MITS 88-DDC, disk controller card set (*sheet 3 of 12*)

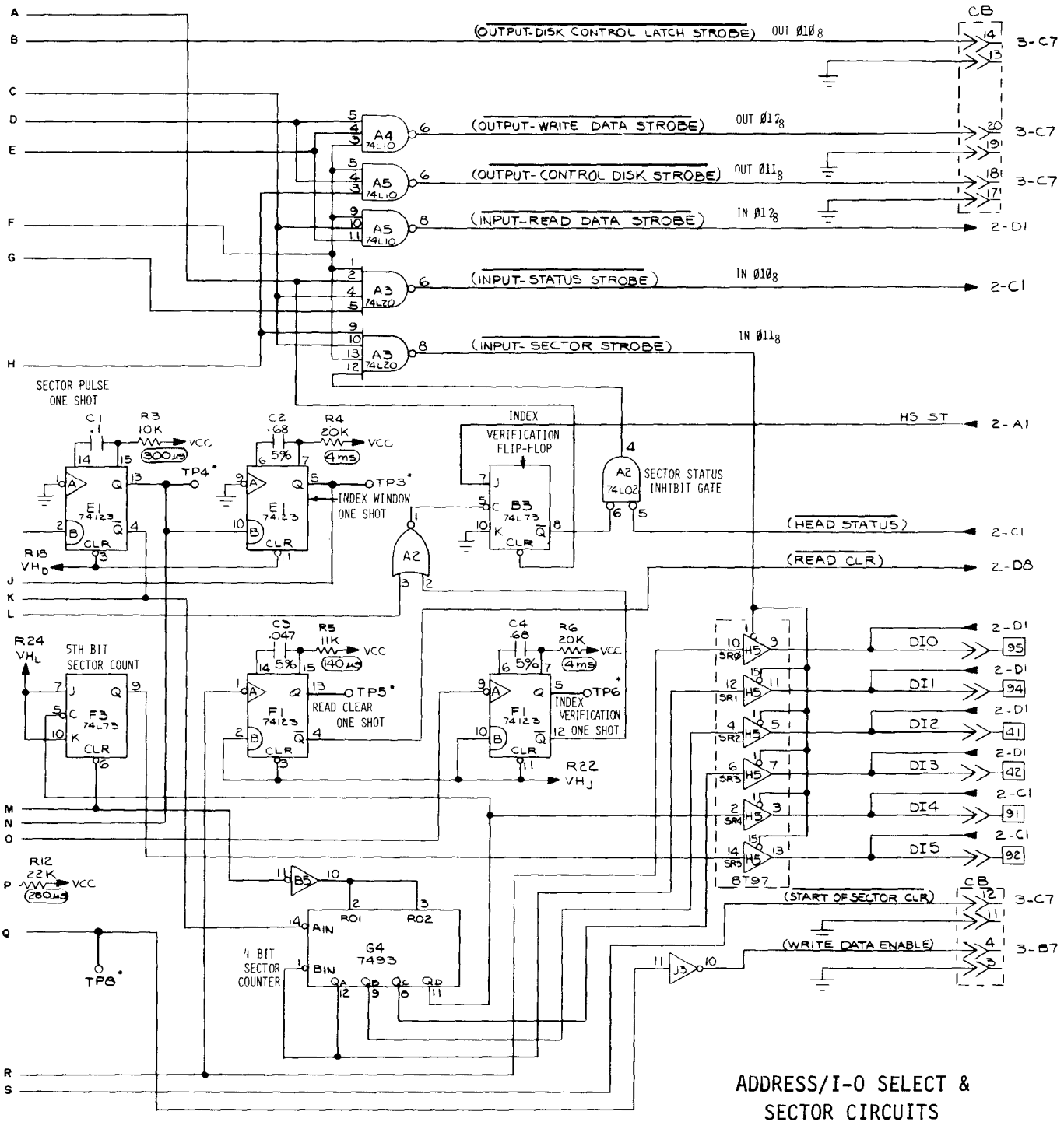


Fig. C.25 Pertec/MITS 88-DDC, disk controller card set (*sheet 4 of 12*)

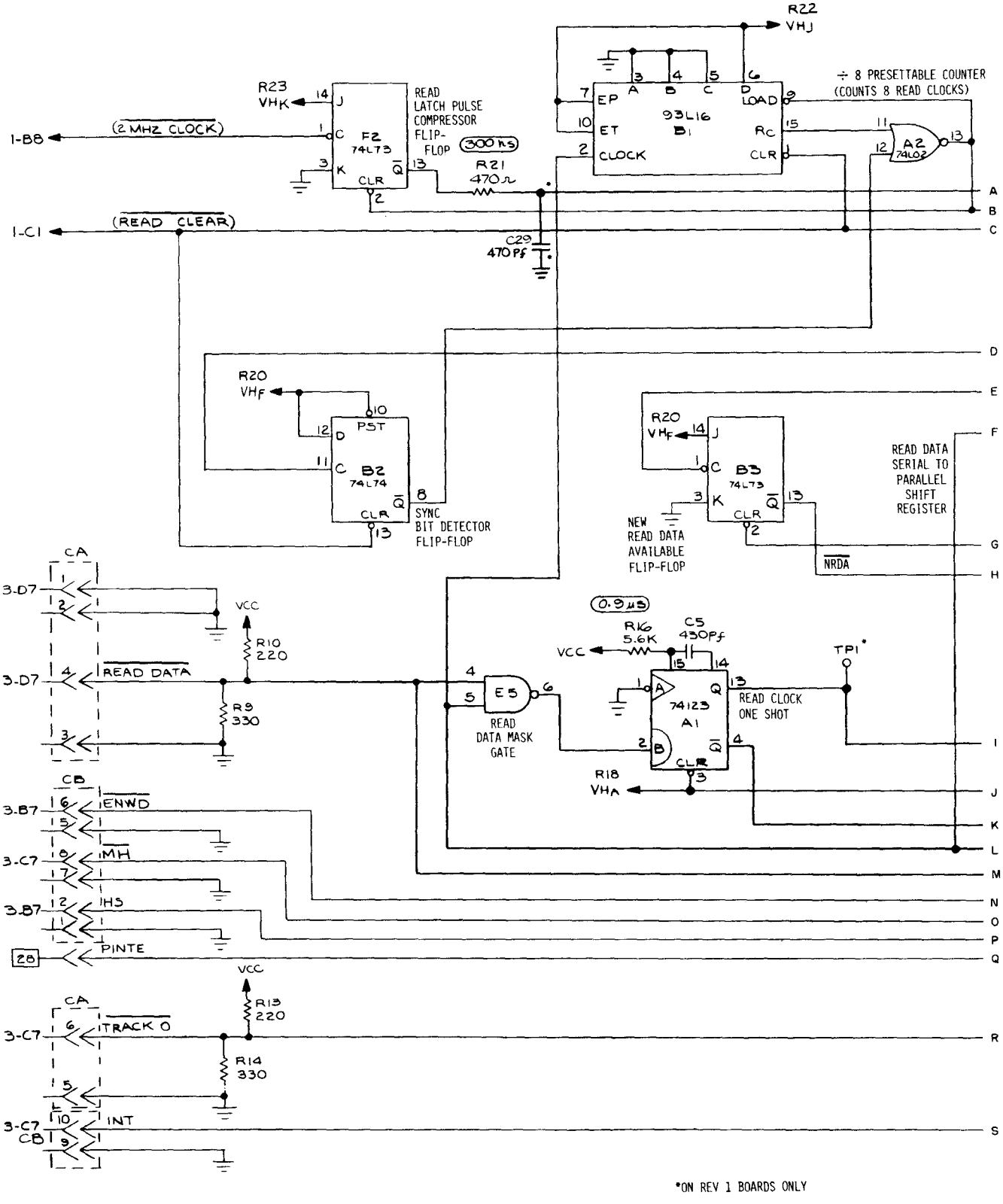


Fig. C.25 Pertec/MITS 88-DDC, disk controller card set (*sheet 5 of 12*)

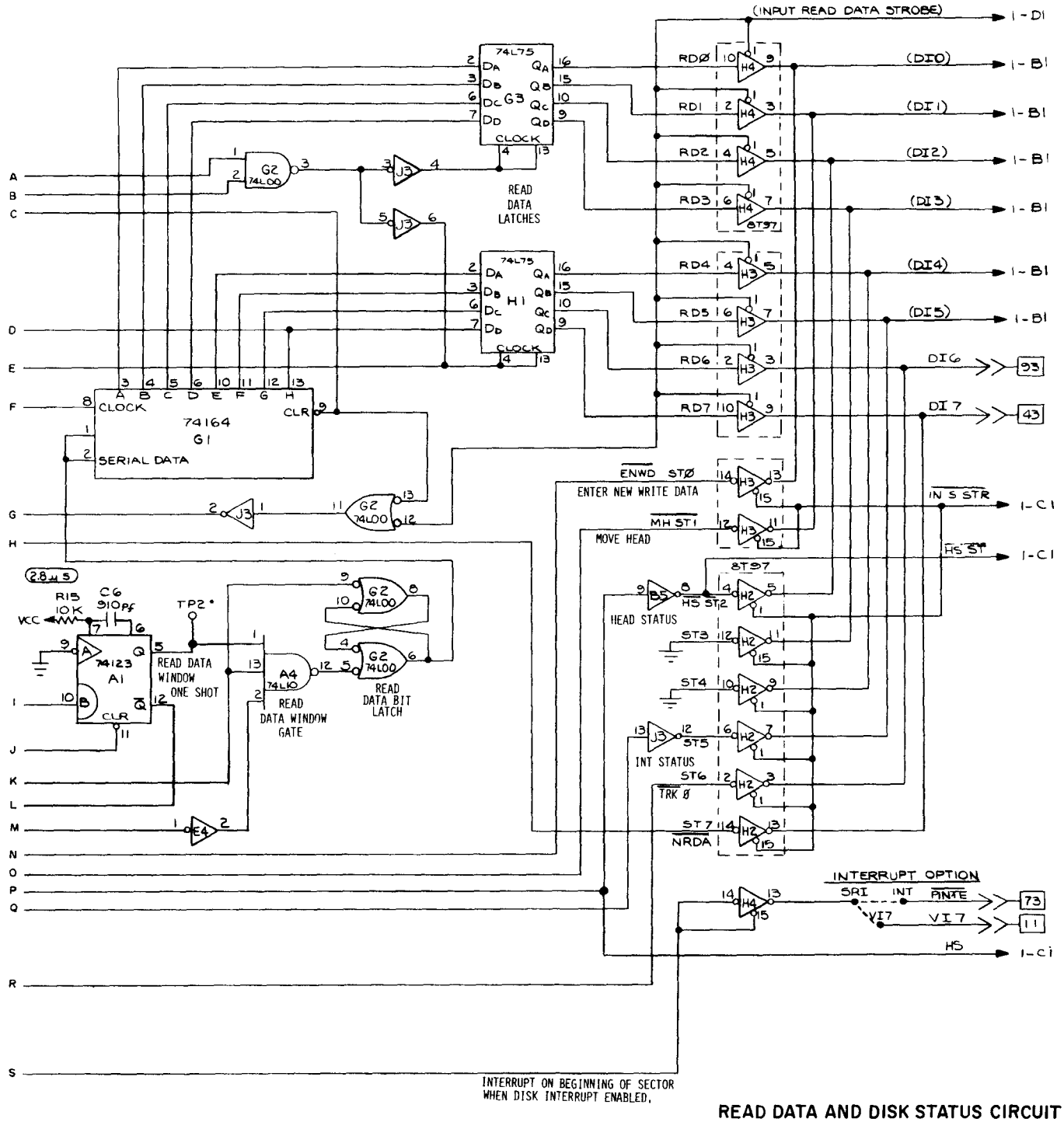


Fig. C.25 Pertec/MITS 88-DDC, disk controller card set (*sheet 6 of 12*)

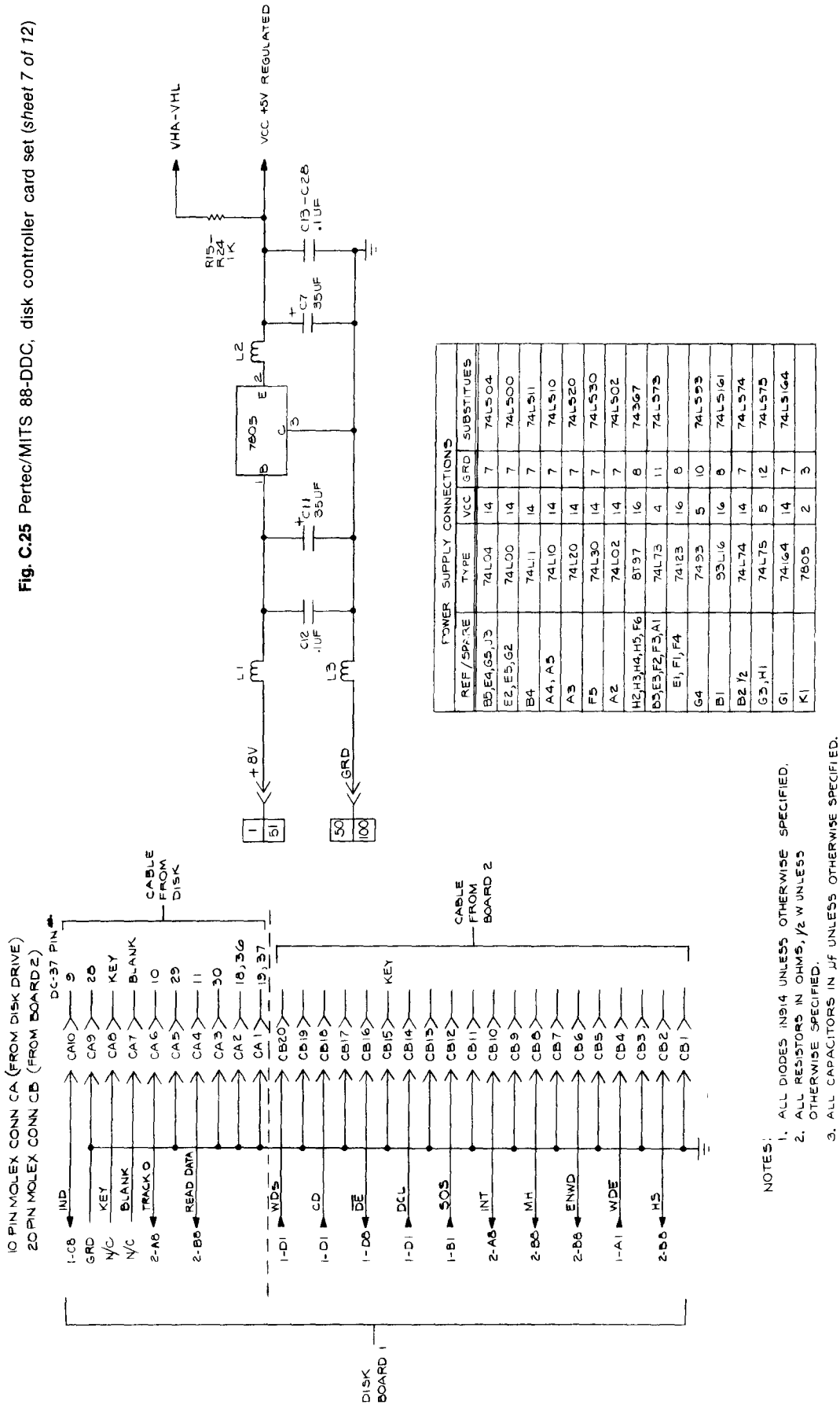


Fig. C.25 Pertec/MITS 88-DDC, disk controller card set (*sheet 7 of 12*)

POWER SUPPLY CONNECTIONS				
REF/SPARE	TYPE	VCC	GRD	SUBSTITUES
B5,E4,G5,J3	74L04	14	7	74LS04
E2,E5,G2	74L00	14	7	74LS00
B4	74L11	14	7	74LS11
A4, A5	74L10	14	7	74LS10
A3	74L20	14	7	74LS20
F5	74L30	14	7	74LS30
A2	74L02	14	7	74LS02
H2,H3,H4,H5,F6	8T97	16	8	74367
B3,E3,F2,F3,A1	74L73	4	11	74LS73
E1, F1, F4	74123	16	8	
G4	7493	5	10	74LS93
B1	93L16	16	8	74LS161
B2 Y2	74L74	14	7	74LS74
G3,H1	74L75	5	12	74LS75
G1	74164	14	7	74LS164
K1	7805	2	3	

NOTES:
1. ALL DIODES IN914 UNLESS OTHERWISE SPECIFIED.
2. ALL RESISTORS IN OHMS, 1/2 W UNLESS OTHERWISE SPECIFIED.
3. ALL CAPACITORS IN μF UNLESS OTHERWISE SPECIFIED.
4. (2500 ns) NOMINAL TIME CONSTANT.

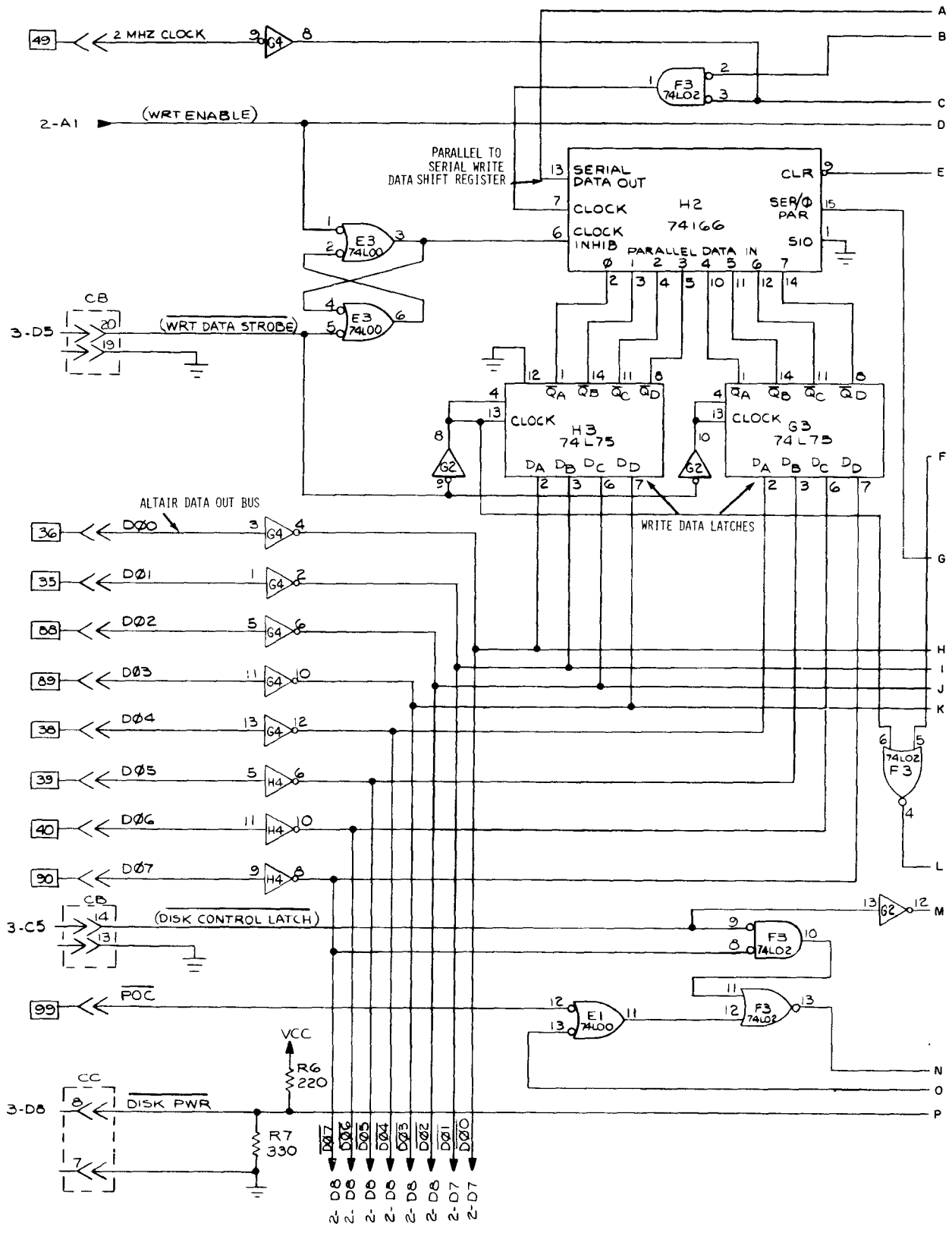


Fig. C.25 Pertec/MITS 88-DDC, disk controller card set (*sheet 8 of 12*)

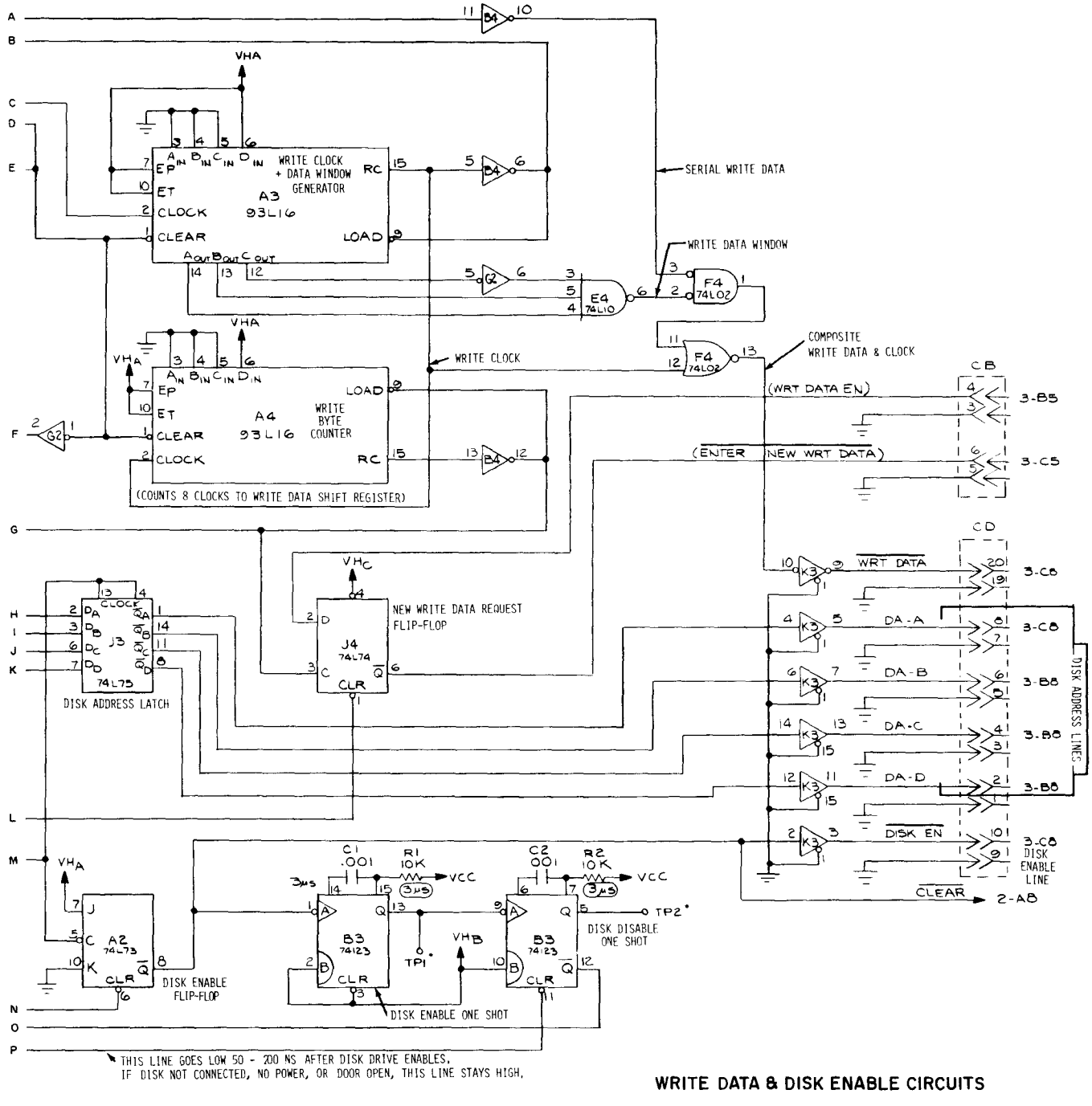


Fig. C.25 Pertec/MITS 88-DDC, disk controller card set (*sheet 9 of 12*)

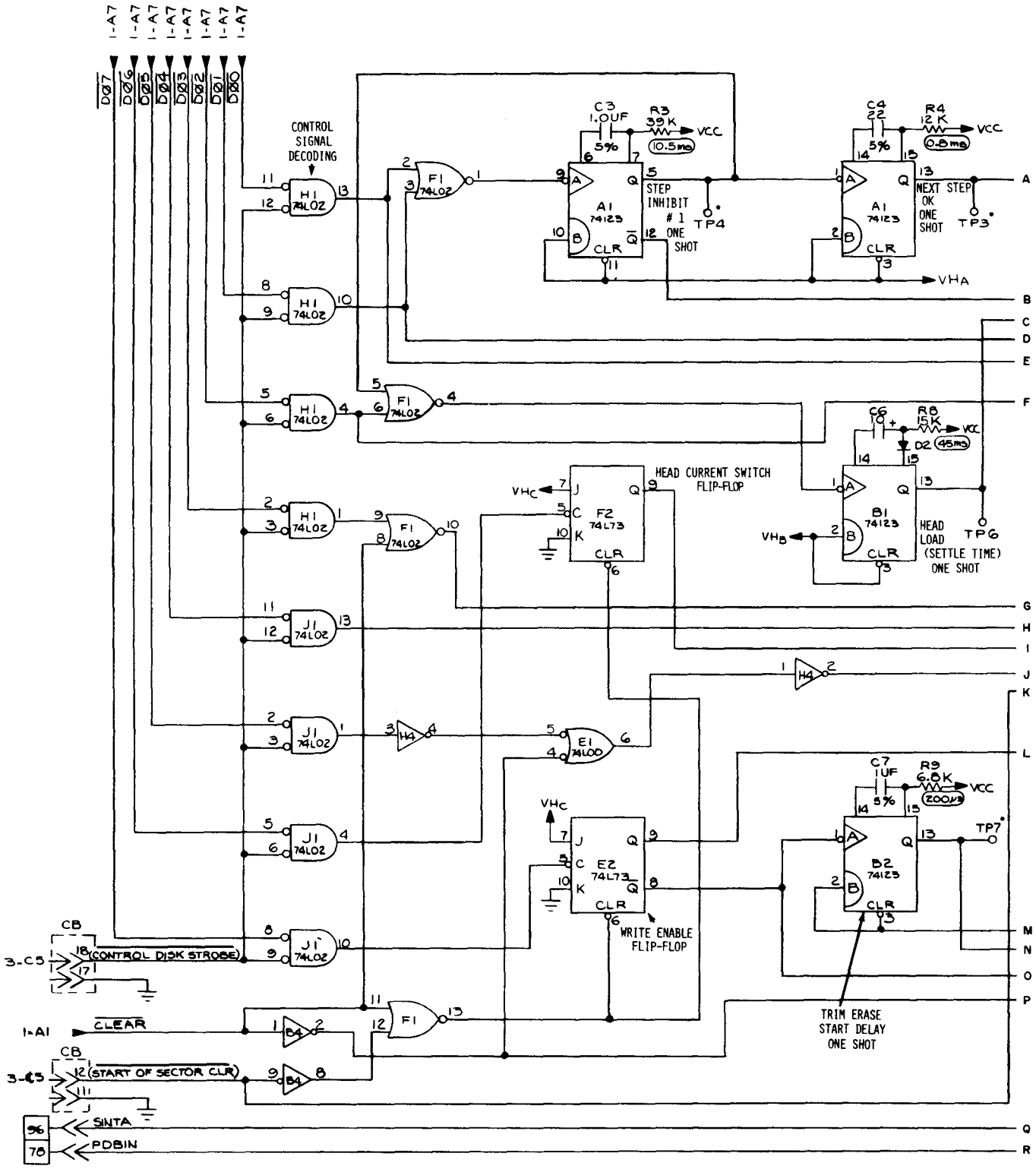


Fig. C.25 Pertec/MITS 88-DDC, disk controller card set (*sheet 10 of 12*)

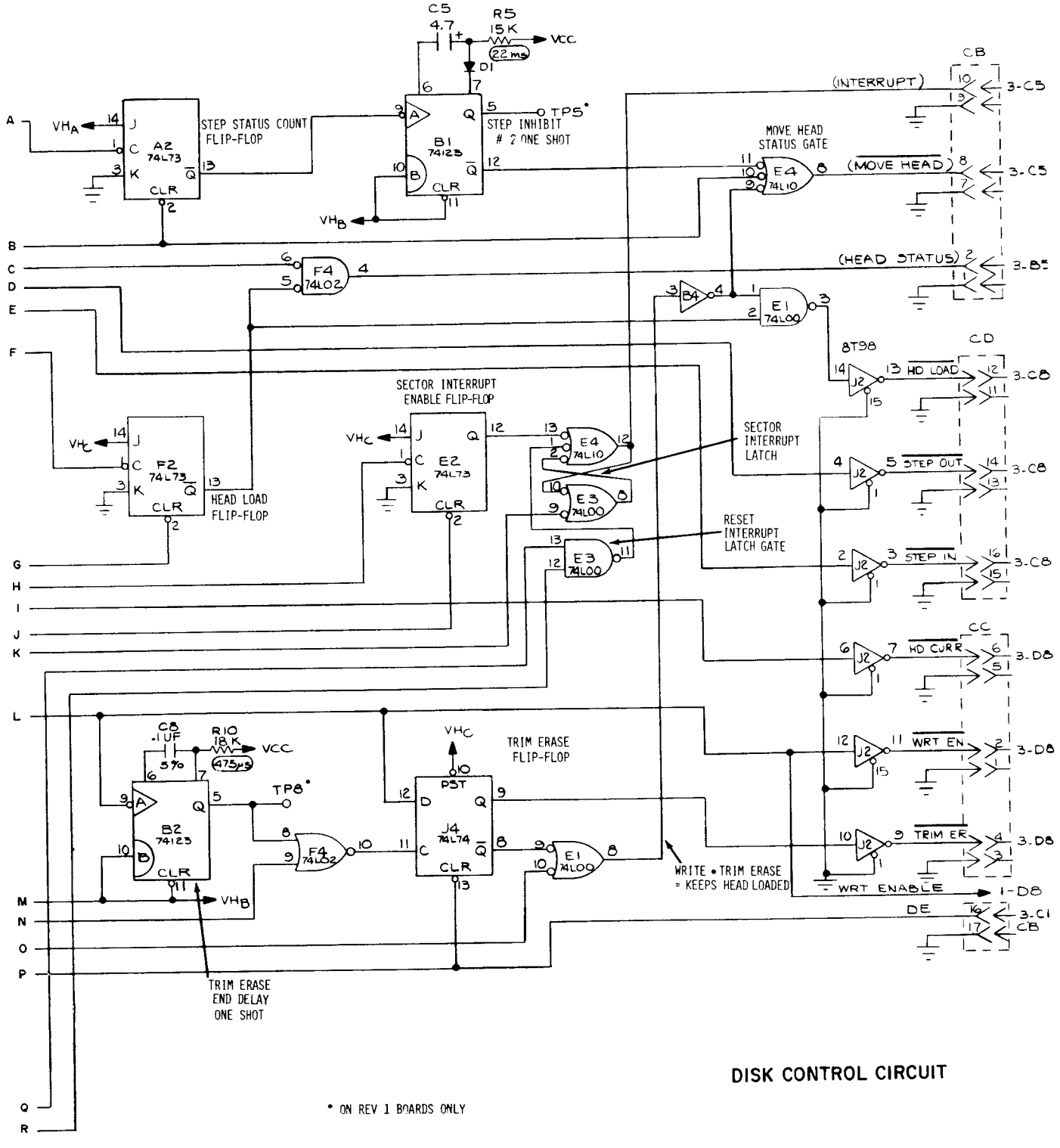


Fig. C.25 Pertec/MITS 88-DDC, disk controller card set (*sheet 11 of 12*)

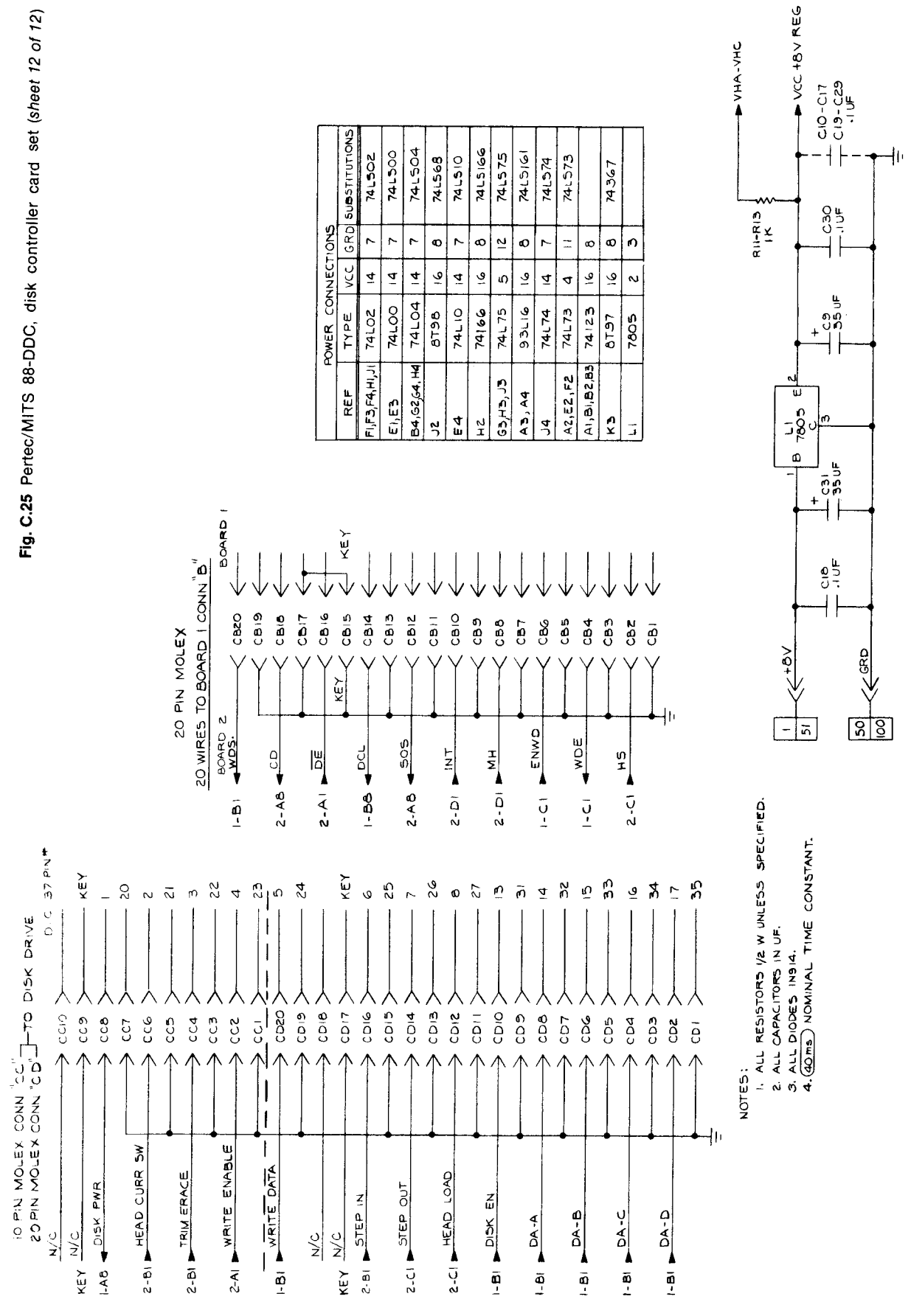
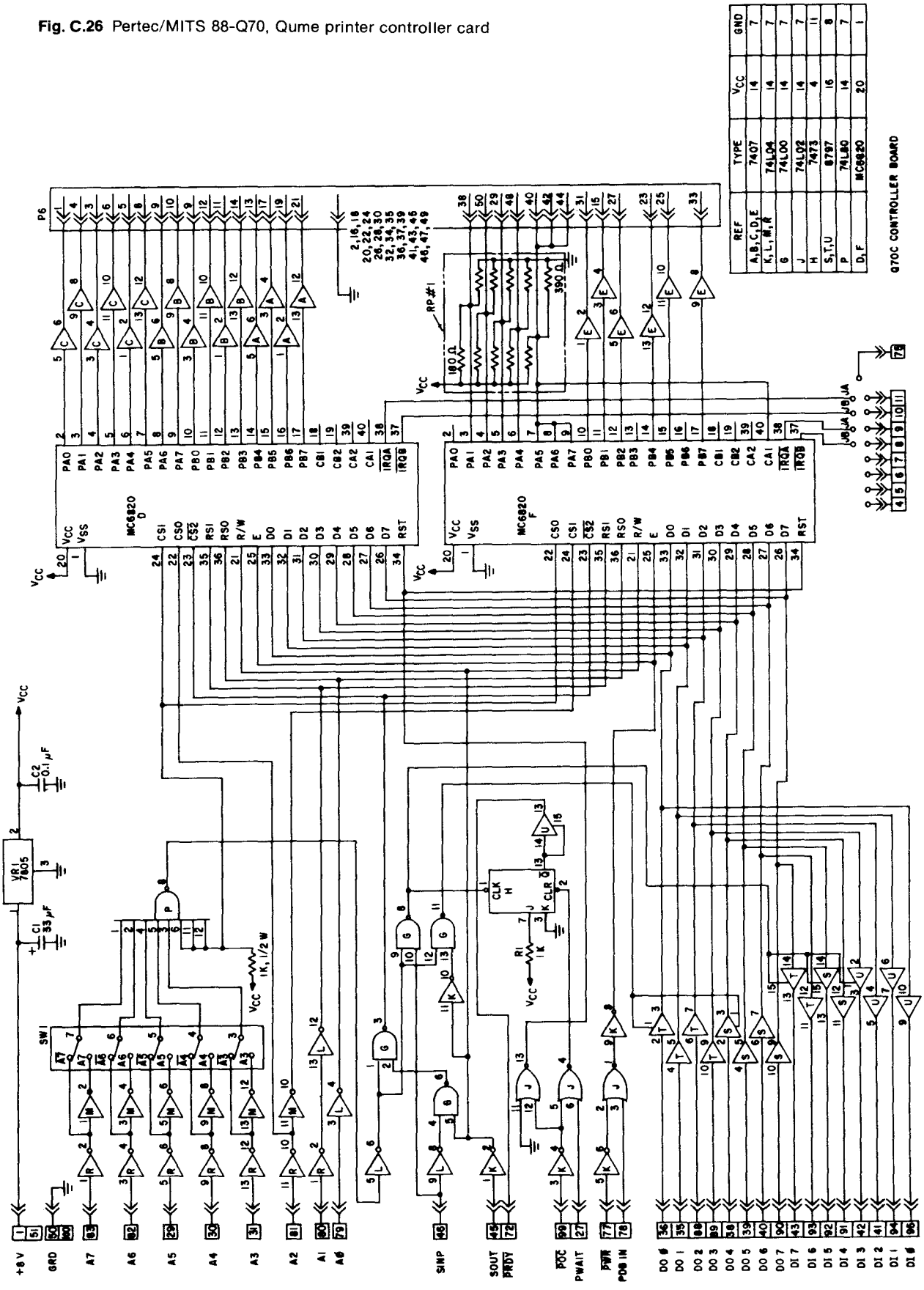


Fig. C.25 Pertec/MITS 88-DDC, disk controller card set (*sheet 12 of 12*)

Fig. C.26 Pertec/MITS 88-Q70, Qume printer controller card



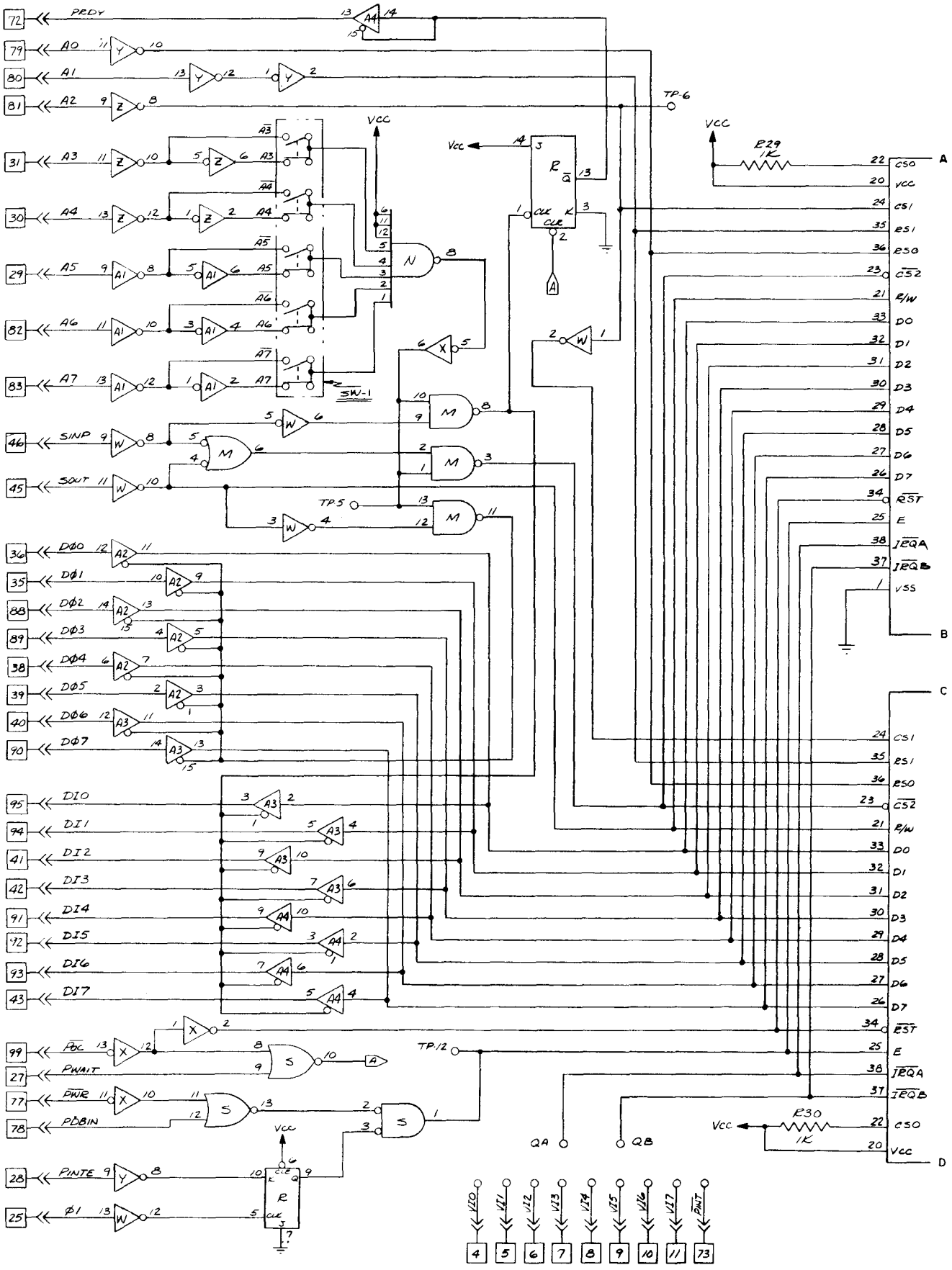


Fig. C.27 Pertec/MITS 88-AD/DA, multiple input/output analog card *(sheet 1 of 2)*

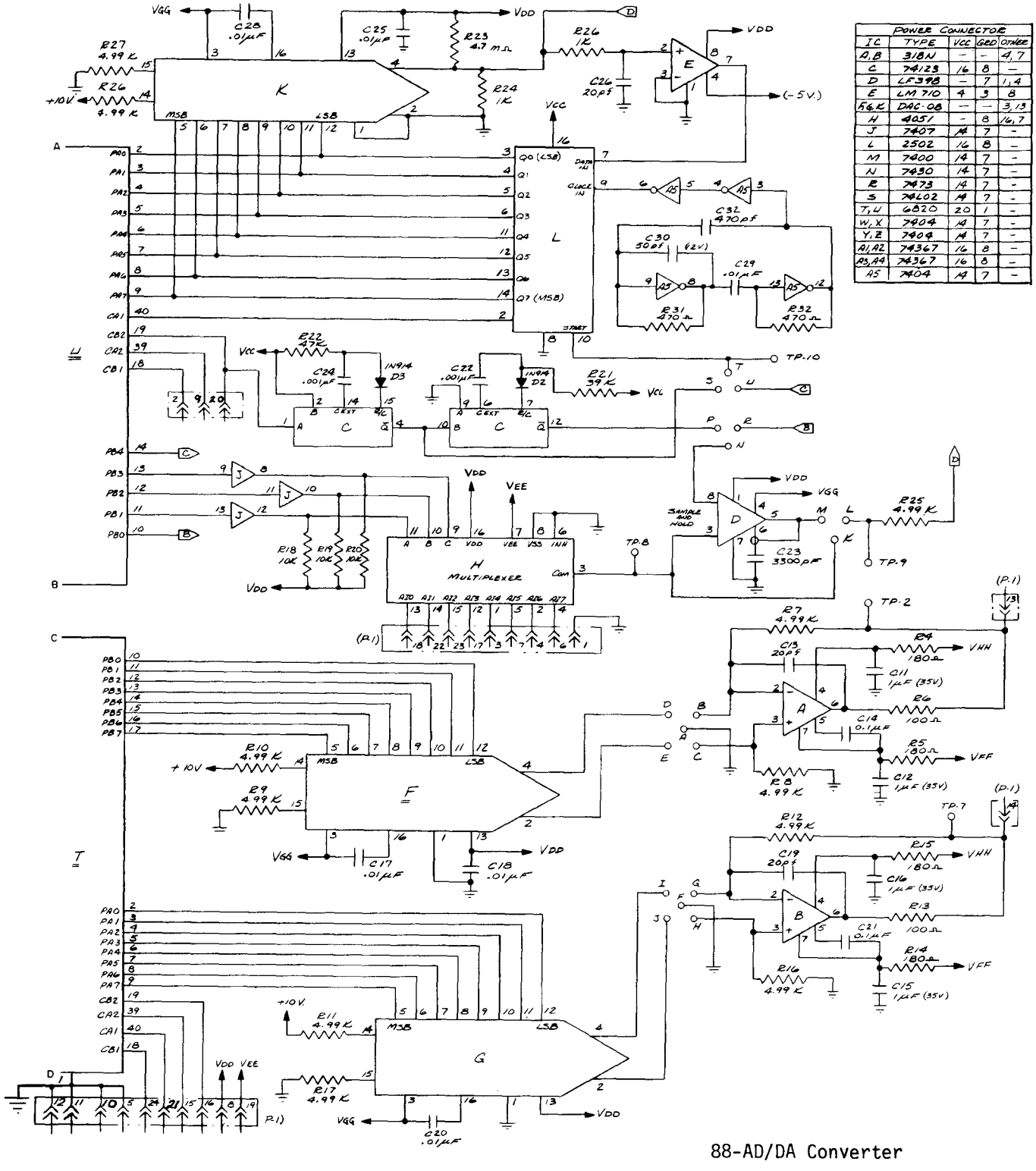


Fig. C.27 Pertec/MITS 88-AD/DA, multiple input/output analog card (*sheet 2 of 2*)

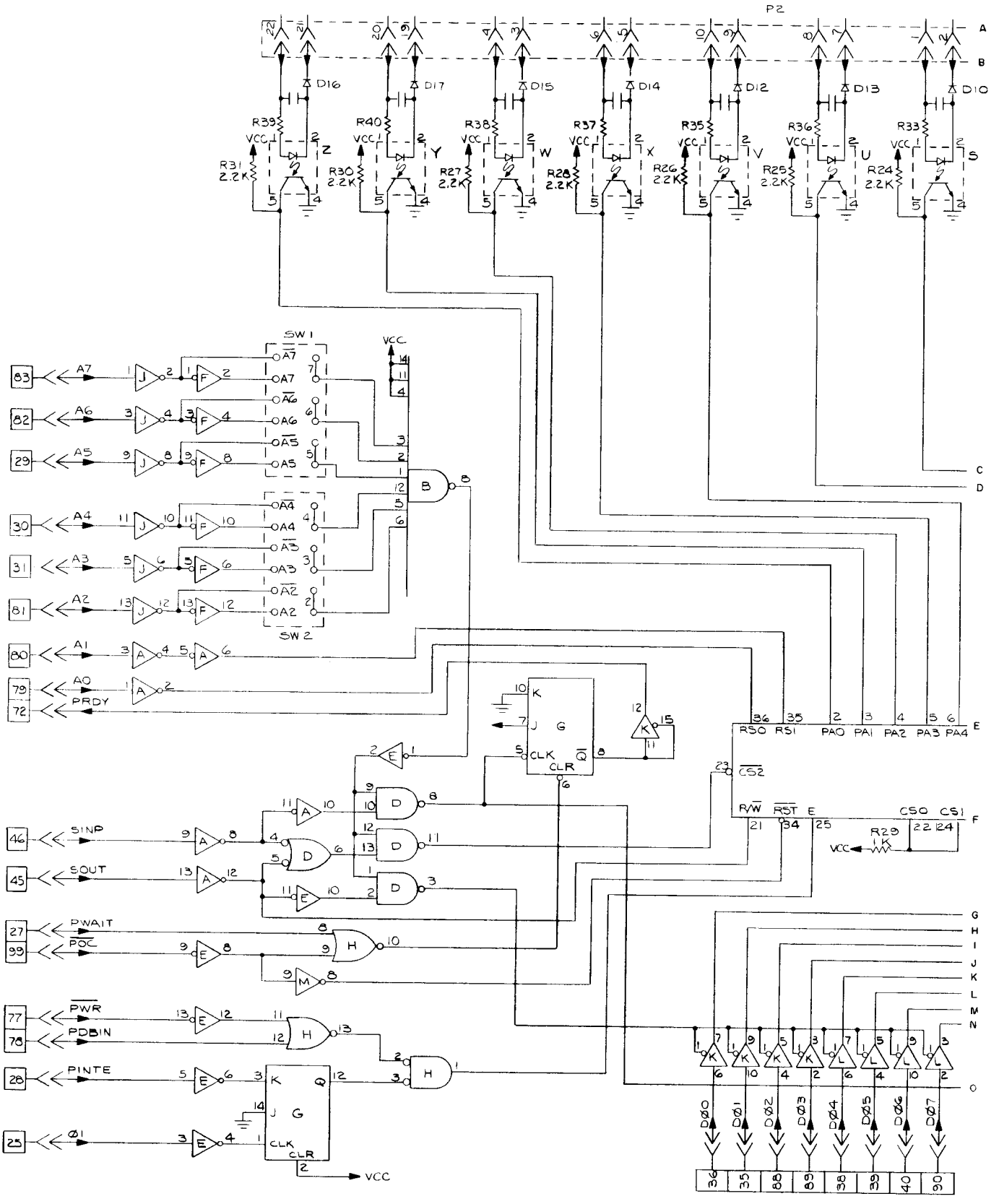


Fig. C.28 Pertec/MITS 88-PCI, process control interface card (*sheet 1 of 2*)

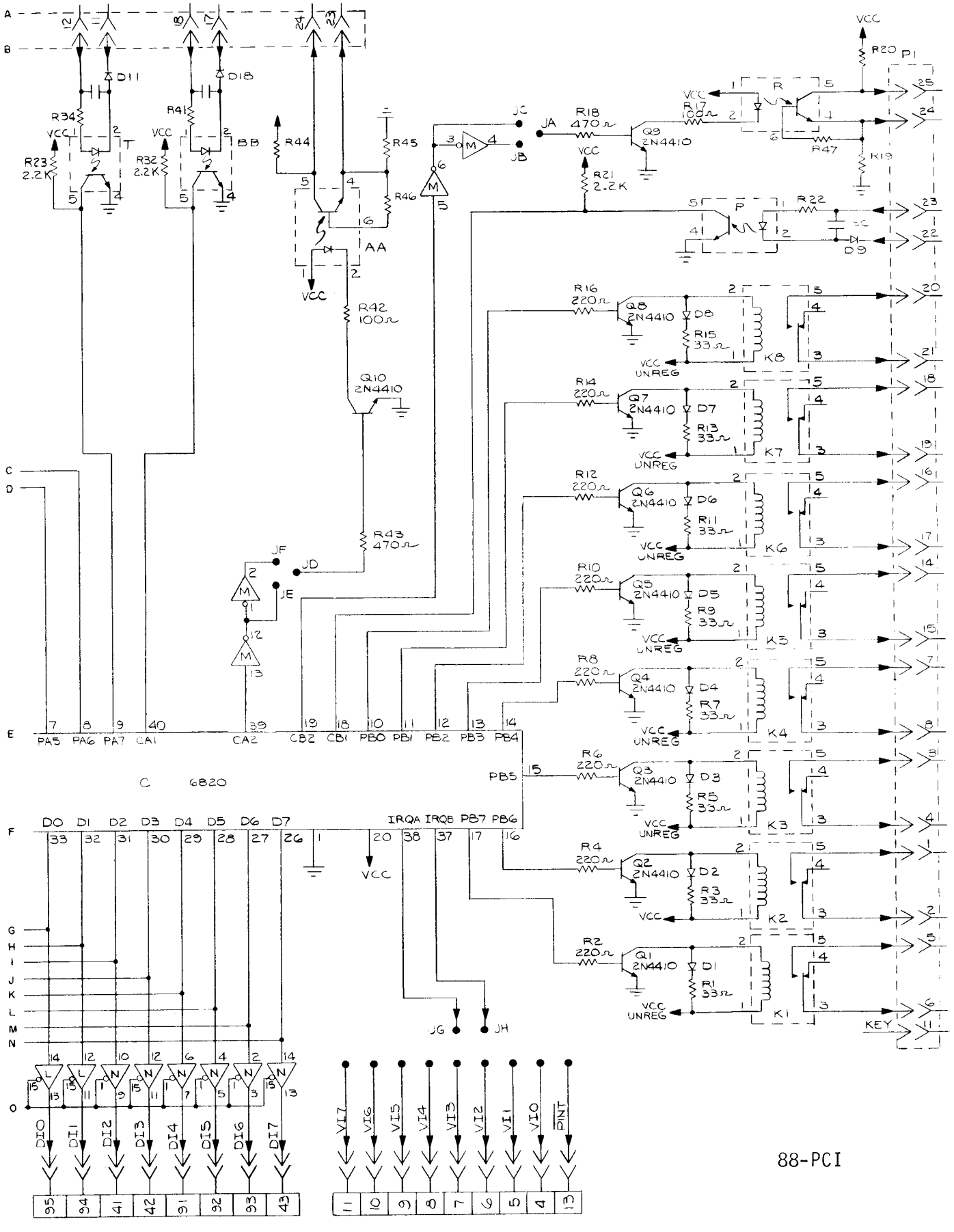


Fig. C.28 Pertec/MITS 88-PCI, process control interface card (*sheet 2 of 2*)

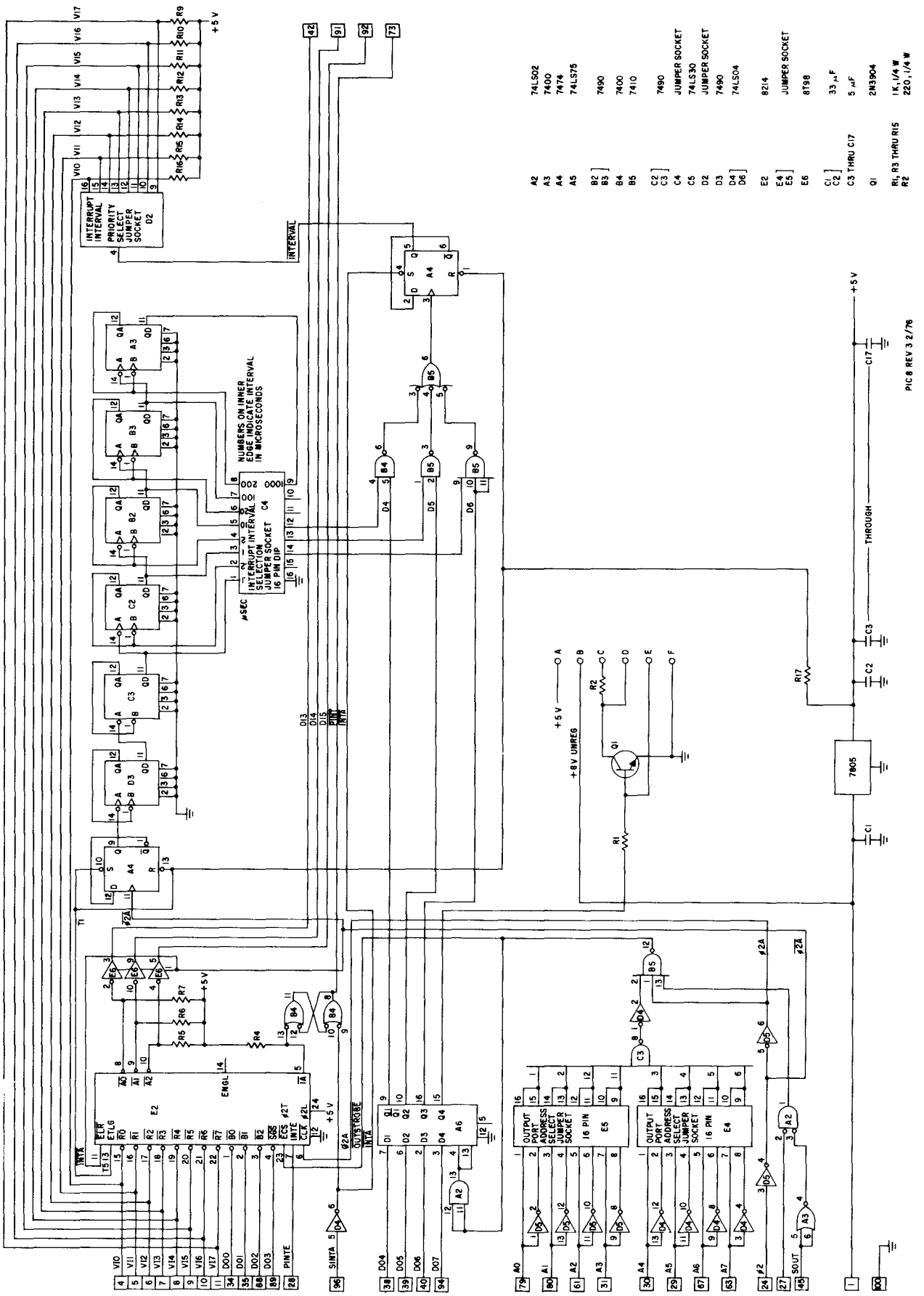


Fig. C.29 Imsai 8080 PIC-8, priority interrupt controller

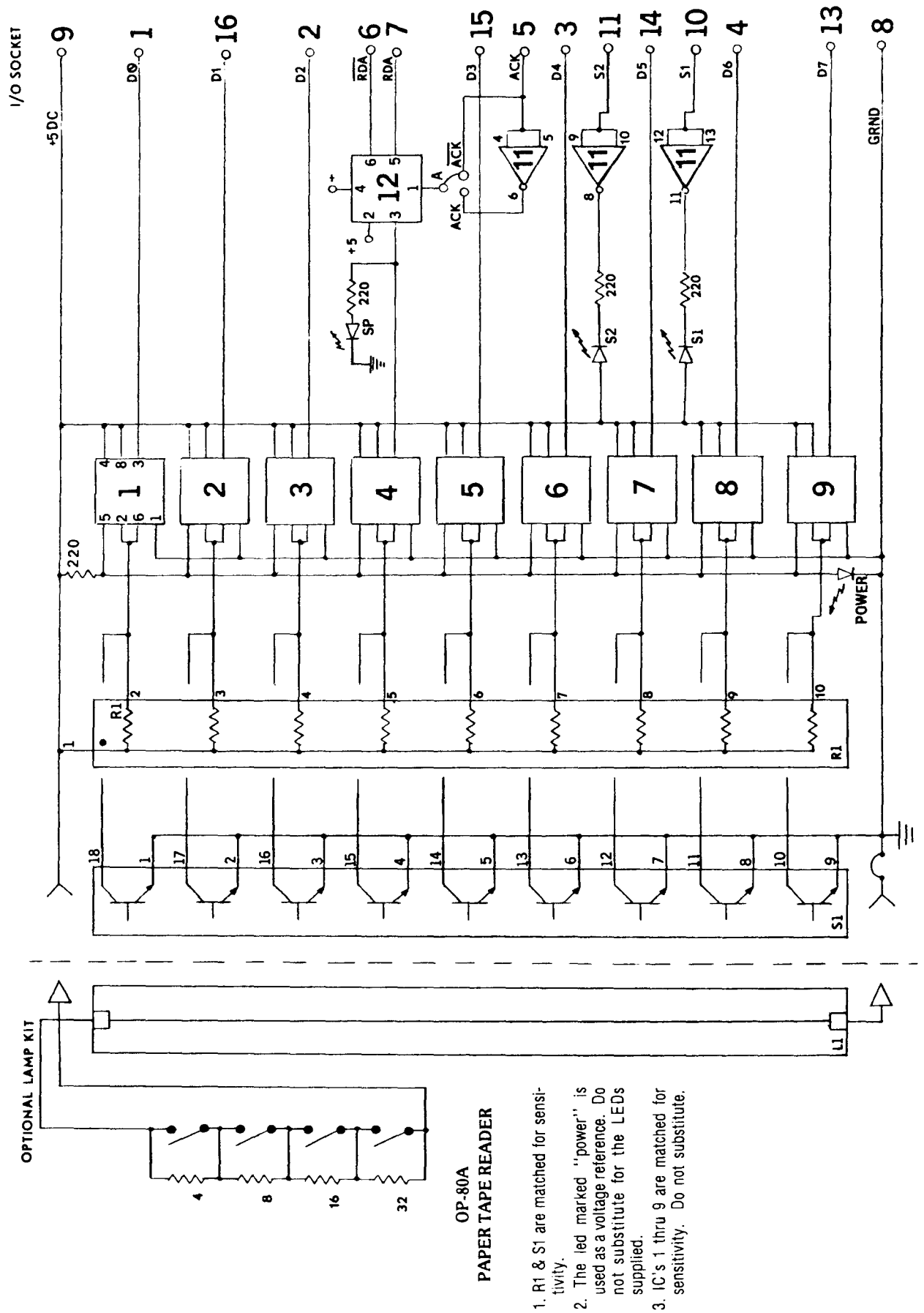


Fig. C.30 Oliver Audio Engineering OP-80A, manual paper-tape reader

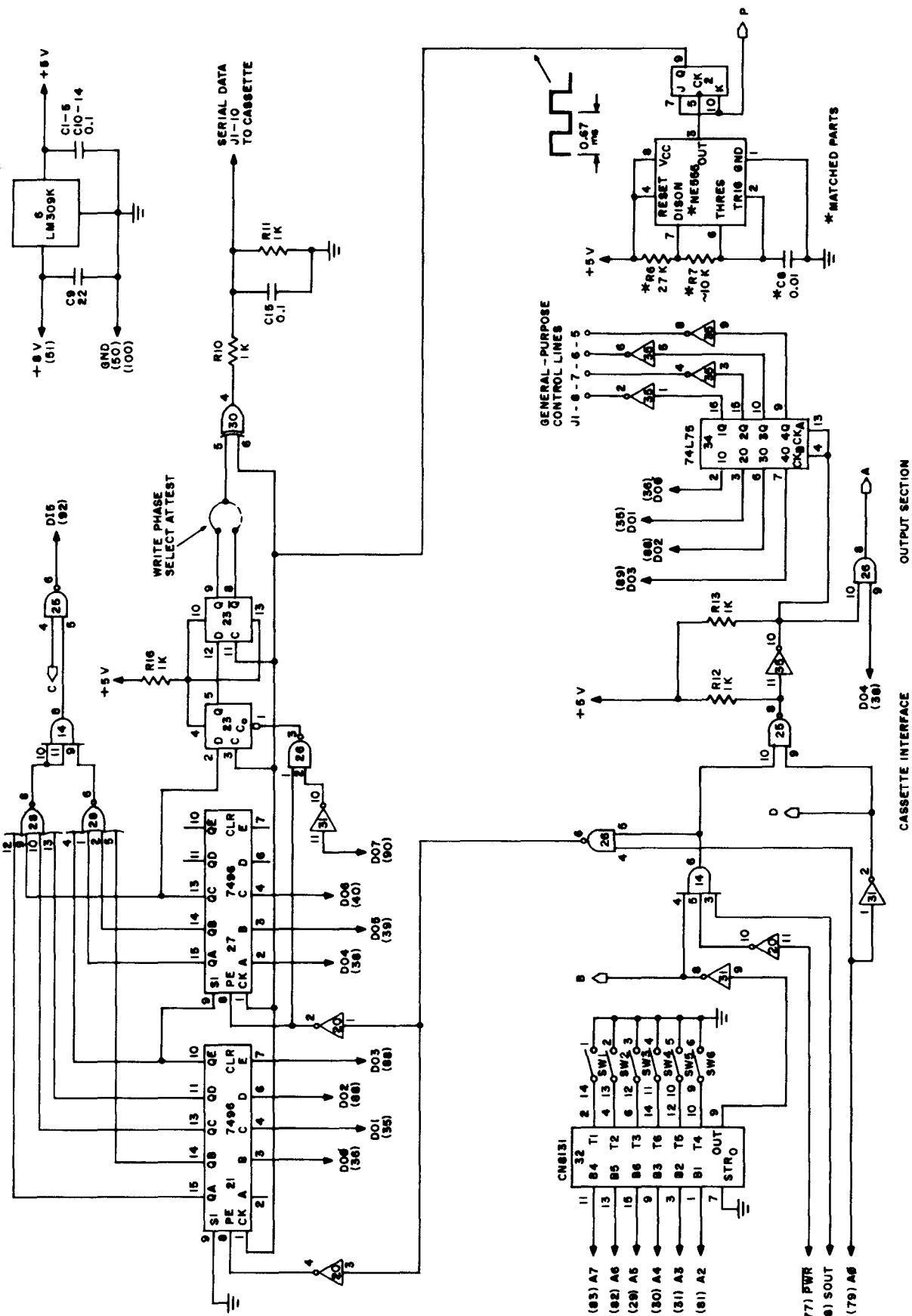


Fig. C.31 Tarbell Electronics TCI, audio cassette interface card (*sheet 1 of 2*)

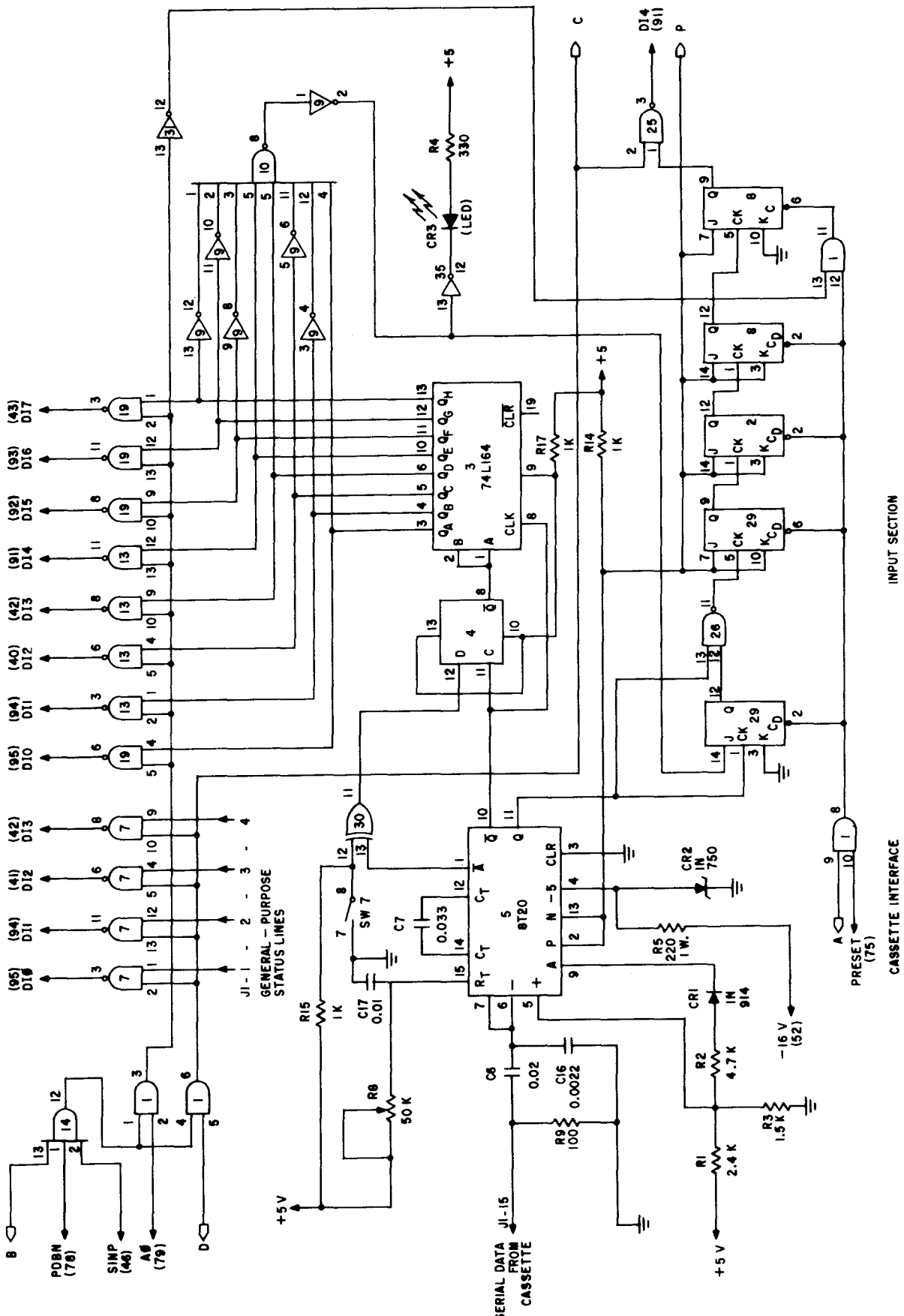


Fig. C.31 Tarbell Electronics TCI, audio cassette interface card (*sheet 2 of 2*)

APPENDIX D

Proposed S-100 Bus Standard

Before going into the actual details for the proposed S-100 bus specification,* I would just like to state that the committee chairmen of IEEE task no. P696 are still interested in suggestions and criticisms of the proposed standard. If you wish to contact them, you may write or call Mr. George Morrow, c/o Thinker Toys, 5221 Central Avenue, Richmond, CA 94804 (415) 524-2101, or Howard Fullmer, c/o Parasitic Engineering, 1201 10th Street, Berkeley, CA 94710 (415) 527-6133.

For the purposes of brevity, just the simplest description of the proposed standard is included here. A full copy of the standard was published in the July 1979 issue of IEEE *Computer* magazine. Copies of this issue can be ordered from the IEEE for a single copy price of $3 to IEEE members or $6 to nonmembers. For a single copy, send payment to Editor, *Computer* Magazine, 5855 Naples Plaza, Suite 301, Long Beach, CA 90803. Additionally, the fully revised standard is available as part of a book published by the IEEE Computer Society in the fall of 1979. This book, *Microprocessors and Microcomputers*, is available at a cost of $9 to IEEE members and $12 to nonmembers.

Although the S-100 bus has now been in use for almost half a dozen years, there has not been one agreed-upon specification that all manufacturers can point to and say that their board meets all requirements. This is changing, though. Just before this book went to press, the Institute of Electrical and Electronic Engineers (IEEE) published what is called the "standard specification for S-100 bus interface devices." This "standard" is a result of IEEE task 696.1/D2 and is almost a two-year culmination of effort to eliminate many of the bus's problems and upgrade it for use with 16-bit microprocessors.

Basically, the changes to the bus include the following upgrades—the extension of the address bus from 16 to 24 bits, ganging of the 8-bit data-in and data-out buses into a 16-bit bidirectional bus, and the addition of two handshaking lines to permit intermixing of 8- and 16-bit memory cards. Also added is a binary-encoded multiple-master arbitration bus that permits up to 16 master CPUs on the bus. The necessary logic to do the arbitration can be built in one IC, so the overhead for the multiple CPU configuration is minimal. Additional ground lines, a power fail line, and an error line have also been added.

Three other lines, termed NDEF (not to be defined), have been allotted for manufacturers and users to create their own special control signals, if needed. Five additional lines are reserved for future use (called RFU lines), and some lines formerly used for front-panel purposes have been deleted, with the intention that such lines can best be handled by a jumper cable from the CPU card to the front panel. Also implemented will be a DMA (direct memory access) protocol that provides overlap of the control lines at the beginning and end of the transition between permanent and temporary masters. This allows address, data, and control buses to settle before information is transferred.

The proposed standard applies to microprocessor interface systems that use the 100-line parallel backplane commonly known as the S-100 bus. Within the system,

- Digital data are exchanged among interconnected devices.
- The total number of interconnected devices is small (22 or fewer).
- The total transmission path length is electrically short (25 in. or less); that is, the transmission line propagation delays are not important.
- The maximum data rate of any signal on the bus is low (less than or equal to 6 MHz).

Thus, the bus provides a means of communication between two basic functional elements that organize and manage the flow of information among devices—a device operating as a bus master and a device acting as a bus slave. A bus master has the capability to generate all interface messages necessary to cause a bus cycle and then transfer device-dependent messages to or from the addressed slave as part of that bus cycle. The master

*Based on "Standard Specification for S-100 Bus Interface Devices," IEEE Task 696.1/D2, K. A. Elmquist, Howard Fullmer, D. B. Gustavson, and George Morrow. *Computer*, July 1979, Vol. 12, No. 7, pp. 28-51.

can also address all slave devices or some portion of them. Bus slaves monitor all bus cycles and can thus be addressed by a master and then transfer device-dependent messages to or from the master. Master and slave capability occur both individually and collectively in devices that are interconnected to the bus.

The S-100 interface system consists of a set of signal lines used to carry all information, interface messages, and device-dependent messages among interconnected devices.

The bus structure is organized into eight sets of signal lines:

1. Data bus—16 signal lines
2. Address bus—16 or 24 signal lines
3. Status bus—8 signal lines
4. Control output bus—5 signal lines
5. Control input bus—6 signal lines
6. DMA control bus—8 signal lines
7. Vectored interrupt bus—8 signal lines
8. Utility bus—20 signal lines

Functional devices interconnected via the interface system are divided into two broad classifications, bus masters and bus slaves, according to their relationship to the generation and reception of interface messages. Devices acting as bus masters are responsible for the initiation of all bus cycles and for the generation of all signals necessary for the conduction of an unambiguous bus cycle. These signals are termed type M signals and consist of the address, status, and control buses. Device-dependent messages are transmitted and received on the data bus.

Bus masters are subdivided into two classifications—permanent masters and temporary masters. A permanent bus master (generally a CPU) is the highest priority master in the interface system. A temporary master may request the bus from the permanent master for an arbitrary number of bus cycles and then returns control of the bus to the permanent master. The transfer of bus control from a permanent master to a temporary master and back to the permanent master is termed a DMA cycle.

The difference between a permanent bus master and a temporary bus master is that:

1. Only one permanent master may exist within the interface system, whereas up to 16 temporary masters may coexist in a single system.
2. A temporary master is not subject to a DMA cycle; that is, there are no nested DMA operations.

Devices acting as bus slaves are bus cycle receptors. A bus slave monitors all bus cycles and, if addressed during a particular bus cycle, accepts or sends the requested device-dependent message on the data lines. While bus masters must generate a specific set of signals in order to assure an unambiguous bus cycle, a bus slave need only examine and generate that subset of bus signals necessary to communicate with bus masters.

The address bus consists of 16 or 24 bit-parallel signal lines used to select a specific location in memory or a specific input/output device for communications during the current bus cycle. All bus masters must assert at least 16 address bits but may assert 24 address bits if extended address capability is desired.

The standard memory address bus consists of 16 lines specifying 1 of 64k memory locations. These 16 lines are named A0 through A15, where A15 is the most significant bit. The extended memory address bus consists of 24 lines specifying 1 of 16 million memory locations. These 24 lines are named A0 through A23, where A23 is the most significant bit.

The standard I/O device address bus consists of lines, A0 through A7, specifying 1 of 256 I/O devices, with A7 used as the most significant bit. Note that, however, the I/O device address has traditionally been duplicated onto the high order address byte A15-A8. While this is considered acceptable procedure, it is not recommended for new designs as it complicates expansion to extended I/O device addressing. The extended I/O device address bus consists of 16 lines, A0 through A15, specifying 1 of 64k devices. A15 is the most significant bit.

The status bus consists of eight lines that identify the nature of the bus cycle in progress and qualify the nature of the address on the address bus. The mnemonics for status lines always begin with a lowercase *s* and consist of:

1. Memory read—sMEMR
2. Op-code fetch—sM1
3. Input—sINP
4. Output—sOUT
5. Write cycle—sWO*
6. Interrupt acknowledge—sINTA
7. Halt acknowledge—sHLTA
8. Sixteen-bit data transfer request—sXTRQ*

All eight status signals must be generated by the current bus master.

One relevant status signal is not directly available on the bus but may be created by the combination of two others. Status Memory Write is defined as:

$$\text{sMemory Write} = (\text{-sOUT}) \cdot \text{sWO (logic equation)}$$

that is, status memory write is true when sOUT is false and sWO (write) is true. Data input and data output are always specified relative to the current bus master. Data transmitted by the current bus master to a bus slave is called data output. Data received by the current bus master from a bus slave is called data input.

The data bus consists of 16 lines grouped as two unidirectional 8-bit buses for byte operations and as a single bidirectional bus for 16-bit word operations. Two unidirectional 8-bit buses are used for byte data transfers. Data output appears on the data output bus (DO0–DO7), where DO7 is the most significant bit.

Data input appears on the data input bus (DI0–DI7), where DI7 is the most significant bit.

For 16-bit data transfers the DI and DO buses are ganged together, creating a single 16-bit bidirectional bus. Two signal lines control the ganging of the data buses, sixteen request (sXTRQ*) and sixteen acknowledge (SIXTN*). When both of these lines are true (in the low state), the data buses are ganged with DO0 corresponding to DATA 0 and DI7 corresponding to DATA 15, the most significant bit.

The five lines of the control output bus determine the timing and movement of data during any bus cycle. The mnemonics for the control output lines always begin with a lowercase *p*. The five lines are:

1. pSYNC, which indicates the start of a new bus cycle.
2. pSTVAL*, which in conjunction with pSYNC indicates that stable address and status may be sampled from the bus in the current cycle.
3. pDBIN, a generalized read strobe that gates data from an addressed slave onto the data bus.
4. pWR*, a generalized write strobe that writes data from the data bus into an addressed slave.
5. pHLDA, the hold acknowledge signal that indicates to the highest priority temporary master that the permanent master is relinquishing control of the bus.

The six lines of the control input bus allow bus slaves to synchronize the operations of bus masters with conditions internal to the bus slave (e.g., data not ready) and to request operations of the permanent master (e.g., interrupt or hold). The six control input lines are:

1. RDY
2. XRDY
3. INT*
4. NMI*
5. HOLD*
6. SIXTN*

The ready lines are used by bus slaves to synchronize bus masters to the response speed of the slave. Thus cycles are suspended and wait states inserted until both ready lines are asserted. The RDY line is the general ready line for bus slaves. It is specified as an open collector line. The XRDY line is a special ready line commonly used by front-panel devices to stop and single step bus masters. It is not specified as an open collector line and should not be used by other bus slaves, since a bus conflict may exist.

The two interrupt lines, INT* and NMI*, are used to request service from the permanent bus master. The INT* line may be masked off by the bus master, usually via an internal software operation. If the master accepts the interrupt request on the INT* line, it may respond with an interrupt acknowledge bus cycle, accepting vectoring information from the data bus. The INT* line is often implemented as a "group interrupt" line in conjunction with the vectored interrupt bus. In this case, INT* indicates the presence of one or more vectored interrupt requests.

The NMI* line is a nonmaskable interrupt request line; that is, it may not be masked off by the bus master. Accepting an interrupt on the NMI* line need not generate an interrupt acknowledge bus cycle. An interrupt request on the INT* line is asserted as a level; that is, the line is asserted until interrupt service is received. An interrupt request on the NMI* line, on the other hand, is asserted as a negative going edge, since no interrupt acknowledge cycle need be generated. Both lines are specified as open collector lines.

The hold request line, HOLD*, is used by temporary bus masters to request control of the bus from the permanent bus master. The HOLD* line may be masked by the permanent bus master to prevent temporary masters from gaining bus control. The HOLD* line is specified as an open collector line and may only be asserted at certain times.

The sixteen acknowledge line, SIXTN*, is a response to the status signal request (sXTRQ*) and indicates that the requested 16-bit data transfer is possible. The SIXTN* line is specified as an open collector line.

The eight lines of the DMA control bus are used in conjunction with control bus signals HOLD* and pHLDA. They arbitrate among simultaneous requests for control of the bus by temporary masters and disable the signal drivers of the permanent bus master, thus effecting an orderly transfer of bus control. All eight lines of the DMA control bus are specified as open collector lines, and these control lines are:

1. DMA0*
2. DMA1*
3. DMA2*
4. DMA3*
5. ADSB
6. DODSB*
7. SDSB*
8. CDSB*

The four lines that arbitrate among simultaneous requests for bus control by temporary masters are DMA0* through DMA3*. The encoded priority of requesters is asserted on these lines and, after settling, they contain the priority number of the highest priority requester.

Four signals are available on the bus to disable the line drivers of the permanent bus master. They are:

1. ADSB*, address disable
2. DODSB*, data out disable
3. SDSB*, status disable
4. CDSB*, control output disable

Use of these lines is tightly specified during the transfer of the bus from a permanent master to a temporary

master, and any transfer involving the control output lines should follow a similar protocol. The address, data, and status signals from the permanent master may be disabled and replaced using these signals as long as the contents of these buses are valid for the current bus cycle as though no replacement had occurred.

The eight lines of the vectored interrupt bus are used in conjunction with the generalized vectored interrupt request, INT*, to arbitrate among eight levels of interrupt request priorities. They are typically implemented as inputs to a bus slave that masks and gives priority to the requests, asserts the generalized interrupt request to the permanent bus master, and responds to the interrupt acknowledge bus cycle with appropriate vectoring data. The eight lines of the vectored interrupt bus are VI0* through VI7, where VI0* is considered the highest priority interrupt. The vectored interrupt lines should be implemented as levels; that is, they should be held active until service is received.

Power in S-100 systems is distributed to bus devices as unregulated voltages. A total of nine bus lines are used:

1. +8 Volts, 2 lines
2. +16 Volts, 1 line
3. −16 Volts, 1 line
4. GROUND, 5 lines

Ground lines are distributed across the edge connector such that low impedance grounds are available on both sides of the edge connector and on both sides of the circuit cards.

The system clock, ϕ, is generated by the permanent master. The control timing for all bus cycles, whether they are cycles of the permanent master or cycles of temporary masters in control of the bus, must be derived from this clock. This signal is never transferred during a bus exchange operation.

Another line, called CLOCK, is specified as a 2-MHz (0.5 percent tolerance) signal with no relationship to any other bus signal. It is to be used by counters, timers, baud-rate generators, etc.

System reset functions are divided into three lines:

1. RESET*, resets all bus masters.
2. SLAVE CLR*, resets all bus slaves.
3. POC*, power-on clear is active only on power-on, asserts SLAVE CLR* and RESET*, and is specified as having a minimum active period of 10 ms.

RESET* and SLAVE CLR* are specified as open-collector lines.

The memory write strobe, MWRT, must be generated somewhere in the system. It is usually generated by front-panel type devices but is optionally generated by permanent masters or motherboards in systems without front panels. Care must be taken that it is generated at only one point in a given system. Memory write is defined as:

$$\text{MWRT} = \text{pWR} \cdot \text{-sOUT} \quad \text{(logic equation)}$$

Another line, PHANTOM*, is provided for overlaying bus slaves at a common address location. When this line is activated, phantom bus slaves are enabled and normal bus slaves are disabled. This line is specified as an open-collector line.

The line ERROR* is a generalized error line that is asserted when an error of some sort (i.e., parity, write to protected memory) occurs in the current bus cycle. This line is specified as an open-collector line.

Three lines that can be specified by individual manufacturers are provided on the bus. These lines, termed NDEF (not to be defined), should only be implemented as options and shall be provided with jumpers so that possible conflicts may be eliminated. Any manufacturer *must* specify in detail any use of these lines. Signals on these lines are limited to 5-V logic levels.

The power fail line (PWRFAIL*) indicates impending power failure and remains true until power is restored and POC* is true. The five remaining lines of the bus are reserved for future use and may not be used for any purpose.

S-100 Bus Pin List

Pin No.	Signal and Type	Active Level		Description
1	+8 VOLTS (B)			Instantaneous minimum greater than 7 volts, instantaneous maximum less than 25 volts, average maximum less than 11 volts.
2	+16 VOLTS (B)			Instantaneous minimum greater than 14.5 volts, instantaneous maximum less than 35 volts, average maximum less than 21.5 volts.
3	XRDY (S)	H		One of two ready inputs to the current bus master. The bus is ready when both these ready inputs are true. See pin 72.
4	VI0*(S)	L	O.C.	Vectored interrupt line 0.
5	VI1*(S)	L	O.C.	Vectored interrupt line 1.
6	VI2*(S)	L	O.C.	Vectored interrupt line 2.
7	VI3*(S)	L	O.C.	Vectored interrupt line 3.
8	VI4*(S)	L	O.C.	Vectored interrupt line 4.
9	VI5*(S)	L	O.C.	Vectored interrupt line 5.
10	VI6*(S)	L	O.C.	Vectored interrupt line 6.
11	VI7*(S)	L	O.C.	Vectored interrupt line 7.
12	NMI*(S)	L	O.C.	Nonmaskable interrupt.
13	PWRFAIL*(B)	L		Power fail bus signal.
14	DMA3* (M)	L	O.C.	Temporary master priority bit 3.
15	A18 (M)	H		Extended address bit 18.
16	A16 (M)	H		Extended address bit 16.
17	A17 (M)	H		Extended address bit 17.
18	SDSB* (M)	L	O.C.	The control signal to disable the 8 status signals.
19	CDSB* (M)	L	O.C.	The control signal to disable the 5 control output signals.
20	GND (B)			Common with pin 100.
21	NDEF			Not to be defined. Manufacturer must specify any use in detail.
22	ADSB* (M)	L	O.C.	The control signal to disable the 16 address signals.
23	DODSB* (M)	L	O.C.	The control signal to disable the 8 data output signals.
24	ϕ (B)	H		The master timing signal for the bus.
25	pSTVAL*(M)	L		Status valid strobe.
26	pHLDA (M)	H		A control signal used in conjunction with HOLD* to coordinate bus master transfer operations.
27	RFU			Reserved for future use.
28	RFU			Reserved for future use.
29	A5 (M)	H		Address bit 5.
30	A4 (M)	H		Address bit 4.
31	A3 (M)	H		Address bit 3.
32	A15 (M)	H		Address bit 15 (most significant for non-extended addressing.)
33	A12 (M)	H		Address bit 12.
34	A9 (M)	H		Address bit 9.
35	DO1 (M)/DATA1 (M/S)	H		Data out bit 1, bidirectional data bit 1.
36	DO0 (M)/DATA0 (M/S)	H		Data out bit 0, bidirectional data bit 0.
37	A10 (M)	H		Address bit 10.
38	DO4 (M)/DATA4 (M/S)	H		Data out bit 4, bidirectional data bit 4.
39	DO5 (M)/DATA5 (M/S)	H		Data out bit 5, bidirectional data bit 5.
40	DO6 (M)/DATA6 (M/S)	H		Data out bit 6, bidirectional data bit 6.
41	DI2 (S)/DATA10 (M/S)	H		Data in bit 2, bidirectional data bit 10.
42	DI3 (S)/DATA11 (M/S)	H		Data in bit 3, bidirectional data bit 11.
43	DI7 (S)/DATA15 (M/S)	H		Data in bit 7, bidirectional data bit 15.
44	sM1 (M)	H		The status signal which indicates that the current cycle is an op-code fetch.
45	sOUT (M)	H		The status signal identifying the data transfer bus cycle to an output device.
46	sINP (M)	H		The status signal identifying the data transfer bus cycle from an input device.
47	sMEMR (M)	H		The status signal identifying bus cycles which transfer data from memory to a bus master, which are not interrupt acknowledge instruction fetch cycle(s).
48	sHLTA (M)	H		The status signal which acknowledges that a HLT instruction has been executed.
49	CLOCK(B)			2 MHz (0.5%) 40-60% duty cycle. Not required to be synchronous with any other bus signal.
50	GND (B)			Common with pin 100.
51	+8 VOLTS (B)			Common with pin 1.
52	-16 VOLTS (B)			Instantaneous maximum less than -14.5 volts, instantaneous minimum greater than -35 volts, average minimum greater than -21.5 volts.
53	GND (B)			Common with pin 100.
54	SLAVE CLR* (B)	L	O.C.	A reset signal to reset bus slaves. Must be active with POC* and may also be generated by external means.

S-100 Bus Pin List (cont'd)

Pin No.	Signal and Type	Active Level		Description
55	DMAO* (M)	L	O.C.	Temporary master priority bit 0.
56	DMA1* (M)	L	O.C.	Temporary master priority bit 1.
57	DMA2* (M)	L	O.C.	Temporary master priority bit 2.
58	sXTRQ* (M)	L		The status signal which requests 16-bit slaves to assert SIXTN*.
59	A19 (M)	H		Extended address bit 19.
60	SIXTN* (S)	L	O.C.	The signal generated by 16-bit slaves in response to the 16-bit request signal sXTRQ*.
61	A20 (M)	H		Extended address bit 20.
62	A21 (M)	H		Extended address bit 21.
63	A22 (M)	H		Extended address bit 22.
64	A23 (M)	H		Extended address bit 23.
65	NDEF			Not to be defined signal.
66	NDEF			Not to be defined signal.
67	PHANTOM* (M/S)	L	O.C.	A bus signal which disables normal slave devices and enables phantom slaves—primarily used for bootstrapping systems without hardware front panels.
68	MWRT (B)	H		pWR·—sOUT (logic equation). This signal must follow pWR* by not more than 30 ns.
69	RFU			Reserved for future use.
70	GND (B)			Common with pin 100.
71	RFU			Reserved for future use.
72	RDY (S)	H	O.C.	See comments for pin 3.
73	INT* (S)	L	O.C.	The primary interrupt request bus signal.
74	HOLD* (M)	L	O.C.	The control signal used in conjunction with pHLDA to coordinate bus master transfer operations.
75	RESET*(B)	L	O.C.	The reset signal to reset bus master devices. This signal must be active with POC* and may also be generated by external means.
76	pSYNC (M)	H		The control signal identifying BS_1.
77	pWR* (M)	L		The control signal signifying the presence of valid data on DO bus or data bus.
78	pDBIN (M)	H		The control signal that requests data on the DI bus or data bus from the currently addressed slave.
79	A0 (M)	H		Address bit 0 (least significant).
80	A1 (M)	H		Address bit 1.
81	A2 (M)	H		Address bit 2.
82	A6 (M)	H		Address bit 6.
83	A7 (M)	H		Address bit 7.
84	A8 (M)	H		Address bit 8.
85	A13 (M)	H		Address bit 13.
86	A14 (M)	H		Address bit 14.
87	A11 (M)	H		Address bit 11.
88	DO2 (M)/DATA2 (M/S)	H		Data out bit 2, bidirectional data bit 2.
89	DO3 (M)/DATA3 (M/S)	H		Data out bit 3, bidirectional data bit 3.
90	DO7 (M)/DATA7 (M/S)	H		Data out bit 7, bidirectional data bit 7.
91	DI4 (S)/DATA12 (M/S)	H		Data in bit 4 and bidirectional data bit 12.
92	DI5 (S)/DATA13 (M/S)	H		Data in bit 5 and bidirectional data bit 13.
93	DI6 (S)/DATA14 (M/S)	H		Data in bit 6 and bidirectional data bit 14.
94	DI1 (S)/DATA9 (M/S)	H		Data in bit 1 and bidirectional data bit 9.
95	DI0 (S)/DATA8 (M/S)	H		Data in bit 0 (least significant for 8-bit data) and bidirectional data bit 8.
96	sINTA (M)	H		The status signal identifying the bus input cycle(s) that may follow an accepted interrupt request presented on INT*.
97	sWO* (M)	L		The status signal identifying a bus cycle which transfers data from a bus master to a slave.
98	ERROR* (S)	L	O.C.	The bus status signal signifying an error condition during present bus cycle.
99	POC* (B)	L		The power-on clear signal for all bus devices; when this signal goes low, it must stay low for at least 10 ms.
100	GND (B)			System ground.

Index