microcomputer design

by Donald F. Martin

Vice President, Product Development
MARTIN RESEARCH

edited and published by

Kerry S. Berland
Vice President, Marketing
MARTIN RESEARCH

SECOND EDITION
(Revised)
October, 1976

martin research
3336 Commercial Avenue
Northbrook, IL 60062
(312) 498-5660
This book covers both basic and advanced information on designing with microprocessors. It assumes that the reader is familiar with digital electronic circuit design, TTL (transistor-transistor logic) in particular.

We have avoided unnecessary duplication of the manufacturers' literature on the 8088 and 8086 microprocessors within the text of this book, preferring to reserve as much space as possible for original hardware design information. The 8088 and 8086 data sheets are reprinted at the end of the book for the reader's convenience. The reader should become familiar with them as he reads this book.

The novice might begin by reading through the 8088 and 8086 data sheets and the following chapters:

1. The 8088 CPU
2. The 8086 CPU
3. Main Timing Logic
4. Software Tricks
5. Minimal Microcomputers
6. Nineteen-Chip Microcomputer
7. Central Processing Unit, Model 471 (an 8080 CPU board)

The reader already familiar with the 8080/8088 might start with these chapters:

5. Main Timing Logic
6. Adding 8086 Instructions
7. Random Access Memory (especially the RAM-PAGE option, Sec. 13.4)
8. Interrupts
9. Minimal Microcomputers
10. Nineteen-Chip Microcomputer
11. Central Processing Unit, Model 471 (an 8080 CPU board)

No book can substitute for practical hands-on experience with microcomputers. The modular micros from Martin Research provide an excellent series of hardware for educational and prototyping purposes. As this book goes to press, both 8080 and 8086 versions are available, and microcomputers based on other CPUs are due for announcement shortly. For more information on the modular micros, contact Martin Research.

Martin Research will appreciate receiving your comments on this book, including criticisms both negative and positive. Ideas and corrections will, with your permission, be used in future editions.
<table>
<thead>
<tr>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
</tr>
<tr>
<td>1 MICROCOMPUTER DESIGN</td>
</tr>
<tr>
<td>2 Designing with Microprocessors</td>
</tr>
<tr>
<td>3 Microcomputers</td>
</tr>
<tr>
<td>4 Notes on the 8008, 8089, and MOS Memories</td>
</tr>
<tr>
<td>2. THE 8008</td>
</tr>
<tr>
<td>1 Brief Introduction to Microcomputers</td>
</tr>
<tr>
<td>2 8008 Architecture</td>
</tr>
<tr>
<td>3 8008 Timing Signals</td>
</tr>
<tr>
<td>4 PDC Instruction Cycle</td>
</tr>
<tr>
<td>5 PDC Read Cycle</td>
</tr>
<tr>
<td>6 PDC Command Cycle</td>
</tr>
<tr>
<td>7 PDC Write Cycle</td>
</tr>
<tr>
<td>8 Instruction Cycle</td>
</tr>
<tr>
<td>9 A Simple Program</td>
</tr>
<tr>
<td>3. THE 8080</td>
</tr>
<tr>
<td>1 The 8080</td>
</tr>
<tr>
<td>2 8080 Hardware</td>
</tr>
<tr>
<td>3 8080 Software</td>
</tr>
<tr>
<td>4 Selected 8080 Technical Notes</td>
</tr>
<tr>
<td>4. MICROPROCESSOR COMPARISONS</td>
</tr>
<tr>
<td>1 Introduction</td>
</tr>
<tr>
<td>2 Systems Design</td>
</tr>
<tr>
<td>3 Processor Comparisons</td>
</tr>
<tr>
<td>4 The 808 vs Microcomputer</td>
</tr>
<tr>
<td>5. MAIN TIMING</td>
</tr>
<tr>
<td>1 8008 Clock Circuits</td>
</tr>
<tr>
<td>2 Enable and Strobe Generation</td>
</tr>
<tr>
<td>3 Master Reset</td>
</tr>
<tr>
<td>4 Additional Circuitry Required for Basic Microcomputer</td>
</tr>
<tr>
<td>5 8008 Clock Generation</td>
</tr>
<tr>
<td>6. BUS STRUCTURES</td>
</tr>
<tr>
<td>1 Introduction to Bus Structures</td>
</tr>
<tr>
<td>2 Latching Logic</td>
</tr>
<tr>
<td>3 Three-State Devices</td>
</tr>
<tr>
<td>4 Advanced Strobe Techniques</td>
</tr>
<tr>
<td>5 Types of Bus Structure</td>
</tr>
<tr>
<td>6 Bus Transceivers and BI-Bus Drivers</td>
</tr>
<tr>
<td>7 The 8008 Data Bus</td>
</tr>
<tr>
<td>7. INPUT/OUTPUT INSTRUCTIONS</td>
</tr>
<tr>
<td>1 I/O Address Modes</td>
</tr>
<tr>
<td>2 8080 Input/Output Instructions</td>
</tr>
<tr>
<td>3 Input and Output Symbols</td>
</tr>
<tr>
<td>4 Generating Input Enables and Output Strobes</td>
</tr>
<tr>
<td>5 Peripheral Strobe Decoding Techniques</td>
</tr>
<tr>
<td>6 8008 I/O Instructions</td>
</tr>
<tr>
<td>8. INPUT DESIGN APPLIQUES</td>
</tr>
<tr>
<td>1 Input Design</td>
</tr>
<tr>
<td>2 8080/8089 Data Bus Input Voltages</td>
</tr>
<tr>
<td>3 Input Multiplexers</td>
</tr>
<tr>
<td>4 Input Enable and Select Signals</td>
</tr>
<tr>
<td>5 Expanding 8008 Input Ports with Conditional Inputs</td>
</tr>
<tr>
<td>6 Additional Input Ports Defined as Memory</td>
</tr>
</tbody>
</table>
### INTRODUCTION

1.1 MICROCOMPUTER DESIGN

This book is about how to design a microcomputer. It focuses on microcomputers based on the first single-chip eight-bit microprocessor to reach high-volume production—the 8080—and on the improved second-generation device, the 8085.

The microprocessor itself is the miniaturized central processing unit (CPU) of a small computer. The significance of the microprocessor in electronic design is that the flexibility, adaptability, and logical processing power of the digital computer are now available in relatively small and inexpensive electronic products.

This book fills a gap in the literature presently available on microprocessors, since it concentrates on the hardware necessary to make the microprocessor a part of a practical and cost-effective electronic system.

### SEC. 1.2 DESIGNING WITH MICROPROCESSORS

1.2.1 Simple Microcomputers

The first 8008 circuits published and produced by the 8080 originator, Intel Corporation, were designed to exhibit the full potential of these CPUs. They used complex circuits to generate asymmetrical clocks, which optimized 8080 speed, but which—together with other such needless complicated designs—made using microprocessors look difficult. This was at a time when the technology was new, and the need was for design concepts to help engineers get started.

The first edition of this book presented the simplest possible microcomputer designs, using the fewest possible standard ICs to achieve the desired features. It is easier to add ICs to a simple design, to increase its potential, than it is to work with a needlessly complex design. Thus, for example, Chapter 5 greatly simplifies 8080 main timing circuitry, and Chapter 7 simplifies the creation of I/O interface signals.

The second generation 8085 is a step forward for the designer. Largely because it is housed in a 40-pin package (while the 8080 comes with 18), the 8085 makes more of its essential signals directly available to external control circuitry. Still, upon reading the manufacturers' 8080 literature, many designers are needlessly confused. The timing diagrams are complex and essential information is scattered among many data sheets.

In this second edition, our goal remains the same: to show how to use microprocessors—8080 and 8085—in practical microcomputers. The section on the 8080, Chapter 3, is expanded, and a data sheet for the Model 871 8080 computer is reprinted near the end of the book.

---

### CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>22. ANALOG INPUTS AND OUTPUTS</td>
<td>.6 Successive Approximation A/D Converter</td>
</tr>
<tr>
<td>23. SOFTWARE</td>
<td>.1 Software Tricks</td>
</tr>
<tr>
<td>24. TESTING</td>
<td>.1 Testing Microprocessors</td>
</tr>
<tr>
<td>25. MINIMAL MICROCOMPUTERS</td>
<td>.1 A Nine-Chip Microcomputer</td>
</tr>
<tr>
<td>26. NINETEEN-CHIP MICROCOMPUTER</td>
<td>.1 A Nineteen-Chip 8088-1 Microcomputer</td>
</tr>
<tr>
<td>471. 8080 CENTRAL PROCESSING UNIT</td>
<td>1.1 Introduction</td>
</tr>
<tr>
<td>8008 DATA SHEET</td>
<td>1.7 Reset Circuitry</td>
</tr>
<tr>
<td>8080 DATA SHEET</td>
<td>1.8 Interrupt Logic</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### MR microcomputer design
1.2 DESIGNING WITH MICROPROCESSORS (cont'd)

1.2.2 LSI Chip Design and Circuit Board Design. The integrated circuits now available which make use of LSI (Large-scale Integration) are impressively complex. One package may contain the equivalent of dozens of ordinary ICs and over a thousand transistors. The 8008 and 8086 microprocessors are good examples of this trend.

There is a distinct difference between an effective circuit design meant to be implemented within an LSI integrated circuit, and a good logic design using conventional ICs on a printed circuit board. The circuit designer needs to be aware of these differences, as his design may not be very efficient.

Take for example the 8008 microprocessor. Eight of the pins on its eighteen-pin package are used for its eight-bit bidirectional data bus. Usually all communication between the 8008 and the outside world takes place via this bus. Information on this bus may be read from the 8008's arithmetic logic unit; its seven-eight-bit data registers; its eight fourteen-bit memory address registers; its memory control logic; and its instruction decoding logic. The data on the bus may represent input information, output data, an instruction code, or a memory address. All of these processes take place at distinct times in the CPU's internal processor operations. (For details, see Chapter 3.)

Within the microprocessor, the eight-bit bus is piped around to all the major sections of the CPU. One of the most important elements in the chip's architecture is the control circuitry that selects which of the various sections within the CPU drive the bus, and which receive from the bus, at each stage in internal processor operations. (See Chapter 6 on bus structures generally.)

Consider for a moment the implementation of a bus control circuit using ordinary TTL ICs on a printed circuit board. These timing signals must be generated into their eight possible state combinations, and used to activate a variety of devices connected to the bus at different locations on the PC board. The board would need a 3-to-8-line decoder, and a likely choice would be a single TTL IC. This chip would contain eight 3-input NAND gates plus six inverters. The disadvantage of using a single-chip decoder is that, since eight decoder output lines are needed in many different places on the IC board, eight rather than three copper wires must be run all over the board. Still, no board designer would choose to separate gates over the one decoder.

The more complex IC and extra PC wires are generally much cheaper than buying, stocking, testing, debugging, and soldering extra ICs. After all, adding just two IC pins means two extra bonding pads on the chip, two bonds to the IC lead frame, two pins, two holes in the PC board, a copper foil pattern, and two solder joints.

Contrast the design decisions which face the chip designer when laying out the mask used in etching the silicon wafer from which a CPU is made. An on-chip 3-to-8 decoder may well be implemented with eight discrete 3-input NAND gates and up to twelve inverters, each gate and any needed inverters located right where the decoded signal is needed. Only the three lines to be decoded are bussed to the far corners of the chip, rather than the eight decoded outputs. Otherwise, the five extra bus lines would probably take up more chip area than any duplication in gates.

These considerations affect the way the microprocessor is organized. First, a relatively large number of special-purpose timing and control signals are developed on the chip, at multiple locations; they are derived from a small number of general-purpose timing signals. Second, only the few undifferentiated timing signals are brought out of the IC package to the outside world. This means that the hardware external to the 8008 must reproduce many special-purpose control signals in order to interface memory, input ports, and output ports with the CPU.

If one looks at the photomicrograph of the 8008, one sees that it was laid out by an excellent chip designer, who uses extra gates sooner than extra bus lines. But it is not effective for the hardware designer to mimic the chip designer in developing his microcomputer control and timing logic. He must develop circuitry which does its job with a minimum number of parts. Thus his timing and control logic may not look exactly the same as that used inside the CPU.

First, the designer must thoroughly grasp the internal processor operations of the CPU. Next, he must understand how the microprocessor's basic timing signals relate to the internal operations. In the case of the 8008, there are six basic timing signals: $d_1$, $d_2$, $d_3$, $d_4$, $d_5$, and $d_6$. Only then should the design of his hardware begin. This approach is much more likely to be productive than copying, or trying to simplify, the 8008 designs published by the chip's manufacturers. And, using this method, the chip count for the basic 8008 microcomputer can be cut from 22 to 64, depending on the application. (See Chapter 6 on main timing logic; Chapters 25 and 18 for design examples.)

The designer who thoroughly understands the operation of the microprocessor is also in a position to do things with the CPU that do not appear in the manufacturers' manuals. (For a simple example, note how extra input instructions can be added to the 8008, in Chapter 8.)
SEC. 1.2 MICROCOMPUTERS

1.2.1 Applications Ultimately the speed and versatility of coming generations of microprocessors will allow designers to build computers which will rival today's best minicomputers. Much of the current excitement in the engineering community about microprocessors seems to be focused on this prospect.

However, many applications exist already for the microcomputers that can be built with today's microprocessors. The products in which microcomputers are used are not necessarily computers themselves; the word computer would not even occur to the average person using many of them. They include a wide variety of digital electronic devices, from automatic vehicles, to computer peripherals, to industrial controls, to communications devices--OEM equipment marketing for as little as $500.00 or less.


Sometimes a microcomputer is employed because this technology has made the use of digital electronics commercially feasible for the first time. In many products, the microcomputer displaces a random logic design because it is more cost-effective--either in simple cost per unit, or because of the design flexibility afforded the manufacturer by the programmable microcomputer.

1.2.2 What a Microcomputer Does A microcomputer is a programmable logic unit based on a microprocessor.

As the name implies, the basic principles of computer technology are used in the microcomputer. The electronic circuitry is configured to perform logical operations on discrete blocks of binary-encoded data (e.g., bits byte, in the 8080/8008). Each operation takes place sequentially at a set speed, and is under control of binary-encoded instructions. The versatility of the microcomputer depends on part on the microprocessor's instruction set, which generally provides basic logical operations (AND, OR, EXCLUSIVE OR, etc.) and arithmetic operations (ADD, SUBTRACT, etc.). Also provided are instructions which receive (INPUT) data and transmit (OUTPUT) data to and from associated equipment.
SEC. 1.4 NOTES ON THE 8008, 8080, AND MOS MEMORIES

The technology used in the manufacture of the 8008 microprocessor is a fourteen-volt P-channel process. That is, the device requires +5 and -5 volt power supplies, for a total potential of 10 volts. This process is also used to produce the 1101 (256 x 1 RAM) and the 1103 (1/2 128 x 1 Dynamic RAM). A similar process is used in the production of the 1700A (256 x 8 PROM).

Microprocessors are produced in relatively low volumes, at least in comparison to the semiconductor memories which make up the bread-and-butter sales for these manufacturers. The 1103 128 x 1 dynamic RAM was the first semiconductor memory to begin successful large-scale competition with core memories, for application in computer mainframes. But since the same 14-volt P-channel process is used to manufacture both these memory devices and the 8008, the extensive efforts to improve the process have resulted in improvements in the 8008. Many standard 8008s test to 8008-1 speed specifications (60% faster). Though you must buy an 8008-1 to be sure you get one, it is not surprising for an ordinary 8008 to beat the 8008-1's 2.5-microsecond cycle time.

Similar comments apply to the 8080 and its 17-volt N-channel process.

For example: the Intel 2104 high-speed 4K RAM uses the same process as the 8008, requiring power supplies of +12, +5, and -5 volts. With competition to capture the 4K RAM market very tight, it is to be expected that the 8080 microprocessor will benefit from further improvements to the 17-volt N-channel process.

For every microprocessor made and sold, a number of memory chips are bought in addition. Even in a small microcomputer, it is typical for there to be a greater dollar investment in memory than in the CPU chip. From the manufacturer's standpoint, moreover, the microprocessor is a relatively expensive device to develop. A large amount of his cost is tied up in Initial Development (especially if he originated the device), in documentation, and in software support (if it is offered). Often, for competitive reasons, there is a great deal of price pressure on the CPU. The result: microcomputer component "packaging." Production-volume quantities from microprocessor manufacturers often include not only the CPU, but various peripheral logic devices (clock generators, bus drivers, etc.), plus an array of solid state memories (ROM and RAM). The current price of a given microprocessor, in large quantities, is often a well-kept secret, varying from transaction to transaction, depending partly on the amount of adjunct circuitry purchased at the same time.

SEC. 3.1 BRIEF INTRODUCTION TO MICROCOMPUTERS

In the past, only the simplest logic design techniques could be used in small and inexpensive electronic systems. The technology thereby many transistors can be included in a single package has made it feasible for the very sophisticated designs in low-cost products. The microprocessor provides the most advanced example of this trend. The paragraphs that follow trace the evolution in logic design that led to the microcomputer.

2.1.1 Binary Logic

Engineers discovered long ago that complex machines may readily be designed through the use of binary logic. Perhaps the best example is the telephone switching system, making use of electro-mechanical switches (relays). Each contact may be in one of two states, open or closed. These states are generally represented as Logic One (1), or true, and as Logic Zero (0), or False.

Switches have long since been replaced with transistors, and voltage levels represent the logical states. Since only positive logic is used in this book, a logic one always represents a higher (more positive) voltage, and a logic zero represents a lower (more negative) voltage.

The reader is presumably familiar with these matters in general, and with transistor-transistor logic (TTL) in particular. For the novice, a good place to start is with the TTL Applications Handbook (August, 1973, Fairchild Semiconductor, 444 Ellie St., Mountain View, CA 94043.)

2.1.2 Combinational Circuits

A circuit which contains only gates, and no latching elements, belongs to that class called combinational circuits. Figure 2.1.2 shows a block containing only combinational logic, with 8 inputs and 8 outputs. For any given set of inputs, there is a unique set of outputs, which does not depend on the previous history of the inputs. (This overlooks the propagation delay of the gates, which is usually a negligible short portion of the history being considered.)

Logic equations, Boolean algebra, truth tables, and Karnaugh maps are useful tools in analyzing combinational logic.
SEC. 2.1 BRIEF INTRODUCTION TO MICROCOMPUTERS (cont'd)

2.1.3 Sequential Circuits Combinational circuits are often combined with latching elements such as flip-flops, registers, latches, one-shots (master/slave multivibrators), etc. A circuit designed with latching elements does not necessarily have a unique set of outputs for a given set of inputs. In general, the outputs depend not only on the current set of inputs, but also on the sequence of inputs received in the past. For this reason such designs are called sequential machines.

Simple sequential circuits often consist of a number of gates and flip-flops arranged in a manner which is specifically related to the structure of the inputs and outputs required. Such circuits are often referred to as random logic. Figure 2.1.2 shows the block diagram of a random logic sequential circuit with \( m \) inputs and \( n \) outputs.

![Fig. 2.1.2—Random Logic Sequential Circuit](image)

As mentioned above, the advent of large scale integration (LSI) has made more compact logic circuits feasible. For very high-volume applications, a random logic design of moderate complexity may be put onto a single integrated circuit. For small-volume applications, this approach is not cost effective. The advantages of LSI may still be obtained.

---

SEC. 2.1 BRIEF INTRODUCTION TO MICROCOMPUTERS (cont'd)

however, using structured sequential circuits. Figure 2.1.3 shows a structured circuit where the memory elements are separated from the combinational logic elements. This allows the integrated circuit manufacturers to design general-purpose logic arrays on some chips (integrated circuits) and general-purpose memory elements on other chips. This means that the same chips may be used by many different customers for totally different logic designs. In this way the low cost advantage of volume production may be realized.

In addition, structured sequential circuits are often more easily understood and modifications or corrections are more easily implemented. Figure 2.1.3 shows a structured sequential circuit where the combinational logic is implemented with a read-only memory (ROM) or programmable logic array (PLA), and the latching memory elements are implemented with chips containing multiple flip-flops with a common clock. Another set of flip-flops for the purpose of synchronizing the inputs, and a two-phase clock, complete the necessary elements of a simple structured sequential circuit.

Whereas this circuit is excellent for a medium-complexity application, there are more complex problems where the size of the combinational logic required becomes unwieldy. In addition the number of memory elements may not be adequate to perform larger tasks.

![Fig. 2.1.3—Simple Structured Sequential Circuit](image)

2.1.4 Computers More complex sequential circuits are often called computers. The structure shown in Figure 2.1.4 is a simplified representation of a computer. Note that, near the center of the diagram, the computer contains an array of combinational logic. More importantly, the overall structure is that of a sequential machine—since memory elements are inserted in a digital feedback path between the output and input of the combinational logic.
In Figure 2.1.4, input data is brought into the system at the left. A multiplexer (MUX) switches between several sources of input data, including the memory array (ROM and RAM) which is part of the system. After being synchronized with the system's time frame, the input data passes to an address (A) register, and thence to an array of combinational logic which is used to decode the information and produce various strobe, enable, and other control signals. These signals, in turn, are used to route input data to other registers and to the arithmetic logic unit (ALU).

The ALU can perform arithmetic operations like addition and subtraction; logical operations like AND, OR, and NOT; and, in complex units, other operations like multiply and divide.

The control signals may also be used to pass data to the outside world through an output register.

The key to the computer is its use of memory. The computer carries out one operation at a time, under control of a program which is stored in memory. After carrying out each operation, the computer fetches another instruction from memory, and then carries out the new operation which is designated by the new instruction. Thus, referring to Figure 2.1.4, the sequence might occur as follows:

1. Old operation is finished.
2. Get ready to fetch a new instruction.
3. Enable the PC Register, which addresses the location in memory where the next instruction is stored.
4. Get ready for the next instruction by incrementing the PC register.
5. Enable memory.
6. Read the new instruction by switching the input multiplexer to the memory bus.
7. Decode the instruction to see what it wants.
8. Execute the instruction, i.e., carry out the designated operation.

Example: an Input 7 instruction. Switch the input multiplexer to input port number 7; create the required strobe signals, and pass this information along for temporary storage in the A register.

10. Fetch the next instruction from memory.

Etc.
SEC. 2.1 BRIEF INTRODUCTION TO MICROCOMPUTERS (cont'd)

As implied by the above example, the computer is a highly structured sequential machine which moves through its operations in a very regular manner. The PC register, or program counter, keeps track of the machine's operations.

The real power of the computer depends on instructions which do not necessarily follow the strict numerical sequence of instructions stored in memory. For example, a JUMP instruction might tell the computer to leave the current instruction number and fetch its next instruction from an entirely different address in memory. Even more powerful are conditional branch instructions, which send the computer to a new memory address only if the contents of an index register meet a specified condition. For example, a JUMP IF ZERO 010 300 instruction would cause the computer to test an index register to see if the current contents are zero. If so, the program counter is loaded with memory address 010 300, and this is where the next instruction comes from. If the register contents were not zero, the computer simply proceeds with the next instruction, in ascending numerical order.

Though Figure 2.1.1 shows the use of memory for storing instructions, for simplicity’s sake it does not show the other important function of memory—particularly random-access memory (RAM). That is, memory is used for temporary data storage by the computer in the course of complicated operations.

SEC. 2.2 8008 ARCHITECTURE

The 8008 was the first eight-bit microprocessor on a single chip. Essentially the miniature central processing unit of a small computer, the microprocessor performs the essential logical and control functions of a microcomputer. Figure 2.2.1 shows the block diagram for the 8008. Though more complex than the sequential machines shown earlier in this chapter, the basic principles are the same. And even while the 8008 is being superseded by the second-generation 8088 CPU (discussed in the next chapter), 8008 architecture provides a useful transition between these sequential machines, and more complex microprocessors.

2.8.1 Bus Structure The 8008 handles data in eight-bit bytes. Thus, most of the elements within the CPU are interconnected via an eight-bit data bus. The data bus buffer, a bidirectional bus driver, is used to drive external devices from the CPU, or (when the driving direction is reversed), to input data from external devices onto the CPU's internal data bus. (See Chapter 6 on bus structures.)
3.2.2 Instruction Decoding  The instruction register is an eight-bit temporary storage register which receives instructions from external memory devices. The instruction decoder determines what operations should be performed by the microprocessor.

3.2.3 Program Counter  The program counter stores the address in memory from which the CPU is currently fetching instructions. The counter is made up of a stack of eight registers, in eight levels. This enables the program counter to keep track of subroutines in the program. For example, a CALL instruction makes the CPU leave off processing the microcomputer's main line program (on level no. 0), and jump to a subroutine whose instructions are stored at a new location in memory. The former memory address remains on level no. 0, and the new memory address is now at level no. 1 in the stack. Thus, the 8008 permits nesting of seven subroutines in the PC stack. At the end of the subroutine, a RETURN instruction causes the CPU to return to level no. 0 in the stack. The stack pointer keeps track of which level in the stack is currently being processed.

Often it is said that a CALL instruction puts one level in the stack, and that the RETURN instruction pops the stack. Actually the stack itself does not change; i.e., the contents of the various levels do not get transferred among each other; only the stack pointer is moved.

Note that each level in the PC stack has a fourteen-bit capacity. This allows the 8008 to address 2^14, or 16 K bytes of memory (see Chapter 18). The memory address is divided into eight low-order bits (PC1) and six high-order bits (PC0).

3.2.4 Index Registers  During the processing of data, it is useful to have a scratch-pad: a small array of easily-accessible memory. The 8008 has seven general-purpose index registers, labeled A, B, C, D, E, F, and H. Transfers of data between these registers are easily accomplished in a direct manner through the 8008's instructions: i.e., LAR (load A register from the B register).

The A register, also called the accumulator, is specially designated as the register used for communications between various elements in the microcomputer. Thus, input data to the 8008 from an input device outside the 8008 ends up in the A register at the end of an input instruction. Output data, intended for an external output port, must first be loaded into the A register before an output instruction is executed. And, the arithmetic logic unit both operates on data in the A register, and stores its results in the A register.

The H and L registers are specially designated for referencing memory external to the CPU, for the purpose of reading data from memory (random-access memory, or RAM), or reading-only memory, ROM) and for writing data into memory (RAM only). (This function should be distinguished from the PCF and PCB registers, which reference memory for the purpose of fetching instructions.)

3.2.6 Arithmetic Logic Unit  The 8008's arithmetic logic unit (ALU) is capable of addition, subtraction, and several logical operations: AND, OR, EXCLUSIVE OR, comparison, and rotation. (For details, see discussion of the instruction set below.)

3.2.6 Flag Flip-Flops  Four flag flip-flops may be set by the operation of the ALU: the carry (C), parity (P), sign (S), and zero (Z) flags. They are described further in the 8008 Manual and in Chapter 17.) In turn, each flag may be tested through the use of conditional branch instructions, with subsequent program flow dependent on the state of the tested flag. For example, a Jump True Zero--JZ--instruction will cause the CPU to jump to another location only if the zero flag is true, i.e., logic one.

3.2.7 Temporary Registers  Two temporary registers provide storage of intermediate data. They are used by the CPU for internal data transfers, and are not directly accessible through program instructions (as are the index registers). Labeled the a and b registers, these temporary registers should not be confused with the index registers, A and B.

3.2.8 Timing and Control  The 8008 requires a two-phase clock, generated by external circuitry (Chapter 6). The E1 and E2 signals are used to synchronize the various stages in the 8008's internal processor operations, as described below.

SEC. 2.3 8008 TIMING SIGNALS

For complete details on 8008 processor operations, the reader is referred to the 8008 data sheet which is reprinted near the end of this book.
SEC. 2.3 8008 TIMING SIGNALS (cont’d)

8.3.1 Clock Signals  The 8008 is an example of four-phase P-channel MOS technology. It requires an external two-phase clock, connected to its $g_1$ and $g_2$ terminals. An internal flip-flop, triggered on the trailing edge of $g_2$, generates a third signal, called SYNC (or simply $y_7$). The four phases are developed as follows:

$g_1 = g_1 \cdot \text{SYNC}$

$g_2 = g_2 \cdot \text{SYNC}$

This method allows four phases to be generated with only three leads to the chip.

Since relatively fast TTL usually generates $g_1$ and $g_2$, while the slower MOS circuitry within the chip generates SYNC, more time must be allowed between $g_1$ and $g_2$ (SYNC going low) and between $g_2$ and $g_1$ (SYNC going high), than between those phases where SYNC does not change state. Hence, optimum speed is obtained by skewing the clock. That is, more time is allowed between $g_2$ and $g_1$ (rising edges) than between $g_1$ and $g_2$. (See Chapter 8 on clock generators.)

The four phases are used to synchronize data transfers within the 8008's internal bus structure (see Advanced Strobe Techniques in Chapter 8), and to synchronize the CPU with external hardware.

8.3.2 8008 States  In the course of internal processor operations, the 8008 travels through eight states, usually entering a new state at the rising edge of the SYNC signal. The 8008 develops three state signals—$s_2$, $s_1$, and $s_0$—which are made available at pins on its package for synchronizing external circuitry. The eight states are shown here:

<table>
<thead>
<tr>
<th>STATE</th>
<th>$s_2$</th>
<th>$s_1$</th>
<th>$s_0$</th>
<th>TYPICAL FUNCTION*</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Low-order memory address</td>
</tr>
<tr>
<td>T0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Same (interrupt recognized)</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>High-order memory address</td>
</tr>
<tr>
<td>WAIT</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Wait for slow memory</td>
</tr>
<tr>
<td>T3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Receive data from outside CPU, or, write into RAM</td>
</tr>
<tr>
<td>STOPPED</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Wait for interrupt</td>
</tr>
<tr>
<td>T4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Internal data transfer</td>
</tr>
<tr>
<td>T5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Internal data transfer</td>
</tr>
</tbody>
</table>

*Note: the functions shown are typical only; see Figure 2.3.8 for details.

8.3.3 Instruction Cycles  The execution of an 8008 Instruction requires one, two, or three machine cycles for completion. These include the four following types of cycles:

![Fig. 2.3.8—8008 Internal Processor Operation](image-url)
The two control bits, ox2 and ox1, are present on the data bus at T2 time. They appear as the two high-order bits, D7 and D6, on the bus during the T2 state. In most 8088 microprocessors, these bits are latched up in the CC-BPC register, external to the 8088, at T2 time, and are used to control external circuitry (see Chapter 6).

The 8088 does not pass through every state, or every machine cycle, during the execution of every instruction. Flow-charts in the 8088 Manual show these transitions which occur. The sections below describe the functions of the four machine cycles.

Figure 2.3.3 is a chart of 8088 internal processor operations. Perhaps the most important single page in the manufacturers’ manuals, this chart is a basic prerequisite for the 8088 designer.

SEC. 3.4 PCI INSTRUCTION CYCLE

PCI is used to fetch the first byte of an instruction from memory. The PCI cycle is always the first cycle in any instruction, and there is only one PCI cycle per instruction. (If there is more than one byte in the instruction code, PCI cycles are used to fetch the remaining bytes.) During PCI, the CPU decodes the instruction internally and (for most instructions) prepares to complete the instruction in subsequent PCC, PFR, or PCI cycles.

3.4.1 PCI-T1 The T1 state is the normal first state of a PCI cycle. During PCI-T1 time, the CPU outputs the low-order eight bits of the memory address where the first byte of the next instruction is located. That is, the 8088 outputs the internal PCI register. At the end of PCI-T1, the PCI register is incremented, in preparation for the next instruction. If an overflow occurs in the PCI register, it sets the address and is used at PCI-T1 time.
2.4.8 PCI-T5  Like T4, the T5 state is also skipped in many instructions, and, when it appears, it is used for internal data transfers. When T5 occurs, it always terminates the instruction.

2.5 PCI READ CYCLE

When a PCI (READ) cycle occurs, it is the second or third cycle of an instruction. PCI cycles are used to read memory. They occur during instructions which carry out the following functions: reading data from memory into an index register; loading an index register with a value contained in the second byte of an instruction code (load immediate instructions); jumping to or calling an address in memory; and performing an arithmetic operation on a value stored in memory, or a value contained in the instruction code.

2.5.1 PCI-T1  The PCI cycle always begins with PCI-T1. If the PCI cycle is being used to read additional bytes of instruction, then the PCI register is outputted at PCI-T1 time. In that case the PCI register is then incremented, as with the PCI-T3 state. If the PCI cycle is being used to read data from memory (for loading into an index register or for an arithmetic operation), then the L register is outputted at PCI-T1 time. (Recall that the PCI register addresses memory for the purpose of fetching instructions, and the L register addresses memory for the purpose of storing data.)

2.5.2 PCI-T2  PCI-T2 always follows PCI-T1 as the second state in a memory read cycle. If the PCI cycle is reading additional bytes of an instruction, then PCI appears on the data bus at PCI-T2 time; if the PCI register overflowed after incrementing, at PCI-T1 time, the PCI register is incremented at the end of PCI-T2. When the PCI cycle is reading data from memory, the six high-order bits of the H register go out on the data bus at PCI-T2 time. The high-order bits are the control bits, co1 and co2, and are 10, designating a PCI cycle.

2.5.3 WAIT  The READY line may be used to make the CPU enter the WAIT state during a PCI cycle, when reading from slow memory.

2.5.4 PCI-T3  PCI-T3 always the third state of a PCI cycle (unless a WAIT state intercedes). If the PCI cycle is the second cycle in the instruction, then data is read into the internal b register. If the PCI is the third cycle, then data is read into the internal c register.

2.6.6 PCI-T4  PCI-T4 takes place only in some PCI cycles. When there is a T4 state, and the PCI is the third cycle of an instruction, it is used to transfer a register to the PCI register on the 8008 chip. If the instruction is a call, the PC stack pointer is pushed at the beginning of T4 time. During an arithmetic operation, PCI-T4 provides a pause to the ALU which allows it to perform its operation.

2.6.7 PCI-T5  PCI-T5 always follows PCI-T4. During arithmetic operations, the flag flip-flops are set according to the result in the ALU, at the same time that the results are transferred to the A register (accumulator). Other internal data transfers take place during other PCI-T5 states.

2.6.8 PCI COMAND CYCLE

PCI cycles are used during input and output instructions. When it occurs, the PCI cycle is the second cycle of a two-cycle instruction. See Chapter 7 for an extended discussion of PCI cycles.

2.6.1 PCI-T1  The contents of the accumulator (A register) appear on the data bus at PCI-T1 time. Note that the PCI register is not referenced and is not incremented.

2.6.2 PCI-T2  The contents of the internal b register appear on the bus. This contains the binary code for the input/output instruction being executed. The two high-order bits, D7 and D6, are the two cycle control bits, co1 and co2, which read 01, indicating a PCI cycle is being executed. Note that all the binary codes for all the I/O instructions begin with this 01 combination.

2.6.3 WAIT  If the READY line has been brought low, the CPU will enter the WAIT state; the bus floats until the WAIT state ends.

2.6.4 PCI-T3  During input instructions, the CPU data bus receives information from an input port external to the CPU at PCI-T3 time. During output instructions, PCI-T3 is present as an idle state only; the CPU will check the READY line before completing the instruction. PCI-T3 time is used by circuitry external to the CPU to complete external data transfers into an output port.
### SEC. 2-8 INSTRUCTION SET (cont'd)

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>DEC.</th>
<th>BINARY</th>
<th>BINARY</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>0000</td>
<td>0000</td>
<td>LYT</td>
</tr>
<tr>
<td>001</td>
<td>000</td>
<td>0000</td>
<td>0000</td>
<td>RLC</td>
</tr>
<tr>
<td>010</td>
<td>000</td>
<td>0000</td>
<td>0000</td>
<td>LDCZ</td>
</tr>
<tr>
<td>011</td>
<td>000</td>
<td>0000</td>
<td>0000</td>
<td>ADDC</td>
</tr>
<tr>
<td>100</td>
<td>000</td>
<td>0000</td>
<td>0000</td>
<td>RST 000</td>
</tr>
<tr>
<td>101</td>
<td>000</td>
<td>0000</td>
<td>0000</td>
<td>LAT</td>
</tr>
<tr>
<td>110</td>
<td>000</td>
<td>0000</td>
<td>0000</td>
<td>RST 010</td>
</tr>
<tr>
<td>111</td>
<td>000</td>
<td>0000</td>
<td>0000</td>
<td>DCR</td>
</tr>
<tr>
<td>000</td>
<td>001</td>
<td>0000</td>
<td>0000</td>
<td>ND</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>0000</td>
<td>0000</td>
<td>ADF</td>
</tr>
<tr>
<td>010</td>
<td>001</td>
<td>0000</td>
<td>0000</td>
<td>RST 015</td>
</tr>
<tr>
<td>011</td>
<td>001</td>
<td>0000</td>
<td>0000</td>
<td>LDCZ</td>
</tr>
<tr>
<td>100</td>
<td>001</td>
<td>0000</td>
<td>0000</td>
<td>RST 020</td>
</tr>
<tr>
<td>101</td>
<td>001</td>
<td>0000</td>
<td>0000</td>
<td>LAT</td>
</tr>
<tr>
<td>110</td>
<td>001</td>
<td>0000</td>
<td>0000</td>
<td>RST 025</td>
</tr>
<tr>
<td>111</td>
<td>001</td>
<td>0000</td>
<td>0000</td>
<td>DCR</td>
</tr>
<tr>
<td>000</td>
<td>010</td>
<td>0000</td>
<td>0000</td>
<td>ND</td>
</tr>
<tr>
<td>001</td>
<td>010</td>
<td>0000</td>
<td>0000</td>
<td>ADF</td>
</tr>
<tr>
<td>010</td>
<td>010</td>
<td>0000</td>
<td>0000</td>
<td>RST 030</td>
</tr>
<tr>
<td>011</td>
<td>010</td>
<td>0000</td>
<td>0000</td>
<td>LDCZ</td>
</tr>
<tr>
<td>100</td>
<td>010</td>
<td>0000</td>
<td>0000</td>
<td>RST 035</td>
</tr>
<tr>
<td>101</td>
<td>010</td>
<td>0000</td>
<td>0000</td>
<td>LAT</td>
</tr>
<tr>
<td>110</td>
<td>010</td>
<td>0000</td>
<td>0000</td>
<td>RST 040</td>
</tr>
<tr>
<td>111</td>
<td>010</td>
<td>0000</td>
<td>0000</td>
<td>DCR</td>
</tr>
<tr>
<td>000</td>
<td>011</td>
<td>0000</td>
<td>0000</td>
<td>ND</td>
</tr>
<tr>
<td>001</td>
<td>011</td>
<td>0000</td>
<td>0000</td>
<td>ADF</td>
</tr>
<tr>
<td>010</td>
<td>011</td>
<td>0000</td>
<td>0000</td>
<td>RST 045</td>
</tr>
<tr>
<td>011</td>
<td>011</td>
<td>0000</td>
<td>0000</td>
<td>LAT</td>
</tr>
<tr>
<td>100</td>
<td>011</td>
<td>0000</td>
<td>0000</td>
<td>RST 050</td>
</tr>
<tr>
<td>101</td>
<td>011</td>
<td>0000</td>
<td>0000</td>
<td>LAT</td>
</tr>
<tr>
<td>110</td>
<td>011</td>
<td>0000</td>
<td>0000</td>
<td>RST 055</td>
</tr>
<tr>
<td>111</td>
<td>011</td>
<td>0000</td>
<td>0000</td>
<td>LAT</td>
</tr>
</tbody>
</table>

### 2.8.4 PCW-T2

The 8008 instruction set is explained in the manufacturers' manuals. For the convenience of the reader, the 268 eight-bit instruction codes are set forth in numerical order in Figure 2.8.1.

*Fig. 2.8.1—The 8008 Instructions in Numerical Order (Page 1)*
SEC. 2.8 INSTRUCTION SET (cont’d)

Octal | Binary | Hexadecimal | Description
------|--------|-------------|-----------------
200  | 1000000 | 0 | Add A register to A register
201  | 1000001 | 1 | Add A register to B register
202  | 1000010 | 2 | Add A register to C register
203  | 1000011 | 3 | Add A register to D register
204  | 1000100 | 4 | Add B register to A register
205  | 1000101 | 5 | Add B register to B register
206  | 1000110 | 6 | Add B register to C register
207  | 1000111 | 7 | Add B register to D register
208  | 1001000 | 8 | Add C register to A register
209  | 1001001 | 9 | Add C register to B register
210  | 1001010 | A | Add C register to C register
211  | 1001011 | B | Add C register to D register
212  | 1001100 | C | Add D register to A register
213  | 1001101 | D | Add D register to B register
214  | 1001110 | E | Add D register to C register
215  | 1001111 | F | Add D register to D register
216  | 1010000 | 10 | Subtract A register from A register
217  | 1010001 | 11 | Subtract A register from B register
218  | 1010010 | 12 | Subtract A register from C register
219  | 1010011 | 13 | Subtract A register from D register
220  | 1010100 | 14 | Subtract B register from A register
221  | 1010101 | 15 | Subtract B register from B register
222  | 1010110 | 16 | Subtract B register from C register
223  | 1010111 | 17 | Subtract B register from D register
224  | 1011000 | 18 | Subtract C register from A register
225  | 1011001 | 19 | Subtract C register from B register
226  | 1011010 | 1A | Subtract C register from C register
227  | 1011011 | 1B | Subtract C register from D register
228  | 1011100 | 1C | Subtract D register from A register
229  | 1011101 | 1D | Subtract D register from B register
230  | 1011110 | 1E | Subtract D register from C register
231  | 1011111 | 1F | Subtract D register from D register
232  | 1100000 | 20 | Load register from memory
233  | 1100001 | 21 | Load A register from memory
234  | 1100010 | 22 | Load B register from memory
235  | 1100011 | 23 | Load C register from memory
236  | 1100100 | 24 | Load D register from memory
237  | 1100101 | 25 | Load register from A register
238  | 1100110 | 26 | Load register from B register
239  | 1100111 | 27 | Load register from C register
240  | 1101000 | 28 | Load register from D register
241  | 1101001 | 29 | Load memory from memory
242  | 1101010 | 2A | Load memory from A register
243  | 1101011 | 2B | Load memory from B register
244  | 1101100 | 2C | Load memory from C register
245  | 1101101 | 2D | Load memory from D register
246  | 1101110 | 2E | Load memory from memory
247  | 1101111 | 2F | Load memory from A register
250  | 1110000 | 30 | Compare register to register
251  | 1110001 | 31 | Compare register to memory
252  | 1110010 | 32 | Compare memory to register
253  | 1110011 | 33 | Compare memory to memory
254  | 1110100 | 34 | Compare register to memory
255  | 1110101 | 35 | Compare memory to memory

Note that in Figure 2.8.1, six instructions are labeled undefined, since the 8008 manufacturers do not specify what these instruction codes do. However, the 808 user may consider these NOP (pronounced "no-op") or no-operation instructions. They can be used as time-fillers during a program. The instructions 092, 093, 094, 095, 096, and 097 operate as follows. Since the 8008 does not decode the instruction, it cannot perform any data manipulations or skip over any states. The CPU goes through all five states, T1 through T5, of a single PCI cycle. T5 terminates the instruction, as it always does. During T1 and T5 times, the CPU decrements its data bus with the octal number 777. This implies that no internal register is either listening or talking at this time.

Note also that several instructions have multiple binary codes. Chapter 11 shows how these may be used to add extra instructions to the 8008 vocabulary. Chapter 16 puts some of these to practical use.

SEC. 2.9 A SIMPLE PROGRAM

In order to provide a feeling of how a microprocessor works, this section reviews how a short program gets executed.

PROGRAM TO SWAP B AND C REGISTERS

LBA LOAD A REGISTER FROM B REGISTER
LBC LOAD REGISTER FROM C REGISTER
LCA LOAD C REGISTER FROM A REGISTER

Fig. 2.8.1 Swap B and C Using A Register

In this example, before execution of the program, the A register contained the number 123, the B register contained 345, and the C register contained 345. The execution of the program is illustrated in the chart in Figure 2.9.2.

![Chart Showing Transfer of Registers](chart.png)

**Table:**

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>234</td>
<td>345</td>
<td>ORIGINAL STATE</td>
<td></td>
</tr>
<tr>
<td>LBA</td>
<td>234</td>
<td>345</td>
<td>LOAD A FROM B</td>
<td></td>
</tr>
<tr>
<td>LBC</td>
<td>234</td>
<td>345</td>
<td>LOAD B FROM C</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>234</td>
<td>345</td>
<td>LOAD C FROM A</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>345</td>
<td>234</td>
<td>FINAL STATE</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.8.1**—The 8008 Instructions in Numerical Order (Page 2)
After the execution of these three instructions, the contents of the B and C registers are reversed from what they were originally. In the process of swapping, the contents of the A register were destroyed. Of course, it was not the object of the program to change (or save) the A register. The fact that a program destroys the contents of a particular register must be remembered by the programmer. Many programmers insert comments at the beginning of each program segment or subroutine telling which registers are needed as information by the program, which registers contain the results of the program's execution, and which registers are used or destroyed during the execution of the program.

Chapter 22 contains some Software Tricks which further illustrate how 8008 programs work.

The 8080 was developed by Intel Corporation as an update of their first-generation microprocessor, the 808. The 8080 has a great many advantages over the 808; there are also a few minor disadvantages. This chapter discusses the main features of the 8080, with comparisons to the 808.

SEC. 2.3 8080 HARDWARE

The 8080 is a much faster CPU than the 808. The extra speed is attained partially by using the 17-volt N-channel MOS process. The 8080 requires three power supply voltages: +12, +5, and -5 volts. (The 8008 requires +5 and -9 volts.) The +12 volt requirement is also for the external two-phase clock generator. Though the large voltage swing required cannot be supplied by ordinary TTL components—and generating the asymmetrical clock would take several TTL ICs—the problem is simplified by the availability of the type 8224 clock generator IC. Many computers need +12 V supplies already, for such accessories as EIA interfaces, analog circuitry, and the like. Further, +5 or -9 V supplies are needed for many memory devices (RAM and PROM) commonly used in microcomputers; these voltages may easily be dropped down from the +12 V supply.

The standard 8080 clock frequency is 2.0 MHz, as compared to the 500 KHz clock used with the standard 8008 (or the 800 Hz frequency of the 808-1).

Another reason for the increased speed of the 8080 is that a full 16-bit memory address is available at separate address terminals on the 8080's 40-pin package. (The 18-pin 8088 requires its data bus to carry the low-order memory address, the high-order address, and a data word in successive cycles.) Thus, while an 8008 takes 20 us for a simple instruction (12.5 us, 808-1), the 8088 takes 2.0 us for a simple instruction. (In either case, a complex instruction prolongs the cycle.)

Since 16 address bits are used for memory addressing, the 8080 can work directly with up to 64 K bytes of memory (65,536 8-bit data words). (The 8008 addresses 16 K.) Because of thermodynamic considerations, a 4-volt N-channel MOS processor inherently has the potential for higher speeds than a 14-volt P-channel processor (like the 8080) or a 5-volt N-channel CPU (such as the 6800). 8080s with instruction cycle times as fast as 1.0 us were available as this book went to press.

Improvements in the 17-volt MOS manufacturing process have resulted in benefits other than higher speed. For example, the size of the silicon die on which the 8080 is etched has been reduced significantly.
Comparing die sizes among three 8080 manufacturers, one is 230 by 210 mils; the second is 370 by 137 mils; or 33,490 square mils; the third is 131 by 169 mils, or 22,189 square mils. Everything else being equal, these reductions in die size mean that more 8080s can be produced from each silicon wafer—which leads to a reduction in cost. Prices have fallen far since the $400 single-quantity price tag for the 8080 was introduced. One advertisement has placed the price in lots of one million at $4.00. Prices in more moderate quantities are not that low, but at any reasonable price the implication is clear: a powerful microcomputer can now be assembled at a cost that would have seemed incredibly low only a few years ago.

The 8080 is available from multiple suppliers, guaranteeing its acceptance as a standard part. These include Intel Corporation (3865 Branding Ave., Santa Clara, CA 95051); Texas Instruments (P.O. Box 5012, Dallas, TX 75222); Advanced Micro Devices (501 Thompson Place, Sunnyvale, CA 94086); and Microcomputers (Five Militia Drive, Lexington, MA 02173); and National Semiconductor (2900 Semiconductor Dr., Santa Clara, CA 95051).

SEC. 3.3 8080 SOFTWARE

The 8080 does not include a program counter (PC) stack. Instead, the stack pointer (SP), which points to the PC stack level from which instructions are currently being fetched, is a full 16 bits wide. Any area in the 8080's 64K memory segment can be used for stack operations. A number of instructions have been added to the 8080 which permit direct manipulation of the stack.

The 8080 is much more limited with regard to stack operations. Addressed directly by a three-bit stack pointer, the 8080 PC stack is not externally addressable. This limits 8080 programs subroutine nesting, a major disadvantage in long and complicated programs, and in interrupt handling.

Two obvious disadvantages accrue from the 8080's external stack. The first is that the minimal 8080 system must have at least some RAM in it—which adds to the cost of small industrial controllers. The second is that the stack pointer must be initialized to a valid RAM address as programing begins. (The 8080 does not require any special attention to its stack pointer, when being initialized—which occurs through an initial interrupt sequence.) Once again, however, these disadvantages are outweighed by the unlimited subroutine nesting and by the availability of PUSH and POP instructions for storing and retrieving registers other than the PC.

A program written for the 8080 can be translated for the 8080 with little trouble. The internal microcoding of the 8080 dictated a rearrangement of opcodes, and the binary machine codes for most 8080 instructions have been changed. In most cases, a one-to-one substitution can be made. There are two notable exceptions. First, since the SP is entirely under program CPU control in the 8080, the stack pointer must be initialized to a valid address in RAM when the machine is started so that the PC may be pushed and popped when CALL and RETURN instructions are executed, as discussed above. Second, 8080 IN and OUT instructions take two bytes, rather than one as they did in the 8080.

The corresponding advantage: an increase in the number of addressable I/O ports.

Figure 3.3.1 shows a list of 8080 instructions, condensed from a listing frequently seen in manufacturer's publications. Though the order in which the instructions appear may seem arbitrary, there is an organizing principle. Namely, the instructions which appear in the first two columns are all found in the 8080 microprocessor. The opcodes have been changed, as well as the mnemonics.

The 8080 IN and OUT instructions, marked with an asterisk, are similar to the 8080 equivalents, except (as noted above) they are two bytes in length.

The instructions in the third and fourth columns of Fig. 3.3.1, starting with LXI B, are those which have been added to the 8080 instruction set.

Note that with the 8080, the program is permitted to address certain pairs of registers—not only the usual address registers, the H and the L, but the B and C together, and the D and E together. Direct loading and storing of the HL register pair at any memory location is also allowed. Together with the double-precision add (DAD) and increment/decrement instructions, these almost make the HL register pair into a 16-bit accumulator (while the A register is certainly an 8-bit accumulator).

Also added are instructions to load immediate the register pairs, and the BC and DE register pairs may be used as addresses for loading and storing the A register.

The 8080 can increment or decrement either the A register, or memory (the data word in memory addressed by the HL register pair) — both of which instruction types were not provided with the 8080.
SEC. 3.3 8080 SOFTWARE (cont'd)

Fig. 3.3.1—8080 Instructions.

Fig. 3.3.2 shows the 8080 instruction set by alphabetical order. The format is designed to aid the 8080 user in learning the 8080 instruction code. As stated in the note on this chart, the 8080 is microcoded in such a manner that its instruction set is more easily memorized when presented in octal (rather than hex) format. Nevertheless, hex is frequently used.

Fig. 3.3.3 presents the 8080 instruction set graphically. The organization of this chart illustrates how various memory-referencing instructions work, and should be useful to the 8080 user just becoming familiar with its instruction set.

A detailed description of each 8080 instruction is presented in the 8080 data sheet, reprinted near the end of this book.
In order to give the clearest possible explanation of a practical 8080 circuit, we have reprinted the data sheet for the Model 471 8080 CPU board near the end of this book. This data sheet includes a schematic diagram with a full description of operations, including such adjuncts as memory-mapped 1/0 addressing and interrupt reset and hold circuitry.

This section presents important technical matters concerning 8080 design generally. Some are to clarify points not made clearly in existing 8080 applications literature; other reflect design experience. We recommend that the reader start with the chapter on the 471 CPU and then familiarize himself with the 8080 data sheet before reading the following discussions.

3.4.1 Memory Access Time

The designer unused to memory system design may have difficulty in divining from the 8080 and memory data sheets whether a given chip meets the 8080 requirements.

An 8080 microcomputer addresses memory with its sixteen-bit address bus; data transfers take place via the 8-bit data bus. The memory address must be valid before the data transfer takes place, so that circuitry external to the CPU can decode the address and provide suitable enabling signals to the memory device that has been selected. The memory chip cannot in instantaneous make the selected data word available, moreover; there is always an appreciable delay between the time the chip is enabled and the time that it can reliably be read from or written into.

The chart below, Fig. 3.4.1, relates 8080 speed to memory speed requirements. The figures are based on an 8080 system making use of the 8224 clock generator.

<table>
<thead>
<tr>
<th>CPU</th>
<th>Clock Cycle</th>
<th>Read</th>
<th>Write</th>
<th>Write</th>
<th>Read</th>
<th>Write</th>
<th>Write</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>8080A</td>
<td>500 ns</td>
<td>610 ns</td>
<td>720 ns</td>
<td>500 ns</td>
<td>500 ns</td>
<td>500 ns</td>
<td>500 ns</td>
<td>500 ns</td>
</tr>
<tr>
<td>8080A-2</td>
<td>375 ns</td>
<td>405 ns</td>
<td>465 ns</td>
<td>375 ns</td>
<td>375 ns</td>
<td>375 ns</td>
<td>375 ns</td>
<td>375 ns</td>
</tr>
<tr>
<td>8080A-1F</td>
<td>325 ns</td>
<td>400 ns</td>
<td>400 ns</td>
<td>325 ns</td>
<td>325 ns</td>
<td>325 ns</td>
<td>325 ns</td>
<td>325 ns</td>
</tr>
<tr>
<td>8080A-2F</td>
<td>250 ns</td>
<td>225 ns</td>
<td>285 ns</td>
<td>250 ns</td>
<td>250 ns</td>
<td>250 ns</td>
<td>250 ns</td>
<td>250 ns</td>
</tr>
</tbody>
</table>

*See text*
The values shown in Figure 3.3.1 are based on data sheets for the 8080 variations shown and on the 8224 clock driver. Note that the 8080 versions marked with an asterisk require an oscillator speed greater than 27 MHz, the speed limit of early Intel 8224 clock generators; a premium device is needed. The read and write access figures shown are based on ideal calculations, based on 8080 and 8224 characteristics alone—plus 40 ns to allow for system delays. The paragraphs that follow support these calculations.

The 8080 accommodates memory timing characteristics with the following timing sequences. Within a machine cycle which addresses memory, there are three states, T1-T3, each lasting one clock period. As R5 goes high during T1, a memory address is clocked onto the 8080 address bus. The time it takes for this address to settle on the bus is (address output delay from 8274), 12. At T2—two clock cycles later—data transfers take place. The interval between T1 and T3 is provided principally so that memory access is assured before a data transfer takes place.

When the 8080 is reading from memory, input data must be available on the data bus before 8274 T3, when the 8080 latches up this data internally. It must be stable prior to 8274 T6 (data set-up time to 8274 T6 during D7N). The preceding factors give the formula for memory access time during 8080 read cycles: 2 to 8274 T6, where to is the clock cycle time. For an 8080 running at the standard 2.0 MHz clock frequency, this is 4 (500 ns) - 200 ns - 150 ns, or 600 ns. However, this figure is ideal; no delays have been allowed for bus drivers, memory decoders, addressing cables, or other factors. A safe practice is to specify memory with a read mode access time of 600 ns or better.

During the memory write mode, the minimum required memory access time is measured from the time that the address becomes stable, to the beginning of the memory write pulse. This memory access time, or in the 8080, is initiated by the 8080 at the leading edge of 8274 during T1. It latches a full complete cycle, 500 ns at standard speed. Thus, the 8080 again provides memory with two clock cycles—here diminished by the address set-up time, and by the amount of time that 824 preceeds 824 (t26, typically 120 ns with the 8080). The formula provided by Intel is 24 - 2 to 8274 + 266 - 140 ns, where 8274 is the clock rise time (20 ns with the 8224 clock generator). Once this figure should be reduced to allow for system delays. Safe practice is to specify a write mode access time of 500 ns or better.

Factors to be considered in a practical memory design include how the memory chip is selected and its address decoded. These matters are discussed further in Chapter 13.

3.4.2 Memory Wait Cycles A slow memory device, which does not meet 8080 speed requirements, can be synchronized with the CPU by pulling down the 8274's READY terminal shortly after it has been selected. Detected by the CPU during T2 time, this condition causes the 8080 to enter one or more T8 states, essentially extra T8 clock cycles. Effectively, the D7N enable signal (or 8274 output strobe) is lengthened by the period 8274 for each wait cycle incurred.

Summary: To produce n wait states in an 8080 system, pull down the D7N input to the 8274 clock generator for a period of approximately (n - 1/6) to 8, starting at the beginning of the ST8 system strobe.

The following paragraphs explain this formula, and the circuit for the 871 CPU board shows a practical wait circuit. (See the 871 data sheet at the end of this book, Sec. 1.13.) Assume that one wait state is required in an 8080 computer operating with a 3.0 MHz clock, in order to add 500 ns to the 8080 memory access cycle.

The 8080 samples the READY line at the falling edge of 8274, during T2 time. To provide proper data set-up time, the D7N input of the 8274 clock generator is generally used to synchronize wait requests with the rising edge of 8274. This means that to cause a wait state, the request must be received by the 8274 before the rising edge of 8274 during T2 time. The selected memory device must enter its request sometime during the period starting when it was selected by an address on the address line—at about T2-6, plus delays—and ending at T2-6, when the 8274 samples for requests. The request must be removed from the rising edge of 8274 at the last required wait cycle, to avoid causing another wait.

The one-shot requesting a wait is triggered by the leading edge of the system strobe, ST8 (ST8 for short), which falls at 81A time, just after the end of T1. This is about 1/2 to 8274 T2-6, and 1/2 to 8274 T2-4; the 8080 samples this request at the leading edge of T2-4; the 8080 sees it at the end of T2-6, entering a wait state. The one-shot requesting the request sometime after 8, begins by the next 8A; the 8274 sees the request; the 8080 realizes that the wait should end; and so it does.

In order to avoid precision timing components, the one-shot period is chosen to end midway between pulses (rising edges). For one wait state, a delay (after the first 8A) of 1/2 to 8274; for two wait states, the delay is 1/2 to 8274; and so on. This generalizes to (n - 1/2) to 8274 for n wait states. But, since ST8 fires the one-shot 1/3 to 8274 time before T2-6, this period must be added: (n - 1/2) to 8274 + 1/3 to 8274 = (n - 1/6) to 8274.
3.4.3 8080 Variations  The 8080 devices available from the different manufacturers may vary in electrical characteristics. For example, the 8090A from Advanced Micro Devices offers improved output drive current capability. Other distinctions are discussed in the 871 data sheet, Sec. 1.2. The reader should secure current specifications from the manufacturer to supplement this information, since new developments are frequent.

Worthy of further note is the 8080 Version 2 from NEC Microcomputers, which operates from a 5-volt symmetrical clock (using circuitry very similar to that discussed in Chapter 5 for the 8080).

3.4.4 Preventing 8080 Hang-up States  An active concern in microcomputer systems used as controllers is that if some unforeseen failure should cause the CPU to leave its normal program path, most likely the system will "walk out" of the bad state and continue its controlling job as well as possible.

Two design goals are important in preventing hang-up states. The first is to minimize the probability that a noise spike can derail the microprocessor. This is both a software and a hardware problem; the hardware aspect requires the engineer to design the power supply, packaging, wiring, and shielding to minimize interference from noise sources. In terms of software, the 8080 is superior to the 8080 in the respect that the 8080 has only one code for the HALT instruction--164--whereas, the 8080 has three--000, 377, and 001--the first two of which might easily result from access to a non-existent memory location.

The other objective is to provide a method whereby the microprocessor will recover its normal program mode. In interrupt systems, the 8080 presents the problem that interrupts may be disabled by software, and if the computer enters the HALT state while interrupts are disabled, then there is no way for the CPU to continue unless it is reset. A solution appears in the data sheet for the 871 CPU board (Sec. 1.7).
4-2 MICROPROCESSOR COMPARISONS

SEC. 4.1 INTRODUCTION (cont'd)

it cannot successfully exit, resuming the task of machine control. A hangup state might cause the machine to malfunction disastrously—or sit idle for hours until a skilled technician can reset it. This would not be so important with a computer (used as such), which encounters little noise—and which, in case of failure, can simply be reset by the operator, restarting the program.

A second way of distinguishing a controller from a computer is the kind of programming it is called upon to execute. The processing power, memory-addressing capability, and (sometimes) the speed of a general-purpose computer may be wasted on many small controllers—whose main function is often to control data I/O functions and to time various machine functions. When the machine is to be produced in high volumes, a simpler processor will optimize the cost of the machine without sacrificing any machine potential.

If the application requires a great deal of complex data handling, the cost of including a computer in the final product depends not just on the hardware, but on the software. Large computer users were forced to realize in the fifties and sixties that the cost of programming is often greater than the cost of the computers themselves. Many microcomputer and minicomputer users are discovering this in the seventies. Fortunately high-level languages are becoming available for use in developing software for these computers. Even so, there remain many applications needing complex software where the availability of tested programs makes minicomputers competitive. This is especially true when the production volume is expected to be relatively low; it makes no sense to develop thousands of dollars of software, to save hundreds of dollars in hardware.

SEC. 4.2 SYSTEMS DESIGN

When planning a computer application, the designer should be thinking at the system level—not just about the CPU. The microprocessor revolution has focused so much attention on the processor that this point is often lost. As a result, one often hears someone planning to interface 32 K bytes of memory, a floppy disk controller, and a 5200 video terminal—to a microcomputer or even a microprocessor. The CPU itself accounts for a very tiny part of the system cost—too small a proportion, in fact. This is analogous to purchasing a high-fidelity music system with a 3000 record changer, a $6000 tape deck, and a $500 stereo receiver—and connecting it to two 45 four-inch speakers. The computer system would be limited by the abilities of the processor—which cannot access 32 K of memory in a reasonable time, or input a megabyte of data rapidly.

4-3 MICROPROCESSOR COMPARISONS

SEC. 4.3 SYSTEMS DESIGN (cont'd)

Along the same lines, it makes little sense for a retail store owner to purchase a microcomputer or microcomputer for doing inventory for his store, plus compiling bookkeeping records at the end of the month—at least, unless he is an expert programmer, or can purchase a specially designed program tailor-made for his application. Here, software availability limits the computer, as much as the hardware; but the limitation is just as significant.

However, a large manufacturer might be able to produce 2,000 such business machines profitably. The costs of developing the software could be spread across a number of products, justifying the effort.

Microcomputers and minicomputers very definitely have their place. Many are ideally suited as controllers, buried inside a piece of business or industrial equipment that the user might not suspect has a computer in it at all. Here is where the minicomputer makes little sense. Its memory-addressing power, its powerful instruction set, and its speed are just overkill for an automobile emission control system, or inside a printer, or for hundreds of other such applications.

In addition, microcomputers are ideal for educational purposes, in teaching computer science principles at all school levels. Microcomputers offer students the first practical opportunity to own their own computers. In a classroom setting, multi-processor arrays can be assembled, allowing many students to perform problems, and to access a central computer for complex subroutines and high-speed peripherals on a time-shared basis.

SEC. 4.4 PROCESSOR COMPARISONS

This section consists of brief sketches of selected current microprocessors (including some that might more properly be termed mini-processors). By no means exhaustive, this compilation is intended only to summarize some important features, and to sketch limitations and advantages.

4.3.1 8008 and 8080 Discussing in detail in this book, the 8080 is generally used in new applications. 8008 machines are being upgraded to the 8080, where the added features are advantageous. A comparison between these processors appears in Chapter 3 above.

The advantages of the 8080 include the necessity for three power supplies: +12, +5, and -5 V. However, the current required at -5 V is very low, and a negative voltage is necessary anyway for many
The 6800 requires a two-phase clock generator with high-capacitance drive capability and fast rise and fall times. Special hybrid clock generators are available, since these requirements are not easily met with standard TTL circuitry; however, they are inherently more costly than monolithic clock drivers such as the 8224 (for the 8080). Just as with the 8080, the requirement for an external clock generator is a disadvantage—especially in small computers where low price is critical.

The 6800 uses an external general purpose stack, in RAM (similar to the 8080), providing for subroutine nesting and facilitating interrupt handling.

The 6800 is second-sourced, and is likely to remain another standard eight-bit microprocessor. Its die size is relatively large (when compared to some 8080 versions), which may put the 6800 at a cost disadvantage in high volumes, compared to the 8080.

### 6500 Series
Patterned after the 6800, the 6500 processors are especially attractive for microcomputer applications. Like the 6800, these processors address an external general purpose stack—but the stack pointer is only eight bits long, limiting the stack to 256 bytes. This is not however a practical limitation. In fact, because the stack pointer is circular—i.e., instead of underflowing into the next page in RAM, starts again at hex FF—the system cannot so easily go wild when an excessive number of stack levels are used. That is, in a microcomputer allowing unlimited subroutine nesting, the stack pointer is free to run into areas of RAM dedicated to other data storage, destroying information; or it may overflow and point to non-existent memory locations.

The 6501 is pin-compatible with the 6800, differing only slightly in hardware interfacing. Its instruction set, while similar to the 6800, is not directly compatible.

The 6502, also in a 40-pin package, features an on-chip clock oscillator. Requiring only an external capacitor (or crystal) and a resistor for main timing, this CPU effectively requires less PCB board area, and saves on component cost. Similar in features to both the 6501 and 6800, it is attractive as a 6800 replacement in new designs. A number of 650X versions are available, including CPUs in 28-pin packages where some 6501/6502 features are sacrificed. Aggressively priced, the 650X series is available both from MOS Technology and from a second-source, Syntek. A microcomputer chip with many microcomputer features, the 650X should prove very competitive.
SEC. 4.3 PROCESSOR COMPARISONS (cont'd)

For the purposes of illustration, a sketch of a 650X-based microcomputer—the M680—appears in Sec. 4.4, below.

4.3.4 2650
The 2650 would rank as an excellent millprocessor—at least, considering its instruction set, which is probably the best of the currently available eight-bit processor chips. However, its weakest link is an on-chip, non-addressable PC stack with only eight levels. This limits subroutine nesting and interrupt handling and the same fault impedes the 8080, and relegates the 2650 to the microcomputer category.

Available from Signetics, other 2650 advantages include a TTL clock; a single power supply; and completely microuncoded indexed addressing.

4.3.5 F8
Available from Fairchild and Mostek, the F8 is a well-designed microprocessor chip set. It requires 6 and 12 V power supplies, and a simple NC combination suffices to complete its on-chip clock oscillator. The F8 lacks a general-purpose stack; instead, each chip in the series (RAM, PROM, etc.) includes an on-chip program counter and one-level program stack. The F8's instruction set is quite versatile. Its disadvantage: it does not interface readily with standard general-purpose memory parts. Though memory interface chips do exist, their use in production designs would remove the advantages of the F8 over other processors, such as the 6500 series. The main F8 application: large-volume microcomputers, especially controller applications.

4.3.6 SC/MP
The SC/MP is an eight-bit microprocessor from National Semiconductor (second source, Rockwell). Designed to fill the gap between four-bit CPUs (4004, 4040) and eight-bit millprocessors, the SC/MP features a 12-bit address bus, interfacing directly to 4 K bytes of standard memory; provision for simple interrupts and DMA; on-chip serial data input port and output port; versatile memory addressing, including relative addressing instructions; and an on-chip clock oscillator, requiring only an external capacitor. Perhaps most important is its single power-supply requirement, +10 to +14 volts, permitting use of an unregulated power supply, and providing CMOS compatibility.

Disadvantages: TTL-compatible version (metal mask option) requires two supplies, +5 and +9 V. Most important, the SC/MP is incompatible with the older P-channel MOS technology, and is therefore much slower than N-channel CPUs. The simplest instruction takes five microseconds, each 2 us long—making the SC/MP about as fast as the 8080. Nevertheless, SC/MP should be attractive in industrial controllers in which speed is not critical.

4.3.7 Z-80
The Z-80 is a third-generation member of the eight-bit microprocessor family which includes the 8086 and 8080. Produced by a new corporation, Zilog, the Z-80 is compatible with the 8080 in terms of software, right down to the instruction code level. In addition, the Z-80 employs some of the 8080's unused opcodes to add a number of very useful instructions, including relative addressing and block transfers. The Z-80 incorporates such useful features as on-chip prioritized interrupt handling, and memory refresh circuitry for dynamic RAM. Furthermore, the Z-80, like the 8080, requires only a single +5 volt supply. Beginning in the last quarter of 1976, the Z-80 is available from Mostek as well as from Zilog.

The advent of the Z-80 is effectively a great advantage to the 8080 family, insofar as the Z-80 combines all the advantages of the 8080 processor with those of the 8080. As this book went to press, Martin Research was introducing a Z-80-based microcomputer at its product line— the M680.

SEC. 4.4 THE M680 MICROCOMPUTER

For the sake of illustration, this section includes a sketch of a small microcomputer based on the 6802 CPU (Fig. 4.4.1). Including only six I/Os in addition to the CPU, the M680 has one page of ROM (256 bytes), a page of PROM, an input port, and an output port. The power supply required is simply +5 volts—unless the PROM employed requires an additional supply.

Full eight-bit I/O ports could be provided by replacing the six-bit latch and driver I2O with eight-bit devices. The 74LS274 can be replaced with a 74LS273, and the 74087 with a 74LS274—all available from Advanced Micro Devices.

The M680 is an example of a small microcomputer suitable for incorporation into industrial controllers. For developing software, however, a computer with full expansion capabilities and a Monitor program is preferable. For typical examples, see the information on the modular micro series at the end of this book.
Both the 8080 and 8086 microprocessors require two-phase clock signals, which must be generated by circuitry outside the microprocessor. The 8086 requires clock voltages at standard TTL voltage levels, while the 8080 requires NM1 and NM2 signals referenced to +5 V.

The first part of this chapter is devoted not only to the generation of clock signals for the 8008, but to other main timing requirements as well. Since the 8008 does not include internal DH and DL address registers, these must be added (Sec. 5.4). 8080 clock circuitry is discussed in Sec. 5.5.

SEC. 5.1 8008 CLOCK CIRCUITS

Symmetrical clocks are the best place for the 8008 designer to start. CPU circuitry may be converted to clock circuits, for slight improvement in system speed during the late stages of circuit design if it is found advantageous.

Most symmetrical clock designs are made up of three sections:

(1) the oscillator;
(2) the counter;
(3) the decoder.

The oscillator section provides the basic signal source at a constant frequency. The counter section then divides the frequency by two to provide two related frequencies, \( f_0 \) and \( 1/2 f_0 \). The decoder section then selects the 01 and 11 states from the counter, providing the 61 and 62 signals required.

Figure 5.1.1 shows the timing diagram for a two-phase clock generator which uses a positive-logic decoder.

Figure 4.4.1--Sketch of the MIKE 6, a 650X-Based Microcomputer

Figure 6.1.1--Two Phase Clock Generator Timing Diagram
SEC. 5.1 8088 CLOCK CIRCuits (cont'd)

5.1.2 Oscillators  Two basic oscillator circuits are described here. Other oscillators which provide approximate square waves may also be used.

The first oscillator shown, in Figure 5.1.2, is very inexpensive and uses readily available components. The disadvantage to this oscillator is that it is inaccurate. Adjustment components may be added to this circuit, as shown in the figure, but the temperature drift may be significant, depending on the application.

![Figure 5.1.2 - Inexpensive CPU Oscillator: Three Inverters and Capacitor](image)

The second type of oscillator is also quite simple, but uses a crystal, which makes it more expensive. The main advantage to this circuit is its stability. (A microcomputer built into instrumentation needing an accurate frequency or time standard will need such an oscillator anyway.) Figure 5.1.3 shows the crystal oscillator schematic. The value of R must be decreased for crystals with high internal resistance. Resistor values may range between 270 and 1500 ohms. If the characteristics of the crystal are unknown, a 270-ohm resistance is generally recommended.

Sometimes a precounter may be used with a crystal oscillator, as shown in Figure 5.1.6 below. The precounter implies the use of a higher-frequency crystal, which is smaller and less expensive. For prototypes, the precounter may be eliminated. For the production version, the cost-effective approach should be used.

5.1.3 Counter Section  The simplest counter is a single flip-flop. Whether the desired flip-flop should be positive- or negative-edge-triggered depends on the type of decoder used. In order to avoid race conditions, a negative-edge-triggered flip-flop is used with a positive decoder, and a positive-edge-triggered flip-flop is used with a negative decoder. (See following section on race conditions.)

5.1.4 Decoders The two phases, $d_1$ and $d_2$, can be decoded from the counter section with only two gates. Since positive pulses are required for the 8088, either AND or NOR gates are used because their exceptional output states are logic one. (The exceptional output state is that state occurring with only one combination of inputs.) Figure 5.1.5 shows the use of AND gates for a positive decoder (exceptional input state of logic one). Figure 5.1.6 shows a negative decoder, which uses a pair of NOR gates (exceptional input state of logic zero). Notice the use of negative-edge-triggered flip-flops with positive decoders, and vice versa, as mentioned previously.

When using TTL decoders, it is advisable to add 1 k ohm pullup resistors (to +5 volts) to ensure an adequate voltage swing to drive the CPU's MOS inputs. If a CMOS decoder is used, these resistors may be eliminated.

The following should explain the cautions on avoiding race conditions in this section. A binary counter made up of flip-flops (including multi-count ripple counters like the 7499 series) exhibits certain inherent race conditions during count transitions. Take for example the simple flip-flop counter shown in Fig. 5.1.5, which goes through four states (see Figure 5.1.4):

<table>
<thead>
<tr>
<th>STATE NUMBER</th>
<th>BINARY COUNTER STATES</th>
<th>POSITIVE DECODERS STATE NAME</th>
<th>COUNT DIRECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td>RACE-PRIME STATE</td>
<td>$d_1$</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1 0</td>
<td>$d_1$, PHASE ONE</td>
<td>$d_2$, PHASE ONE</td>
</tr>
<tr>
<td>2</td>
<td>1 0 0 1</td>
<td>RACE-PRIME STATE</td>
<td>$d_1$</td>
</tr>
<tr>
<td>3</td>
<td>1 1 1 0</td>
<td>$d_1$, PHASE TWO</td>
<td>$d_2$, PHASE TWO</td>
</tr>
</tbody>
</table>

![Figure 5.1.4 - Two-Phase Clock Generator Timing Diagram](image)
SEC. 5.1 8008 CLOCK CIRCUITS (cont'd)

During the transition between state number 1 (binary output code 01) and state number 2 (binary output code 10), both bits must change state. This transition is not immediate, and in this kind of counter, the low-order bit must change before the high-order bit can change. This means the count is really 01, (00), 10 (where the 00 count is very short, just a glitch). If the decoder were looking for the 00 state, a glitch would appear at this point. Similarly, in the transition between state 3 (11) and state 0 (00), the counter passes through the 10 count. Glitches are avoided by making sure not to decode the 00 or 10 counts.

A similar situation occurs when positive-edge-triggered flip-flops are used, except that the safe states are now the even counts, and the counting direction is reversed (Figure 5.1.4, to the right). Schematic: Figure 5.1.6.

5.1.5 Monolithic Clocks Instead of the TTL-based two-phase clock circuits shown above, the designer may choose to specify a monolithic clock generator. One such is the National Semiconductor MH8803 two-phase oscillator/clock driver. A circuit diagram is shown in Figure 5.1.7. With the frequency control pin left open, as shown, the oscillator runs at 300 kHz, with a 750-nanosecond pulse width.

Figure 5.1.7—Monolithic Two-Phase Clock Generator

SEC. 5.2 ENABLE AND STROBE GENERATION

One of the functions of the main timing logic in the 8008-based microcomputer is to develop the signals necessary to activate associated input selectors, memory, and output ports at the proper times.

Chapter 2 has already covered the basic timing characteristics of the 8008. Here the hardware is developed to accommodate the 8008's requirements.
5.2.1 Eight-State Decoder  
Most of the 8008 designs previously published have used decoders to transform the S1, S2 and S3 state outputs from the microprocessor and develop eight separate state signals. The schematic for this circuit is shown in Figure 5.2.1. Usually the WAITING and STOPPED signals are then used to control lamp driver circuits which inform the user when the MPU is waiting for memory or has hit a HLT instruction. Then a large number of inverters, gates, and flip-flops are added to generate strobe and enable signals to perform the appropriate data transfers to and from the MPU at the proper times. The presence of this large and often confusing mess of TTL has probably discouraged many designers from using the 8008 microprocessor than any other single factor.

5.2.2 A Practical Strobe Generator  
A very significant savings in gates can be accomplished by using the strobe inputs of the decoder to eliminate the need for these additional gates, as shown in Figure 5.2.2.

---

![Figure 5.2.1--Catal Decoder Generates State Signals](image1)

![Figure 5.2.2--Strobed State Decoder Simplifies MPU Logic Design](image2)

The only potential objection to this technique is that the STOP and WAIT lines become a string of pulses rather than a steady level. If someone is using an oscilloscope to look for STOP or WAIT states, he will actually find it advantageous, since a string of pulses is more quickly recognized as a level, and pulses will trigger his scope sweep more readily than a DC level signal. If STOP or WAIT indicator lamps are required, the pulse signals again will work better than level signals in driving LED readouts. The designer merely calls for a smaller current-limiting resistor in series with the LED display, since it is being driven in the pulse mode.
SEC. 5.2 ENABLE AND STROBE GENERATION (cont'd)

Figure 5.2.3 shows a circuit to perform these functions.

The flip-flop is set by ST2 and reset by ST3. During the interval between these signals, while the output of the flip-flop remains high, the SYNC signal is passed through the AND gate to the output. The pulse width of T3A is approximately equal to the PHASE TWO clock period.

It may be desirable to suppress the T3A enable signal during a PCW (memory write) cycle, when the CPU is outputting onto its bus rather than inputting. In order to accomplish this, the reset of the flip-flop is connected to PCW rather than to ST3. ST3 still resets the flip-flop, now a D-type device, through its clock. Figure 5.2.5 shows this revision.

6.2.3 Generating the T3A Enable/Strobe
When the CPU is inputting data during T3 time, it needs an enable pulse which comes between the ST2 strobe and ST3 strobe. This signal anticipates the T3 state in order to give time for the information coming into the CPU to settle on the bus. This signal is called T3A. See Chapter 2 for a further description of T3A.

There is more than one way to generate the T3A enable pulse, but perhaps the most convenient one is shown in Figure 5.2.4.

Figure 5.2.4--A T3A Enable Generator to Enable CPU Input Sources
Section 5.2.4 describes the generation of RAM write pulses. Since this usually requires the generation of the PCW signal anyway, the TIA generation circuit of Figure 5.2.5 is still minimal.

It is necessary when using the circuit of 5.2.5 to load the CC2 and CCI bits on the beginning of the STSW pulse. When these bits are loaded, this is usually accomplished with the STSW signal. Figure 5.2.6 shows the timing diagram associated with the circuit of 5.2.5. The instructions being executed are an LDA, followed by the fetch portion of the next instruction (which might be any instruction). Notice how the PCW cycle suppresses TIA. The use of this circuit is more fully explained in Chapter 26.

5.2.4 Generating the "STSW" Strobe The other signal which is often (but not always) needed in an 8008 microprocessor system is STSW. This strobe is generated when the CPU is outputting information (during 73 cycle) to be stored in RAM. This output condition occurs only when a PCW cycle is being executed. Since the PCW cycle is denoted by CC2 and CCI being high, strobe generation may be as simple as a NAND gate and an inverter. Figure 5.2.7 shows two possible circuits which both fit equivalent logic equations.

\[
\text{STSW} = CC2 \cdot CCI \cdot ST3
\]

\[
\text{STSW} = CC2 \cdot CCI \cdot ST3
\]

A) METHOD No. 1

B) METHOD No. 2

Figure 5.2.7-Methods for Generating "STSW"

The limitation of this circuit is that the STSW pulse width is only as long as ST3 (the width of a PHASE TWO pulse). For an 8008-speed clock this pulse would be 500 nanoseconds wide. This pulse is quite sufficient for many RAMS. The STSW signal is called "STSW" probably because it is more easily pronounced.
6.2.5 Generating the "STWED" Strobe

For those RAMs requiring wider write pulses, the circuit in Figure 5.2.8 may be used. The normal STS pulse is generated, as in Figure 5.2.7, and a set-reset flip-flop is added to broaden the write pulse.

The flip-flop shown is an AND-OR design. (For a description, see the article "Another Way to Build a Two-Gate Flip-Flop" by the present author in Electronics, June 15, 1974, page 12A.) It is reset to a low level by the negative-going STS pulse, starting the STSE pulse. The positive-going edge of the next PHASE 1 clock pulse sets the flip-flop to a high level and terminates the strobe. For systems using a symmetrical clock, this flip-flop doubles the write pulse width. Therefore, with a clock designed for normal 8008 speed, the write pulse will be a full microsecond in width. In any case, the STWED signal always stagers around a little longer for systems with slower memory.

Figure 5.2.8--Generating "STWED," a Wider Write Pulse

Figure 5.2.9--Basic 8008 Timing with Little Peripheral Logic
SEC. 5.2 ENABLE AND STROBE GENERATION (cont'd)

5.2.6 Summary The above main timing circuits are combined in Figure 5.2.9, which shows:

(1) the clock which produces $t_1$ and $t_2$;
(2) the CPU itself;
(3) the strobed decoder;
(4) the STB generator; and
(5) the STBR generator.

Figure 5.2.10 shows the main timing logic signals used by an 8008. The relative simplicity of firing up an 8008 CPU is becoming more clear.

SEC. 5.3 MASTER RESET

Nearly every complex logical system requires some method for initialization when power is first applied. Otherwise flip-flops, latches counters, and other devices with bistable logical states will take on arbitrary and possibly undesirable values. The system might even enter an illegal state, where the machine hangs up and refuses to respond to inputs at all—a most frustrating habit, as every designer who has encountered it knows well. The solution is generally a master reset circuit which develops a reset signal for a fraction of a second when power is first applied.

5.3.1 Simple Master Reset Circuits A simple master reset circuit which uses only one or two TTL gates is shown in Figure 5.3.1. The capacitor is initially discharged, so that the 7404 output is initially high. After the capacitor charges to the 7404 logic one threshold value, the output goes low, and remains low as long as the power is on. The output signal is commonly called MR, and is used for devices (such as a 7490 decade counter) with positive-logic resets. When a negative-logic signal is needed (as to reset a 7474 flip-flop), the MR signal can be inverted using another 7404 stage, producing $\overline{MR}$.

Figure 5.3.1—(A) Simple Master Reset Circuit Using TTL; (B) CMOS Version

---

Fig. 5.2.10--Basic Main Timing Signals for an 8008 Microcomputer

Figure 5.3.1—(A) Simple Master Reset Circuit Using TTL; (B) CMOS Version
SEC. 5.3  MASTER RESET (cont'd.)

With a standard TTL gate, as shown, R should normally be not less than 1 K ohms (to protect the gate against power supply transients), nor more than 25 K ohms (to provide sufficient input current). The diode shown is optional, and forces the capacitor C to follow the power supply voltage down rapidly when power is turned off.

Note that the circuit is power-supply rise-time dependent. If the +5-volt supply tends to come up more rapidly than other supplies in the machine, consideration should be given to increasing the capacitor's value.

The circuit of Figure 5.3.1-A, using CMOS, allows the use of smaller capacitors and larger resistors. Of course, drive capabilities are also reduced (2 TTL loads with the chip shown). The circuit of Figure 5.3.2 can be used to minimize power supply rise-time dependency.

![Figure 5.3.2—Master Reset with Reduced Power Supply Dependency](image)

5.3.2 8008 Initialization  The 8008, which contains a number of internal registers and other bistable elements, has its own initialization provisions. As the power supply and system clock come on, the CPU executes a HALT instruction automatically. During the ensuing STOPPED condition, the CPU automatically clears its internal memories in sixteen clock periods. The CPU may be started by causing its INTERRUPT terminal to go to logic one.

For more information on initial interrupts with the 8008, see Chapter 16. See also the practical example in Chapter 25.

SEC. 5.4  ADDITIONAL CIRCUITRY REQUIRED FOR BASIC MICROCOMPUTER

5.4.1 Address/Data Registers  During TI time, the CPU outputs information on its data bus which is intended for temporary storage in an external register, called the DL Register.

In most instances, the DL register contains the low-order portion (eight bits) of the address in memory which is being read or written into by the CPU. When an output instruction is being executed, the data transferred to the DL register—during memory cycle two—is intended for an output port. (For clarification, the reader may study the chart on internal processor operation in the 8008 manual.)

The strobe used to load the DL register from the data bus at the proper time is $ST1$. It occurs far enough into TI time to ensure that the data which the CPU is outputting has had time to settle on the bus.

Figure 5.4.1 shows how simply the DL register may be added to the CPU. The eight-bit latch used should be a low-power device, whose input current requirements will not overload the CPU output bus drive capabilities.

![Figure 5.4.1—The DL and DH Registers Connected to the CPU Bus](image)

(A) DL Register  
(B) DH Register
SEC. 5.4 ADDITIONAL CIRCUITRY REQUIRED FOR BASIC MICROCOMPUTER (cont'd)

During T2 time, the CPU outputs the address or instruction intended for the DI register. The two high order bits (D17 and D16) are, in all cases, used to determine the type of instruction cycle being performed. Since they are control bits, rather than memory-addressing bits, D17 and D16 are usually referred to as C22 and C21 (respectively) in this book.

The remaining six low-order bits of the DI register usually contain the high-order memory address information. In the case of input or output instructions, the CPU loads the DI register with the I/O instruction itself, in its eight-bit binary code.

The strobe used to clock the data bus information into the DI register is called STP.

Figure 5.4.1(b) shows an eight-bit latch being used as the DI register. Cautions similar to those mentioned for the DI register must be observed with regard to loading down the CPU data bus.

6.4.8 Basic 8080 Microcomputer A working 8080 microcomputer requires the addition of read-only memory (ROM) for program storage, and an input/output/memorty select decoder. These matters are outside the scope of a chapter on main timing logic, and are discussed elsewhere in the book. However, we are now very close to a design for a minimally configured microcomputer. See Chapter 26 for a design example.

SEC. 5.5 8080 CLOCK GENERATION

The 8080 CPU requires a two-phase clock referenced to ±12 V. System timing requirements dictate an asymmetrical clock, where PH1 has a duty cycle of 2/3, followed immediately by PH2 with a duty cycle of 1/3, followed with a gap of 2/3. The standard 8080 clock cycle is 500 ns (2.0 MHz frequency), derived from a crystal resonator running at nine times that frequency, or 18.0 MHz. PH1 and PH2 are produced by decoding the outputs of a divide-by-nine counter.
SEC. 6.1 INTRODUCTION TO BUS STRUCTURES

6.1.1 What is a Bus? A bus consists of a number of actual wires which may be driven by more than one source. The following sections will concentrate on the types of buses most commonly applicable to microprocessors.

The wired-AND function (sometimes called the wired OR) used in DTL is a primitive example of a one-bit bus. Figure 6.1.1 shows two RTL inverters with their outputs wired together. RTL is used in this example for simplicity: the gates have passive (resistor) pullups rather than active (transistor) pullups as in TTL devices with totem-pole outputs.

![Diagram of a One-Bit Bus](image)

Figure 6.1.1--A "One-Bit Bus" Example

Referring to the figure: if either point A or point B is raised to logic one, the associated transistor will turn on and the "one-bit bus" will be actively pulled down to logic zero. The portion of the figure enclosed in the dashed rectangle is considered the bus. This orientation is important in clearly defining the difference between a bus input and a bus output.

Thus, when the RTL inverter outputs are connected to the bus, they become bus inputs. The RTL inverters themselves function here as bus drivers. Naturally the bus is not very useful until a bus receiver is connected to it. A bus receiver uses the output of the bus to drive its input. Many bus receivers may be attached to a bus.
SEC. 6.1 INTRODUCTION TO BUS STRUCTURES (cont'd)

The real value of a bus structure becomes apparent when one uses the same bus to transmit different information at different, well-defined times. Information can then be time multiplexed onto the bus. Only one bus driver can be active at one time, and bus receivers should pay attention to the information on the bus only when it pertains to them.

Figure 6.1.2 shows a bus circuit with two bus drivers and two bus receivers. The timing circuit consists of a clock driving a single flip-flop. The flip-flop alternately enables the two 7401 gates. When STROBE A is high, the information on INPUT A is impressed onto the bus, and the top-most 7400 gate is enabled. OUTPUT A then follows INPUT A logically. When the next clock pulse occurs, the flip-flop complements, which disables STROBE A and enables STROBE B. Now the information on INPUT B goes onto the bus and is recognized by OUTPUT B.

Figure 6.1.2--Two Drivers and Two Receivers on One-Bit Bus

6.1.2 Why Bother? All this effort is hardly justified with a one-bit bus; three wires are used (a bus and two strobe lines) to transport information that could have been communicated over two wires with no extra logic. However, consider an eight-bit bus: similarly configured, it would need ten wires (the eight bus lines and two strobe lines), as opposed to sixteen wires. As the system becomes more complex, the savings become more apparent. And when the designer is using large-scale integration (LSI) chips, the cost of the real estate of the silicon chip is less expensive than the extra printed circuit board copper, PC board connectors, and backplane wiring. And LSI chips intended for time-multiplexed bus use are themselves less expensive, ultimately, because fewer pins and bonds are required.

SEC. 6.2 LATCHING LOGIC

The next step is to improve the circuit of Figure 6.1.2 by adding a latch to the bus receiver. Naturally it is necessary to latch up the information at OUTPUT B only when it is valid. This may be difficult to do with the circuit in Figure 6.1.2. The problem which arises is that the information on OUTPUT A should not be latched up until ENABLE A has been on long enough for the data to propagate down the bus and settle on the output of the 7400.

Figure 6.2.1--One-Bit Bus with Enables and Latches

Figure 6.2.1--One-Bit Bus with Enables and Latches
SEC. 6.2 LATCHING LOGIC (cont'd)

In order to do this a timing signal is needed for each driver and each receiver. Figure 6.2.1 shows a circuit which not only transfers the information, but latches it in flip-flops so that it is continuously available at OUTPUT A and OUTPUT B.

The timing diagram in Figure 6.2.2 shows the sequence of events as information is transferred from each of the bus driver inputs to the appropriate bus receiver output latch without any problem. The clock must be slow enough to let the enabled data settle on the bus before the receiver strobe causes the data to be latched.

CLOCK

ENABLE A

STROBE A

ENABLE B

STROBE B

Figure 6.2.2-Timing Diagram for Figure 6.2.1

The arrows in Figure 6.2.2 indicate the points at which the output flip-flops are loaded. Note that loading occurs a significant time--one-half clock cycle--after the associated bus driver has been enabled onto the bus.

The bus-structured designs discussed up to this point have used open-collector bus drivers. Integrated circuits with open-collector outputs have only two output states: that is, the gate can either pull the bus down to logic zero, using an active circuit element (the gate's internal output transistor); or it can allow the bus to be pulled back up to logic one by a passive circuit element (an external pull-up resistor). To disable the bus driver, a signal is applied to an extra input terminal which

SEC. 6.3 THREE-STATE DEVICES

A relatively new class of logic devices is available which offers better performance in bus-structured designs than the open-collector devices described in the previous sections of this chapter. The integrated circuit with three-state outputs has both active pull-up transistors, and active pull-down transistors, in its output stage-like ordinary TTL with totem-pole outputs. To disable the outputs, an extra terminal is used, commonly termed OUTPUT DISABLE. When the outputs are disabled, the output floats; that is, the output stages neither sink nor source a significant amount of current, but allow the output voltage to be determined by another circuit element connected to the output bus. This third state is often called the high-Z or high-impedance state.

The term TRI-STATE is often used; this is the registered trademark of National Semiconductor Corporation for three-state devices.

Figure 6.3.1 shows some common symbols for three-state buffers. Circuit (c) is similar in circuit function to the 7401 open-collector gate described above.

(a)

(b)

(c)

(d)

Figure 6.3.1--Common Symbols for Three-State Buffers
SEC. 6.3 THREE-STATE DEVICES (cont'd)

A very valuable property of almost all three-state buffers is that, when disabled, not only the outputs but the inputs are in a high-Z state. If a number of three-state devices are connected to an ordinary TTL output, the load factor is defined by the maximum number of three-state devices which may be enabled at one time, not by the total number. This property is very handy in the design of large bus-structured systems.

To minimize the possible conflict between two three-state devices trying to drive the same bus line at the same time, most manufacturers have designed their devices such that the output enable time is longer than the output disable delay time. The designer should design his system timing and strobe signals to make use of these properties and avoid any output overlap between two bus drivers. Otherwise current spikes may appear. While not likely to destroy the buffers--whose outputs are generally current-limited--these spikes can easily disrupt normal system functions.

SEC. 6.4 ADVANCED STROBE TECHNIQUES

The bus circuits discussed earlier are extended further here through the use of three-state buffers. The circuit in Figure 6.4.1 is similar

![Diagram](image)

**Figure 6.4.1—Three-State Buffers Replace Open-Collector Devices—Timing Control Shown Separately**

---

SEC. 6.4 ADVANCED STROBE TECHNIQUES (cont'd)

to that in Figure 6.2.1, except that three-state buffers are used in place of open-collector gates. The timing logic is separated into a third section of the schematic. The timing circuitry could have been included either with the bus drivers, or the bus receivers.

The 747 flip-flop in the timing control section of Figure 6.4.1 may be considered a one-bit counter, and the pair of 748 gates may be considered a two-bit decoder. Note the similarity between the above schematic and the 8006 main timing circuitry in Chapter 5.

The capabilities of the timing section shown above can be expanded by using multiple-bit counters and decoders, as shown in Figure 6.4.2.

**ENABLE DECODER**

![Diagram](image)

**Figure 6.4.2—Expanding Number of Bus Drivers and Receivers with Only a Few Chips**
6.5.1 Input Bus An input bus is a bus which drives only one set of receivers, but has more than one driver. An example: the three-state data outputs of many RAMs and ROMs are connected together to the inputs of a single set of buffers. Figure 6.5.1 shows a CPU Input bus which accepts information from a ROM or one of two RAMs, all of which have three-state outputs. The decoder must provide enable signals to ensure that no more than one memory IC is driving the input bus at one time.

6.5.2 Output Bus An output bus is a bus which contains only a single set of drivers, but several sets of receivers. In the example in Figure 6.5.1, the CPU is connected to eight buffers, whose outputs constitute the CPU output bus. The RAM chips receive data to be stored from this CPU output bus. The WRITE STROBE line both enables the RAM, and strobes the data to be written onto the chip at a time when the data on the CPU output bus is stable.

6.5.3 Bi-Directional Bus A bi-directional bus employs more than one set of drivers and more than one set of receivers. Non-overlapping enable signals should be provided to the drivers, and correctly-timed strobes should be provided to each of the receivers at a time when the bus is stable.

Note that, in Figure 6.5.1, the symbol 1B is used to represent an input bus; the symbol 1B denotes an output bus; and 1B is a bi-directional bus.

6.5.4 Tri-Bus A bus structure whose hardware is designed using three-state devices is called a tri-bus in this book. Unlike the above bus designations, this term refers primarily to hardware (electrical) considerations. A tri-bus normally has several bus drivers connected to one or several bus receivers, and in otherwise unternated--ie, no pull-up resistors are used to pull the bus to ground. This distinguishes the tri-bus not only from open-collector bus structures (discussed above) but from other bus configurations with 120-ohm impedance characteristics.

For a good introduction to tri-bus circuitry, the reader is referred to the data sheets for the National 8830 series (and to Application Note 83, Data Bus and Differential Line Drivers and Receivers).

Figure 6.5.2 gives a brief summary of the types of busses defined above.

Fig. 6.5.1--Examples of different kinds of bus configuration, in a hypothetical four-bit data system.
### 6.6 Types of Bus Structure (cont'd)

<table>
<thead>
<tr>
<th># of Receivers On Bus</th>
<th># of Transmitters On Bus</th>
<th>Name of Bus</th>
<th>Type of Bus</th>
<th>Control Signals Needed</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only One</td>
<td>More Than One</td>
<td>Input Bus</td>
<td>Tri-Bus</td>
<td>Enables, Strobe</td>
<td>&quot;NB&quot; Bus; Data from memories</td>
</tr>
<tr>
<td>More Than One</td>
<td>Only One</td>
<td>Output Bus</td>
<td>Direct Bus</td>
<td>Strobes</td>
<td>&quot;TB&quot; Bus; outputs of DL register; data to RAM memories</td>
</tr>
<tr>
<td>More Than One</td>
<td>More Than One</td>
<td>Bi-Direc-</td>
<td>Tri-Bus</td>
<td>Enables, Strobes</td>
<td>&quot;BB&quot; Bus (CPU Bus, D7-D0)</td>
</tr>
</tbody>
</table>

Figure 6.6.2—Names and Types of Bus Structures

### 6.6 Bus Transceivers and Bi-Bus Drivers

#### 6.6.1 Bus Transceivers

Various interconnections of the terminals of bus receivers and bus drivers, on the same integrated circuit, produce devices which are useful in bus-structured communications. For example, a bus transceiver is formed when the output of a bus driver is connected to the input of an associated bus receiver. Figure 6.6.2 shows a bus transceiver, often used to add peripheral devices to provide three-state buffering and interface to long bus lines. The dotted line is often drawn solid in device schematics, to indicate that the drive section does not load the driver input when the driver is disabled.

#### 6.6.2 Bi-Bus Drivers

A very useful device in bus-structured designs is the bi-bus driver. We use this term to refer to the circuit of Figure 6.6.2(a), where two three-state bus drivers are connected in parallel, with complementary driving direction. The two enable terminals are shown separately in Figure 6.6.2, and part (b) shows a convenient symbol to designate the bi-bus driver. When ENABLE #1 is activated,
the tri-bus to the left drives the tri-bus to the right, and when ENABLE #2 is activated, the direction is reversed. When neither is enabled, no data transfer occurs between the two busses. The control signals are designed such that at no time are both ENABLE terminals activated.

Figure 6.6.3 shows a bi-bus driver where the two ENABLE signals are connected to the same terminal, and where one driver is enabled by an active low input. Thus the enable terminal is labeled DIRECTION CONTROL, and one or the other of the drivers is normally on at any given time. (As with other three state devices, internal circuitry assures that each driver can be disabled more quickly than it can be enabled, so as to avoid driver overlap.)

Figure 6.6.4—Bi-Bus Drives Save Chip Area by Handling Interface with External Bus

SEC. 6.7 THE 8008 DATA BUS

6.7.1 Bi-Bus Drivers in the 8008 The 8008 microprocessor employs what amount to single-enable bi-bus drivers at its eight data bus pins. These bi-bus drivers connect the bi-directional bus internal to the CPU chip, D7 through D0, with the bi-directional data bus external to the CPU, called in this book BB7 through BB0. The single-enable signal which controls the driving direction of the bi-bus drivers is developed within the CPU.

The CPU bi-bus driver drives outwards during most of its states. That is, the internal eight-bit data bus is transmitted outward to the external BB bus.

The CPU bi-bus driver drives inwards during the T3 state, except during PCW cycles. That is, the data on the external BB bus drives the CPU's internal data bus. The directionality is inwards also during the WAIT and the STOPPED state, when the external BB bus is not being driven.
SEC. 6.7 THE 8080 DATA BUS (cont'd)

6.7.2 T4 and T5 Times The T4 and T5 states occur in the 8080 during idle states, and during internal processor operations which require data transfers internal to the chip. That is, during T4 and T5 times, bus drivers within the 8080 are transmitting to bus receivers on the internal data bus. During these states, the 8080's bi-bus drivers clearly must be enabled in such a manner that information on the external 88 bus is not driven inwards, or else the internal data bus would be disrupted. Thus, the bi-bus drivers drive outputs during T4 and T5 times, and the state of the internal data bus is available on the external 88 bus.

The chip manufacturers state, in a footnote to the chart called INTERNAL PROCESSOR OPERATIONS (reprinted in Chapter 2), that the "Content of the internal data bus at T4 and T5 is available at the data bus. This is designed for testing purposes only." The above discussion should demonstrate that this fact is a logical consequence of the bus structure used within the 8080. An unnecessarily complex bi-bus structure would be required to design this feature out (one with dual enables and extra gating). Thus one need not fear that the availability of this information is likely to change on future 8080 devices.

The availability of internal bus data during T4 and T5 times is used in Chapter 11 of this book for adding extra instructions to the 8080.

6.7.3 8080 Data Bus Buffering As discussed in Chapter 2 above, the 8080's internal bi-bus drivers are just capable of sinking 1.5 mA under normal voltage and temperature conditions. Many 8080 designs therefore add low-power TTL buffers (or CMOS buffers followed by regular TTL buffers) to the 8080's eight data outputs, producing an output bus which, in this book, would be labeled TB7 through TB0. The direct connections to the 8080 data terminals now form the input bus, labeled NB7 through NB0.

Another approach, used in the design of this book, is to use low-power Schottky integrated circuits for interfacing with the 8080 data bus. Up to three such devices may be driven directly by the 8080. The additional cost of these devices must be considered, but so must the cost of the buffers (including stocking, testing, and insertion) and the PC board space saved.

When buffers are not used, the same bi-directional bus (88) serves both for inputs and outputs to the CPU. This leads to added simplicity in the bus structure on the microcomputer PC board.

Input bus considerations are discussed further in Chapter 8 below.

SEC. 7.1 I/O ADDRESS MODES

Communications between a microprocessor and data-handling circuitry external to the central processing unit are commonly implemented by executing input and output (I/O) instructions. Special circuitry generates timing signals necessary for synchronizing the I/O device with the CPU—a strobe signal which causes an output port to latch up information on a bus at the appropriate time, or an enable signal which activates the bus-driving circuitry associated with an input port, causing the input data to be transmitted to the CPU.

Just as when the CPU is reading from or writing into memory, a means must be provided for addressing the desired I/O port. This I/O port address must be available outside the CPU board at the same time that the I/O data transfer takes place—in practice, for proper synchronization, a little earlier.

Examples of I/O ports are printers, keyboards, television typewriters and graphic displays; analog to digital converters and digital to analog converters; digital displays; cassette recorders; and so on.

There are at least three common modes for addressing I/O ports, each being used by a different standard eight-bit microprocessor.

8008 Mode—In the 8008, the simplest of these processors, the I/O port address appears on five bits of the high-order address bus, namely DH5 through DH1. Output data appears on the low-order address register, DL7-DL0. Input data is transmitted to the CPU via the CPU data bus, DB7-DB0.

8080 Mode—In the 8080, the I/O port address appears in duplicate on both address buses, DB7-DB0 and DL7-DL0. Both input and output data transfers take place via the CPU data bus, DB7-DB0. The 8080 microprocessor provides signals which are used to create 1/0 strobes for activating these data transfers at the appropriate time. Data transfer is unidirectional—i.e., input only or output only.

6800 Mode—In the 6800, there are no I/O instructions as such; I/O ports are simply addressed as memory. Typically, one section of memory is set aside for I/O use. Just as with memory, I/O data transfers take place via the data bus, DB7-DB0.

The first part of this chapter concentrates on 8080-type instructions. A method of decoding the strobe and enable signals for these instructions is described next. Finally, information is presented on 8080-type I/O instructions. A method of generating a universal 1/0 strobe enable signal, compatible with all three 1/0 ADDRESSING MODES, ends the chapter.
SEC. 7.3 8008 INPUT/OUTPUT INSTRUCTIONS

7.3.1 Instruction Codes

The 8008 has a complement of eight input instructions and 24 output instructions, as shown in Figure 7.2.1.

<table>
<thead>
<tr>
<th>BINARY OCTAL</th>
<th>INSTRUCTION</th>
<th>BINARY OCTAL</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111111</td>
<td>177</td>
<td>OUT 37</td>
<td>01011111</td>
</tr>
<tr>
<td>01111110</td>
<td>176</td>
<td>OUT 36</td>
<td>01011110</td>
</tr>
<tr>
<td>01111101</td>
<td>175</td>
<td>OUT 35</td>
<td>01011101</td>
</tr>
<tr>
<td>01111100</td>
<td>174</td>
<td>OUT 34</td>
<td>01011100</td>
</tr>
<tr>
<td>01111011</td>
<td>173</td>
<td>OUT 33</td>
<td>01011011</td>
</tr>
<tr>
<td>01111010</td>
<td>172</td>
<td>OUT 32</td>
<td>01011010</td>
</tr>
<tr>
<td>01111001</td>
<td>171</td>
<td>OUT 31</td>
<td>01011001</td>
</tr>
<tr>
<td>01111000</td>
<td>170</td>
<td>OUT 30</td>
<td>01011000</td>
</tr>
<tr>
<td>01110111</td>
<td>157</td>
<td>OUT 27</td>
<td>01010111</td>
</tr>
<tr>
<td>01110110</td>
<td>156</td>
<td>OUT 26</td>
<td>01010110</td>
</tr>
<tr>
<td>01110101</td>
<td>155</td>
<td>OUT 25</td>
<td>01010101</td>
</tr>
<tr>
<td>01110100</td>
<td>154</td>
<td>OUT 24</td>
<td>01010100</td>
</tr>
<tr>
<td>01110011</td>
<td>147</td>
<td>OUT 23</td>
<td>01010011</td>
</tr>
<tr>
<td>01110010</td>
<td>146</td>
<td>OUT 22</td>
<td>01010010</td>
</tr>
<tr>
<td>01110001</td>
<td>143</td>
<td>OUT 21</td>
<td>01010001</td>
</tr>
<tr>
<td>01110000</td>
<td>140</td>
<td>OUT 20</td>
<td>01010000</td>
</tr>
</tbody>
</table>

Fig. 7.2.1--The 52 8008 I/O Instructions

These instructions are one byte long, as shown in the chart above. The binary instruction code takes the form 011 HMM MM1. If FR = 00, the instruction is an input instruction--Input 7 through Input 0, where the input number is defined by the binary combination of MMM. If FR = 01, 10, or 11, the instruction is one of the 24 possible output instructions, Output 37 through Output 20.

The 8008 user may find the following rule convenient in remembering I/O instruction codes. Take the number of the input or output port; double it (using base 8 arithmetic); add 101. The result is the octal instruction code.

7.3.2 The PCI Memory Cycle

I/O instructions in an 8008 microcomputer system require two memory cycles to execute.

The first memory cycle, a PCI (Instruction) cycle, is used to fetch the I/O instruction from memory, and is just like the first part of any other 8008 instruction. It consists of the T1, T2, and T3 states only—T4 and T5 are skipped. See Figure 7.2.3.

Fig. 7.2.3--8008 Input/Output Instructions: Internal Operations
During the second memory cycle, a PCC (command) cycle, an I/O instruction departs significantly from the other 8088 instructions.

During PCC-T1 time, the data in the A register goes out on the 8088 data bus. (This information was loaded into the A register by a previous instruction.) As always, the DL register external to the 8088 latches up whatever is on the data bus at STI time. Usually the information on the bus during T1 time is part of a memory address, so this use of the DL register is somewhat unexpected.

During PCC-T2 time, the instruction itself goes out on the 8088 data bus. It is latched up in the external DL register at STI time. Now the DL register serves (temporarily) as a source from which the control logic can derive the binary code for the I/O instruction. The relevant bits are W05 through W11.

7.2.2 Output Instruction Sequence

Consider the case where data is being outputted. The output information has been loaded into the A register by a previous instruction. During T1 time of the PCC cycle, this data leaves the CPU. But the external hardware does not yet have any indication where this information is going. So, the information is stored temporarily in the DL register. Then, at T2 time of the PCC cycle, the instruction is latched into the DL register. Now the destination can be decoded. At T3 time, which during output instructions is an idle state for the CPU, the appropriate output port is strobed. The information then passes from the DL register to the output port. See the top half of Figure 7.2.2 below.

Thus, output information from the 8088 normally travels via the DL register, rather than directly from the CPU data bus to the output ports. This is necessary to accommodate the 8088's internal processor operations. This feature is actually convenient to the hardware designer because the eight-bit latch which makes up the DL register serves as a buffer which keeps the output ports from overloading the CPU data bus.

After T3 time of the PCC cycle the output instruction has done its job, and the T4 and T5 states are skipped.

7.2.3 Input Instruction Sequence

During T2 time, PCC memory cycle, of an input instruction, the instruction code is used to select the input port designated by the

*An input instruction may also be used to output data, as shown by the steps marked with an asterisk. See Chapter 9.
SEC. 7.3 8008 INPUT/OUTPUT INSTRUCTIONS (cont'd)

Instruction. Then, at T3A time, the CPU bus inputs data to its internal b register. At T5 time, the input data is transferred to the A register, and the input instruction has achieved its purpose.

(At T4 time of a PCI cycle, the four flags appear on the data bus. This is an auxiliary function unrelated to I/O data transfers. It is useful in equipment where the flag conditions are displayed on a front panel, or in flag-saving hardware: see Chapter 17.)

Figure 7.3.2 illustrates the data transfers which take place during I/O instructions.

SEC. 7.3 INPUT AND OUTPUT SYMBOLS

An eight-bit digital input device from which the CPU derives data is called an input port. Generally, it is a multiplexer with open-collector output circuitry, or an integrated circuit with three-state output circuitry (as discussed in Chapter 8). The eight-bit digital output devices to which the CPU sends data are called output ports. Digital ports are represented by the symbol D, and analog ports by A. Inputs are referred to by the symbol I and outputs by the symbol O. Thus DI refers to a digital input, and AO to an analog output.

DT17 refers to digital output 17, and when used without a suffix, is used to denote the strobe signal used for loading output 17. Similarly, an enable signal for digital input 5 would be represented as DIN. Since such enable and strobe signals are usually provided by the outputs of active-low decoders, the signals are often shown in their complemented state—NMT, DIN, etc.

Figure 7.4.1 in the next section shows all possible digital input and output strobes.

In order to refer to an input or output strobe in a general way, the characters I, Y, or Z replace numerical designations. Thus, AX4 is a reference to an active-low strobe for an undefined analog input port.

A particular digital input or output bit is referenced by adding a dashed suffix to the associated enable or strobe symbol. Hence, bit 7 of digital input 3 would be called DIN7. When these symbols are used with analog inputs, they refer to one of several analog signals in a multi-channel (multiplexed) analog system. AN4-7 would refer to the eight-bit digital value at digital input port #4 which derives from analog input channel number 7. (See Chapter 12.)

During input instructions the 8008 CPU receives data on its bidirectional bus during T3 time, PCI cycle. In practice, the input devices are enabled slightly in advance of the beginning of T3 time, that is, during T3A time. Output ports are also loaded, from the DL register, at T3A time of an output instruction, PCI cycle. A general-purpose strobe/enable signal which occurs only at T3A time of I/O instructions may be formed by gating the T3A state (developed by the main timing circuitry discussed in Chapter 5) with DIN and C1 (the latter two signals defining a PCI cycle). In this book, this signal is called DIN or DIN when it is an active-low signal.

Figure 7.4.2 shows DIN generated by a three-input NAND gate, and being used to enable I/O strobe/enable decoders. The encoded number of the input or output port is obtained from the DIN register, where the I/O instruction was stored during T2 time, PCI cycle. Note how simply the signals for all 32 input/output instructions are generated.

In a practical microcomputer design, not all 32 ports would necessarily be used. For example, the lower decoder in Figure 7.4.2 would be omitted in a system needing all eight inputs but only eight output ports.

Figure 7.4.4 shows a 74LS138 3-to-8 decoder being used to generate four output strobes and four input enables—all with only one chip. Note that in this example, the DIN signal is not accessible: it is generated within the 74LS138 chip.
SEC. 7.4 GENERATING 8008 INPUT ENABLES AND OUTPUT STROBES (cont'd)

Note that the circuit of Figure 7.4.1 ignores IHS and IHW. This means that the decoder does not distinguish between I/O instructions which are identical with regard to IHS, IHW, and IHI, but may differ so far as IHS and IHW are concerned. This decoder is restricted as to the number of inputs and outputs available; and different I/O instructions will produce the same strobe. For example, a DTR pulse will be generated by INF 000, INF 004, OUT 010, or OUT 030. Any one of these instructions can be used as an output instruction: the data left in the A register by a previous instruction can be strobed into an output port at POC-T3A (DIN) time. But, as discussed in the previous sections of this chapter, only in the case of the two input instructions—INF 000 or INF 004—all input information present on the data bus at POC-T3A time end up in the A register when the instruction is finished. With this distinction in mind, the programmer might find places where he would use an OUT 010 and others where the INF 000 instruction would be needed.

SEC. 7.5 PERIPHERAL STROBE DECODING TECHNIQUES

7.5.1 Using DIN and the DH Register When peripheral devices are being connected to the main logic board in a microcomputer, it may be very cumbersome to decode all the needed strobe signals on the CPU board and provide all these extra pins on the connector. A more efficient approach is to provide the IHI register and the DIN signal. Then the needed strobes are decoded by the peripheral itself.

This approach also leads to the design of a universal and flexible peripheral bus structure. See Chapter 10 for a more complete description of how peripheral buses should be designed.

7.5.2 Developing a Peripheral Strobe Approach Every input or output instruction in an 8008 microcomputer contains five address bits. If all combinations of inputs and outputs are to be utilized, then any strobe generated should depend on all five bits of this instruction address. (The example in Figure 7.4.1 does not uniquely decode inputs and outputs.) The strobe should depend not only on the five bit addresses; it should be enabled only when the DIN signal is active (logic zero). Therefore there are six signals which must be gated together to provide any one uniquely-defined I/O strobe. These six signals are IHS, IHW, IHI, DTR,
SEC. 7.6 PERIPHERAL STROBE DECODING TECHNIQUES (cont'd)

DIN, and DNT. Figure 7.5.1 shows a six-input NAND gate and three inverters being used to decode an INPUT 007 strobe (DNT). By inserting and deleting inverters in all the various combinations at the DIN through DIN inputs, all 32 of the 8088 I/O strobe/enable signals could be generated.

Fig. 7.5.1--A NAND Gate Decodes the Six Signals Necessary to Produce a Strobe or Enable

This NAND gate is obviously quite awkward. What is needed is a single integrated circuit which combines the six required input bits and provides a number of input/output strobe/enable signals. This one IC could be located in the peripheral device and decode the few I/O control signals which it might need.

The 74LS138 TTL integrated circuit is a 3-to-8 decoder with three strobe inputs. Two of the strobes are active-low, and one is active-high. This turns out to be very convenient for many decoding applications, and therefore members of the '138 family appear frequently in this book. They include the 74LS138 (Schottky) as well as the commonly-used 74LS138 or 3205 (8205) (Low-power Schottky).

Fig. 7.5.2--74LS138 3-to-8 Decoder Schematic

This integrated circuit satisfies the requirements outlined above for a peripheral strobe decoder, since it has exactly the six inputs desired, and provides eight active low outputs.

In application, the active-low DIN strobe must be gated with all outputs, and thus must be connected to one of the 74138's two active-low enable inputs. Then the BI register bits (D15 through DIN) are connected to the remaining input terminals in any order.

7
7.5.3 Selecting the Desired Combination of Strobes

There are twenty unique ways in which the five DH register bits may be connected to the 74138 peripheral strobe decoder. They are shown in Figure 7.5.3. Eight of these combinations provide output strobes, while the remaining twelve combinations provide some inputs and some outputs.

Note that with this scheme, INP 000 is never decoded. This strobe can be generated only when all five DH register bits are logic zero, but the 74138 requires that there be at least one active high input bit (at pin 6). For a similar reason, the DT7 strobe cannot be generated. Therefore INP 000 and OUT 037 are usually designated as I/O ports located right on the main CPU logic board.

In evaluating the usefulness of the various combinations of signals provided, note that an "output instruction" in the 8080 is really an output-only instruction: it cannot be used to input data to the CPU. On the other hand, an "input instruction" is really an input/output instruction, and may be used for either inputting or outputting (or both). This point is elaborated further in Chapter 9.

For the further convenience of the designer, the twenty rows in Figure 7.5.3 are realigned in Figure 7.5.4 as columns. This diagram is a graphic representation of how the decoder inputs may be wired, and what control signals result.

The decoders on the various peripheral circuit boards of a system need not completely avoid overlap in input/output signals. All that is necessary is to develop a sufficient number of unique input/output control signals to fulfill the needs of each peripheral device.

As design examples of this peripheral strobe technique, Figures 7.5.5 and 7.5.6 show the circuit diagrams which correspond to columns 15 and 16 respectively of the table in Figure 7.5.4. If these two decoders were used in two different peripherals, each would have eight I/O ports which do not conflict with the other peripheral.

Chapter 20 more fully describes the bus structure to which these peripherals might be attached, and gives a block diagram of an efficient peripheral bus system.

With slight modifications, the same bus structure is used in the modular micro series of microcomputers, available from Martin Research. The Model 471 CPU board, based on the 8080, is described near the end of this book. The 471 supports not only 8080 I/O instructions of the standard variety, but also 8080-compatible I/O, as well as memory-mapped I/O instructions using the EIN strobe. Therefore the strobe decoding technique described in this chapter are very useful in an 8080 system.
Fig. 7.5.6--10 Ways to Produce 8 I/O Strobe/Enables with the 74LS138
The 8080 instruction set includes the two instructions IN (input) and OUT (output). Each is a two-byte instruction, the second byte of which is an eight-bit I/O port address. Thus, the 8080 is able to address 256 input ports and 256 output ports. The 8080 develops a distinct output strobe signal (output write, \texttt{OUT}) and a separate input enable signal (input read, \texttt{IN}). These are related to IN and OUT instructions in the same way that memory read and write signals (\texttt{MRE} and \texttt{MWR}) relate to memory-referencing instructions. These distinct strobe signals mean that the I/O ports are addressed independently from memory.

This 8080 I/O mode is similar to the 8088 in its separation from memory-referencing instructions. However, in the 8088, output data appears on the data bus, rather than on the low-order address bus.

Also, with the 8080, there are eight I/O address bits (rather than five) which appear redundantly on both the high-order and low-order address lines (A17-A0 and I/O-I/U). To produce output strobe signals, a decoder combines the output strobe, \texttt{OUT}, with various combinations of the eight address lines. Similarly, to create input enable signals, a decoder combines \texttt{IN} with the address lines.

Unless an unusually large number of I/O ports is required in an 8080 system, it is unnecessary to use all eight address lines in the I/O decoding circuitry. Five lines are often more than adequate—producing 32 input ports and 32 output ports. This allows the 74LS138 to be used in an 8080 system, just as described earlier in this chapter (Sec. 7.5).

One method is simply to ignore three of the address lines in the I/O decoding circuitry. Then referencing to the decoding circuitry in Sec. 7.4, \texttt{IN} or \texttt{OUT} replaces the 8080 signal \texttt{M}. Another technique is to create an 8080 version of the signal \texttt{IN}, and use it for memory-mapped I/O instructions. That is, a given area in memory is assigned to I/O ports, and memory read/write instructions are used instead of \texttt{IN} and \texttt{OUT}. The signal \texttt{IN} is produced by combining 8080 memory read/write signals with three selected address bits, thus uniquely defining a portion of the memory map for I/O ports. The remaining five address bits are combined with \texttt{IN} by the 74LS138 peripheral strobe decoder to produce any eight of the 32 possible I/O strobe/enable signals.

These techniques are discussed in further detail, with examples, in the 471 data sheet, Sec. 1.6, near the end of this book.

A method of creating a 16-bit output port for the 8080 is discussed at the end of Chapter 9.
SEC. 8.1 INPUT DESIGN

This chapter describes several methods for inputting digital information to a microcomputer. It is assumed that the information is available at TTL logic levels.

Both the 8088 and the 8080 microprocessors receive input data through an eight-bit bidirectional data bus. The data bus is used not only for receiving data, but also for outputting data from the CPU and for addressing memory, all at different stages in the internal processor operations. This imposes the first input design requirement: the input devices on the bidirectional data bus must be disabled when the CPU is not supposed to be accepting data.

Since more than one eight-bit data source is usually required, the second requirement is some method of selecting the desired input source.

After a preliminary discussion of 8088/8080 data bus input voltages, this chapter discusses two basic input approaches: using multiplexers, and using three-state bus structures.

SEC. 8.2 8080/8088 DATA BUS INPUT VOLTAGES

The 8088 and 8080 microprocessors have been designed for compatibility with TTL circuitry. However, their internal MOS circuitry needs a little help in recognizing logic high voltages. TTL devices typically output logic high voltages of about +5.4 volts, but are not guaranteed to deliver more than +1.4 volts. The 8088 requires a logic high input of V\text{IH} = 1.5 \text{ V}, or +3.5 \text{ V} with a +5 \text{ V} supply. The 8080 requires +3.3 \text{ V}. (The AMD8080A from Advanced Micro Devices requires only +5 \text{ V}, but to allow interchangeability, the designer should provide for +3.3.)

In early 8088 designs, an array of pullup resistors was usually connected from each data bus line to +5 \text{ V}. In one published circuit, the resistors were connected through diodes to a transistor switch, which was activated during input selection cycles. The 8080 internalizes this feature with active pullup resistors which are switched on during INR0 cycles. Still, these resistors are not guaranteed to turn on until the minimum V\text{IH} is reached. (The NEC UP8080A omits them.)

When the CPU data bus terminals are connected to bus drivers, to increase the CPU's ability to address an expanded system, a bidirectional bus driver IC may be chosen which is rated to provide voltage levels of over 3 volts when driving inwards. Examples: Signetics 8726 and 8728; Intel 8226 and 8228. (The 8228 is a special driver/controller for the 8080.) In small systems, MOS memory connected directly to the data bus (2122 RAM, 2708 ROM, for example) provides the required voltage levels; MOS memory ICs are manufactured with the same (or very similar) process as the 8080 or 8088 itself. The 4099 CMOS driver (Fairchild) also interfaces to MOS microprocessors.
SEC. 8.2 8004/8080 DATA BUS INPUT VOLTAGES (cont’d)

There are two basic methods of driving these MOS microprocessors with TTL I/Os. The first is to use pullup resistors, as do the manufacturers in their prototype designs. Resistors in the range of 10 to 22 kΩ are suitable for data buses using three-state input devices.

The other approach is to make sure that the devices driving the CPU data input bus are specified for a logic high output voltage of +5.5 volts or better. Many off-the-production-line TTL devices meet this specification already.

As manufacturers are called upon to deliver integrated circuits in quantity which can drive MOS LSIs without pullup resistors, they will probably comply by supplying respecified parts (perhaps at premium prices). Intel Corporation has announced the availability of I/Os guaranteed to deliver a +3.65-volt logic high level, which provides a safety margin for protection against noise. An example of this is the 8212, an octal latch with three-state outputs, available in a 24-pin package.

SEC. 8.3 INPUT MULTIPLEXERS

8.3.1 Three-State Multiplexers The multiplexer is essentially a digital electronic switch, which selects the input designated by its data select code. TTL designers will be familiar with such devices as the 74153, a dual 4-to-1 multiplexer with TTL totem-pole outputs. A multiplexer which is to interface with the bidirectional CPU bus must be capable of being disabled. Figure 8.3.1 outlines a design in which multiplexers with three-state outputs connect to the CPU bus.

The block diagram in Figure 8.3.1 shows a three-state multiplexer array with six eight-bit input ports. Input ports #2 and #4 are connected directly to the multiplexer. The circuitry used here is totem-pole TTL.

Multiplexer input ports #0 and #1 are connected to a three-state input bus. (See Chapter 6 for definitions of bus types.) In the diagram, the symbol 3-state indicates a three-state device. A second input bus furnishes information from inputs #6 and #7 into the multiplexer.

Figure 8.3.1—Input Design Uses Both Multiplexers and Three-State Input Bus

Figure 8.3.2 shows the essential details of the multiplexer portion of the preceding figure. This circuit uses 74153 three-state multiplexers in a 32-to-4-line array. A few sample input connections are shown. The symbol DS/0-7 refers to the high-order bit (bit 7) of either input #1 or input #6. The line labeled 880 refers to the low-order bit of the CPU’s bidirectional data bus. Other lines are labeled in similar notation.

8.3.2 Open-Collector Multiplexers As implied in the general discussion of bus structures in Chapter 6, open-collector multiplexers may be used to interface with the CPU bidirectional data bus. For example, two 8267 open-collector multiplexers could be connected to the CPU data bus. The designer may choose to emulate this approach; but consider that it may necessitate pullup resistors on the data bus.
SEC. 8.4 INPUT ENABLE AND SELECT SIGNALS

8.4.1 Decoding the Input Enable Signal—8088 CPU The 8088 reads input data during T3A time of an input instruction. Therefore, in Figure 8.3.2, the 74LS253's three-state output enable pins (1 and 15) could be connected to a 3-input NAND gate, one of whose inputs is connected to T3A. Only during memory cycle, two of the input (or output) instruction (PCC cycle) and its high-order control bit, CC2 (sometimes referred to as IMT) be logic zero, AND CC1 be logic one. Therefore the other two inputs to the 3-input NAND gate be 1's and 0's, and now the multiplexers will be enabled during T3A time of either an input or an output instruction. If, at this time, INS and IRP are low, then the instruction being executed is an input instruction. INS and IRP could have been added as additional inputs to the NAND gate (which would then have to be a 4-input NAND)—but this logic is unnecessary. It does not matter if the multiplexer tries to input data during an output instruction. The T3 state of the PCC cycle during an output instruction is an idle state. The 8088's bidirectional data bus driver reads data in parallel onto the internal data bus, but nobody is listening inside. The designer can thus reduce a 3-input NAND to a 2-input NAND. In a complex 8086 system, this concept can save a number of ICs.

8088 CPU—The I/O strobe signal to be used depends on which I/O addressing mode is used with the 8080, as discussed in Chapter 7. IOE is the normal 8080 input enable signal, but DIN or MM# may also be used. (See the 411 data sheet, Sec. 1.6.)

8.4.2 Selecting Input Ports—8088 CPU As discussed in Chapter 7, the eight-bit binary code for the input instruction itself appears on the data bus at T2 time of memory cycle two (PCC cycle). It is latched up in the IM register, and the IMH and IML bits contain the three-bit binary code for one of the eight possible 8088 input ports. In the circuit of Figure 8.3.2, IMH and IML are used as the multiplexer data select lines, thereby selecting one of four combinations of two input ports (7 and 6; 5 and 4; 3 and 2; 1 and 0). To select one input port from any of these pairs, the low-order input select bit, IMI, must be used. The 74126 three-state buffer at the top left of Figure 8.3.2 uses IMI to select one bit of input port 6. These techniques may easily be extended to provide an array of up to eight 8-bit input ports, the normal 8088 input capacity.

8088 CPU—The 8088 outputs the I/O port number on both the HI and IL address registers, redundantly, at T1 time during an I/O instruction. A full eight-bit decoder may be used, or, a predecoded strobe signal (DIN) may be used for 8086-compatible I/O instructions or for straight memory-mapped I/O using FER# and FERW. See the 411 data sheet, Sec. 1.6.

Fig. 8.3.2—Essential circuitry for Figure 8.3.1
8.6 BUS-ONLY INPUT DESIGN

8.6.1 Direct Bus Approach
In small microcomputers, all of the input devices may simply be connected directly to the CPU bidirectional data bus, using three-state devices. Only the selected input device is enabled during an input instruction (and, permissibly, during output instructions: see paragraph 8.4.1 above). At other times, all of the input devices are disabled, and draw only a small leakage current from the bidirectional data bus. For an example of this approach, see the microcomputer design in Chapter 6.

The practical limitations to the direct three-state bus approach are bus line capacitance and the leakage currents demanded by disabled three-state devices.

8.6.2 Bus Capacitance
The data bus terminals of both the 8080 and 8085 are specified at rated speeds for up to 100 picofarads of load capacitance. It is difficult to predict the capacitance of a given bus circuit with accuracy, because of the influence of printed circuit geometry. As a rule of thumb, the designer can estimate 7 picofarads of capacitance for each IC pin added to the bus. Large arrays, or long bus lines, are likely to cause trouble.

Capacitance loads greater than 100 pf will of course cause increased delay in data transfer. The 8080 specifications indicate that an increase of 100 to 200 pf causes the output delay to move from about 0.85 us to 1.01 us. In the 8085, adding 50 pf will add only 20 ns of output delay. These delays may be tolerable in slow-speed systems, but if the CPU is operating close to its maximum speed, the clock may have to be slowed down accordingly to compensate. When many three-state devices are connected to the 88 bus, it is best to create a CPU input bus system using a three-state bus driver, as described in paragraph 8.5.4. This is the practice in all but very small microcomputers.

8.6.3 Three-State Leakage
A second limitation to the direct bus approach results from the leakage currents demanded by disabled three-state devices. Each output terminal of a typical three-state IC may demand up to 40 microamps of leakage current when disabled. A large array of input devices on the CPU data bus may cause total leakage currents that imperil the CPU's ability to drive external devices.

The Intel 8080A is capable of sinking 1.9 ma to 0.45 V, while the AMD 8080A will sink 3.2 ma to 0.4 V. The designer should check current data sheets to note any updates to these figures. The specs differ between manufacturers, and between the 8080A and older 8080.

Figure 8.6.1—Block Diagram of Input System Using NB Bus

The 8085 microprocessor is capable of sinking approximately 1.5 milliamps to a logic low level of 0.4 volts (+5 volt power supplies, 0-75°C). It is capable of sourcing approximately 1.7 milliamps from a logic high level of 3.0 volts. Any problems with leakage current are likely to derive from the logic low specification. Take for example a circuit where the 8085 is required to drive three low-power Schottky devices with 360-microamp logic low input currents. This leaves a margin of some 420 microamps—enough to supply leakage currents to about ten disabled three-state input devices.

Another possibility is that leakage currents will endanger the ability of external ICs to drive the CPU. This problem is most likely to occur with MOS memory chips connected directly to the CPU data bus.

The bus structure designer must calculate these factors carefully when designing small microcomputers with multiple three-state input devices directly on the CPU data bus.

8.6.4 Adding an Input Bus
When many three-state devices are to be used as inputs to the microprocessor, it is best to create a CPU input bus system. An eight-bit three-state buffer array is connected between the individual three-state input devices and the CPU data bus. The three-state buffers drive the CPU data bus when enabled by a PIA enable signal, as shown in Figure 8.5.1. The inputs to the buffer array make up the input bus (NB), relabeled because it is no longer bidirectional. The same NB bus is often connected not only to input ports, but to three-state memory devices, as shown in Figure 8.5.1.
INPUT DESIGN APPROACHES

SEC 8.5 BUS-ONLY INPUT DESIGN (cont'd)

So far as the loading of the CPU bus is a factor, the three-state buffer circuitry of Figure 8.5.1 is similar to the three-state multiplexer circuitry in Figure 8.3.1 above. Each places only one three-state device on the S8 bus. The choice of circuitry depends on the number and type of input port devices to be used in the microcomputer being designed.

SEC 8.6 EXPANDING 8008 INPUT PORTS WITH CONDITIONAL INPUTS

Some microcomputers require more than the eight input ports which can easily be provided in an 8008 system. The problem is not the input multiplexer chain which can easily be extended. An input bus structure can be expanded even more readily simply by adding more three-state devices to the bus. The problem comes in performing an input instruction with the 8008 which will create more than eight distinct input enable signals.

During memory cycle two of an input instruction—a PCC cycle—the instruction itself appears at T2 time. Only three bits of that instruction (DR1, DR2, and DL) are available for selection of input ports, as discussed in Chapter 7 and earlier in this chapter.

A simple and convenient method of expanding inputs is to decode the DL register during the execution of input instructions. As an example, the following design decodes the seven conventional input instructions, INP 006 through INP 002. The eighth input instruction, INP 007, is broken down into eight different instructions, numbered GEN 077 through GEN 070.

Immediately before executing a conventional INP 007 instruction, the programmer must set up the three low-order bits of the A register. During execution of the input instruction, the content of the A register appears during memory cycle two (PCC cycle) at T1 time. This information is latched up in the DL register, which is always strobed at T1 time.

In the circuit shown in Figure 8.6.1, the decoder to the left produces the conventional eight input strobes. The input #7 strobe is fed to an additional 3-to-8 decoder chip, which decodes the three low-order DL register bits—DL1, DL2, and DL2—to produce the eight additional input strobes labeled GEN77 through GEN70.

8080 CPUs—Although this conditional input technique could be used in an 8080 system, it is generally unnecessary. The normal 8080 1/0 addressing mode supports up to 256 input ports and 256 output ports. If 8088-compatible 1/0 ports are used in an 8080 system—using DIN—all 32 1/0 addresses can be used for inputs, as well as outputs. See the 471 data sheet near the end of this book, Sec. 15.1.

Figure 8.6.1—Input Strobe Decoder Provides Fifteen Input Strobes

The circuit shown expands the number of 8008 input ports from eight to fifteen. Obviously the number can be expanded still further with this approach, with a hardware cost of one 3205 decoder for each seven new input instructions afforded. Bits other than the three low-order DL register bits may also be used.

Figure 8.6.2 shows 8008 software examples. The instruction pairs shown are used to create conditional input strobes.

<table>
<thead>
<tr>
<th>INP 007</th>
<th>INP 008</th>
<th>GEN 077 STROBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>007</td>
<td>006</td>
<td></td>
</tr>
<tr>
<td>007</td>
<td>005</td>
<td></td>
</tr>
<tr>
<td>007</td>
<td>004</td>
<td></td>
</tr>
<tr>
<td>007</td>
<td>003</td>
<td></td>
</tr>
<tr>
<td>007</td>
<td>002</td>
<td></td>
</tr>
<tr>
<td>007</td>
<td>001</td>
<td></td>
</tr>
<tr>
<td>007</td>
<td>000</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.6.2—Instruction Pairs Used in Expanding Input Ports

This approach is an interesting example of how the designer can make full use of the 8008 microprocessor by thoroughly understanding its internal processor operations. Consider what happens during memory cycle two (PCC cycle) of an input instruction. At PCC-T1 time, the content of the 8008's internal A register is outputted on the CPU bus and latched up in the external DL register. This data transfer...
SEC. 8.6 EXPANDING 8008 INPUT PORTS WITH CONDITIONAL INPUTS (cont'd)

is not ordinarily of any importance in an input instruction, whose purpose is usually to select data from a designated input port, and transfer that information to the CPU's internal A register at PICO-TI time. Luckily for the conditional input approach, the A register can be loaded with a conditional input address during the previous instruction.

It might be asked why the 8008 bothers to output the content of the A register during PICO-TI time of an input instruction in the first place. Certainly this information is not normally needed outside the 8008 at this time. The principal reason is that it was easiest to design the microprocessor so that input and output instructions are as similar as possible. The transfer of the A to the DL register is an essential part of an output instruction (Chapters 7 and 9).

SEC. 8.7 ADDITIONAL INPUT PORTS DEFINED AS MEMORY

Input ports may be added to a microcomputer simply by defining certain memory locations as inputs. This is an important technique in systems using the 8088 microprocessor, which has only eight regular input instructions.

An input port can be added using the technique shown in Figure 8.7.1. At left is a circuit showing a 1702A 256 x 8 SRAM with its eight address bit terminals connected to the microcomputer's DL register. The active low chip enable terminal is connected to a memory select signal PAG77—which enables the chip when it is addressed during memory read cycles. The outputs of the memory chip are connected to the microprocessor input bus.

At right a four-bit three-state buffer replaces the memory chip. It can be expanded to a full eight-bit input port by adding another 74LS245. The 74HC245, an eight-bit buffer, may also be specified (Schottky clamping; Schmitt-trigger inputs; 20-pin, 0.3" Skmg-20F package).

This kind of input port is not addressed with an input instruction like a normal input, but with a memory-reading instruction. In an 8086-based microcomputer, this would mean a LR instruction. The input port designation would not be included in the binary-encoded instruction, as with input instructions. Instead, the appropriate memory location must be set up in the H register before executing the memory-reading instruction.

Figure 8.7.1—The Memory at Left Replaced by Three-State Buffer "Input Port" at Right

In the example shown, the H register is loaded with 077 (PAGE 77), an LR instruction is executed, and the four bits (DN7-4 through DN7-1) are read into register P. The disadvantage of this input port technique is that it requires the use of a high-order memory address. An advantage is that the input information may be read into any internal registers—not only the A register, as with input instructions, but the B, C, D, E, H, or L registers.

The memory-mapped I/O technique is applicable to nearly all microprocessors, with variations as to connections and software. Schematics, circuit descriptions, and typical software for the 8080 are presented in the 471 data sheet, near the end of this book (Sec. 1.6).
This chapter begins with a section devoted to 8080 output signals. Starting with Sec. 9.1, a number of output techniques are discussed which are applicable in any microcomputer. Sec. 9.4 discusses variations applicable to microcomputers based on the 8080.

SEC. 9.1 8080 OUTPUTS

Chapter 7 outlines the similarity in internal processor operations between the 8080's eight input instructions and 24 output instructions. From the point of view of this section, an 8080 output, it may be said that the 8080 has 32 output instructions, eight of which also perform the input function.

During memory cycle two of an output instruction, the STZ strobe will load DC2 with zero, and CG1 with a logic one. This bit combination defines an I/O operation. This chapter refers to all 32 I/O instructions as outputs: the first eight as input/output, and the last 24 as output-only instructions (see Figure 9.1.1). Therefore, when DC2 = 0 and CG1 = 1 (defining a POC cycle), it always indicates that an output instruction is being performed.

Recalling from Chapter 7: input/output and output-only instructions are identical in their first five states. At T4 time of memory cycle two (POC cycle), the contents of the A register are latched up in the external DL register. At T4 time the instruction itself is latched up in the IH register. In input/output instructions, T3, T4, and T5 times are used to input the data from the input port to the CPU's internal A register.

During memory cycle two of an output-only instruction, T4 and T5 times are omitted. T3 time is not used to input information into the CPU, as with input/output and many other instructions. T3 time is an idle state within the CPU, which effectively waits for hardware/external to the CPU to take the output data which has been stored in the DL register, back at T1 time, and load it into an output port.

A typical microcomputer output port would be an eight-bit latch, the data inputs of which are connected to the appropriate bits of the DL register. The eight output bits of the DL register are connected to the data inputs of all the microcomputer's output ports, and as such constitute an output bus (as defined in Chapter 6). Each output port latches up the information on the output bus only when it is strobed at T3 time, memory cycle two, of the appropriate output instruction.

For the sake of convenience, the T3A signal is generally used to produce output strobes. The T3A signal is developed in 8080 microcomputer primarily to produce input enable signals. T3A begins late in T2 time, before T3 proper begins. This is because, during input/output instructions, one desires to put the input data onto the CPU data bus for a short period before the CPU begins to input data at T3 time, to be sure that the input data has time to settle on the data bus.
Chapter 7 describes how INX and INY signals are produced at T3A time, for use as strobe/enable signals. When used to load information from the output bus into an output port, these signals are functioning as strobes, following the nomenclature of this book.

This method of generating strobes makes it possible to realize a number of kinds of output hardware with ease. The following sections show a number of techniques for generating both levels and pulses. They all are based on the principle that output data is derived from the output bus (DL register), and is valid when the IN or IN strobe goes low, signifying execution of the appropriate output instruction.

![Diagram showing the DL (Output) Bus](image_url)
SEC. 9.2 PULSE OUTPUTS

9.2.1 Using the Strobes
Figure 9.2.1 shows the simplest possible pulse output. It consists simply of one of the output strobes. Up to 32 such outputs may be provided in an 8088 microcomputer, and they are so simple that they are often forgotten.

Fig. 9.2.1—Very Simple Pulse Output

The pulse width of this output is 2.0 us for an 8088 running at its normal speed (500 Hz symmetrical clock); 1.25 us for an 8089 (800 Hz); and 500 ns for an 8080 (2.0 MHz clock cycle). This output technique is inexpensive in hardware, but costs a whole output instruction for only one pulse. It is useful in an 8080 system, where there are usually plenty of I/O ports, and in an 8088 system where there are sufficient I/O ports.

The four examples in Figure 9.2.2 show pulses which are controlled by one of the output bits. The pulses in Parts (a) and (b), for example, occur during an Output 16 instruction, only when the high-order data bit being outputted is low. Up to eight independent pulse outputs may be driven with one particular output strobe.

9.2.2 Conditional Output Pulses
A conditional output is one where the particular output, either pulse or latch, depends on more than one of the DL register bits. An addressable output is one which uses a decoder to provide more than one conditional output. The circuit of Figure 9.2.3 is conditional because the enable inputs of the 7415 are connected to DL4 and DL3. It is also addressable because the decoder address inputs are connected to DL2, DL3, and DL4.

Fig. 9.2.2—Pulse Outputs Using Gates

Fig. 9.2.3—Conditional Pulse Outputs
SEC. 9.2 PULSE OUTPUTS (cont'd)

Eight distinct pulse outputs are provided by one IC in Figure 9.2.3. A pulse may be generated by an Output 16 instruction, but only when the output word contains a particular two-bit combination. DT15-4 = 0, DT 15-3 = 1. Only one of the eight pulses occurs, as selected by the three lower order bits of the output word. Thus, in an 8080 microcomputer, the instruction sequence 1AA 014, OUT 16 would cause a PULSE1 pulse.

9.2.3 Modified Pulse Widths Pulses longer or shorter than the available output strobe signal may be obtained by using one-shot multivibrators such as the 74123 (or 76123 for more accuracy). Notice, in Figure 9.2.4, how conveniently the inverting A input may be connected to the output strobe, and the non-inverting B input to a particular bit on the output bus (D1 register). In the example shown, an output pulse will occur only if the sign bit of the A register is set when an OUT 013 instruction is executed. (In other words, DT15-7 = 1.) The RESET input may be connected to MR, from the microcomputer master reset circuitry, or it might be connected to a negative-going pulse output. If a 74221 dual one-shot is used, it cannot be retriggered once fired by a previous output pulse, unless reset.

![Fig. 9.2.4—Other Pulse Widths Generated With a One-Shot]

SEC. 9.3 LATCHING OUTPUTS (cont’d)

9.3.1 Latching Registers It is often desirable to have an output port latch up the data it receives when strobed, and to store the output data until it receives another output command. Figure 9.3.1 shows D-type latches, such as the 7475, being used for this kind of output-with-memory. Since the 7475 has two strobe connections, it can function (as shown) as a dual two-bit output port. The uppermost two latches are controlled by the Output 16 strobe, bits 7 and 6. Both true and inverting outputs are presented. The two lower latches are strobed during an INP 2 instruction. Here an input/output instruction is being used as an output instruction—that is, an OUT 002 instruction.

![Fig. 9.3.1—Four Latched Output Bits]

Of course, the DL and IN registers themselves are examples of this kind of latching circuitry. These registers, however, receive their input data directly from the CPU data bus, at T1 and T2 times respectively, while output latches receive data from the DL register at T3A time.
9.3.3 Conditional Latches  The conditional output approach can be applied to output latches, as shown in Figure 9.3.3. Here, the value stored in one of the four latches can be changed by an OUT 035 instruction, only when the two high-order bits of the A register are zero. This may be considered a simple output multiplexing circuit. Note that the output bits are labeled as conditional outputs, CTF1-4, etc. The four latches are cleared to zero by an OUT 010 instruction. In this circuit, the three-state capabilities of the 74173 are not used.

Fig. 9.3.3—Four Conditional Output Latches

9.3.3 Addressable Latches  A form of conditional latch, the 9334 addressable latch in Figure 9.3.3 drives eight current-limited light-emitting diodes (LEDs). The states of the three low-order bits of the A register, before executing an OUT 021 instruction, define which lamp is to be addressed. The next higher-order bit defines whether the lamp is to be turned on or off (0 = on, 1 = off). Thus the instruction sequence LAT 015, OUT 021 would cause output CTF1-3 to go high, turning off the associated lamp. LAT 001, OUT 021 would turn lamp CTF1-1 on.

Fig. 9.3.4—Addressable Latch Controlling Current-Limited LEDs

The addressable latch concept is useful for driving outputs other than LEDs, of course.

9.3.4 Lamp-Driver Latches  The circuit of Figure 9.3.4 uses an integrated circuit designed specifically for driving LEDs. The DM8859 has a pin for adjusting the current to the LEDs.

Fig. 9.3.4—LED Driver-Latch IC as an Output Port
Chapter 10 contains an extended discussion of digital displays.

9.3.5 Flip-Flop Latches An ordinary D-type flip-flop provides the simplest one-bit latch. As shown in Figure 9.3.5, one half of a 7474 connects to an output strobe and to a DL bit to provide a single output bit and its complement.

Fig. 9.3.5--A D-Type Flip-Flop Provides One-Bit Latch

Hex or octal D-type flip-flops may be connected to provide many output bits with one IC. The circuit shown in Figure 9.3.6 shows a 74174 hex D flip-flop, which provides six output bits. The reset line on the flip-flop may be connected to MASTER RESET (MR) so that all outputs will reset to zero during power-on initialization. A 74273 octal D flip-flop would provide eight output bits.

Fig. 9.3.6--Hex Flip-Flop Provides Six Output Bits

9.3.6 Alternate-Action Flip-Flops Various special output functions may be provided by using simple flip-flops and gates. One example is shown in Figure 9.3.7 This flip-flop...
Though an eight-bit output word usually suffices, some microcomputer applications require a larger word length. For example, precision digital-to-analog converters often require a resolution better than eight bits (one part in 256: see Ch. 22 for representative eight-bit circuitry). Ten or 11 bits are typically required. Since DAC circuitry usually requires parallel input data, with all input data available simultaneously during conversion, two output instructions would be required if only eight-bit output data were available, with the complete data word becoming available only after a second output instruction.

Though the 8080 is an eight-bit microprocessor, it is capable of outputting 16-bit data. See Figure 9.5.1.

Fig. 9.5.1—16-Bit Output Port for 8080

The normal 8080 output instruction might be said to waste eight bits of output data. The output data itself appears on the data bus, and the output port number appears redundantly on both the IX and DL address lines. In Fig. 9.5.1, the low-order address bits (D7-D0) is used for eight bits of output data, and the data bus (D7-D0) is used for another eight bits.

A memory write instruction is used to output to the 16-bit output port. Assume the instruction to be MOV M,B (move to memory the contents of the B register). The L register—normally a low-order memory address—contains the eight high-order output bits. The H register contains the eight low-order output bits. The H register—a memory page address—decodes by external circuitry to produce a suitable output strobe signal—here, DTV. 16 bits of output data appear in Output Port 7.

A convenient way of producing DTV is to use 8274—essentially a predecoded strobe signal, addressing up to 32 I/O ports with memory-mapped instructions. (See the data sheet for the 8274 CPU board, Sec. 1.6.4, near the end of this book.)

Alternatively, the port address and high-order output data can be set up in the BC register pair, with the low-order output data in the A register. Then a STX B instruction would produce the 16-bit output.

Notice that all 16 output bits become valid at the same time. This could prove important in a DAC (digital-to-analog converter) application. If the data were outputted to the DAC in two stages—eight bits at a time—the DAC would create an extraneous intermediate output voltage after the first byte had arrived. Of course, extra hardware could be added to delay data transfers to the DAC until the second output byte arrived—but this is unnecessary when the circuit in Fig. 9.5.1 is used.

Another way of creating a 16-bit output port is through straight memory-mapped I/O. Six memory locations, O12345 and O12346, serve as memory-mapped output ports. The 8080 instruction SLD O12345 would output the data stored in the H and L registers to these two locations. The L register value would move to the I/O port at location O12345 during one memory cycle, and the H register value to location O12346 during a subsequent memory cycle. This technique—essentially a 6400-node output instruction—works with the 8080, but differs from the technique described above (Fig. 9.5.1) in that the 16 bits do not appear simultaneously.
SEC. 10.1 COMBINED INPUT/OUTPUT TECHNIQUES

With the 8088 microprocessor, every input instruction is also an output instruction—as discussed in Chapters 7 and 8 above. Advantage may be taken of this fact by adding certain features to the microcomputer. The basic principle, with the 8088, is that during the FBC cycle of an input instruction, data flows both into and out of the CPU.

8088-compatible combined input/output instructions may be added to an 8080 CPU board. (See the 47l data sheet near the end of this book, 1.6.)

SEC. 10.2 TABLE LOOKUPS

One combined I/O technique involves table lookups. The microcomputer outputs a digital value, then inputs a new digital word which corresponds to some desired function of the original word. In a sense, the microcomputer is looking up the answer in a numerical table. The first of the following two subsections describes how a table lookup program would normally be written for an 8088 microprocessor. The second describes how the same function could be performed more efficiently using a combined I/O technique.

10.2.1 Table Lookup with 808 Program A table containing the squares of the integers 0 through 5 is shown in Figure 10.2.1. The LOCATION column corresponds to an address in the microcomputer's read-only memory (ROM). The CONTENTS column shows the digital value stored at that location (in octal). If the 8088's R and I registers are loaded with a number between 000000 and 000006, and an LMI instruction is executed (Load A from Memory), the A register will then contain the square of the number.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>CONTENTS in Octal</th>
<th>CONTENTS in Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>000001</td>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>000002</td>
<td>004</td>
<td>4</td>
</tr>
<tr>
<td>000003</td>
<td>011</td>
<td>9</td>
</tr>
<tr>
<td>000004</td>
<td>020</td>
<td>16</td>
</tr>
<tr>
<td>000005</td>
<td>031</td>
<td>25</td>
</tr>
<tr>
<td>000006</td>
<td>044</td>
<td>36</td>
</tr>
</tbody>
</table>

Fig. 10.2.1—Table of Integer Squares Stored in Memory
If the number to be squared was originally in the A register, then the program shown in Fig. 10.2.2 will convert this number to its square. (8086 mnemonics are shown.)

---
NUMBER IS IN A REGISTER.
LHI 000 POINT TOWARDS TABLE START.
LAA POINT TOWARDS TABLE ENTRY.
LAM LOAD SQUARE OF NUMBER.
---

Fig. 10.2.2--Program for Squaring a Number with a Lookup Table

The procedure used here to perform squaring can also be used to find the sine, cosine, or any trigonometric, algebraic, or arbitrary function of an eight-bit variable. A 256-byte lookup table is convenient for many desired functions because the variable to be looked up may be placed in the CPU's L register, and the page number of the lookup table in the H register. Thus any convenient page in memory may be used, with the table values stored in PROM or ROM. Usually the H register is not used in referring to the variable itself, but addresses the lookup table in the same manner that other pages of memory are addressed. Of course, the lookup table is usually located at some other page number than 0000—so the table of numbers shown in Figure 10.2.1 would more likely start at location 002000, and the LHI 000 instruction in Figure 10.2.2 would be replaced with LHI 020.

10.2.2 Table Lookup Using a Combined I/O Technique  The 256-byte table lookup program discussed above could instead be implemented using a combined I/O technique. Figure 10.2.3 shows a 256-byte PROM—the same memory element that could have been used in the above subsection—as an input/output port. This configuration, called the Table Lookup option, requires little or no additional hardware in PROM-configured systems.

In an 8080 system, it makes more sense to address the lookup table as a page in memory. This is because several instructions must be used to address an 8086-compatible combined I/O port with an 8080, and the advantages of this technique are lost. (For the programming required, see the 471 data sheet, Sec. 1.6.) Though 8086-compatible combined I/O instructions make sense for the 8080 in some applications, they are impractical for table lookups.

With the 8088 CPU, addressing a table lookup with a combined I/O instruction produces a savings in programming. Only one instruction is required to transform the contents of the A register, instead of the three required when addressing the table lookup as memory (Figure 10.2.2). For the circuit shown in Figure 10.2.3, that one instruction is INF 006. During PCO-7 time, the contents of the A register go out on the CPU bus and are latched up in the EI register. The EI register addresses the 1702 PROM. Then, during PCO-7 time, the high input enable pulse reads the contents at the selected table address back into the A register. Thus, when the input instruction is complete, the A register contains the same data word which corresponds to the desired function of the old A register.

Not only are there two instructions saved, but there has been no need to disturb the H and L registers. Since these are the only two 8088 registers addressing memory, this enhances programming flexibility.

10.2.3 Four Table Lookups with an 8 K ROM  The Table Lookup option may be applied to large ROMs as well. For example, in Figure 10.2.4, a 1024 x 8 ROM is used to perform four table lookups. The four pages (each 256 x 8) within the PROM cannot be enabled separately through the B08B's two chip enable terminals (GIE and GEE), so a different addressing scheme is devised.
SEC. 10.3  TABLE LOOKUPS (cont'd)

This section applies to the addition of an 8308 to an 8088 system. To begin with, DMA is always high during input instructions, and so cannot be used to distinguish table lookups from one another. Second, since OUTPUT instructions ignore any data that is on the 8088 data bus during FOC-TIA time (that is, DIN time), it is unnecessary to distinguish between input and output instruction modes in this circuit. Since the only purpose of using IHS or DHI in addressing the FRM would be to make this distinction, they too are unused. If it should happen that an output instruction is executed whose instruction code looks just like one of the four table lookup instructions except for IHS and DHI—one of the 8308's stored data words will be addressed; the 8308 will be enabled with DHI; and the table lookup value will be impressed onto the 88 bus. But the CPU is simply not listening.

Fig. 10.2.4--A Single 8 K ROM Chip Serves as Four-Page Table Lookup

The IHS bit connects to the noninverting chip enable terminal of the 8 K ROM. This prevents the chip from being activated during input instructions 3 through 6. Therefore, the circuit of Figure 10.2.4 provides four table lookups, activated by INP 007, INP 006, INP 005, and INP 004. The circuit is quite simple and requires only one integrated circuit.

SEC. 10.3  TABLE LOOKUPS (cont'd)

10.2.4  Trigonometric Tables  In the following example, a lookup table allows a microcomputer to find the sine of an angle, by addressing a 256-location FRM.

Only the sines of angles between 0° and 90° are stored in this table. Values in the other three trigonometric quadrants (90°-360°) may be readily calculated from the first quadrant values. And since sin (90°-θ) = cos θ, the cosine function is also easily derived with a few added instructions.

Both the number representing the angle, and the value of the sine function, are limited to eight bits in this example. The address bits of the FRM must be weighted such that the decimal numbers 090 to 277 correspond to angles between 0° and 90°. Figure 10.2.5 shows a typical scheme.

Address bit 8  A7  A6  A5  A4  A3  A2  A1  A0
Weighting factor 90° 90° 90° 90° 90° 90° 90° 90°

Fig. 10.2.5--Weighting Factors for Address Bits of Sine Function Lookup Table

The range of the values of the sine function is from zero to one. The usual scheme allows all eight bits in the table to represent a binary fraction, with the binary point to the left of the high-order bit. This provides possible sine values between 0 and 255/256 (decimal). Figure 10.2.6 shows the weighting of each data bit which provides for this range of values.

Data bit D7  D6  D5  D4  D3  D2  D1  D0
Weighting factor 1 4 1/8 1/16 1/32 1/64 1/128 1/256

Fig. 10.2.6--Weighting of Data Bits in Sine FRM

The FRM is then programmed to give the best match between angles and their sines. For example, say the FRM address is 01010101, or 125 octal. This corresponds to an angle of 125/400 octal of 90°, or 85/256 decimal of 90°, or about 28°31'. The sine of this angle is 0.40844 decimal (from a trig table). This value is converted into a binary
fraction with a resolution of $1/256$ decimal: $0.49899 \times 256 = 127.61$. The approximation used is $128/256$ decimal, or $0.499$ exactly. This sine value corresponds to a decimal value of 0.500, which is what would be expected for an angle of $90^\circ$. The error is minimal.

The accuracy of the sine value stored in the PROM, assuming that the desired angle is represented exactly by the eight address bits, is plus or minus one part in 512, or better than 0.2%. Assuming as the worst case that the error in representing any given angle with eight bits is additive with the maximum error in the sine table, the lookup table still gives an accuracy better than 0.4%. This is more than adequate for many applications encountered using eight-bit microcomputers.

### Section 10.6 Bits-Swapping

A Swap register is often a very convenient addition to an 8088 microcomputer. The Swap register permits the programmer to exchange, or swap, the contents of the A and Z (Swap) registers with one instruction at any desired place in the program.

A Swap register is connected to input/output port number 5 in Figure 10.5.1. The content of this register is read during PCC-T3A time of an INP 005 instruction. Meanwhile the previous value of the A register has been saved in the DL register at PCC-T1 time. At the end of the PCC-T3A enable pulse, this old value of the A register is loaded into the S register.

Fig. 10.5.1—The Swap Register Connected as I/O Port #5
SEC. 10.3 BYTE-SWAPPING (cont'd)

The chart in Figure 10.3.2 shows in detail how the original contents of the A register (called A1) and the original contents of the S register (called A2) are exchanged with the INF 005 instructions.

The state transition diagram in Figure 10.3.3 shows in a more simplified form the data transfers which take place during an INF 005 instruction, using the Swap register. The order in which the data transfers take place is shown by the numbers in parentheses.

![Diagram showing data transfers](image)

Fig. 10.3.3—State Transition Diagram for Swap Register

SEC. 10.4 ADDING A UART

In many communications applications, data is transmitted asynchronously. For example, a teletypewriter sends and receives characters with pauses of variable length between. On the other hand, the microprocessor is a highly synchronous device, employing a relatively complex clocking scheme and requiring interfacing devices to be synchronized with the CPU. And, in communications applications, it is often convenient to use only one data line—one pair of wires. Data is transmitted serially, with each data word marked off with START and STOP codes. The 8086/8088 microprocessors, again, are eight-bit parallel devices, processing data made up of eight-bit words.

The Universal Asynchronous Receiver/Transmitter (UART) is an extremely useful adjunct to a microcomputer because it interfaces the microprocessor to the serial data world. Since the UART is both an input device and an output device, it provides a specialized example of a combined I/O technique.
SEC. 10.4 ADDING A UART

The UART converts eight-bit microprocessor output words into data by loading the parallel data into a shift register, adding START and STOP codes at the beginning and end of each word. Then all the bits are shifted onto the transmit line at regularly-spaced intervals. Each word is totally self-contained, since it is marked off by its own START and STOP bits, and therefore it is possible to insert a pause of any desired length between words being transmitted. The UART on the other end of the line watches for another START bit, which synchronizes its own receiver clock for proper reception. The relation between two START bits in a stream of data need not be synchronous, because of the variable-length pause which may occur between words.

An example of the most common type of UART is shown in Figure 10.4-1.

There are actually two related devices, with the same pinouts, designed for slightly different serial data characteristics: the 2502 and 2517. The device shown is able to transmit 1.5 stop bits with 5-bit data codes.

This UART is very easily interfaced with microprocessors because of its bus-structured design. Nearly all the output terminals are three-state, and may be connected to the CPU input bus (DB) directly, or to a bidirectional data bus. (Definitions: Chapter 6.) The TR OUT terminal (serial data output) does not, however, employ three-state circuitry, it is the output to the transmission line, and may be connected directly to the TTL-compatible serial data input of a peripheral machine, such as a teletypewriter. The TR EMPTY (transmit register empty) flag also does not employ three-state outputs, so that it can be used to generate an interrupt in the microcomputer, or strobe more data from a F14 register, without first enabling the control section's output circuitry at the 2F ENAB (status flag) terminal.

In the schematic of Figure 10.4-1, the three-state outputs from the UART receiver (RDB through RDO) are connected directly to the CPU DB bus. So also are the three-state inputs from the control section of the UART. These status flags are tested by the microprocessor to control the flow of serial data.

When an INP 003 instruction is executed by the microprocessor, the RXS signal goes low during the 2B time, and the status flags from the UART's control section are enabled onto the CPU data bus. The CPU places this information into its (internal) A register, as it does during any input instruction. The microcomputer tests this information and proceeds through its functions accordingly.

The INP 004 instruction reads the received data from the UART into the microprocessor data bus. It also resets the DMA Enable and DATA READY flags within the UART. To do this, the DBN strobe line is connected to both the READ ENABLE and DATA READY flags on the UART. This allows the microcomputer to receive an eight-bit byte of information and to acknowledge...
reception to the UART—reading the UART for the next byte—with one simple input instruction. This design is an example of how a three-state bus-structured input device may be combined with the pulse output design discussed in Chapter 9.

The control word is loaded into the UART with an OUT 012 instruction. This control word defines the word length, parity, and length of stop bit. When the CPU has determined that the transmit buffer is empty (using an INP 003 instruction and testing bit 5), it may output another word to the transmit buffer using an OUT 012 instruction.

In this design, one LSI integrated circuit, two input instructions, and no further hardware are sufficient to provide an 8080-based microcomputer with asynchronous data communications capabilities. Note that this simple interface depends not only on the internal complexity of the LSI chips involved—the CMOS71—and the microprocessor—but on the bus-structured input approach and the use of devices with three-state outputs.

The circuit shown in Figure 10.4.1 could be used with an 8080 microprocessor as well as with an 8086. However, the microprocessor lines which drive the UART would be changed: substitute B80 for D20, B81 for D21, etc.

The major intent of this section is to show how easily the UART may be connected to a microcomputer. For this reason we are not duplicating information which is readily available in the manufacturer's data sheet (GMS Microsystems Corp., 56 North Blvd., Hapeville, GA 30354). See also the data sheets for the 1601 and 1402 (Western Digital Corp., 19488 Red Hill Ave., Newport Beach, CA 92660) the IMS 6011 (Texas Instruments, PO Box 6011, Dallas, TX 75222); and 551013 (General Instrument Corp., 600 W. John St., Hicksville, NY 11802).

---

**SEC. 11.1 ADDING INSTRUCTIONS TO THE 8080**

A number of 8080 instructions are not used in normal programming. Some of these instructions have been added to provide extra facilities, such as the 8080 chip designers have already used the extra codes which were left over on the 8086, and are shown in Figure 11.1.2. There are no redundant opcodes for the JMP, CALL, or RET instructions in the 8080 instruction set. By reducing redundancy, and by making input/output instructions into two-byte codes, room has been made for a number of extra 8080 instructions. (See Chapter 3.)
### SEC. 11.1 ADDING INSTRUCTIONS TO THE 808 (cont'd)

<table>
<thead>
<tr>
<th>OCTAL CODE</th>
<th>MNEMONIC</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>JMP</td>
<td>USUAL CODE.</td>
</tr>
<tr>
<td>114</td>
<td>JMP</td>
<td>NOT NORMALLY USED.</td>
</tr>
<tr>
<td>124</td>
<td>JMP</td>
<td>&quot;</td>
</tr>
<tr>
<td>134</td>
<td>JMP</td>
<td>&quot;</td>
</tr>
<tr>
<td>144</td>
<td>JMP</td>
<td>&quot;</td>
</tr>
<tr>
<td>154</td>
<td>JMP</td>
<td>&quot;</td>
</tr>
<tr>
<td>164</td>
<td>JMP</td>
<td>&quot;</td>
</tr>
<tr>
<td>174</td>
<td>JMP</td>
<td>&quot;</td>
</tr>
<tr>
<td>106</td>
<td>CAL</td>
<td>USUAL CODE.</td>
</tr>
<tr>
<td>116</td>
<td>CAL</td>
<td>NOT NORMALLY USED.</td>
</tr>
<tr>
<td>126</td>
<td>CAL</td>
<td>&quot;</td>
</tr>
<tr>
<td>136</td>
<td>CAL</td>
<td>&quot;</td>
</tr>
<tr>
<td>146</td>
<td>CAL</td>
<td>&quot;</td>
</tr>
<tr>
<td>156</td>
<td>CAL</td>
<td>&quot;</td>
</tr>
<tr>
<td>166</td>
<td>CAL</td>
<td>&quot;</td>
</tr>
<tr>
<td>176</td>
<td>CAL</td>
<td>&quot;</td>
</tr>
<tr>
<td>007</td>
<td>RET</td>
<td>USUAL CODE.</td>
</tr>
<tr>
<td>017</td>
<td>RET</td>
<td>NOT NORMALLY USED.</td>
</tr>
<tr>
<td>027</td>
<td>RET</td>
<td>&quot;</td>
</tr>
<tr>
<td>037</td>
<td>RET</td>
<td>&quot;</td>
</tr>
<tr>
<td>047</td>
<td>RET</td>
<td>&quot;</td>
</tr>
<tr>
<td>057</td>
<td>RET</td>
<td>&quot;</td>
</tr>
<tr>
<td>067</td>
<td>RET</td>
<td>&quot;</td>
</tr>
<tr>
<td>077</td>
<td>RET</td>
<td>&quot;</td>
</tr>
<tr>
<td>000</td>
<td>HLT</td>
<td>OPTED USED.</td>
</tr>
<tr>
<td>001</td>
<td>HLT</td>
<td>NOT NORMALLY USED.</td>
</tr>
<tr>
<td>377</td>
<td>HLT</td>
<td>OPTED USED.</td>
</tr>
</tbody>
</table>

**Figure 11.1.1—Additional Unused Instructions**

The basic technique for adding instructions to an 808 microcomputer is to decode the execution of these normally unused instructions, and to use additional hardware to implement the desired operation.

---

### SEC. 11.2 OUTPUT ANY REGISTER

An interesting way to put the 808's normally unused instructions to work is to use them to output any register with one instruction. When a self-transfer instruction is executed—IAA, LLB, etc.—the contents of that index register will appear at a specially designated output port.

The key to the Output-Any-Register Option is to use a four- or five-bit comparator IC to decode the instruction. Figure 11.2.1 shows a 93L24 five-bit comparator connected to the output bus of an 808 microprocessor. Its output at pin 1A goes high only during an instruction FETCH cycle (when PCE is low) and only when the following equations are valid: D7 = D6 = D5 = D4 = D3 = D2 = D1 = D0. The first equation implies that the instruction's octal code begins with a 3. The following pairs of three equations means that the second and third digits of the instruction's octal code are the same. This will be seen to define the seven self-transfer instructions, IAA through LLL (see Figure 11.1.1 above).

At PCI-T3 time of a self-transfer instruction, when the instruction code appears on the data bus, the comparator output goes high. At the end of T3A time the 74107 flip-flop is set, causing its complementary output terminal to go low. At ST4 time the flip-flop is reset, and the SDI strobe goes high. This signal clocks a 74773 eight-bit latch, which latches up the data word on the output bus at PCI-T4 time. This word happens to be the contents of the index register referred to by the self-transfer instruction. The reader is referred to the first line of the 808 INTERNAL PROCESSOR OPERATION chart (in the 808 manual, and reprinted in Chapter 2).

Note that the comparator outputs may go high when the proper bit combinations appear on the output bus at PCI-T1 time, when the low-order memory address happens to match up with the code for a self-transfer instruction. But the 74107 is clocked only at the end of T3A time—by which time this memory address will have gone away, the comparator output will be low, and the output port will not be clocked.

**Important:** when the circuit shown in Figure 11.2.1 is being used, the microcomputer should NOT use the 377 HALT instruction. Consider what happens when a 377 instruction is executed. By extrapolation from the binary codes for the seven self-transfer instructions, 377 might be expected to stand for an IAM instruction: LOAD MEMORY FROM MEMORY. There is no such 8008 instruction, however; this is one of the three opcodes for the HALT instruction (Figure 11.1.2). Nevertheless the 93L24 will decode 377 as a self-transfer instruction at PCI-T3 time, and the 74107 will be set at the end of PCI-T3A time. The HALT instruction will stop the microprocessor at the end of PCI-T3 time, and the 808 will wait for an interrupt before proceeding. No T4 time occurs during the HALT instruction, so the 74107 remains set. At T4 time of the very next instruction executed, the flip-flop is reset and an extraneous value...
SEC. 11.2 OUTPUT ANY REGISTER (cont'd)

is strobed through the 74273 output port. Thus, either the 000 or 001 codes should be used for HALT instructions.

Only one output port is provided with the Output Any Register option, unlike normal 8080 output instructions, where the output word can be steered to any one of up to 32 output ports with a single-byte instruction.

Note that though the 8080 has self-transfer instructions—opcodes 111, 110, 111, 11, 10, and 177—this technique is difficult to apply. The 8080 has only one opcode for HALT, namely, 186, which would have to be trapped out and distinguished from the others to avoid the problem mentioned above.

SEC. 11.3 ONE-BYTE PUSH-POP INSTRUCTIONS

When a Last-In, First-Out (LIFO) register is connected in place of the 74273 latch, the Output-Any-Register option allows the programmer to push any index register with one single-byte instruction, and to pop registers with another single-byte instruction.

A schematic for this circuit appears in Figure 11.3.1. (For a discussion of push-pop registers in general, and the LIFO portion of this circuit in particular, see Chapter 11.) A one-byte self-transfer instruction pushes the contents of the appropriate index register into the LIFO, and a single-byte INF 000 instruction pops the last-loaded register and reads it back into the microprocessor.

This circuit is particularly useful in systems making extensive use of the 8080's interrupt-handling capabilities. (See Chapter 17 for a discussion of saving and restoring index registers and flags using push-pop software and hardware.)

Figure 11.3.2 shows the opcodes for the new PUSH and POP instructions provided by the hardware in Figure 11.3.1.

<table>
<thead>
<tr>
<th>OCTAL CODE</th>
<th>NEW MNEMONIC</th>
<th>COMMENT</th>
<th>OLD MNEMONIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>PUSH A</td>
<td>PUSH A REGISTER</td>
<td>LAA</td>
</tr>
<tr>
<td>311</td>
<td>PUSH B</td>
<td>PUSH B REGISTER</td>
<td>LBB</td>
</tr>
<tr>
<td>322</td>
<td>PUSH C</td>
<td>PUSH C REGISTER</td>
<td>LCC</td>
</tr>
<tr>
<td>333</td>
<td>PUSH D</td>
<td>PUSH D REGISTER</td>
<td>LDD</td>
</tr>
<tr>
<td>344</td>
<td>PUSH E</td>
<td>PUSH E REGISTER</td>
<td>LEE</td>
</tr>
<tr>
<td>355</td>
<td>PUSH H</td>
<td>PUSH H REGISTER</td>
<td>LHH</td>
</tr>
<tr>
<td>366</td>
<td>PUSH L</td>
<td>PUSH L REGISTER</td>
<td>LLL</td>
</tr>
<tr>
<td>101</td>
<td>POP</td>
<td>POP LAST-LOADED REGISTER</td>
<td>INF 000</td>
</tr>
</tbody>
</table>

Fig. 11.3.1—One Instruction Outputs Any Index Register to Special Output Port

Fig. 11.3.2—One-Byte PUSH-POP Instructions
The RET, CAL, and JMP instructions for the 8008 microprocessor have in common the fact that the three middle bits—5, 4, and 3—may be varied without changing the normal functioning of the instruction. Thus there are eight codes for each of these instructions. A useful technique is to decode these separate instructions as they are executed and use the numbered instructions for special purposes.

<table>
<thead>
<tr>
<th>NEW MNEMONIC CODE</th>
<th>NEW MNEMONIC CODE</th>
<th>NEW MNEMONIC CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RET 007</td>
<td>JMP 104</td>
<td>CAL 106</td>
</tr>
<tr>
<td>RDT 017</td>
<td>JFP 114</td>
<td>CAL1 118</td>
</tr>
<tr>
<td>RDT2 027</td>
<td>JFP2 124</td>
<td>CAL2 126</td>
</tr>
<tr>
<td>RDT3 037</td>
<td>JFP3 134</td>
<td>CAL3 138</td>
</tr>
<tr>
<td>RDT4 047</td>
<td>JFP4 144</td>
<td>CAL4 146</td>
</tr>
<tr>
<td>RDT5 057</td>
<td>JFP5 154</td>
<td>CAL5 158</td>
</tr>
<tr>
<td>RDT6 067</td>
<td>JFP6 164</td>
<td>CAL6 166</td>
</tr>
<tr>
<td>RDT7 077</td>
<td>JFP7 174</td>
<td>CAL7 176</td>
</tr>
</tbody>
</table>

Fig. 11.4.1—Mnemonics, Codes for Additional Numbered Instructions

The numbered return instructions may be used for resetting an interrupt level, as described in Chapter 16. The numbered jump and call instructions may be used for pushing, popping, or outputting, or as tools in debugging programs and hardware (Chapter 24).

Figures 11.4.2 and 11.4.3 show circuits for decoding the execution of the numbered JMP and CAL instructions.
Both the 8080 and the 8085 have a READY terminal which, when brought low, causes the CPU to cease executing instructions. This function is intended primarily to allow the CPU to operate with slow memory or I/O devices. When READY goes low before T2 time, the CPU cycles in the T3 state and does not proceed further until the READY line goes high once more.

With simple hardware additions, the microprocessor can execute a simple WAIT instruction, causing it to enter the WAIT state. In an 8080 computer, the instruction is only one byte long; with an 8085, using the normal I/O mode, it is two bytes long. Figure 11.5.1 (A) shows a one-shot triggered by an 8080-mode INPUT 7 instruction; the CPU enters a wait state for a period determined by R and C. (For how to calculate the required one-shot period for an 8080, see Chapter 3.) In the circuit shown, the wait state is entered only when the low-order output bit is high, as set up by a previous instruction.

Figure 11.5.1 (B) shows a similar circuit. Here, however, the CPU cannot leave the wait state until a peripheral is ready. This may be useful in interfacing to a floppy disk.

Fig. 11.4.2--Decoding the 8080 Numbered Jump Instructions

Fig. 11.4.3--Decoding the 8080 Numbered Call Instructions
The first eight-bit microprocessor to hit volume production, the 8008 lacks some of the features introduced later with the 8080 CPU. This chapter contains several ideas for expanding the capabilities of the 8008, and may be useful not only for educational purposes, but to the designer considering design updates to an existing 8008 system.

**SEC. 12.2 ONE-BYTE PUSH-POP (LIFO) REGISTER**

12.2.1 LIFO and FIFO Registers  A LIFO register is a temporary data storage device, usually having a capacity of at least several data words. It is organized as a number of single-word registers, often referred to as a stack. As each word is loaded into the LIFO, it pushes down the stack. When a word is read out of the LIFO, the last word loaded in is popped out. Thus the names Last-In, First-Out (LIFO) and push-pop. In operation, the LIFO can be analogized to a spring-loaded cafeteria tray stand, where the last tray laid down is the first picked up. Or the LIFO may be likened to an office elevator at quitting time. When the elevator reaches the main floor, the people who got on last are usually the first to get off. (The analogy breaks down in the morning, because as everyone knows, it is usually the first person who got on, standing at the back of the elevator, who wants to get off on the second floor. The crushed toes which result are the penalty for using the LIFO in a random-access application.)

---

**Fig. 12.2.1—Comparison of LIFO and FIFO Registers**
SEC. 12.2 ONE-BYTE PUSH-POP (LIFO) REGISTER (cont'd)

As a contrast, the FIFO (First-In, First-Out) register is also shown in the drawing. The FIFO may be analogized to a queue of people waiting at a counter: first one, first served. The same SIZO register is also used, by analogy to a silo on a farm, where the crop first stored is first removed from the bottom.

Both kinds of registers are often handy in a computer. For example, when returning from a subroutine in a program, the Return instruction refers automatically to the program location which called the subroutine in the first place. The memory location of the last call instruction needs to be stored--stored last and unloaded first. In practice, there is usually a stack with a capacity of many words of data, so that subroutines may be nested many levels deep. Subroutine A calls subroutine B, which calls subroutine C, etc.; then they return to the main line program in reverse order. In a related application, the contents of several data registers may need to be stored at the beginning of a subroutine, then restored at the end. (For example, during interrupts: see Chapter 17.) These examples call for a LIFO register, where the data most recently stored is the first retrieved.

On the other hand, data coming from a peripheral device is frequently handled on a first-come, first-served basis. and then a FIFO register is desirable.

12.2.1 One-Byte LIFO Register A one-byte register may be considered either a LIFO or FIFO register, of course. In an 8080-based microcomputer, a single-byte register is useful for temporary storage of the A register--for example, during interrupts. Since this function seems more closely analogous to the usual functions of the LIFO, the one-byte design in Figure 12.2.2 is labeled as such. This design is for handling eight-bit bytes, as are the other LIFO and FIFO designs in this chapter.

Fig. 12.2.2--One-Chip, One-Byte LIFO Register

This single-chip storage register is loaded when an output instruction OUT 017, is executed. The information is retrieved using an input instruction, INP 007. Unlike most multi-word LIFO and FIFO registers, the data stored may be retrieved (read out) as many times as desired without changing the information; only a new OUT 017 instruction changes the data. (If this one-byte register is both a LIFO and a FIFO, it is also a one-byte RAM, which is addressed using I/O instructions rather than memory read/write instructions.)

The circuit shown in Figure 12.2.3 is functionally identical, but shows the separate storage and retrieval functions more clearly. The one-byte register consists merely of an eight-bit latch and an eight-bit buffer with three-state outputs. The latch constitutes an output port, connected directly to the buffers, an input port. If a second byte of storage is needed, another 74S374 chip could be connected in a manner similar to the circuit in Figure 12.2.2, but using different input and output instructions.
Fig. 12.3.2—A One-Byte LIFO Using Separate Latches and Buffers

SEC. 12.3  SIXTEEN-BYTE LIFO REGISTER

If more than two locations of LIFO storage are needed, then a sixteen-byte LIFO register may be constructed using three chips and a pair of gates (Figure 12.3.1). The 16 x 4 RAM memory chips are used for data storage, and a four-bit up/down counter (plus two logic gates) are used to provide the last-in, first-out characteristic.

When information is pushed (loaded) into the LIFO, or popped out (retrieved), the data does not actually move within the RAM. Instead, the pointer to the memory is stored in the 93L65 up/down counter, and moves with each operation. When an OUT 017 instruction is executed, the eight-bit data word is stored in the RAM, and then the RAM address is incremented. Since the DT27 strobe pulse is inverted, the incrementing of the 93L65 does not occur until the end of the strobe. And since the 74S189 is faster than the 93L65, the incrementing of the counter does not affect the RAM address until after the AE (write enable) signals has disable the 74S189 RAM chips. In this way a potential race condition is avoided.
When a byte of data is being popped from the LIFO, the address must be decremented before the data is read. This is accomplished by stroking the 9366 count-down command (OUT) with a microcomputer main timing signal—here, 8T of the 8008 clock. This occurs early enough in the T3A phase to precede the actual reading of data by the CPU. If this technique unduly restricts the CPU speed, a flip-flop may be used to move the decrementing operation back to the beginning of t1 time (Figure 12.3.2(a)). Even more speed may be obtained by using a one-shot for this function (Figure 12.3.2(b)).

Fig. 12.3.1—Timing Modifications to Sixteen-Byte LIPO for Greater Speed

Note that the MR (master reset) pin of the up/down counter in Figure 12.3.1 is connected to the microcomputer master reset signal. This resets the counter to 0000 when power is first applied. Of course, the circuit would still function if the initial counter output level were any random value. The microcomputer doesn’t see the counter output address; the important thing is that the counter moves up and down through the stack as it receives the appropriate instructions. However, making the counter output addresses consistent from power-on to power-on, and machine to machine, facilitates debugging of defective chips.

An added feature of this circuit is that the LIFO address can be preset to a given level in the sixteen-level stack by using an OUT O27 instruction, whose four low-order bits load the up/down counter. If this function is unneeded, the counter’s PL (parallel load) and parallel data (Pd through PD) can be connected to LONE (logic one: +5 volts through L01D0 resistor). On the other hand, if the parallel load feature is included, it may be used to replace the MR signal by including a few instructions in the microcomputer’s initialization.
SEC. 12.4 A 32-BYTE FIFO REGISTER (cont'd)

Fig. 12.4.1--A 32-Byte FIFO Register for Internal CPU Operations

SEC. 12.5 OTHER 8008 IMPROVEMENTS

This book contains a number of other methods for expanding 8008 capabilities, which are organized topically in other chapters of this book. They may be used singly, and in many cases in combination, to make the most of the 8008's capabilities. Most of the ideas presented involve the addition of only one or several integrated circuits to the basic 8008 microcomputer, plus, in many cases, some added instructions in the microcomputer's software. The purpose is twofold: first, to show how these extra functions can be added to a practical 8008 microcomputer; and second, for educational reasons, to demonstrate how basic microcomputer elements work.

Some expansion ideas are the following: expanding the number of input ports (Chapter 8); expanding output ports (Chapter 9); added instructions (Chapter 11); and interrupt handling (Chapters 15 and 17). Other sections of the book are devoted to simplifying the basic 8008 microcomputer—which is another way of getting the most out of a CPU.
NEED FOR MEMORY IN MICROCOMPUTERS

Naturally the microprocessor is the heart of any microprocessor system. But memory is equally essential, and, in many microcomputers, represents more of the system cost than the microprocessor itself. A microcomputer needs digital storage elements external to the microprocessor for two reasons:

(1) Programming—Because the microprocessor is a general-purpose computing device, it must be programmed to perform the particular functions needed in the system in which the microcomputer is used. The CPU derives its instructions from memory, and includes an internal program counter which keeps track of (reference) the memory location from which instructions are being read.

(2) Temporary data storage—Because the microprocessor itself has relatively little internal data storage capacity, most microcomputers need additional temporary data storage, usually provided by RAM. The CPU includes internal registers which reference the memory location for writing data into memory (in the case of RAM) and reading data from memory (ROM or RAM).

The following section details the way in which microprocessors use memory, both RAM and ROM. Later sections concentrate on RAM, and Chapter 14 focuses on ROM.

MEMORY REFERENCING

The Program Counter (PC) register within the CPU contains the address of the next instruction to be fetched and executed. The address of memory into which data can be written (or from which data can be read) is typically stored in the H and L registers in an 8080—though the BC and DE registers pairs may also be used. (Only the H and L registers are used for this purpose in an 8080.) During any instruction which references memory as a source to read from or write into, the appropriate memory address appears on the microcomputer's address lines, so that the selected location can be activated for the upcoming data transfer. The 8080 has 16 address terminals on its 40-pin package for this purpose, while the 8008 requires external BR and LR registers (Ch. 5) to address 16 lines. The ways in which memory is referenced are summarized below.

1) PROGRAM COUNTER (16 bits, 8080; 14 bits, 8008)
   Address for READING instructions from memory.

2) H AND L REGISTERS (or B and C, D and E in 8080)
   a) Address for READING data from memory.
   b) Address for WRITING data into memory.
SEC. 13.2 MEMORY REFERENCING (cont'd)

An 8080 instruction such as ADD (add memory) first reads a byte from memory, at the location specified by the H and L registers; then the CPU adds this value to the contents of the A register. The 8080 instruction ADD M is identical in substance.

1  LMR Load Memory from register r.
2  LME Load Memory Immediate.
3  LdM Load register r from Memory.
4  ADM Add Memory to the A register.
5  ADD Add (with Carry) Memory to the A register.
6  SSBM Subtract Memory from the A register.
7  SSM Subtract (with Borrow) Memory from the A register.
8  NSM AND Memory with A register.
9  XORM EXCLUSIVE OR Memory with A register.
10  ORM INCLUSIVE OR Memory with A register.
11  CMF Compare Memory with A register.

Figure 13.2.8—The 11 8080 Instructions Which Reference Memory Using the H and L Registers as an Address

The 8080 includes all of the above instructions—however, note that the mnemonics usually used differ from the 8080 mnemonics. In addition, the 8080 includes a number of instructions which refer to memory in other ways. For a graphic presentation of these 8080 instructions—including double-precision transfer instructions; indirect transfers; and direct load and store instructions—see the chart in Chapter 3 (Fig. 3.3.5) above.

SEC. 13.3 MEMORY ADDRESS CONVENTIONS

Within the 8080, memory is referenced by the fourteen bits of the internal program counter (with regard to fetching instructions) or the fourteen bits of the internal H and L registers (with regard to reading and writing data). These fourteen bits address up to $2^{14}$, or 16 x 1024, or 16 K bytes of memory.

In the 8080, the 16 address lines reference $2^{16}$, or 64 K bytes. These address lines may flexibly be loaded from various CPU registers as well as the H and L. (Again, see Chapter 3.)

SEC. 13.3 MEMORY ADDRESS CONVENTIONS (cont'd)

The six high-order memory bits (HSH through DH0) address 64 (decimal) pages of memory, where a page is a group of 256 bytes addressed by the eight low-order memory (DL) bits. The addresses are often written in octal notation to correspond to the eight-bit architecture of the microcomputer; thus, each page begins with location 0000 and runs to 3777 (for 256 decimal bytes).

The full memory address is, in this book, transcribed in split octal notation, where there are six octal digits. The right-hand triplet corresponds to the H register and the left-hand triplet corresponds to the L register. Thus a kilobyte RAM array (Figure 13.5.3) whose first location is 000000 runs through location 005377 in split octal notation. This notation clearly indicates that a 1 kilobyte RAM consists of four pages (000, 001, 002, 003).

Figure 13.3.1 shows the correspondence between the states of the fourteen binary address bits, and the split octal notation for that memory address. The asterisks appear above a sixth high-order octal digit which is often seen in 8080 literature, and is used to address PROM programming stations if included in the system. The 8080 uses all 16 bits.

<table>
<thead>
<tr>
<th>HSH</th>
<th>DHS</th>
<th>DH1</th>
<th>DH0</th>
<th>DL7</th>
<th>DL6</th>
<th>DL5</th>
<th>DL4</th>
<th>DL3</th>
<th>DL2</th>
<th>DL1</th>
<th>DL0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 13.3.1—Split Octal Notation

SEC. 13.4 RAM-PAGE OPTION

13.4.1 How It Works The preceding sections set the stage for the introduction of a valuable design technique called the RAM-PAGE OPTION. This technique involves both hardware and software changes, and both eliminates hardware, and cuts down on program size and execution time for an 8080 microcomputer. RAM-PAGE also frees up the use of one of the index registers internal to the CPU (usually the D register), which is useful in handling interrupts. The limitation of the design is that it can be used only in systems with a single page (256 bytes) of RAM, or less.
As discussed previously, instructions which read from memory may refer either to RAM or to ROM. Write instructions, however, are normally used only with RAM, since by its very nature ROM (read-only memory) cannot be written into by the microcomputer. Since, with the RAM-PAGE option, there is only one page of RAM, the microcomputer may be designed to refer to that page automatically whenever a memory write instruction is executed. In other words, the page number is ignored during memory write cycles.

The 8080 instructions which cause a memory write cycle are IMA, ICR, LDR, LSR, LNE, DS, and UNA. When these instructions are used in a microcomputer equipped with the RAM-PAGE option, the programmer need not set up the H register to point to the RAM address.

Figure 13.4.3 shows two sample 8080 programs written for systems with and without this option. The program written without uses two more bytes of program storage and takes 50% more time to execute.

The amount of savings attainable with RAM-PAGE varies greatly with the application and the style of program used. There is no doubt that many low cost, high production-volume microcomputers are being designed with only one page of RAM. As long as there is no intention of expanding the amount of RAM in the future, one should always opt for the RAM-PAGE approach. This is especially true since this option requires no additional hardware.

(However, note that other special purpose options described in Chapter II may conflict with the RAM-PAGE option.)

### 13.4.2 Another Ram-PAGE Advantage

The RAM-PAGE option is most attractive in systems which have a need for interrupts. When an interrupt occurs, one has only to save the L register before being able to store the remaining registers in RAM. Without RAM-PAGE it is necessary to save both H and L registers before the remaining registers can be stored in RAM. In many current designs the interrupt routine moves the H register to the D register and the L register to the E register, then sets the H and L to point toward RAM, and stores the other registers in RAM. With the RAM-PAGE option only the L register need be moved to the E register before the saving of other registers in RAM can begin. This frees up the use of the D register for some other purpose. This extra register makes the programmer's job much easier and can cut the instruction count significantly. See Chapter 19 for more information on how the RAM-PAGE Option is used with interrupt programs.

In 8080 systems, a page or more in RAM is usually reserved for the program stack. The 8080 PUSH and POP instructions facilitate interrupts.
13.4.3 RAM-PAGE Hardware Design

Usually single 256 x 8 (or dual 256 x 4) bit RAM chips are used for designs needing only one page (256 bytes) of RAM. The more modern designs usually use a pair of 256 x 4 bi-directional bus-type RAMS connected directly to the CPU bus. For example, see Figure 13.4.1.

Figure 13.4.1—One Page of RAM Added Directly to CPU Bus

The RAMSEL (RAM select) signal in Figure 13.4.1 should go to logic one only when the IN register is selecting RAM, and COL is logic zero.

The microcomputer design example in Chapter 38 shows how the RAM-PAGE option is implemented in hardware. See also paragraph 13.6.4.

13.5 RANDOM ACCESS MEMORIES

13.5.1 General

There are many RAM ICs available today. Of these, there are two basic types: (1) static and (2) dynamic.

The static RAM holds the data which has been stored in it as long as power is being applied. Normally, the data in a static RAM is stored in memory cells similar to flip-flops. Some of these RAMs will hold data when only a fraction of the normal voltage is being applied (e.g., Advanced Micro Devices 9102).

In the dynamic RAM, information is usually held by stored charge on the capacitance of each memory cell. The charge on this capacitor leaks off after awhile so that it is necessary to refresh the information stored periodically.

13.5.2 RAM For Small Microcomputers

Many small microprocessor systems require only a few locations of RAM. Some are so small as to require none. (See the NMI in Chapter 36.)

One of the smallest 8-bit RAM arrays consists of two 16 x 4 bipolar RAM chips and associated read-write logic. Figure 13.5.1 shows a simple

Figure 13.5.1—Sixteen Bytes of RAM with Two 16 x 4 RAM Chips
circuit which uses these chips. 16 x 4 bipolar RAM chips are easily obtained from a number of manufacturers, but have the drawback of inverting stored data. Inverters may be added to the inputs on the program to compensate for inverting data. With the 8008, an MCR 777B instruction will complement (reinvert) the contents of the A register.

The 74S189 is a three-state device; the 3101 and 7485 are open-collector types, which require pull-up resistors on the output bus (not shown).

SEC. 13.6 THE 256 x 4 RAM

A number of 256 x 4 RAMs are available. Most of these chips use the 5-volt N-channel technology. These memories are not only fast enough for the 8008, but are available in high-speed versions compatible with the 8000.

The fact that RAM may be added to the microcomputer with only two RAM chips and a few gates often makes these memory ICs very attractive for small 8-bit microprocessor systems.

There are three basic types of 256 x 4 RAM available. They may be generally categorized by the number of pins in their IC packages.

13.6.1 22 Pins The 22-pin versions have separate input and output pins for data. Figure 13.6.1 shows the schematic of the basic 256 x 4 RAM.

The 2101 chip manufactured by Intel has one chip enable (CE), and although there is no separate output enable, the chip enable pin will float (three-state) the outputs when the chip is not selected. The chip schematic is the same as shown in Figure 13.6.1 except for the lack of OE and CE. The 2101 is usually less attractive for microprocessor designs with a small amount of RAM than are the 16-pin or 18-pin versions. However, a pin-compatible CMS version, the Intel 5101, is now available, and is very attractive for battery-backed memory systems.

Figure 13.6.1--Schematic of 256 x 4 Random Access Memory Chip. The Chip May Be Packaged in 22, 18, and 16-pin Configurations.
For systems with a larger amount of RAM, the 2122 (1024 x 1) chip is more attractive, because of its smaller package (16 pins) and lower price.

13.6.2 18-Pin The 18-pin versions of the 256 x 4 RAM are now available from many suppliers. They are pin-for-pin compatible, following the design shown in Figure 13.6.1. The 18-pin version is just like the 22-pin version, except that DI4 and DI0 are connected internally in the IC package, as are the other pairs (DI3, DO3; DI2, DO2; DI1, DO1). This creates a 4-bit bidirectional bus structure, as compared to the 22-pin version, which may be connected to two separate busses.

13.6.3 18-Pin The 18-pin versions of the 256 x 4 RAM are probably the most attractive designs for 8008 and 8080 designs requiring only a small amount of RAM. Some designers prefer the 18-pin version for 8080 designs, but consideration should be given to the 16-pin configuration before making a decision. There are currently two types in the 16-pin version (Signetics and Intel). The Signetics 2606 chip was the first 256 x 4 RAM. The Intel version is not pin compatible with it. The major advantage to the Signetics 2606 and the Intel 2112-1 is their faster speed. The circuit of Figure 13.6.2 may be used with the 2606 and 2112-1 and 8080 microprocessor, even though the chip enable pulse on writing is only 500 nanoseconds.

The slower 2112 specification requires a 750-nanosecond pulse, which means that more complicated circuitry is needed to drive these RAM chips. Figure 13.6.3 shows a design for driving the 2112 RAM.

The Intel 18-pin version, the 2111, requires that READ ENABLE be connected to the OE pin (see Figure 13.6.9).

Figure 13.6.2--16 Pin Medium-Speed RAMs (Two 256 x 4 Chips)

Figure 13.6.2--Two 256 x 4 Medium-Speed RAMs for an 8008 Microcomputer with the RAM-PAGE Option
Of the circuits shown in Figures 13.6.2, 13.6.3, and 13.6.4, the first is the simplest and would seem to be the most attractive. Whether it is really suitable depends in part on the 256 x 4 RAM's clock specifications. Figure 13.6.6 shows the timing diagram for the circuit of Figure 13.6.2.

The symbol $T_{cy}$ is used here to designate the minimum time for which the RAM must be enabled to write properly. The CPU clock circuit is assumed to be of the symmetrical variety (as discussed in Chapter 5). It may be seen that the $T_{cy}$ pulse is one-eighth of the chip's cycle time, $T_{cy}$. For a standard 8008 microcomputer, where $T_{cy}$ is 4.0 microseconds, $T_{cy}$ would come to 500 nanoseconds.
13.6.6 Random Access Memory

13.6.6 256 x 4 Circuit Without RAM-PAGE Option

When it is desired to add a page of RAM to a microcomputer which does not use the RAM-PAGE option, the circuit in Figure 13.6.6 can be used. The signal labeled RAM ENABLE may derive from one of the eight outputs of a decoder, such as the 74LS138. The inputs to the decoder would come from the I/H register. The T3A enable signal is combined, in this circuit, with the other RAM select logic. The control signals are for an 8080 microcomputer system.

---

13.7 Random Access Memory

13.7.1 The 2102 1K RAM

The 2102 static RAM has gained much popularity among designers of microprocessor systems and of other machines requiring large configurations of medium-speed memory. The 2102 is available from many suppliers. Power supplies for the 2102 are convenient: +5 volts and ground. The 5-volt N-channel technology used in the fabrication of this chip has improved considerably; while still slower than the 17-volt N-channel process used for high-speed RAMs, the 2102 is not only adequate for 8080-based microcomputers, but in the 2102 A version, compatible with the 8080.

A one-kilobyte (1024 x 8) RAM array using 2102 chips is shown in Figure 13.7.1. The outputs are three-state and may therefore be connected to the memory data input bus (MD bus) or directly to the CPU bus (8-8 bus).

13.7.2 Access Time and Address Bit Connections

In working with the 2102 and other RAM chips with more than 256 locations, the designer should be aware of the considerations which affect access time.

For example, if the address bits have been stable for several microseconds or more, and the chip enable (CE) pin is brought low, the access time is effectively 75ns, which is usually much faster than the chip's rated cycle time.

In an 8080-based microcomputer—to review what has been presented earlier in this chapter and in Chapters 2 and 5—the eight-bit low-order memory address comes out on the CPU bus at T1 time. The high-order bits are present on the bus at T2 time. The main timing circuitry develops strobe signals called ST1 and ST2, which occur slightly before the end of T1 and T2 times (respectively). These strobes are used to activate external eight-bit latches, called the IL and IL registers, which latch up the memory addresses. The delay in developing the strobes ensures that the information has had time to settle on the CPU data bus before it is latched up in the registers.

Thus the eight low-order address bits are available at ST1 time, and the high-order bits are available at ST2 time. Memory accessing itself usually takes place during the low-order memory address cycles, and ST3R for write cycles. This means that the low-order address bits have plenty of time to travel from the IL register and settle down reliably into the low-order memory address ports. Therefore memory access time is effectively determined by the chip select time and by the settling time of the high-order bits.

In an 8080 microcomputer, all 16 address bits become valid at the same time. For 8080 memory speed requirements, see Chapter 3.
In the diagram shown in Figure 13.7.2, the M register is loaded on the falling edge ST2 (normally defined as SY times 82). It is obvious that the high-order access time Hacc is the most critical.

Figure 13.7.1--A One-Kilobyte RAM Using Eight 2102 RAM Chips

Figure 13.7.2--Block Diagram and Logic Symbol for 2102 1K RAM
SEC. 13.7 THE 1024 x 1 STATIC RAM (cont'd)

For most memories and most 8008 systems, the access time is long enough for reliable operation. However, by using 61 rather than 60 to develop the STIA strobe, the designer increases this almost 100%. Or, STIA may be used: see Figures 13.7.4 and 13.7.6.

Most RAM chips have some addresses which are more quickly accessed than other addresses. For example, see the block diagram of a 2102 RAM in Figure 13.7.3. The access time for the column decoder section is shorter than that for the row decoder. This means that the high-order (IN) address bits should always be connected to the column decoder terminals, leaving the remaining column decoder terminals (the slower ones if this is known) and the row decoder terminals for the low-order address bits.

Figure 13.7.4 shows the results of access time experiments on a typical 2102 RAM chip.

<table>
<thead>
<tr>
<th>ADDRESS BIT</th>
<th>ADDRESS TO OUTPUT DELAY (nanoseconds)</th>
<th>ADDRESS DECODER AFFECTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>A9</td>
<td>400</td>
<td>COLUMN</td>
</tr>
<tr>
<td>A8</td>
<td>415</td>
<td>COLUMN</td>
</tr>
<tr>
<td>A7</td>
<td>415</td>
<td>COLUMN</td>
</tr>
<tr>
<td>A6</td>
<td>425</td>
<td>COLUMN</td>
</tr>
<tr>
<td>A5</td>
<td>415</td>
<td>COLUMN</td>
</tr>
<tr>
<td>A4</td>
<td>535</td>
<td>ROW</td>
</tr>
<tr>
<td>A3</td>
<td>535</td>
<td>ROW</td>
</tr>
<tr>
<td>A2</td>
<td>505</td>
<td>ROW</td>
</tr>
<tr>
<td>A1</td>
<td>515</td>
<td>ROW</td>
</tr>
<tr>
<td>A0</td>
<td>525</td>
<td>ROW</td>
</tr>
</tbody>
</table>

Figure 13.7.4--Access Times for Different Address Bits of a Typical 2102 RAM Chip

If the chip were connected in an 8008 system with bits A0 and A9 connected to IN1 and IN9 (as shown in Figure 13.7.1), the address-to-output delay would be 415 nanoseconds (worst case between A8 and A9). This delay would, if reliably repeated in production quantities, allow use of the 2102 in an 8008 system operating at standard 8008 speed. The designer could do without having to specify the premium 2102-1, guaranteed for a maximum address-to-output delay of 500 nanoseconds.

Fig. 13.7.6--Generating the STIA Strobe for Slow Memory Chips

Fig. 13.7.6--How the STIA Strobe Fits into the 8008's Main Timing
SEC. 13.7 THE 1024 x 1 STATIC RAM (cont'd)

Though manufacturers of 2102 RAMs may not be anxious to select devices based on their A9 and A8 access times, it is useful to the microcomputer designer to know about the practical margin of up to 120 ns between the chip's specified speed, and the speed needed in an 8080 system.

The above timing considerations relate to the way memory is addressed in 8080 systems. The 8080 outputs a low-order address first, then the high-order address a full clock cycle later. In an 8080 system, all 16 address bits become valid simultaneously, usually about the time [0] is ending within T1 time. During T2 time, the CPU checks to see whether the memory requests a wait cycle; and at T3, the memory data transfer occurs. Factors influencing memory access time in an 8080 system are discussed in Chapter 3 above.

SEC. 13.8 THE 4096 x 1 DYNAMIC RAM

A number of 4K RAMs have been designed to date. It is not yet clear which designs are destined to become industry standards. The only thing that seems certain is that the single-transistor cell has been accepted as the most efficient design approach.

The key to the mass production of large memories using one-transistor cells is the use of on-chip differential amplifiers. The information read is compared with a voltage half-way between logic one and zero. In some designs, the actual value of the reference voltage is subject to the same process variables as the cell voltage levels. This means that process variations tend to cancel out. Another interesting design concept is to make the reference voltage depend on VCC. The threshold voltage of the chip may be varied by changing the power supply voltage, thus testing the safety margins of the cells' voltage levels. The manufacturer can then test chips to pass given internal noise margin specifications.

The first single-transistor-cell 4K RAM designs to appear on the market were the Texas Instruments 4030 and the Mostek 4096. Both have already established a place in the 4K RAM market.

The major difference between the 4030 and 4096 is the number of pins on the package. Figure 13.8.1 shows the approximate relative size of the 22-pin and 18-pin packages. The 22-pin package terminals are on 400-µin dual-in-line centers, while on the 16-pin package the pin rows are 800 miles apart. The 22-pin package is about 1.1 inches long, while the 16-pin version is about 800 miles long. (Approximate dimensions are given because of variations in packaging style.)

Fig. 13.8.1—Relative Size of 22- and 16-Pin Packages (Approxi. 2 to 1)

4 K dynamic RAM designs usually use a six-bit multiplexer to switch the column address bits from the computer address bus to the memory refresh bus. (The refresh bus is usually connected to the outputs of a six-bit binary counter.) The Mostek designers decided that, since a 6 x 2-bit multiplexer is required for the refresh circuit anyway, why not use a 6 x 4-bit device and multiplex the twelve bit addresses onto the same six-bit bus. This design philosophy reduces the connections to the outside world, and also cuts in half the number of address drivers required.

The Mostek design also provides for TTL-compatible logic levels on all of the input and output terminals of the 4096. The 22-pin designs require a 12-volt swing on the chip enable pin. Still another advantage of the 16-pin design is that data is stored and read back in true logic form, while the data read from the 22-pin version is inverted from the data written. This last is particularly bothersome in small memory systems, which might otherwise be designed without any input or output buffering at all.
### SEC. 13.8 THE 4096 x 1 DYNAMIC RAM (cont'd)

<table>
<thead>
<tr>
<th>MOSTEK 4096</th>
<th>TI 4090</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pins on package</td>
<td>16</td>
</tr>
<tr>
<td>Chip enable voltage</td>
<td>TTL</td>
</tr>
<tr>
<td>Address line drivers</td>
<td>6</td>
</tr>
<tr>
<td>Input to output data</td>
<td>true</td>
</tr>
<tr>
<td>Data inverters needed</td>
<td>none</td>
</tr>
<tr>
<td>Board area</td>
<td>less</td>
</tr>
<tr>
<td>Refresh circuitry needed</td>
<td>... about the same ...</td>
</tr>
<tr>
<td>Other manufacturers, second sources</td>
<td>Fairchild, Electronic, and many others</td>
</tr>
</tbody>
</table>

Fig. 13.8.1—Comparison Between 16- and 22-Pin 4 K RAMs

Texas Instruments produces the 4050, an 18-pin 4K RAM, and a 4060—which is like the 4090, except with VBB specs of 5 volts. Mostek has gone with VBB = 5 V, and Intel second-sources this 18-pin 4K RAM version.

It is possible to read convincing claims that both the 18-pin 4K RAM, and the 22-pin 4K RAM, are the industry standard, each in exclusion of the other. The market in question includes large computer mainframes, the expectation being that the 4K RAM will finish off core memory. However, acceptance of 4K RAMs has been slower than initially expected—partly because of the lack of agreement on an industry standard device. Already the focus has shifted to the 16K RAM, being delivered in prototype quantities as this book went to press.

Note that TI offers an 18-pin 4K RAM package. But because the 16K RAMs now being delivered are in 16-pin packages, the 18-pin design seems unlikely to gain wide acceptance.

### SEC. 13.9 TRANSPARENT DYNAMIC RAM REFRESH

Dynamic RAM arrays may be refreshed in a microcomputer without causing the CPU to halt, wait, or otherwise pause. A transparent refresh circuit works by inserting memory-refreshing cycles into gaps in CPU timing without disturbing the flow of processor instructions.

With an 8080, the key is to use the CPU's bidirectional data bus during T2B time. This is that portion of the T3 state immediately following T1A time; or, in other words, the second half of T3 time.

Since the ST2 strobe occurs during T3B time, and this strobe (or a derivative of this strobe) is used when writing into memory during P3 cycles, memory cannot be refreshed at PM-T3B time. Thus a new signal is defined, X88, which is equal to T2B + T2.

A practical RAM refresh circuit is implemented by scrambling the low-order address bits while the chip is disabled, but in the WRITE mode. This scrambling may be accomplished by driving a three-state binary counter onto the EB bus, and latching the EL register during E3B time. Between each E3B refresh cycle, the three-state counter is incremented.

Since the CPU has already completed its use of the EL register by E3B time, there will be no conflict between program execution and RAM refresh.

Figure 13.9.1 shows the memory refresh address counter. Other circuitry necessary includes a strobe and some gating to load the EL register at E3B time; gating to continue the refresh during the STOPPED state, if necessary; and specialized refresh driving circuitry suited to the design of the dynamic RAM chips being employed.

In an 8080 computer, transparency may not be complete, depending on the speed of the processor and the refresh time required by the memory. A convenient gap in processor timing occurs during the T4 state of an MI (Instruction fetch) machine cycle. The refresh circuit should request a hold from the 8080 during the MI cycle; the 8080 acknowledges this request, floating its address and control buses, allowing the refresh controller to address memory and perform the desired refresh cycling. Since only internal data transfers take place during T4 and T3 time, the CPU will continue through these cycles. If refresh is accomplished quickly, then the processor can be allowed to return to normal operation without skipping a beat. Otherwise a machine cycle will be taken up by the refresh circuitry.

The designer should study the hold timing chart in the 8080 data sheet. See also the 871 data sheet, Sec. 1.3.
While core memories are being replaced by solid state memories in more and more data storage applications, it is easy to forget a prime advantage of these magnetic devices. That is, core can both be read from and written into quickly at speeds compatible with data processing needs, and can also retain information in the stored magnetism of the individual core elements when the system power supply is turned off.

With semiconductor memories, as indicated in the previous chapter, the designer chooses between RAM and ROM. Random access memory (RAM) has only one of the capabilities of core memory—fast reading and writing—but not the other, non-powered storage. The read-only memory (ROM) retains its information even when power is not applied; but is not easily changed (written into) during normal computer operations. There is little doubt that RAM memories—random access nonvolatile devices—will eventually become available. However, practical microcomputer designs at present depend on ROM and RAM.

A typical microcomputer system contains a block of ROM where instructions and data are stored permanently, and another block of memory, made up of RAM, which is used for temporary storage. Because these two types of memory are separate, rather than intermingled, the instructions found in microprocessors are somewhat different from those used in computers based on core memory. A very specific example would be an instruction like Store Forward ??, which might be found in a core-based microcomputer. This instruction causes a digital word to be addressed at a location which is ?? (total) locations forward with reference to the current program counter location. In a microcomputer, since there are two different types of memory—one of which cannot be written into—this kind of relative memory addressing is not practical. In the 8086, for example, memory is always referenced through address pointers (H and L) and instructions like DMA are found in the instruction set.

Though a small microcomputer can conceivably do without any RAM at all, it must have at least a small amount of ROM—at least when general-purpose microprocessors like the 8086 and 8088 are used. Without ROM, the microprocessor could not be programmed at all. This chapter presents some practical ROM designs, starting with the very small.
### SEC. 14.1 ROMS AND OTHER MEMORIES (cont'd)

<table>
<thead>
<tr>
<th>TYPE OF MEMORY</th>
<th>DESCRIPTION</th>
</tr>
</thead>
</table>
| CORE | Magnetic core memory.  
Non-volatile: retains data without power.  
Each bit or word alterable. |
| RAM | Semiconductor random access memory.  
Volatile: needs power to retain data.  
Each bit or word alterable. |
| ROM | Mask-programmed read-only memory.  
Non-volatile: retains data permanently.  
Programmed at IC factory.  
Not alterable. |
| F/ROM | Field-programmable read-only memory.  
Non-volatile.  
May be programmed after packaging.  
Once any given bit is altered from its original state, cannot be re-altered. |
| RePROM | Reprogrammable ROM (light-erasable).  
Non-volatile.  
May be programmed after packaging.  
Once any given bit is altered from its original state, cannot be re-altered by the PROM programmer.  
All of the bits may be erased simultaneously to the original logic zero state by exposing the chip to UV light. |
| E/PROM | Electronically erasable version of RePROM. |
| E/ROM | Electrically alterable ROM.  
Non-volatile.  
Bits individually alterable. |
| RAN | Random access non-volatile memory.  
Core replacement. |

**Fig. 14.1.1--Types of Memory**

### SEC. 14.2 DIODE PROGRAMMABLE ROMS (DAROM)

A simple ROM array can be constructed using TTL decoder circuits and diodes. The program can be changed easily merely by removing and adding diodes. The circuit shown in Figure 14.2.1 provides sixteen bytes of DAROM.

One sixteen-pin DIP socket is dedicated to each memory location. The eight pins on one side of the socket are all connected to one of the output lines of the 7414 decoder. The eight pins on the other side of the socket are connected to the eight separate data bus (BB) lines of the microcomputer. When using an 8088 or 8086, the BB lines should be provided with pullup resistors in the range of 10K to 22 K ohms which define the logic high state on the BB bus. The zero state is defined by the presence of a diode at the selected output address of the decoder, opposite the BB bit in question. If location 100000 was read from the circuit shown in Fig. 14.2.1, the result would be a binary 00000011 on the BB bus.

Though handy in breadboarding and prototyping, this design is hardly cost-effective in production, of course. Another limitation has to do with the logic LOW input voltage requirements of the microprocessor. For example, the 8080 is specified to recognize a voltage of 0.4 volts, or 0.8 volts, as the logic low voltage. Germanium diodes would have to be used for assurance that the 0.8-volt limit was not being exceeded. In practice silicon diodes are adequate for breadboards to be tested at room temperature. However, this design is recommended only for prototyping, in low-noise environments.

### SEC. 14.3 FIELD-PROGRAMMABLE ROMS (F/ROMS)

#### 14.3.1 F/ROMS

The field-programmable ROM can be programmed electrically after the IC is packaged. Most F/ROMs incorporate a matrix of tiny fuses which may be blown out selectively to produce the desired bit pattern of ones and zeros. Most F/ROMs initially contain all zeros, and programming permanently stores ones in the locations whose fuses have been blown. The voltages required to program the F/ROM differ from type to type.

For eight-bit microprocessors, F/ROMs with an eight-bit output structure are most convenient. 32 x 8 F/ROMs are often used for program storage, and especially for system characterization. That is, a line of similar machines may contain identical circuitry, with system definitions provided by a large ROM or PROM. Those relatively few key variables which allow the implementation of special options--system characterization--are stored in the F/ROM, where they can easily be changed in the plant or in the field.
Sec. 14.3 Field-Programmable ROMs (F-ROMs) (cont'd)

14.3.1 F-ROM Programmers

A F-ROM programmer circuit is very useful when a microcomputer must be reprogrammed in the field. A complete design should both program the desired bit locations, and then verify the F-ROM contents. The design discussed here is a microcomputer peripheral. That is, it may be placed in the cabinet housing the microcomputer. The F-ROM to be programmed is inserted into a socket on the programmer. The programmer uses the microcomputer to select the F-ROM locations to be programmed and verified, and is thus much more convenient to use than a bit-by-bit or byte-by-byte stand-alone programmer.

The programmer is more easily designed if programming may be accomplished mainly with voltages which are within TTL levels. The programmer circuitry may be composed mostly of TTL devices; the chance of circuit failure is decreased, and the verifying circuitry is simplified. F-ROMs which come from the factory as all logic zeros usually require a voltage larger than Vcc to be applied to output terminals during programming. But easier to design with are those F-ROMs which start out as all zeros, and require the programmed output to be held near ground during programming.

Figure 14.3.1 shows the circuit diagram for a F-ROM programmer designed for use with the 745274 32 x 8 F-ROM. In the steps below, a sample programming sequence is given which describes the operation of the circuit. The notation used is for an 8088 microprocessor.

1. The F-ROM location to be programmed is read to determine its present state. The proper location must be addressed, the chip's three-state outputs must be enabled, and chip Vcc must be set to its normal TTL level of about +5 volts. First, the A register is loaded: the five low-order bits (A4 through A0) contain the F-ROM location, A5 is low to enable the F-ROM. A6 is low. A7 is low to set the Vcc to +5 volts. An Output 36 instruction is executed, which strobes the 74273 eight-bit latch. The 7415 inverting open-collector buffers at Q7 pull current through the 1.8-V zener and a series diode, setting the voltage at the base of the NPN power darlington stage at about 6.3 volts. This value is dropped through two diode drops and powers the F-ROM.

2. The microcomputer executes an Input S instruction, which activates an array of 74LS three-state buffers and places the F-ROM word for the address previously selected on the microprocessor input bus. The contents of the F-ROM may be compared with the desired contents and if equal the programming of this word may be skipped.
SEC. 14.3 FIELD-PROGRAMMABLE ROMS (F/ROM) (cont'd)

(3) The F/ROM may now be programmed, one bit at a time. The bit to be programmed is addressed by setting up its address in the three low-order bits of the A register then performing an Output 36 instruction. A3 is set to 10H to program the addressed bit to zero. The selected output of the 9394/ 74229 addressable latch remains high until another Output 36 instruction is performed. The selected output bit is pulled down near ground by the 7416 inverting open-collector buffer. More than one latch output can be high at any one time, but care must be taken not to overload the power supply or overheat the F/ROM.

(4) The F/ROM VCC is now increased to +12.5V so as to swap an internal protective zener diode and provide enough current to blast the fuse for the bit to be programmed. Up to 750 ma is required. This is done by setting the five low-order bits of the accumulator to the correct F/ROM location (as in step 1), and executing an Output 36 instruction. This time, however, the chip should be disabled: A5 should be low. A6 should be high, which allows the 1K ohm resistor to pull up to +14 v, charging the 0.01 microfarad capacitor. The NM dual-traction stage ramps VCC up to about 12.5 volts.

(5) Within 10 microseconds to 1 millisecond after VCC has reached 12.5 volts, the F/ROM should be enabled. Another Output 36 instruction is executed, the settings remaining the same except that A5 should be low.

(6) One millisecond later, A5 is toggled once again; another Output 36 instruction is performed; and the F/ROM is disabled. The microcomputer may develop this time delay by using an interval timer (Chapter 18) or by executing a software loop calculated to take the required time.

(7) Within 10 microseconds to 1 millisecond later after the F/ROM is disabled, VCC should be returned to +5 volts by running another Output 36 instruction, with the high order bit set low.

(8) Within 10 microseconds to 1 millisecond after VCC returns to +5 volts, the F/ROM is once again enabled by toggling bit 5 of the accumulator and running an Output 36 instruction. Then an Input 3 instruction is executed in order to test the programmed bit.

(9) Before programming another bit, the program should wait a certain period of time to keep the F/ROM from burning up. VCC should not remain at +12.5 volts for more than 10% of the total programming cycle. In order to further cool the chip, VCC should be removed altogether between bit-programming times. This is done by performing an Output 36 instruction with bit 6 high, which clamps VCC near zero.

(10) If step 8 above reveals that the bit did not program, steps 4 through 8 may be repeated for that bit, allowing the proper delay described in step 9 to avoid over-heating. If it once again fails to program, steps 4 to 8 may be repeated once again, this time applying +12.5 volts for 50 to 75 milliseconds. The +12.5-volt 50% duty cycle still must not be exceeded—that is, the corresponding wait time between programming bits must also be increased.

(11) Once programming is accomplished for the bit originally selected, another bit may be programmed, using steps 1 through 10. Changing the bit to be programmed (step 9) requires two Output 36 instructions. The first such instruction addresses the old bit number, but the accumulator is first set so that A3 (the 9394 D input) is low. This turns off the latch for the old bit. Then the accumulator points to the new bit address, setting the latch data input high, and performs another Output 36. Now the new bit address is ready, and the programming procedure may be resumed, starting with step 4.

Note that, when the microcomputer is first turned on, a master reset pulse will cause the addressable latch to output all zeros. The eight-bit latch points to location zero, the F/ROM is enabled, and VCC is +5 volts.

A chart showing output bit assignments for the F/ROM programmer appears in Figure 14.3.2.

SEC. 14.4 REPROGRAMMABLE PROMS

The reprogrammable PROM has achieved wide popularity. A typical example is the 1702A, a 2 K ROM in a 28 x 8 configuration. This ROM is very useful in the boardhousing stages of eight-bit microcomputer development. In the low-volume applications, the 1702A is popular even though PROMS like the 74S471 may be less expensive and take up less PC board area. One advantage of the 1702A is the availability of the 1200 ROM, a
mask-programmable chip which is a direct plug-in replacement for the 1702A, and may be substituted when a design is finalized and high-volum production begins.

The 1702A is electrically programmable using a programmer of medium complexity, requiring a relatively high-voltage power supply. For this reason, programming of these chips is usually accomplished in the lab, or at centralized facilities. Suitable ReROM programmers are available from several sources, and may be purchased as accessories to microcomputer prototyping equipment sold by the microprocessor manufacturers.

The 1702A is erased to an all-zero state with an ultraviolet light source.

SEC. 14.5 COMPARISON OF ROM TYPES

The intended application will determine which type of ROM is most suitable. Figure 14.5.1 illustrates some of the many factors that influence the decision. Naturally the choice is most difficult in low-volume microcomputer production, when the cost of a mask-programmed ROM is not clearly justified.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>DROM</th>
<th>FRM</th>
<th>PROM</th>
<th>ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Diode-Programmed</td>
<td>Field-Programmable</td>
<td>Re-Programmable</td>
<td>Mask-Programmed</td>
</tr>
<tr>
<td>Memory</td>
<td>Read-Only Memory</td>
<td>Read-Only Memory</td>
<td>Read-Only Memory</td>
<td>Read-Only Memory</td>
</tr>
<tr>
<td>Number of locations for which ROM is practical</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small</td>
<td>Small- medium</td>
<td>Small- medium</td>
<td>Small- large</td>
<td></td>
</tr>
<tr>
<td>Suffixes, add to part number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>Medium</td>
<td>Medium- high</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extra costs, add to purchase price</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spare needed</td>
<td>Programmer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power supplies</td>
<td>$5 V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost of programming</td>
<td>Nothing</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 14.5.1—Some Considerations When Choosing Type of ROM to Be Used

SEC. 14.6 THE ROMIN OPTION

14.6.1 Flexible Memory Referencing There are two basic ways in which to load one of the 8008's internal index registers from memory. The first is to use the LDW instructions. These instructions are good mainly for handling constants stored in ROM as part of the program, and do not constitute a very flexible method for loading memory. The second way is to use the LIM instructions. These load the register with the contents of the location in memory addressed by the H and L registers. This method is more flexible, but may still be somewhat cumbersome because the H and L registers must constantly be manipulated.

The improved instruction set designed into the 8080 microprocessor allows the A register to be loaded from memory locations which are addressed either by the B and C registers, or by the D and E registers, in a manner similar to the use of the H and L registers. Although these added instructions can be used only to load the A register, the added flexibility is a great advantage when moving a block of data. One pair of registers (e.g., D and E) points to where data is being picked up, and the other pair of registers (e.g., H and L) points toward the RAM location where the data is being placed.

With the 8080 this flexibility is not automatically available. Several options are available, however, to help make up for this deficiency. The RAM-PAGE option described in Chapter 13 is often helpful in small systems, since that option eliminates having to manipulate the H register when addressing RAM. But RAM-PAGE is no help in systems requiring more than one page of RAM.

Another useful design idea for increasing memory-addressing flexibility in an 8008 system is the ROMIN option. Essentially, ROMIN is treated as an input port, and a conditional input approach allows the A register to address the memory.

14.6.2 Implementing the ROMIN Option With the ROMIN option, a system may reference memory in either of two ways. The first is through the normal method: using the LIM or LWM instructions in conjunction with the H and L registers. The second is by using input (IOP) instructions. While 8008 systems are usually designed so that memory may be referenced only during IOP instruction fetch and PCR (memory read) cycles, systems with the ROMIN option also permit memory to be read during FCC (I/O command) cycles.
The memory timing circuitry for the MKI 4 (see Chapter 28) has been replaced in Figure 14.6.1. For comparison, the circuit has been modified for the ROMIN option. Note how easily ROMIN may be implemented. Only one wiring change has been made.

In the original circuit, the DSH signal (DHS modified) is generated by NANDing the DSH with MIN. MIN is a timing signal which is active (i.e., high) during PCI-TSA and PCR-TSA times. DSH, in turn, is used to enable ROM. (See Chapter 28 for elaboration.) Thus ROM may be enabled during PCI cycles (ROM as source of program instructions) or PCR cycles (ROM as source of data read from memory).

With the ROMIN option, the DSH signal which enables ROM is created using T3A, rather than MIN. T3A is active not only during PCI and PCR cycles, but also during PCI cycles. (As explained in Chapter 5, the T3A signal is disabled during ROM cycles.) This change permits ROM to be read during input instructions (whence ROMIN).

Fig. 14.6.1—Original MKI 4 Memory Timing; ROMIN Option Modification

SEC. 14.6 THE ROMIN OPTION (cont'd)

It turns out that INF 000, INF 001, INF 002, and INF 003 will read ROM using the above circuit, because of the bit combinations of DSH, DHL, and D3 which may occur. If the resultant limitation in the use of input instructions is overly stringent, the designer might expand the input port array with the conditional input concept (Chapter 8).

A memory map is shown in Figure 14.6.2. Note that the chart shows that the four input instructions reference four pages in ROM.

Fig. 14.6.2—Memory Map for MKI 4 with ROMIN Option

14.6.3 Programming with ROMIN The ROMIN option permits an input instruction to address memory. The high-order memory address (DSH through DSH) is defined by the code for the input instruction itself, which (in binary form) is 01 000 M11 (where 0001 defines which one of the eight input instructions is being executed). Since the DSH bit is always logic 0, only odd-numbered pages of memory may be referenced by a ROMIN input instruction. The 12-order memory address bits are simply set up in the A register before executing the input instruction. This is possible because, as with all I/O instructions, the A register is transferred to the DL register during PCI-TSA time.
SEC. 16.6 THE ROMIN OPTION (cont'd)

PURPOSE OF PROGRAM: LOAD CONTENTS OF MEMORY AT LOCATION 003125 INTO A REGISTER

METHOD #2 (ROMIN)

LHI 003
LLI 125
LAM

(5 bytes)

METHOD #2 (ROMIN)

LAI 125
INF 001

(3 bytes)

Fig. 16.6.3---Simple ROMIN Program Comparison

SEC. 16.1 DIRECT MEMORY ACCESS

Regardless of how fast a microprocessor or minicomputer may be, there is some high-speed peripheral which requires data at a rate which is too fast for the CPU, or which takes up too much of the CPU's time. Direct memory access (DMA) can effectively increase the speed of the computer in handling blocks of data, and is therefore very useful in microcomputer systems where a fast peripheral puts a strain on the capabilities of the microprocessor.

A computer with DMA can write data into memory, or read from it, without separate input or output instructions for each byte. For example, a peripheral may be connected to DMA hardware, with the DMA circuitry under control of the CPU. A typical instruction to the DMA controller might be "read in 32 bytes from the first sector of the floppy disc, and place them in memory, starting at location 007000." After the microprocessor gives this command to the DMA controller, it continues to execute its normal flow of instructions. The DMA system provides a flag (input bit) which the CPU may interrogate periodically to find out whether the job has been completed. Another alternative is to have the DMA controller create an interrupt when its job is done.

It is not the purpose of this chapter to describe all of the various DMA techniques which the designer may develop. Instead, the ways of obtaining direct access to memory in 8080 and 8085 microcomputers are described.

SEC. 15.2 DMA WITH THE 8080

The address bits on the 8080 chip use three-state output circuitry. The DMA controller is connected to the HOLD terminal of the 8080, as shown in Figure 15.2.1. When the HOLD line is brought high, the CPU will acknowledge with a logic one on the HOLD output. The CPU address and data bus will be floating in the high-Z state by the end of the next 52 pulse after HOLD goes high. Once the 8080 address lines are floating, the address lines coming from the DMA controller may be enabled. This circuit allows the DMA system to gain access to both the memory address bus and the memory input and output data.

The 471 data sheet, near the end of this book, includes basic information on the 8080 hold function (Sec. 1.9).
SEC. 15.2 DMA WITH THE 8080 (cont'd)

![Diagram of DMA Address System for 8080](image)

**Fig. 15.2.1--DMA Address System for 8080**

SEC. 16.1 NEED FOR INTERRUPTS

The 8080 and 8088 microprocessors contain an input terminal which may be used by external circuitry to interrupt the program currently being executed. The need usually arises in a system where certain high-priority events occur, demanding immediate attention of the CPU. The interrupt feature is also useful when the software burden of checking for many low-priority events becomes excessive or too time-consuming. For example, when a slow-speed peripheral device—such as a keyboard—must be handled without interrupts, the mainline program must leave its principal task periodically to go and check to see whether there is new data from the keyboard. This may place unwelcome constraints on the system's software.

In an 8080 microcomputer, the CPU indicates its recognition of an interrupt request by causing the INTA machine status bit to go high during the next instruction fetch cycle. In other words, INTA goes high at T1H time. However, the CPU will not recognize interrupts unless an EI (enable interrupt) instruction has previously been executed. (For further details, see the 471 data sheet near the end of this book, Sec. 1.8.)

When the INTERRUPT terminal of the 8088 is brought to logic one, the 8088 recognizes the interrupt at the next instruction fetch (PCI) cycle by outputting S2, S1, S0 = 110, at what would otherwise have been T1 time. This alternate T1 time is called TII. Whereas normally the program counter (PC) is incremented automatically after the T1 state, in preparation for the fetching of the next instruction, the PC is incremented automatically after the TII state. This permits the CPU to remember where it was in the mainline program when interrupted. The following PCI cycle is used to insert the interrupt instruction that causes the CPU to jump to the interrupt routine.

In both cases, 8080 and 8088, it is left to external hardware to jam the interrupt instruction onto the CPU data bus at the proper time.

SEC. 16.2 INITIAL INTERRUPT

The 8088 uses the interrupt function to escape the initial STOPPED (halted) state which occurs automatically when power is first applied. Sixteen clock periods after this state begins, the 8088 instruction register; the A, B, C, D, E, H and L registers; and the PC register are all cleared to zero. An initial interrupt is required to escape the STOPPED state. The most common approach is to use a master reset pulse to deliver an interrupt to the CPU, and to jam in an initial instruction which calls an initialization subroutine in the software. This program proceeds to clear RAM and output latches as needed.
SEC. 16.2 INITIAL INTERRUPT (cont'd)

In the 8080, an interrupt is not used to initialize the CPU. Instead, the RESET input is used; this terminal clears only the 8080 PC register. All other internal registers must be initialized through software. (See the 471 data sheet near the end of the book, Sec. 1.7.)

SEC. 16.3 HOW THE INTERRUPT IS USED

The program which the microcomputer enters through the initial interrupt, which contains the instructions normally executed by the microcomputer in carrying out its principal functions, is called the mainline program. When an interrupt occurs, the microprocessor finishes its current instruction, and then allows an extra instruction to be inserted (jumped to). This extra instruction is usually a RESTART instruction which calls a subroutine in memory; that is, the CPU jumps from its location in the mainline program to the location where the subroutine is stored. The interrupt subroutine is said to service the interrupt which caused that subroutine to be called. For example: typing a key on a keyboard may cause an interrupt, which calls an interrupt subroutine for processing keyboard characters. The interrupt subroutine services the keyboard by inputting the character typed, and storing it into a list of typed characters in RAM. Then the subroutine returns to the mainline program. Depending on the design of the system, either the mainline program, or another interrupt routine, picks up the characters stored in RAM for further processing later on.

The use of the microprocessor's interrupt capability is an important element in the overall design of the system. Whether interrupts are used at all, or one interrupt level is used, or a number of interrupt levels are used, depends on the application for which the microprocessor is intended. An example of the choices involved is illustrated by the two approaches for adding a keyboard input to a microcomputer which are discussed in this book. In Chapter 21, the microprocessor periodically checks for new keyboard data, and a FIFO provides some temporary keyboard data storage. This approach avoids some of the complications of dealing with interrupts. In Chapter 26, the MINI microcomputer has a similar keyboard input, but each new keyboard character interrupts the microcomputer. Chapter 17 gives several design examples for interrupt systems, concentrating on the means of saving status during interrupts. Note in these examples that the system design frequently involves tradeoffs between software complexity and added hardware. The interrupt feature is often an important element in the design.

SEC. 16.4 SYNCHRONIZING INTERRUPT REQUESTS (8080)

The 8080's interrupt line is strobed into two different parts of the CPU's internal circuitry. The interrupt strobe takes place on the falling edge of $\delta_4$. If the interrupt signal changes state shortly before that time, one section of the CPU might recognize the request, and another part not. For this reason 8080 interrupts must be synchronized with external circuitry to $\delta_4$, or to the rising edge of $\delta_4$.

The 8080 contains its own internal interrupt synchronization.

SEC. 16.5 SIMPLE INTERRUPTS

One example of a simple interrupt system is a microcomputer where an interrupt is recognized only when the CPU is in the STOPPED state. This is suitable when the interrupt request need not be serviced immediately when the demand arises. Even in very complex systems, of course, an interrupt will not be serviced immediately, if a higher priority interrupt is already being processed.

If the main line program can be interrupted only at certain points during its execution, the problems associated with having to save internal microprocessor status, during execution of the interrupt, are minimized. (See Chapter 17 on saving status.)

In Figure 16.5.1, part (2), a flip-flop is connected between an interrupt source and the interrupt terminal of the CPU. An interrupt request will be passed on to the CPU only when the CPU halts before the request was received. In an 8080 system, the HALT condition is decoded in the state decoder circuitry (Oh. 5) as a string of pulses, produced by $\delta_7$ or $\delta_6$; these pulses will permit the request to be passed on very shortly after it occurs. In an 8080 system, the signal HALT (halt acknowledge) could be combined with PMT to produce a similar pulse train for use in this circuit.
SEC. 16.6 INTRODUCTION TO PRIORITY INTERRUPT SYSTEMS (cont'd)

A microcomputer which is able to recognize interrupts from more than one source is called a multiple interrupt system. A system with two or more interrupt sources generally requires a method for assigning priorities as to which interrupt request should be acted upon first. A priority interrupt system arranges multiple interrupts in a hierarchy so that each interrupt has a certain level of importance.

Each interrupt is assigned to a given priority level, and when a program is being executed as the result of a particular interrupt, the microcomputer is said to be on that interrupt level.

Fig. 16.6.1--Block Diagram of Priority Interrupt System
SEC. 16.6 INTRODUCTION TO PRIORITY INTERRUPT SYSTEMS (cont’d)

In the examples given in this chapter, interrupt level number 0 has the highest priority; levels with larger numbers have successively lower priorities. If the microcomputer is executing a program on interrupt level 2, a new interrupt on levels 1 or 0 should be recognized—the program should jump to the higher-priority interrupt (returning to interrupt level 2 later on). A level 2 interrupt cannot, however, be interrupted by level 3 or 4 interrupts, or by any other lower-priority requests. The system is designed so that a new interrupt on the same level as that already being executed will not be recognized.

The basic parts of a priority interrupt system are shown in Figure 16.6.1. Each element is described below.

SEC. 16.7 INTERRUPT REQUEST REGISTER

The original source of an interrupt may be an interval timer, a simple pushbutton switch, or a peripheral device. The interrupt request signals are fed into an Interrupt Request Register, an array of latches—one for each interrupt input. The purpose of this register is to store interrupt requests temporarily, since the microprocessor may not be able to service any given interrupt request immediately. Each interrupt latch is reset when the CPU finishes servicing that particular interrupt level.

A design for an Interrupt Request Register is shown in Figure 16.7.1. The register is made up of G flip-flops, one for each interrupt level. Though the drawing shows only two such latches, eight are required for a full eight-level priority interrupt system (four 7479 packages). Each flip-flop may be set on the rising edge of an interrupt request pulse (labeled REQ 0, REQ 1, etc.). Each is reset by an active-lower-level signal which indicates that the microprocessor is returning from that interrupt level, having finished executing the designated subroutine. Note that MR also sets flip-flop #0; this is the only one so connected; see Sec. 16.12.

Fig. 16.7.1—Edge-Triggered Interrupt Request Register

SEC. 16.8 INTERRUPT REQUEST SYNCHRONIZATION REGISTER

The flip-flops within the Interrupt Request Register may be set at any random time. To prevent race conditions from occurring in the rest of the priority interrupt circuitry, an Interrupt Request Synchronization Register is required.

The least complicated kind of synchronization register is a strobed latch with sufficient bit capacity for the number of interrupt levels being handled. An eight-bit example is the 74273 in Figure 16.8.1 below.

SEC. 16.9 PRIORITY ENCODERS

A system with two or more interrupt sources generally requires a method for assigning priorities to which interrupt request should be acted upon first. Since the heart of a priority interrupt system is a priority encoder, this device is introduced here. The easiest
SEC. 16.9 PRIORITY ENCODERS (cont'd)

way to understand the function of a priority encoder is to see how it may be connected so as to reverse the action of an ordinary decoder. Figure 16.9.1 shows a 74138/32305 three-to-eight decoder with its outputs connected to the inputs of a 74148/3218 priority encoder.

![Diagram of decoder and priority encoder connections]

Fig. 16.9.1--Priority Encoder Reverses the Action of Decoder

When the 74138 is enabled, one and only one of its eight output terminals is low. The 9318 detects this condition, and outputs the same three-bit address that appeared at the 74138 inputs. The 9318 ENABLE OUTPUT terminal will be active (high) when the 74138 ENABLE INPUT terminal is a logic one. And, when none of the 74148's input lines is active (low), the ENABLE OUTPUT terminal will be disabled (low).

The manufacturers usually show the 74148/9318 with both its inputs and its outputs as active low. The notation in Figure 16.9.1 shows active-low inputs and active-high outputs. As a consequence, the usual input numbering is reversed: 7 for 0, 6 for 1, etc. Though both notations are correct, the one used here is more convenient for the circuits shown in this chapter. The proper pin numbers are given to avoid confusion.

A priority encoder does more than simply reverse the functions of a decoder. A priority encoder is usually connected to circuitry that permits more than one of the encoder's inputs to be active simultaneously—unlike the circuit of Figure 16.9.1 (where only one 74138 decoder output can go low at once). Thus, the 74148/9318 assigns a priority among the possible input codes. The output address will correspond to the highest-priority active input. The lower-numbered inputs have the highest priority. If for example the input line labeled 0 is brought low, the output address will be 000 regardless of the state of the other input lines. Figure 16.9.2 illustrates this feature by using the symbol X to denote "don't care" input conditions.

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>OUTPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2 1 0</td>
<td>C B A EO</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 NO INPUT ACTIVE</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 1</td>
<td>1 0 1 1 LOWEST PRIORITY</td>
</tr>
<tr>
<td>X 0 1 1 1 1 1 1</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>X X 0 1 1 1 1 1</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>X X X 0 1 1 1 1</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>X X X 0 1 1 1 1</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>X X X 0 1 1 1 1</td>
<td>0 0 0 1 HIGHEST PRIORITY</td>
</tr>
</tbody>
</table>

Fig. 16.9.2--Truth Table for Priority Encoder

SEC. 16.10 PRIORITY REGISTERS

Another useful element in priority interrupt schemes is the priority register. As contrasted to the priority encoder, the priority register does not encode its inputs (produce a binary code representing the number of the highest priority active input). Instead, it merely gates its inputs according to their priority, allowing only the highest-priority active input signal to pass through to its outputs. An example is the 74278, as shown in Figure 16.10.1.
SEC. 16.10  PRIORITY REGISTR (cont'd)

Fig. 16.10.1---Eight-Bit Prioritised Interrupt Request Synchronisation Register

Of the cascaded-pair's eight outputs (LEV7 through LEV0), no more than one may be active at a time: the highest-priority output whose corresponding input is activated. In other words, if the highest-priority input signal (D2 on the 74278 to the right of the diagram) is high, it is passed through unaffected. The second-highest-priority output (D1 on the right-hand 74278) is true if its input signal (D2) is true and Y1 is false. Since the registers are cascaded, the left-hand 74278 outputs are inactive unless the other 74278 enables the chip with a high PIA signal.

PRIORITY INPUT FROM LAST STAGE PD
HIGHEST PRIORITY
Y1 = D1
Y2 = D2
Y3 = D3
LOWEST PRIORITY
Y4 = D4

PRIORITY OUTPUT TO NEXT STAGE P1 = Y4

Fig. 16.10.2---Priority Register Logic Equations

The 74278 also incorporates a latching function. The input data values are latched up by the SYNCHRO signal, and thus the circuit in Figure 16.10.1 constitutes a Prioritised Interrupt Request Synchronisation Register.

SEC. 16.11  INTERRUPTS BEING SERVICED REGISTER

ientos 

Fig. 16.11.1---Interrupts Being Serviced Register
As stated above, at any given time several interrupt levels may be in process of being serviced. For example, the microcomputer might have begun servicing interrupt level 7, when an interrupt request on level 4 was received. The system must remember that both levels 4 and 7 are active, so that when the highest-priority interrupt subroutine is completed, the CPU can return to and finish all the lower-level interrupts. It is not adequate to truncate the system by combining the functions of the Interrupts Being Serviced Register and the Interrupt Request Register, because this would prevent the system from distinguishing readily between new interrupt requests and interrupts already in progress. The result would be lost time and erroneous results.

In its simplest form, the Interrupts Being Serviced Register is made up of an array of flip-flops, similar to the Interrupt Request Register discussed in Sec. 16.7 above.

Figure 16.11.1 (center) shows such a register in the form of two 74273 latches (eight latches). These latches are loaded as follows. The three inputs at the left of the schematic—PE2, PE1, and RPO—derive from a 9318 priority encoder (not shown), which in turn is connected to an Interrupt Request Synchronization Register (Sec. 16.8). Thus PE2 through RPO represent a three-bit binary code for the highest-priority interrupt request received. When the microprocessor begins to service a new high-priority interrupt level, it causes the VM signal (described later), here used to strobe the Interrupts Being Serviced Register.

The outputs of the Interrupts Being Serviced Register are in turn priority-encoded by a 9318, as shown to the right in Figure 16.11.1. Only four I/Os are required by this circuit.

SEC. 16.12 PRIORITY COMPARATOR

The Priority Comparator continuously compares the interrupt level being serviced, with the interrupt requests which have been received. Whenever an interrupt request is received which is higher in priority than that currently being serviced, the system sends an interrupt request signal directly to the microprocessor.

An ordinary digital comparator may be used for this function. See for example the 93L24s at work in Figure 16.16.1 (center) and Figure 16.17.1 (center).

Note that when the microcomputer is first turned on, it should recognize an interrupt on level zero so as to carry out its initialization scheme. Thus the Priority Comparator should, at power-on, receive an interrupt request on level zero. Also, the priority interrupt system must not think it is already on interrupt level zero, or no interrupt

SEC. 16.13 PRIORITY COMPARATOR (cont'd)

pulse would be transmitted to the CPU.

Thus, the ONO flip-flop in the interrupt level being serviced register (Sec. 16.11) must be reset by the master reset signal. Also, the master reset signal must generate an interrupt request to (i.e., set) the zero-level flip-flop in the interrupt request register (Sec. 16.7).

SEC. 16.14 INTERRUPT JAN LOGIC

When the Priority Comparator strobes the CPU's INTERRUPT terminal with an interrupt request, the microprocessor responds with an interrupt acknowledge signal—RE7X for the 8080, SII for the 8086. The priority system jumps in an instruction which causes the CPU to jump to an interrupt subroutine. In microcomputers based on the 8080 or 8086, the instruction is usually a RST (RESTART) instruction.

Two such schemes are shown in the three- and eight-level priority interrupt systems below (Figure 16.16.1 and 16.17.1). The instruction jammed in is a RESTART N instruction, where N is the interrupt level that is just beginning to be serviced. This instruction causes the CPU to jump to location 000000 in split octal notation, where a suitable interrupt subroutine is stored. The circuitry shown is for the 8080.

SEC. 16.14 NUMBERED RETURN INSTRUCTIONS

When a given interrupt subroutine is complete, the interrupt system logic must recognize this fact and clear out the appropriate latches in the registers which keep track of that interrupt level. Since subroutines generally end with RETURN instructions, a suitable technique is to decode the execution of RETURN instructions by the microprocessor, assigning unique numbered RETURN instructions to each interrupt level. That is, the very last instruction executed on interrupt level 3 is a RE7X instruction. This instruction is decoded using a technique described in Chapter 11 above.

The RE7X signal is used to reset the level 7 latches in both the Interrupt Request Register, and the Interrupts Being Serviced Register. RE7X resets the latches for interrupts level 6, and so forth.

The RE00 instruction is the last instruction in the power-on initialization subroutine, which takes place on interrupt level 0. The normal RETURN instruction— which is the same as RE7X— should be avoided until initialization is complete. Since interrupt level 0 is normally used only for initialization, an interrupt request for level 0 will not
SEC. 16.14
NUMBERED RETURN INSTRUCTIONS (cont'd)

occur afterwards. After this point, the RET or RET# instruction may be utilized freely for normal subroutine returns. Note that this part of the interrupt system cannot be used with the 8080 microprocessor, which does not have numbered return instructions.

RETURN INSTRUCTION RECOGNITION
NUMBERED RETURN INSTRUCTION DECODER

SEC. 16.16
AN ADDRESSABLE LATCH AS AN INTERRUPT REGISTER

Figure 16.16.1 shows an Interrupts Being Serviced Register which combines several of the functions discussed above. The register itself is a 9334 eight-bit addressable latch. It is addressed from either of two sources, as selected by a 74158 multiplexer. Any one bit within the 9334 may be set by the priority-encoded interrupt request address, R0 through RF. This can occur only when the IRF (interrupt request present) signal is high (see this signal in Figures 16.16.1 and 16.17.1). A 9334 latch bit may be reset by the execution of the appropriate numbered RETURN instruction, whose address appears on three lines of the microcomputer output bus, T8 through T9. This in turn can happen only when the RIR (return instruction recognized) signal is high (Figure 16.14.1.)

The outputs from the Interrupts Being Serviced Register are priority-encoded by a 9318. This circuit accomplishes a respectable portion of the functions of a priority interrupt system with only three ICs.

Fig. 16.15.1—Interrupts Being Serviced Register: Multiplexer Sets and Resets Addressable Latch.
INTERRUPTS

SEC. 16.16 THREE-LEVEL PRIORITY INTERRUPT SYSTEM

RETURN INSTRUCTION RECOGNITION

NUMBERED RETURN INSTRUCTION DECODER

74138/2

5-BIT

COMPARE

74L24

93L24

5-BIT

COMPARE

93L24

5-BIT

COMPARE

INTERRUPT REQUEST PRESENT

Fig. 16.16.1--3-Level Priority Interrupt System

INTERRUPTS

SEC. 16.16 THREE-LEVEL PRIORITY INTERRUPT SYSTEM (cont'd)

The circuit for a three-level priority interrupt system is shown in Figure 16.16.1. This design uses many of the elements discussed in the previous sections of this chapter.

A 93L24 five-bit comparator is connected so as to recognize numbered RETURN instructions. The number of the RETURN instruction is decoded using one side of a 74139 dual decoder; only four such instructions are employed: RET1, RETN, RDT1, and RDTH. The three interrupt request inputs take the form of negative-pulse signals fed to the S7 terminals of three 74279 set-reset flip-flops, whose outputs are marked B7R, B7N, and B6R. These make up an asynchronous Interrupt Request Register.

The outputs from these flip-flops are connected to a 74279 priority register, where they are both synchronized by S6 and S5, and prioritized. The outputs of this register go, in turn, to the Priority Comparator and three NAND gates which feed the UV flip-flops in the lower left-hand corner of the diagram. The second 93L24 comparator checks to see whether the requested interrupt is of a higher priority than any interrupt level already on. If it is, then the INT flip-flop is set, which requests an Interrupt of the 8080 CPU. When the CPU acknowledges the Interrupt, with an ET1 strobe, the UIMJ flip-flop is set. This prepares the JAM gate to jam the appropriate RST(NT) instruction onto the input bus (NB1 through NB3) at the next T6A time. In order to suppress the output of the Priority Comparator during the interrupt and jam sequence, the INT and UIMJ signals are fed into the 93L24's two high-order bits (A4 and A5). This prevents the occurrence of multiple interrupts during the interrupt sequence.

Three 74125 three-state buffers are used to enable the UIV (interrupt level) bits into the RST jam instruction. The diodes provide the remaining constant bits of the RST instruction. (Recall that the octal code for the RST instruction is 456, where N is the level number: Chapter 11.)

The STS strobe resets the UIMJ flip-flop and terminates the interrupt sequence.

SEC. 16.17 FULL EIGHT-LEVEL PRIORITY INTERRUPT SYSTEM

The schematic for a full eight-level priority interrupt system for the 8080 is shown in Figure 16.17.1. The circuit can readily be modified for the 8085.

The eight-bit Interrupt Request Register consists of D flip-flops. This register provides rising-edge clock inputs (RDQ0-RDQ7), useful in avoiding problems related to pulse width and multiple interrupts.
SEC. 16.17 FULL EIGHT-LEVEL PRIORITY INTERRUPT SYSTEM (cont'd)

The highest-priority interrupt request, level 0, is assigned to the master reset function. The register also allows for negative-pulse-triggered interrupt request signals (N0-0-N7), for added flexibility.

The output of the priority encoder must be stabilized before it can be considered valid for subsequent processing. For this reason, a 7492 Interrupt Request Synchronization Register is used. This octal latch is strobed from RST or, if the CPU has hit a HALT instruction, from the repetitive STOP pulse train. In either case, the signals reaching the priority encoder can change only at a time which is related to MACH TWO of the CPU clock: ST0 = ST0 - ST1; SPO = STOP0 - SY - S2.

The remainder of the eight-bit priority interrupt system is similar to the three-bit example above. One difference is that a 74524L octal bus driver is used to jam the RST instruction onto the CPU input bus.

SEC. 16.18 DAISY-CHAIN INTERRUPT SYSTEM

Another way of configuring a priority interrupt system is to employ a daisy-chain structure. The block diagram for a four-level daisy-chain interrupt system is shown in Figure 16.17.1.

![Daisy-Chain Interrupt Structure](image)

Each member of the chain can block interrupt requests coming from a lower-priority member of the chain. At the same time, each level can block interrupt-acknowledge signals returning from the CPU, so that lower-priority daisy levels will not think that their interrupt requests have reached the CPU, and will keep trying. Since daisy level 0 is closest to the CPU logic, it is the highest-priority interrupt level. This kind of structure is used for interrupts in the Fairchild F-8 microprocessor chip set.

A diagram for one stage of such a system appears in Figure 16.18.2.

The interrupt-acknowledge signals need not necessarily be daisy-chained. It could be a number on the bus combined with a strobe signal. Such variations are left to the imagination of the designer.
In many microcomputers, the CPU must be able to service an interrupt at any random point during the execution of the mainline program. The CPU’s internal registers and flag bits might have any of the possible combinations of values at the time when the interrupt instruction is received. The content of the registers and flags, taken together at any particular time, is called the machine status, or simply status. If the mainline program is to proceed normally when the interrupt subroutine ends, the interrupt subroutine must save status as it begins, and restore status before returning control to the mainline program.

The 8080 makes status saving relatively easy. (See Chapter 23, Sec. 23.11, for a practical example.) However, the 8080’s structure and instruction set require a number of special considerations. This chapter is devoted to the software and hardware necessary to enable the 8080 to save status during interrupts.

The 8080 has a number of registers on the chip. These include the PC register stack, the seven index registers, and the four flags—all discussed in the following sections.

17.3 THE PROGRAM COUNTER STACK

There are eight PC (program counter) registers in the 8080’s internal PC register stack. When a subroutine is called by the microprocessor’s program, the program counter goes up one level in the stack. This stack will overflow if subroutines are nested too many times. It must be remembered, therefore, that a subroutine interrupt uses at least one level in the stack, and if the software includes too many nested subroutines, the PC register stack may overflow.

Given that overflow is avoided, no special care need be taken to save the status of the PC register during an interrupt, since the PC stack pointer logic does the saving and restoring automatically. The RET (return) instruction tagged in by the interrupt pushes the program counter by pointing up one level in the PC stack, thus saving the mainline program location in the register below. The interrupt subroutine ends with an RET (return) instruction, which pops the PC register by pointing down one level in the stack to where the main line program location was stored.
SEC. 17.3 INDEX REGISTERS

The seven general-purpose index registers of the 8008, the A, B, C, D, E, H, and I registers, may be in use by the main line program when an interrupt occurs. The interrupt subroutine either must leave the contents of any given register unchanged, or must save the register, use it, then save or restore the register before returning to the main line program. If only a few registers are to be used during the interrupt subroutine, then only those registers need be saved and unsaved. The configuration of hardware set software that will be necessary to save status during interrupts depends therefore on the complexity of the interrupt subroutine. If a register is not used in the main line program it is, of course, not necessary to save it in the interrupt subroutine.

SEC. 17.4 FLAGS (CONDITION FLIP-FLOPS)

The 8008 has four internal flip-flops which store flag bits set during the execution of arithmetic and logic instructions. These four flags are C (carry), P (parity), Z (zero), and S (sign). The 8008 manual describes how these flags are set and tested by various instructions.

The parity of a word in the 8008 is based on the eight-bit result of an arithmetic or logic instruction. (This point is not described fully in current editions of the 8008 manual.) Even though the carry flag may be affected by the same instruction, the carry bit is not used in calculating the value of the parity bit. If the number of logical ones in the eight-bit result is even, then the parity is even, and the parity flag will be set to one (logic true). If the number of ones is odd, the parity flag will be reset to zero. If the eight-bit result is all zeros, i.e., there are zero ones, this is considered even parity, and the parity flag will be set.

Of the sixteen combinations of four bits, only ten flag combinations are really possible due to the logical constraints on the flags. For example, if the zero flag is set, the parity must be even, and therefore the parity flag must also be set. If the sign flag is set, the zero flag cannot be set. Figure 17.4.1 shows all combinations of the four 8008 flags, both possible and impossible.

SEC. 17.4 FLAGS (CONDITION FLIP-FLOPS) (cont'd)

<table>
<thead>
<tr>
<th>C P Z S</th>
<th>DESCRIPTION OF WORD*</th>
<th>NO. OF WORD*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>Half the positive numbers except for zero; no carry</td>
<td>63</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>Half the negative numbers; no carry</td>
<td>64</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>Impossible: if zero flag is set, the parity flag must also be set</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>Impossible: if zero flag is set, the parity flag must also be set</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>Half the positive numbers; no carry</td>
<td>64</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>Half the negative numbers; no carry</td>
<td>64</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>Zero; no carry</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>Impossible: if sign flag is set, zero flag cannot be set</td>
<td>0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>Half the positive numbers except zero; with carry</td>
<td>63</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>Half the negative numbers, with carry</td>
<td>64</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>Impossible: if zero flag is set, the parity flag must also be set</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>Impossible: if zero flag is set, the parity flag must also be set</td>
<td>0</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>Half the positive numbers, with carry</td>
<td>64</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>Half the negative numbers, with carry</td>
<td>64</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>Zero; with carry</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>Impossible: if sign flag is set, zero flag cannot be set</td>
<td>0</td>
</tr>
</tbody>
</table>

* A word is defined here as a unique combination of eight bits in the accumulator plus the carry bit. There are 512 possible words. The right-hand column shows the number of words which correspond to each combination of flags, and the middle column describes those words.

Fig. 17.4.1--The Sixteen Combinations of Flags, Including Six Combinations Which Are not Possible
SEC. 17.6 SOFTWARE TECHNIQUES (cont'd.)

Part two determines the status of those flags to be saved. Flags not used by the main line program need not be saved. Near the end of part two, the A register contains all the information required to restore the flags later on. Part two ends with JNL and LMA instructions, which store the status of the flags in RAM to be retrieved later. The JNL instruction can be used here, for the first time, because the flag status has already been determined, and the fact that the JNL instruction may change the flags no longer matters.

Part three saves all of the other registers used by both the main line program and the interrupt subroutine. Naturally, if the main line program never uses the C register, for example, the interrupt subroutine need not save it. On the other hand, if the interrupt subroutine does not use the C register, it would not be necessary to save it even if the main line program did use it. Again, the JNL instruction can now be used since part two already saved the flag conditions.

Note: in the examples given in this chapter, all Arabic numerals are in decimal unless otherwise specified.

17.6.3 Saving Status, Part One  An example of part one is shown in Figure 17.6.3:

| LIX | SAVE H REGISTER IN D REGISTER. |
| LEL | SAVE L REGISTER IN E REGISTER. |
| LMI 040 | POINT TO RAM LOCATION. |
| LLS 370 | USED FOR SAVING A REGISTER. |

Figure 17.6.3—Save Status with Software, Part One

In Figure 17.6.1, it is assumed that the main line program does not use the D and E registers, but does use the H and L. The D and E registers are therefore used to save the H and L registers, so that they may be loaded with an address in RAM where the A register may be saved.
17.6.4 Saving Status, Part Two

An example of part two appears in Figure 17.6.3:

SAV LAI 000
ASSUME ZERO FLAG IS SET.
JZJ SAVC IF TRUE, GET CARRY BIT.
LAI 060
ASSUME SIGN BIT IS ZERO (POSITIVE WORD).
JTS SAVC IF TRUE, GET CARRY BIT.
LAI 300
SIGN BIT WAS SET.
SAVC VAR
GET CARRY BIT.
JTP SAWW JUMP IF PARITY IS EVEN.
XRI 001
SET PARITY 110.
SAW INL
POINT TOWARDS NEXT RAM BYTE.
LMA
SAVE FLAG WORD.

Figure 17.6.2—Save Status with Software, Part Two

In this example of part two, the status of all four flags is packed into one eight-bit word and stored into memory. This word is rearranged in a way that allows the restoration of flags with a simple two-instruction program. A step-by-step description of how flags are saved and restored using this program appears in Figure 17.6.3. The instructions appear to the left of the figure. The ten possible combinations of flags appear at the top, both in octal form, and broken down into the four binary flag bits. The ten columns of the figure show how the program affects each of the ten combinations, instruction by instruction.

The three-digit numbers in the columns in Figure 17.6.3 are in a notation which conveys nine bits of information, that is, the eight bits which make up the contents of the A register, plus the carry bit. Values of 400 or above represent a set carry flag. Each three-digit number reflects the contents of the A register (plus carry flag) after the instruction shown at the left of that row has been executed.

---

**Fig. 17.6.3—Saving Flags with Software Illustrated for All Ten Possible Flag Combinations**

<table>
<thead>
<tr>
<th>ALL FLAGS</th>
<th>00</th>
<th>01</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>10</th>
<th>11</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARRY</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PARITY</td>
<td>P</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ZERO</td>
<td>Z</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SIGN</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

SAV LAI 000
SAVC VAR
JZJ SAVC
LAI 060
SAVC VAR
LAI 300
SAVC VAR
JTP SAWW
XRI 001
SAW INL
LMA
UNSAV LAY
AIA

---

In this example of part two, the status of all four flags is packed into one eight-bit word and stored into memory. This word is rearranged in a way that allows the restoration of flags with a simple two-instruction program. A step-by-step description of how flags are saved and restored using this program appears in Figure 17.6.3. The instructions appear to the left of the figure. The ten possible combinations of flags appear at the top, both in octal form, and broken down into the four binary flag bits. The ten columns of the figure show how the program affects each of the ten combinations, instruction by instruction.

The three-digit numbers in the columns in Figure 17.6.3 are in a notation which conveys nine bits of information, that is, the eight bits which make up the contents of the A register, plus the carry bit. Values of 400 or above represent a set carry flag. Each three-digit number reflects the contents of the A register (plus carry flag) after the instruction shown at the left of that row has been executed.
SEC. 17.6 SOFTWARE TECHNIQUES (cont'd)

The program has finished saving flags at the point where the dotted line appears, after the LMA instruction. Note that the content of the A register at this point is different for each of the ten flag combinations. This value is stored in memory. At this point comes part three of the status-saving program, which saves further registers; then comes the main body of the interrupt subroutine.

17.6.5 Saving Status, Part Three An example of part three of the status-saving procedure is shown in Figure 17.6.4:

INL  POINT TOWARDS NEXT RAM LOCATION.
IMB  SAVE THE B REGISTER.
INL  POINT TOWARDS NEXT RAM LOCATION.
IMC  SAVE THE C REGISTER.
INL  MOVE THE RAM POINTER AGAIN.
IMD  SAVE THE H REGISTER (FROM PART ONE).
INL  MOVE THE RAM POINTER AGAIN.
IME  SAVE THE L REGISTER (FROM PART ONE).

Figure 17.6.4—Saving Status with Software, Part Three

In Figure 17.6.4, note that the H and L registers are saved indirectly by saving the D and E registers. The H and L registers were loaded into the D and E registers previously, in part one.

A number of locations of RAM must be set aside to save registers and flags. The following registers are being used in this generalized example (Figure 17.6.5):

<table>
<thead>
<tr>
<th>PAGE</th>
<th>BYTE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>040</td>
<td>370</td>
<td>SAVE A REGISTER</td>
</tr>
<tr>
<td>040</td>
<td>371</td>
<td>SAVE FLAGS</td>
</tr>
<tr>
<td>040</td>
<td>372</td>
<td>SAVE B REGISTER</td>
</tr>
<tr>
<td>040</td>
<td>373</td>
<td>SAVE C REGISTER</td>
</tr>
<tr>
<td>040</td>
<td>374</td>
<td>SAVE H REGISTER</td>
</tr>
<tr>
<td>040</td>
<td>375</td>
<td>SAVE L REGISTER</td>
</tr>
</tbody>
</table>

Fig. 17.6.6—RAM Locations Used in Saving Registers and Flags

17.6.6 Interrupt Subroutine After part three of the status-saving procedure described above has taken place, the main line program status has been saved in RAM. The next part of the interrupt subroutine performs the functions for which the interrupt was intended by the designer. If, for example, the interrupt was caused by a key having been pressed on a Teletype, then the Teletype data might be shifted in, and the resultant character stored in a RAM buffer for subsequent processing by the main line program.

In any case, once the main body of the interrupt subroutine has been executed, it is necessary to restore the status saved in the three parts of the status-saving routine described above.

The restore program is also broken up into three parts. Since the last things saved are the first things restored, the three parts of the restore program are numbered inversely, three, two, one.

17.6.7 Unsaving Status, Part Three The first part of the program to restore status is part three. It unsaves the L, H, C, and B registers. Notice that this is the reverse of the order in which they were saved in Section 17.6.5 above.

A generalized example of unsaving status, part three, is given in Figure 17.6.6.
17.6 SOFTWARE TECHNIQUES (cont'd)

17.6.8 Unsaving Status, Part Two

Part two of the unsaving status program restores flags in a manner which corresponds to the way they were treated in the second part of the save routine. An example of a flag restoring program which works with the flag saving instructions shown in Section 17.6.4 is shown in Figure 17.6.7:

DCL  POINT TOWARDS FLAGS BYTE.
LAM  PICK UP FLAG INFORMATION.
ADA  RESTORE ALL FLAGS.

Figure 17.6.7—Unsave Status with Software, Part Two

The DCL instruction points to the location in memory where the flag-saving word was stored. The LAM instruction unsees the flag-saving word from memory. The ADA instruction adds the A register to itself. This doubling of the A register is equivalent to shifting every bit left one place. The bit which was in bit A0 is shifted into the carry bit. The A0 bit becomes the sign bit, A1, and sets the sign flag accordingly. Because all other bits have been paired so as not to affect parity, the parity is determined by the A0 bit alone, as it is shifted into the A1 position.
17.7.1 General Hardware is used to save status during interrupts in systems where software-only schemes are too restrictive or too time-consuming.

An example: the LIFO register design described in Chapter 12 may be used to save as many registers as might reasonably be desired. The LIFO may be used by more than one subroutine, and may even be used with subroutines which call themselves without losing any status information.

17.7.2 One-Byte LIFO Figure 17.7.1 shows a circuit which saves the A register with an OUTPUT X instruction, then reads the silver data back into the A register with an INPUT Y instruction. This circuit requires only one chip, the new 74S374, an eight-bit latch with three-state output circuitry. The '374 comes in a 20-pin Slim-DIP package on standard 0.3-inch row-to-row centers. Alternately, the Intel 8212 IC may be used—a smaller octal latch with three-state outputs, this one in a larger 24-pin package. Finally, the octal latch could be implemented with a separate octal latch, like the '74273, connected to an input multiplexer or to a discrete three-state buffer.

17.8.1 Hardware During the T4 state of memory cycle two of each INPUT instruction, the 8080 outputs the flags on the CPU bus. The Carry flag (C) appears on bit D1, Parity (P) on D2, Zero (Z) on D3, and the Sign flag (S) on D0. Flag-saving hardware is usually designed to latch up the flags during only one of the eight available INPUT instructions. In the examples that follow, INPUT 7 is designated as the flag-saving instruction, and the associated input strobe signal is INT.

The technique then is to latch up the information on the four lower-order bits of the CPU bus during the first T4 state after every INT strobe. Specific flag-saving designs can be implemented using the many TTL chips available. Figures 17.9.1 and 17.9.2 illustrate some possible designs using some of the convenient integrated circuits.
SEC. 17.9 LATCHING UP THE FLAGS (cont'd)

17.9.8 Software The program to save and unsave flags using the hardware in the preceding paragraphs is shown in Figure 17.9.3. This example assumes that the system uses only 256 bytes of RAM (the RAMPAGE option introduced in Chapter 13), and that the system does not have hardware push-pop options. The main line program does not use the E register.

```
INR LEL SAVE L REGISTER IN E.
LLI 370 POINT TO SAVE A REGISTER LOCATION.
LPA SAVE A REGISTER.
INP 007 SAVE FLAGS.
INL POINT TO SAVE H REGISTER LOCATION.
LPE SAVE H REGISTER.
```

Main Body of Interrupt Program may use all flags and A, L, H registers.

```
INP 006 GET SAVED FLAG BITS.
NDI 00B RESTORE FLAGS 6 AND 7.
LLI 040 POINT TO SAVED...
LLI 370 . A REGISTER LOCATION.
LAM RESTORE A REGISTER.
LLI 371 POINT TO SAVED H REGISTER.
LHM RESTORE H REGISTER.
LLE RESTORE L REGISTER.
RET RETURN TO MAIN LINE.
```

Figure 17.9.3--Software for Saving Status Using Above Hardware

In Figure 17.9.3, the INP 007 instruction (INP 007) not only saves the flags, but also destroys the A register as would any other input instruction. It is therefore necessary to save the L register, set it up, and save the A register before executing the INP 7 instruction. Once the flags are saved, then the INL instruction, which could change the flags, can be used.

The INP 7 instruction should not, in general, be used in the main body of the interrupt program, lest the saved flags be lost. As an academic exercise, the reader might want to write a program which does use INP 7 within the body of the interrupt program. The program uses two instructions, plus the INP 7 instruction, and avoids destroying the saved flags. However, since any input instruction could have been used it is best to choose one which is not needed in the interrupt subroutine.
SEC. 17.9 LATCHING UP THE FLAGS (cont'd)

The circuit of Figure 17.9.1 is a little more flexible than that of Figure 17.9.2 because the same input strobe could be used for both saving and unsaving flags. The second time the input instruction is executed, it will destroy the flags—but not until after the previously-saved information has been read into the microcomputer's input bus. This is because the clock (C) stores information in the 74172 in Figure 17.9.1 at ST4 time, whereas the information has already been read at T3A time. The T3A enable pulse always occurs before the ST4 strobe.

The circuit in Figure 17.9.1 is clocked with a logic signal which is normally high, but goes to zero at the beginning of the IN7 pulse, which corresponds closely to the T3A pulse. The clock signal returns to logic one at the beginning of the 74172 pulse. It is at this time that the flag information is loaded into the 74172's flip-flops.

The circuit in Figure 17.9.2 differs in the fact that the enable input, G, receives a positive pulse. The G input is the complement of the clock pulse used in the circuit of Figure 17.9.1. The 74175 latches begin following the inputs as soon as the enable line goes high, and latches (holds) the information that is on the inputs when the G line goes low at the beginning of the ST4 strobe. The fact that the outputs start following the inputs at the beginning of T3A time (IN7 time) means that the same input instruction cannot successfully read out the data before it changes, as was possible in the previous example.

SEC. 17.10 SAVING ALL FOUR FLAGS WITH HARDWARE

17.10.1 Hardware The circuit shown in Figure 17.10.1 saves all four flags in hardware. It uses a 74176 quad flip-flop to store the four bits of information. The clock for this circuit is developed like the negative pulse for Figure 17.9.1 above. The "Restore Flags Input Port" shown in Figure 17.10.1 consists of either a three-state buffer, or a multiplexer input port.

Figure 17.10.1--Quad Flip-Flop Saves All Four Flags

The bits representing the Sign flag (S) and the complement of the Zero flag (Z) are entered into the input twice so that they will not disturb the parity of the input word, as established by the saved Parity (P) flag.

The resultant input word (INPUT Y) may be represented as follows:

<table>
<thead>
<tr>
<th>BIT NUMBER</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTENTS</td>
<td>C</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
</tbody>
</table>

Figure 17.10.2--Values of Bits Stored in Hardware
**Sect. 17.10 Saving All Four Flags with Hardware (cont'd)**

The two low-order bits are connected to ground, and are therefore read as logical zero.

**17.10.2 Software** The flag-restoring procedure involves executing an INPUT Y instruction, which introduces the manipulated flag values into the A register. The next instruction is ADA, that is, add the A register to itself. This instruction effectively doubles the contents of the A register and is similar to a nine-bit binary left shift one. (See Section 17.6.8 for a similar operation, used in the all-software status-saving procedure.) The ADA instruction affects all the flags, and restores all four to their original status.

Figure 17.10.3 shows the ten possible combinations of flags being saved, then restored.

<table>
<thead>
<tr>
<th>ORIGIAL</th>
<th>WORD STORED</th>
<th>RESULT IN A</th>
<th>RESTORED</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLGS</td>
<td>IN HARDWARE</td>
<td>AFTER ADA</td>
<td>FLGS</td>
</tr>
<tr>
<td>CPZS</td>
<td>C76543210</td>
<td>C76543210</td>
<td>CPZS</td>
</tr>
<tr>
<td>0000</td>
<td>X00011100</td>
<td>00111100</td>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
<td>X01111100</td>
<td>01111100</td>
<td>0001</td>
</tr>
<tr>
<td>0100</td>
<td>X00011100</td>
<td>00111100</td>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
<td>X01111100</td>
<td>01111100</td>
<td>0101</td>
</tr>
<tr>
<td>0110</td>
<td>X00000000</td>
<td>00000000</td>
<td>0110</td>
</tr>
<tr>
<td>1000</td>
<td>X10011100</td>
<td>10011100</td>
<td>1000</td>
</tr>
<tr>
<td>1001</td>
<td>X11111100</td>
<td>11111100</td>
<td>1001</td>
</tr>
<tr>
<td>1100</td>
<td>X1001100</td>
<td>10011100</td>
<td>1100</td>
</tr>
<tr>
<td>1101</td>
<td>X1111100</td>
<td>11111100</td>
<td>1101</td>
</tr>
<tr>
<td>1110</td>
<td>X1010100</td>
<td>10110100</td>
<td>1110</td>
</tr>
</tbody>
</table>

Figure 17.10.3--Saving and Restoring all Four Flags Using Hardware

**Sect. 17.10 Saving All Four Flags with Hardware (cont'd)**

The values of the original flags are shown in the left-hand column in Figure 17.10.3. The word stored in the 74174 is shown in the next column; the X in the C (Carry) column represents that the Carry bit is not defined at this time. That is, it remains whatever it was in the program which precedes the flag-restoring routine. (The value of the original Carry bit has been stored in bit number 7.) After the ADA instruction has been performed, the Carry bit is now accurately defined, as are the other three flags. Note that the left and right-hand columns correspond, indicating that this design saves and restores flags successfully.

**Sect. 17.11 Saving Four Flags, Bus Structured Hardware**

**17.11.1 Hardware** This circuit shown in Figure 17.11.1 operates in a manner similar to the design shown and discussed in Section 17.10. However, it is designed for bus-structured microcomputer inputs, and uses a four-bit latch with three-state outputs, that is, the 74173. This design uses only four input bits; the four low-order bits may be used for other purposes. The restore program masks out these bits when restoring flags.

![Figure 17.11.1--Saving Four Flags: Three-State Outputs, Bus Structure](image-url)
In this circuit, an EXCLUSIVE OR gate, such as one quarter of a 7486 TTL device, helps perform the required logical manipulations of the flag values. The bit loaded into the J3 input on the 74173 is compensated so that the sign and zero bits will not disturb the parity bit.

17.11.8 Software The program used for saving and restoring flags is as follows:

```
INF X  SAVE FLAGS
INF Y  GET FLAG WORD
NDI 360 MASK OUT LOW-ORDER BITS
ADA  RESTORE FLAGS
```

Figure 17.11.3--Instructions for Saving and Restoring Four Flags

The chart in Figure 17.11.3 shows how each of the ten possible flag combinations is stored in the four-bit latch. The values of the four bits are manipulated from the original flag values, as shown. The following symbol denotes the EXCLUSIVE OR operation: ⊕. The results in the A register after the ADA instruction are shown in the following column, and the flags that result from this instruction are shown at right.

<table>
<thead>
<tr>
<th>ORIGINAL FLGS</th>
<th>STORED IN LATCHES</th>
<th>AFTER ADA INSTRUCTION</th>
<th>RESTORED FLGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C P Z S</td>
<td>1D 2D 3D 4D</td>
<td></td>
<td>C P Z S</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
<td>0 0 1 0 0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
<td>0 1 1 0 0 0 0 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0 0 0</td>
<td>0 0 1 0 0 0 0 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>0 0 0 0</td>
<td>0 1 0 1 0 0 0 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 0 0 0</td>
<td>1 0 0 1 0 0 0 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 0 0 0</td>
<td>1 1 1 1 0 0 0 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
<td>1 0 0 1 0 0 0 0</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>1 0 0 1</td>
<td>1 1 0 1 0 0 0 0</td>
<td>1 1 0 1</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>1 0 0 1</td>
<td>1 1 0 1 0 0 0 0</td>
<td>1 1 1 0</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>1 0 0 1</td>
<td>1 0 0 0 0 0 0 0</td>
<td>1 1 1 0</td>
</tr>
</tbody>
</table>

Fig. 17.11.4--Four Flag Bits Manipulated, Stored, Unsaved

17.11.8 Saving Three Flags Only If the parity bit is not used, the OR and XOR gates may be omitted from the circuit of Figure 17.11.1, and the remaining flags will be saved and restored properly. In that case, the software would have to be changed to mask out what would have been the parity bit: NDI 320 would replace the NDI 360 in Figure 17.11.2.
SEC. 17.12 STATUS-SAVING SYSTEM, EXAMPLE I

Note: in this and the following sections, examples are given of practical status-saving systems, based on the information presented in all of the preceding sections of this chapter, both software and hardware.

17.12.1 Description The first status-saving example is for a fully-developed 8088-based microcomputer which has more than 256 bytes of RAM, and a main line program which uses all of the flags and registers. Therefore all are saved. Figure 17.12.1 is a STATUS USAGE CHART whose basic format is repeated, for comparison purposes, in the remaining sections of this chapter.

<table>
<thead>
<tr>
<th>USE</th>
<th>STATUS</th>
<th>REGISTERS</th>
<th>FLAGS</th>
<th>PROGRAM CONTROL STACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN LINE</td>
<td></td>
<td>A B C D E H L C P Z S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERRUPT</td>
<td></td>
<td>A B C D E H L C P Z S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 17.12.1--Status Usage Chart, Example I

17.12.2 Hardware To make this system possible, the hardware contains a register connected to the OUTPUT 16 and INPUT 6 ports, and a second register connected to the OUTPUT 17 and INPUT 7 ports.

17.12.3 Interrupt Subroutine Restrictions The interrupt subroutine can use all index registers and flags. There are eight total program counter stack registers available, and both the main line program, and the interrupt program, must use at least one PC register apiece. This leaves six additional PC stack levels which can be distributed between the main line program and the interrupt program, according to the demands of the software. The boxes in the chart above are left blank to be filled in by the user.

17.12.4 Software Required The remaining figures in this section show the software needed for saving and unsaving status using this design example.

Figure 17.12.2--Example I: Saving Status, Part I, A, B, L Registers

- LAT 000 ASSUME ZERO.
- JZS SAVC IF TRUE GET CARRY BIT.
- LAT 040 ASSUME POSITIVE.
- JPS SAVC IF TRUE GET CARRY BIT.
- LAT 000 SIGN FLAG WAS SET.
- SAVC RAR GET CARRY BIT.
- JTP SAVF JUMP IF PARITY IS EVEN.
- XRI 001 SET PARITY 000.
- SAVF INL POINT TOWARD NEXT RAM BYTE.
- INM SAVE PARITY WORD.

Figure 17.12.3--Example I: Saving Status, Part 2, Flags

- INL
- INM SAVE B REGISTER.
- INL
- INM SAVE C REGISTER.
- INL
- INM SAVE D REGISTER.
- INL
- INM SAVE E REGISTER.

RAM Locations Used (assume RAM addressed at 040 000):

- SAV DEF 0 SAVE H REGISTER BYTE.
- DEF 0 SAVE FLAG WORD.
- DEF 0 SAVE B REGISTER.
- DEF 0 SAVE C REGISTER.
- DEF 0 SAVE D REGISTER.
- SAVE DEF SAVE E REGISTER.

Figure 17.12.4--Example I: Saving Status, Part 3; B, C, D, E Registers
SEC. 17.18 STATUS-SAVING SYSTEM, EXAMPLE I (cont’d)

LHI 040
LLL 000
LMX UNSAVE E
DCL
LMI UNSAVE D
DCL
LCM UNSAVE C
DCL
LMM UNSAVE B

Fig. 17.18.5--Example I: Restoring Status, Part S; B, C, D, E
Registers

DCL
LMX GET FLAG WORD.
DCL POINT TOWARD H REGISTER.
ADA RESTORE ALL FLAGS.

Fig. 17.18.6--Example I: Restoring Status, Part S; Flags

LMH RESTORE H REGISTER.
INF 7 GET SAVED L REGISTER.
LLA RESTORE L REGISTER.
INF 6 RESTORE A REGISTER.
RET RETURN TO MAIN LINE PROGRAM.

Fig. 17.18.7--Example I: Restoring Status, Part T; A, H, L
Registers

SEC. 17.18 STATUS-SAVING SYSTEM, EXAMPLE II

17.18.1 Description This microcomputer has only one page (256 bytes) of RAM and is designed in the manner described in the section in Chapter 13 on the RAM-FAE option. The main line program uses only the Sign, Carry, and Zero flags, never testing the Parity flag. In addition, the main line program never uses the E register.

<table>
<thead>
<tr>
<th>USE</th>
<th>STATUS</th>
<th>REGISTERS</th>
<th>FLAGS</th>
<th>PROGRAM</th>
<th>STACK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>IN USE IN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAIN LINE</td>
<td>PROGRAM</td>
<td>GET</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN USE IN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERRUPT</td>
<td>PROGRAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 17.18.1--Status Usage Chart, Example II

17.18.2 Hardware No hardware save registers are needed in this example. Because the main line program does not use the E register, and the RAM-FAE option is used, software alone is sufficient to save status.

17.18.3 Interrupt Subroutine Restrictions The interrupt subroutine can use only the A, B, H, and L registers. It always sets the Parity flag to logic one before returning to the main line program. (This is indicated graphically in the chart above.)

17.18.4 Software Required The following figures show the software needed for saving and unsaving status using this design example.

LEI MOVE L TO E TEMPORARILY.
LLL SAVE.
LMA SAVE A REGISTER.

Fig. 17.18.2--Example II: Saving Status, Part 1; A and L Registers
SEC. 17.12 STATUS-SAVING SYSTEM, EXAMPLE II (cont'd)

LAI 000 ASSUME ZERO.
JTS SAVC IF TRUE GET CARRY BIT.
LAI 000B ASSUME POSITIVE.
JFS SAVC IF TRUE GET CARRY BIT.
LAI 30DB SIGN FLAG WAS SET.
SAVC RAM GET CARRY BIT.
INL POINT TOWARD RAM FLAG BYTE.
LMA SAVE FLAG INFORMATION.

Figure 17.13.5--Example II: Saving Status, Part 2; C, Z, S Flags

INL
LMB SAVE B REGISTER
INL
LHM SAVE H REGISTER
RAM Locations used in saving status
SAVA DEF 0 SAVE A REGISTER BYTE.
SAVT DEF 0 SAVE T FLAGS.
SAVB DEF 0 SAVE B REGISTER BYTE.
SAVH DEF 0 SAVE H REGISTER BYTE.

Figure 17.13.6--Example II: Saving Status, Part 3; B and H Registers

LHI SAVB
LLZ SAVB
LIM

Figure 17.13.7--Example II: Restoring Status, Part 3; B Register

DCL POINT TOWARD SAVT.
LAM GET FLAG WORD.
DCL POINT TOWARDS SAVED A REGISTER.
ADA RESTORE S, C, Z, S FLAGS, SET PARITY BIT TO ONE.

Figure 17.13.8--Example II: Restoring Status, Part 2; C, Z, S Flags

The fact that the Parity bit is always set by the restore status
routine in Figure 17.13.6 above implies that the occurrence of an
interrupt can be detected by the main line program. A program which
shows the normal way of waiting until after an interrupt has occurred
before proceeding is shown below, in Figure 17.13.8:

HLT WAIT FOR INTERRUPT BEFORE CONTINUING.

Figure 17.13.8--Example II: Main Line Program Waits for Interrupt

The main line program will halt until interrupted. After the inter-
rupt, the program will continue with the instruction which follows the HLT.
The program in Figure 17.13.9 also provides a pause in the main line
program while waiting for an interrupt, but at the same time, checks for
a switch closure occurring on input 4, bit 7.

LAI 001 LOAD ODD PARITY WORD.
ORA RESIST PARITY FLAG
LOOP INP 4 GET INPUT 4 WORD.
RAL MOVE BIT 7 TO CARRY.
CFC SACH CALL ROUTINE IF SWITCH IS CLOSED.
JTF LOOP CHECK SWITCH AGAIN UNLESS INTERRUPT HAS OCCURRED.

Figure 17.13.9--Main Line Program Checks Switch While Waiting
Interrupt
SVC 17.14 STATUS-SAVING SYSTEM, EXAMPLE III

17.14.1 Description
In this simple system, the main line program does not use the D or E registers. It uses only the
Sign and Zero flags.

<table>
<thead>
<tr>
<th>USE</th>
<th>REGISTERS</th>
<th>FLAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A B C D E H L C P Z S</td>
<td></td>
</tr>
<tr>
<td>USED IN MAIN LINE PROGRAM</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>USED IN INTERRUPT PROGRAM</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 17.14.1--Status Usage Chart, Example III

17.14.2 Hardware
No special hardware functions are added.

17.14.3 Interrupt Subroutine Restrictions
The interrupt subroutine uses only the A register; it may use all four flags.

17.14.4 Software Required
See the remaining figures in this section.

LDA SAVE A REGISTER.

Fig. 17.14.2--Example III: Saving Status, Part 1; A Register

LAI 000 ASSUME ZERO.
JZJ SAVZ PUT AWAY IF ZERO.
LAI 060 ASSUME POSITIVE.
JPS SAVR PUT AWAY IF POSITIVE.
LI 300B SET NEGATIVE.
SAVE LDA SAVE FLAG WORD IN D REGISTER.

Fig. 17.14.3--Example III: Saving Status, Part 2; 2, S Flags

SEC. 17.14 STATUS-SAVING SYSTEM, EXAMPLE III (cont'd)

(Not Needed.)

Fig. 17.14.4--Example III: Saving Status, Part 3

(Not Needed.)

Fig. 17.14.5--Example III: Restoring Status, Part 3

LAD GET FLAG WORD.
ADA RESTORE S, Z FLAGS.

Fig. 17.14.6--Example III: Restoring Status, Part 2; 2 and S Flags

LAE RESTORE A REGISTER.
RET RETURN TO MAIN LINE PROGRAM.

Fig. 17.14.7--Example III: Restoring Status, Part 1; A Register

Example III shows how simple a status-saving procedure can be, when only the essential flags and registers are saved. It is important to use the simplest approach, whenever possible.
SEC. 17.16  STATUS-SAVING SYSTEM, EXAMPLE IV

17.16.1  Description
The interrupt is caused by an interval timer.
The interrupt program is designed to test whether
the transmit buffer is empty on a UART (Universal Asynchronous Receiver/
Transmitter). If it is not empty, i.e., there is information ready for
transmission, the interrupt routine returns to the main line program.
If the UART transmit buffer is empty, the transmit buffer is loaded from
a location in RAM, and the interrupt subroutine returns to the main line
program.

The system has more than one page of RAM.
The UART transmit buffer empty flag is connected to the high-order
bit of input port number 5. The main line program uses the Parity, Zero,
and Sign flags, but never uses the Carry bit. The D and E registers are
never used in the main line program.

<table>
<thead>
<tr>
<th>USE</th>
<th>STATUS</th>
<th>REGISTERS</th>
<th>FLAGS</th>
<th>PROGRAM COUNTER STACK</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A B C D E H L C P Z S</td>
<td>&lt;= 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USED IN MAIN LINE PROGRAM</td>
<td></td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USED IN INTERRUPT PROGRAM</td>
<td></td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>✔</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 17.16.1--Status Usage Chart, Example IV

17.16.2  Hardware
The UART is connected to the microcomputer input port
as described above. No special hardware is needed for
the specific purpose of saving or restoring status.

17.16.3  Interrupt Subroutine Restrictions
The interrupt subroutine can use the A, H, and L registers,
and all flags. It uses only one PC stack register.

17.16.4  Software Required
The first part of the interrupt subroutine
(Figure 17.15.2) tests the UART TR buffer
to see whether it is empty. If it is not empty, it returns to the main
line program. This part of the program is very fast, yet it restores
all necessary status before returning.

000 070 LEA SAVE A REGISTER.
000 071 INF 005 GET TR EMPTY FLAG FROM UART.
000 072 BAL MOVE FLAG INTO CARRY BIT, WITHOUT DISTURBING
          P, Z, S FLAGS.
000 073 LAC RESTORE A REGISTER.
000 074 RRC RETURN IF TR BUFFER NOT EMPTY.

Fig. 17.15.2--Example IV: Saves, Unsave Status While Testing UART

In the main body of the interrupt program in Figure 17.15.3, the
reader may, as an exercise, identify the instructions which save and
unsave each needed register and flag, and compare these to the numbered
steps in the preceding sections of this chapter.
SEC. 17.16  STATUS-SAVING SYSTEM, EXAMPLE IV (cont'd)

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>INSTRUCTION</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 075</td>
<td>LDI</td>
<td>SAVE H REGISTER.</td>
</tr>
<tr>
<td>000 076</td>
<td>LEL</td>
<td>SAVE L REGISTER.</td>
</tr>
<tr>
<td>000 077</td>
<td>LLI 040</td>
<td>POINT TOWARD PAGE IN RAM.</td>
</tr>
<tr>
<td>000 101</td>
<td>LLI 370</td>
<td>POINT TOWARD LOCATION IN PAGE.</td>
</tr>
<tr>
<td>000 103</td>
<td>LMA</td>
<td>SAVE A REGISTER.</td>
</tr>
<tr>
<td>000 104</td>
<td>LAL 000</td>
<td>ASSUME ZERO.</td>
</tr>
<tr>
<td>000 106</td>
<td>JZS 000125</td>
<td>IF ZERO GO SAVE.</td>
</tr>
<tr>
<td>000 111</td>
<td>LAI 060</td>
<td>ASSUME POSITIVE.</td>
</tr>
<tr>
<td>000 113</td>
<td>JYS 000120</td>
<td>JUMP IF POSITIVE.</td>
</tr>
<tr>
<td>000 115</td>
<td>LSA 300</td>
<td>SIGN FLAG HAS SET.</td>
</tr>
<tr>
<td>000 120</td>
<td>JTP 000125</td>
<td>JUMP IF PARTY IS EVEN.</td>
</tr>
<tr>
<td>000 123</td>
<td>XRI 001</td>
<td>SET PARTY ODD.</td>
</tr>
<tr>
<td>000 125</td>
<td>DI 001</td>
<td>POINT TOWARD NEXT RAM BYTE.</td>
</tr>
<tr>
<td>000 126</td>
<td>LMA</td>
<td>SAVE FLAG WORD.</td>
</tr>
<tr>
<td>000 127</td>
<td>DNL</td>
<td>LOCATION OF BUFFER POINTER (040 372).</td>
</tr>
<tr>
<td>000 130</td>
<td>LAM</td>
<td>PAGE CONTAINING TRANSMIT BUFFER.</td>
</tr>
<tr>
<td>000 131</td>
<td>LAM</td>
<td>GET NEXT CHARACTER TO TRANSMIT.</td>
</tr>
<tr>
<td>000 134</td>
<td>OMA</td>
<td>SET FLAGS.</td>
</tr>
<tr>
<td>000 135</td>
<td>JZS 000152</td>
<td>IF ZERO MEANS BUFFER EMPTY, GO RESTORE STATUS AND EXIT.</td>
</tr>
<tr>
<td>000 140</td>
<td>OUT 015</td>
<td>OUTPUT NEW CHARACTER TO UART.</td>
</tr>
<tr>
<td>000 141</td>
<td>XRA</td>
<td>LOAD ZERO TO A REGISTER.</td>
</tr>
<tr>
<td>000 142</td>
<td>LMA</td>
<td>NOTE CHARACTER HAS BEEN TRANSMITTED.</td>
</tr>
<tr>
<td>000 143</td>
<td>DCH</td>
<td>POINT TOWARD ...</td>
</tr>
<tr>
<td>000 144</td>
<td>LLI 372</td>
<td>... BUFFER POINTER.</td>
</tr>
<tr>
<td>000 146</td>
<td>LAM</td>
<td>GET BUFFER POINTER.</td>
</tr>
<tr>
<td>000 147</td>
<td>ADD 001</td>
<td>INCREMENT POINTER.</td>
</tr>
<tr>
<td>000 151</td>
<td>LMA</td>
<td>REPLACE POINTER.</td>
</tr>
<tr>
<td>000 152</td>
<td>LLI 040</td>
<td>POINT TOWARD ...</td>
</tr>
<tr>
<td>000 154</td>
<td>LLI 371</td>
<td>... SAVED FLAGS.</td>
</tr>
<tr>
<td>000 156</td>
<td>LAM</td>
<td>GET FLAG WORD.</td>
</tr>
<tr>
<td>000 157</td>
<td>DCH</td>
<td>POINT TOWARD SAVED A REGISTER.</td>
</tr>
<tr>
<td>000 160</td>
<td>ADA</td>
<td>RESTORE FLAGS.</td>
</tr>
<tr>
<td>000 162</td>
<td>LAM</td>
<td>RESTORE A REGISTER.</td>
</tr>
<tr>
<td>000 163</td>
<td>LHI</td>
<td>RESTORE H REGISTER.</td>
</tr>
<tr>
<td>000 166</td>
<td>LLE</td>
<td>RESTORE L REGISTER.</td>
</tr>
<tr>
<td>000 168</td>
<td>RET</td>
<td>RETURN TO MAIN-LINE PROGRAM.</td>
</tr>
</tbody>
</table>

Figure 17.16.3—Main Body of Interrupt Program, Including Saving, Restoring Status

SEC. 18.1  INTERVAL Timers

Two types of interval timer are discussed in this chapter. The first is used to interrupt the microprocessor. The second is connected to an input port, so that the microcomputer can find out that time it is by executing an input instruction. Either interval timer may be preset to a given value with an output instruction.

SEC. 18.2  INTERRUPT INTERVAL TIMER

An eight-bit presettable interval timer requiring only two integrated circuits is shown in Figure 18.2.1. Its only output is the underflow (overflow) terminal, which is connected to an interrupt port of the CPU. (See 18 on interrupts generally.) The NAND gate avoids the possibility of a false interrupt occurring during the parallel load.

NOTE: To use this internal timer in an 8080 microcomputer, the input bits shown connected to the low-order address bus, D7-D0, may be connected to the data bus, DB7-DB0—depending on the I/O instruction mode being used. See Chapter 7.
SEC. 18.2 INTERRUPT INTERVAL TIMER (cont'd)

The counters may be loaded to any desired eight-bit value with an output instruction. The output strobe, DNX, enables the counters' active-low PARALLEL LOAD (TLO) terminals. The clock causes the counters to decrement towards zero, and at the clock time following the counters' having reached an all-zero count, the BORROW terminal goes low. Figure 18.2.2 shows the number 005 (octal) being loaded into the interval timer, and an interrupt strobe being generated six clock periods later. The counter outputs, labeled Q7 through Q0 in Figure 18.2.1, need not be connected to the microprocessor at all. In this example, they are used only to illustrate circuit timing in Figure 18.2.2.

OUTPUT TO INTERRUPT

CLOCK

DNX

Q7

Q6

Q5

Q4

Q3

Q2

Q1

Q0

COUNT 005 004 003 002 001 000 377 376 375 374

---

Fig. 18.2.2--Counters Loaded with 005; Count to Zero; Cause Interrupt

SEC. 18.3 TIMERS FOR INPUT PORTS

18.3.1 MULTIPLEXER INPUTS Another approach to interval timer design is to require the microcomputer to read the time by executing an input instruction. Figure 18.3.1 shows a circuit with TTL outputs suitable for use with an eight-bit microcomputer with a multiplexer input structure. (Input design: see Chapter 8.) Like the circuit in the previous section, the microcomputer can preset this timer by executing an Output X instruction, stroking the counters' PARALLEL LOAD terminals. Here the counters are connected to count up, and the timer's eight-bit output is connected to the input terminals of a multiplexer array. An Input X instruction switches the multiplexer to the DNF position, and the setting of the interval timer is read by the microprocessor.

NOTE: To use this interval timer in an 8080 microcomputer, the input bits shown connected to the low-order address bus, DL7-DL0, may be connected to the data bus, DB7-DB0, depending on the I/O instruction mode used. See Chapter 7.

In an 8080 system, use PS2I rather than SY0--which, in an 8080 system, stops in the Hold mode.

---

Fig. 18.3.1--Interval Timer for Multiplexed Input
SEC. 18.3 TIMERS FOR INPUT PORTS (cont'd)

It is necessary to ensure that the counters do not change state during an input instruction, or else the CPU might garble its input data. The appropriate method for doing this depends on the speed of the clock pulse and on whether the clock can be related conveniently to the timing pulses already available within the microcomputer. In Figure 18.3.1, a relatively low-speed clock is connected to the B input of a D-type flip-flop. In the circuit shown, the clock signal derives from the SYNC (SY) signal developed in the main timing circuitry of an 8008 microcomputer.

When the repetition rate of the SY signal (4.8 us for a standard 8008 clock) is orders of magnitude higher than the clock source, the slight jitter caused by the asynchronous relationship between the clock and the SYNC signal is negligible. But the circuit can be made synchronous by deriving the low-speed clock from the main timing oscillator.

The clock is gated through a D-type flip-flop, using a synchronization signal from the CPU to assure stable counter outputs during data transfer onto the microcomputer bus. In Figure 18.3.1, the rising-edge positive pulse, needed by the 8080 to count, can appear only when the input clock signal has just changed from zero to one, and SY goes high. This occurs at the beginning of T3A time, as the input cycle begins. In an 8080 microcomputer, the signal PRT (the clock signal) may be used in place of SY. This signal occurs even when the CPU is in the hold mode, unlike SY.

18.3.2 Bus-Structure Inputs The interval timer in Figure 18.3.2 is designed to output directly onto the microprocessor input bus. The counters used are therefore three-state devices. The 74LS74 shown are binary (sixteen-count) devices, the 74LS292, a pin-compatible decimal (BDC) counter, could also be used. These types include internal latches between the counters and the output stages. When the IE (enable output) pin is low, the outputs cannot change. This pin is strobed by a timing signal derived from the microprocessor, to make sure that the outputs do not change while the interval timer is driving the microcomputer bus.

In the circuit of Figure 18.3.2, a synchronized clock signal (CLKS) is developed by the same flip-flop circuit discussed in a previous section.
SEC. 18.3 TIMERS FOR INPUT PORTS (cont'd)

CLOCK
SYNC
CLKS
LOW ORDER TIMER BIT
DNX
BBO

Figure 18.3.1—Timing Diagram for Figure 18.3.2 (8088 System)

SEC. 18.4 CLOCKS FOR INTERVAL TIMERS

The microcomputer itself needs a clock to drive its main timing circuitry, as discussed in Chapter 5 above. In many cases the same clock can do double duty by driving the interval timer. The frequency count-down inherent in the microcomputer main timing logic can be utilized by using a signal like 61 or 62. Further time division will require precounters.

SEC. 18.5 PRECOUNTERS

In many cases, the interval timer must develop signals several orders of magnitude slower than those available either from the main timing logic, or from a separate crystal oscillator using a small, reasonably-priced crystal. If accuracy is not a prime concern, of course, a multivibrator based on a timer IC like the 555 could be used. But usually a precounter is more suitable, inserted between a fast signal source in the main timing logic and a presettable interval timer circuit of the type shown earlier in this chapter.

A precounter can be cleared or reset by the same signal (or its complement) used to strobe the presettable stages, in order to assure a predictable pulse interval.
SEC. 19.1 DIGITAL DISPLAYS

An increasing number of electronic products use digital displays to convey information to their operators. Several technologies are available: notably the cold-cathode discharge tube, typified by the Nixie tube trademarked by the Burroughs Corporation; the fluorescent tube; the liquid crystal display, which is coming into its own in low-power applications; and the light-emitting diode (LED) display. We limit discussion to LEDs, not because they are best in every application, but because they interface most easily with TTL circuitry. Cold cathode and fluorescent displays require high-voltage power supplies; liquid crystals require AC drive circuitry and are difficult to multiplex-and though these are not complex problems to surmount, they take us beyond the scope of this book. A small suitcase of valuable applications literature can easily be collected from the manufacturers of the displays and of the integrated circuits used to drive the displays.

SEC. 19.2 SEGMENT DECODERS

The standard method of driving a seven-segment LED display employs a seven-segment decoder such as the 7447 or 7457 TTL IC. A four-bit binary-coded decimal (BCD) signal is presented to its inputs; the chip's active low outputs are connected through current-limiting resistors directly to the LED display. See Figure 19.2.1.

![Diagram of 7447 decoder driving LED display](image)

Fig. 19.2.1—Decoder Drives LED Display

In a microprocessor-based system the BCD input to the decoder usually derives from a latching device. The microprocessor executes an output instruction which strobes the latch and transfers the data on the output bus into the display. The digit displayed does not change until the next output instruction is executed. One method is to use separate latches and decoders, as shown below in Figure 19.2.2.
SEC. 19.2 SEGMENT DECODERS (cont'd)

A reduction in power consumption results when an LED display is pulse-modulated. For example, if an LED is operated with a 25% duty cycle, the peak current required for brightness equal to that obtained with a DC supply would apparently be four times the DC requirement. However, the human eye tends to respond to peak intensities, so that the required power for apparent equal brightness will be as much as 30% less than the DC value. A disadvantage: extra bypass capacitors may be required on the power supply to suppress switching transients. A convenient method of achieving pulse modulation is to strobe the ripple-blanking input (OSI terminal) of the 7447 segment decoder, as shown in Figure 19.2.2.

This schematic also shows the connection of the ripple-banking terminals of the 7447 decoders to suppress non-significant zeros. Thus the number 042.500 is displayed 42.5. The RBO (ripple-banking output) terminals of the 7447s are connected to DTL-type circuitry within the decoders, so that open-collector buffers may be used (as shown) to pull these terminals low. A separate open-collector buffer is provided for each segment decoder, to allow a switching arrangement to change the position of the decimal point. Note that shifting the decimal point with this all-hardware approach might require a relatively complex switching arrangement.

Figure 19.2.3—Zero-Suppression Using Hardware

SEC. 19.3 MULTIPLEXING

In a multi-digit system, a significant savings in gates results from multiplexing. The segment decoders are time-shared by switching rapidly between digits. A typical scheme is shown in Figure 19.3.1. This figure shows the display connected to the output circuitry of a microcomputer. Two digits are addressed at one time with the output instruction; the latches store their values and the displays remain stable until addressed again. Note that, with the 7447 decoder, a decimal 15 (1111) input blanks the digit. The digits can therefore be blanked, when power is first applied to the equipment, by incorporating a few instructions in the power-on initialization subroutine stored in the microcomputer's ROM. These instructions need only load the accumulator with all logic ones, and then output to the digital displays, two-digits at a time. Blankning nonsignificant zeros with a few extra instructions is not overly difficult.

If the 7447's ability to produce a blank display with a binary 1111 input is not needed, it may be replaced with an 8759 decoder, with active-high current-source outputs, and the anode drive transistors eliminated. The 8759 produces a creditable seven-segment display from a four-bit binary input (..., 9, A, B, C, D, E, F).
SEC. 10.3 MULTIPLEXING (cont'd)

NOTE: In an 8080 microcomputer, output data may derive from the DB bus, rather than the DD bus: see Chapters 7, 8.

Figure 10.3.1—Multiplexed LCD Display
goal is to simplify microcomputer input and output selection. The standard method uses digital multiplexers, like those of the 74151 series. Thus for a 8-bit microprocessor with 8 different input sources, the microcomputer would include the digital equivalent of an eight-pole, eight-throw switch. Up to sixty-four separate input lines could be needed to route data from the various inputs to the microprocessor. In a more complex system, extra multiplexers would be needed on the microcomputer board, and would take space and power even when some of the input ports were not in use. And with TTL ICs, all multiplexer inputs which might not be connected in certain system configurations would need pull-up resistors. (Chapter 9 more fully describes these various input techniques.)

Modularity and flexibility are enhanced through use of a bus-structured system. The input devices are all connected through three-state integrated circuits to the same input bus, which leads directly to the input terminals of the microprocessor. Each three-state device is normally inactive and its output is floating—that is, its high impedance state allows some other IC attached to the same bus to determine the logical state. Now only sixteen connections are needed—the eight input bus lines, and up to eight input strobe lines. This both simplifies the wiring on the microcomputer board, and minimizes the number of connections needed between the microcomputer and the other circuit boards in the machine. This technique has been described more fully in Chapter 7.)

SEC. 20.3 AN EFFICIENT MICROCOMPUTER BUS STRUCTURE

An efficient bus structure for microcomputers is shown in Figure 20.3.1. Nearly all of the sections of the block diagram are described in detail in various sections of this book. Though the diagram shows external IN and IN address registers—as would be present in an 8080-based microcomputer—the structure is compatible with the 8080. A very similar design forms the basis of the modular micro series, as exemplified by the Model 471 CPU board, described near the end of this book.

20.3.1 Main Board/Peripheral Organization In this example, the system will be organized as follows—with control signals appropriate for an 8080 system:

IN3 LOW during PCC ... I/O on main CPU board.
IN3 HIGH during PCC ... I/O from a peripheral.
IN5 LOW during PCI, PCC, POW ... memory on main board.
IN5 HIGH during PCI, PCC, POW ... memory in peripherals.

Fig. 20.3.1--An Effective 8080 Bus Structure for Peripheral Interfacing
The above set of conditions for IN3 means that the following inputs and outputs can be used by various peripheral devices: OUT 26, 25, and 34; OUT 27, 28, 29, and 24; OUT 17, 16, 15, and 14; and INP 7, 8, 5, and 4. Each peripheral device develops its own strobe signals on board, using the peripheral strobe decoding techniques described in Chapter 7. Note the omission of OUT 37 above, which, as explained in Chapter 7, cannot readily be developed by the peripheral strobe decoder.

The remaining inputs and outputs may be used on the main CPU board. These include OUT 33, 32, 31, and 30; OUT 23, 22, 21; and 20; OUT 13, 12, 11, and 10; and INP 3, 2, 1, and 0. Since the strobes for OUT 37 and INP 0 cannot readily be developed on the peripheral boards, these two I/O ports may specifically be designated for use on the main CPU board—as shown in 20.3.1.

The constraints shown above for IN3 mean that the upper 8K of memory may be located on the peripheral boards, with the lower 8K of memory on the main CPU board. An example of a memory map which is consistent with this scheme is shown in Figure 20.3.2. Different configurations are possible, obviously.

---

### Data Bus Control

In order to realize the benefits of a bus-structured system, the system shown in Figure 20.3.1 includes a bidirectional bus driver, effectively extending the CPU's bidirectional bus (BB) and providing what is here labeled simply as the data bus (DB). An eight-bit bidirectional bus driver made up of two 74S241 integrated circuits is shown in Figure 20.3.3. The left-hand part of the drawing shows each element of the IC, and the right-hand part of the drawing presents the same circuit element in a more convenient schematic notation. Though not shown in the diagram, each three-state driver and direction control gate in the 74S241 employs Schmitt trigger circuitry. (The bidirectional bus driver was introduced in Chapter 8 above.)
SEC. 20.3 AN EFFICIENT MICROCOMPUTER BUS STRUCTURE (cont’d)

PCC—during an input instruction, where the input port has been assigned to the peripherals.

PCI—when the instruction address refers to a location in memory which is assigned to the peripherals.

POC—when the CPU is reading memory from a location assigned to the peripherals.

Given the system organization defined above, a logic diagram for generating DBC is shown here:

```
    PCC       7400
     |        |     |
  7400      |        |
     |        |     |
T3A       7400   DBC
     |        |  OUT/IN
  7400      |        |
     |        |     |
CC1
     |        |
  7400      |
     |        |
DH5
```

Fig. 20.3.6—Logic diagram for Developing DBC Signal

In the above figure, some simplification results from the fact that CCI is high only during PCI and POC cycles. If the POC signal is not already available on the main CPU board, it may be developed here by ANDing C27 and CCI.

Note also in the figure above that the bidirectional bus driver is allowed to point inwards during output instructions, when the output port being addressed is assigned to the peripherals. This is permissible, as discussed in the section in Chapter 8 on developing input enable signals. During T3A time of an output instruction, the CPU itself is not receiving from the data bus anyway.

20.3.3 Other Control Signals The other control signals shown in Figure 20.3.1 have all been discussed in other sections of this book. Specifically: ENO, EN7, and EN, in Chapter 7 (IN under peripheral strobe strobing techniques); MIN, in Chapters 14 and 26; IDFM, in Chapter 26; F00, SMD, S1L, S1, and S2, in Chapter 5.
Because a fast typist does not always release one key before hitting the next, the ideal keyboard responds only to key closures, without regard for how long each key is held down. The K5740 implements this feature, N-key rollover, by feeding the key detect pulse into a 50-bit shift register and comparing the present status of each key with its condition the last time scanned. Only a transition from 0 to 1 (key closure) strobos the ROM word into the output latches.

Keystroke contact bounce, which could cause unwanted repeated letters, is masked by initiating a delay of several milliseconds following detection of a depressed key, during which further closures are ignored. The capacitor shown at pin 17 produces the recommended delay.

The K5740 requires an external clock oscillator in the 10- to 200 MHz range to drive the scanning counters. This signal is usually available elsewhere in the microcomputer—so it can be derived from the microprocessor main timing circuit. Keyboard encoder chips which include clock oscillators, requiring only an external RC combination, are also available—though, so far, are less popular than the National K5740; the S9021 (American Micro-Systems, S9021, Homestead Rd., Santa Clara, CA 95051); the IA2000 (Elliswood Systems, 591 Ellis St., Mountain View, CA 94042); and the K7270-XX (Standard Microsystems, 36 Norman Blvd., Rampsouge, NY 11787). Each differs in features and performance.

In the K5740, the shift, shift lock, and control make keys are not scanned, but are sensed continuously by the encoder. In the circuit shown, a solid-state lamp displays the shift lock condition—instead of using mechanically interlocking shift and shift lock switches. Just as on an office typewriter, the operator removes the shift lock condition by stroking the shift key.

When the encoder recognizes a keystroke, it sets its DATA STROBE output to a logic one. This terminal is wired directly to the encoder's BUS STROBE CONTROL, an input terminal which resets the encoder output on the next falling edge of the keyboard clock. This data word is thus available at the encoder output for one keyboard clock period only, which allows the encoder to operate at maximum speed. However, if no more temporary storage were provided than the one character stored by the encoder, the microprocessor would have to test for keyboard data at a rapid rate, imposing unnecessarily severe restraints on its software.

![Fig. 21.2.1--Keyboard Interface with Microprocessor: Only Three Chips](image)
SEC. 21.3 THE FIFO

The 2812 FIFO provides up to 32 characters of storage and enables-to-order interfacing. "Application of First-In First-Out Memories," John Springer, Advanced Micro Devices, 901 Thompson Place, Sunnyvale, CA 94086. Also, data sheet for the 2813 32 x 8-bit First-In First-Out Memory. Its PARALLEL IN/OUT input is stroked by the encoder's DATA STROBE output, and loads a keyboard data word every time a keystroke is recognized. After a character is loaded into the FIFO, it bubbles down through the 12 positions until it reaches the output, or is stopped by a previous character. When there is at least one character stored in the FIFO, the 2812 OUTPUT READY signal will go to logic one. This signal is periodically tested by the microprocessor to see whether there is new data from the keyboard. With the 2812 providing buffer storage, the microprocessor needs to test for input data far less frequently.

The FIFO's PARALLEL DUMP (PD) control is permanently enabled by wiring it to 4.5 volts. But the parallel dump function is also internally gated with the OUTPUT ENABLE terminal, so that the first-received character will not be dumped until the OE terminal is activated. Thus a single strobe to the 2812 first reads the keyboard word into the microcomputer, and then dumps the word out of the FIFO, moving the next keyboard character into the output position. Figure 21.3.1 shows the timing. The delay (TD) between the leading edge of the DATA INPUT STROBE and the appearance of valid data on the microcomputer input bus is less than 40 ns for the 2812 (Figure 21.3.2).

The FIFO's registers are cleared out when power is first applied by a signal, WR, from the microcomputer's master reset circuit. This signal goes LOW for a fraction of a second and prevents the FIFO from taking on initial random states that could be interpreted as keyboard data.

SEC. 21.4 INPUT TO MICROPROCESSOR

The design discussed here is for an eight-bit microprocessor, like the 8008 or 8080, with bus-structured input circuitry. The input devices are connected together onto the same input bus, leading directly into the microprocessor's data input terminals. The input devices use three-state buffers.

Fig. 21.3.1—Loading Keyboard Encoder Loads One Character into FIFO

Fig. 21.3.2—Inherent Bus Delay and Guaranteed Stability of Data During Transfer.
SEC. 21.4 INPUT TO MICROPROCESSOR (cont'd)

This system uses two input instructions for loading information from the keyboard: STATUS INPUT and DATA INPUT. The microprocessor periodically does a STATUS INPUT instruction, sending a pulse down the STATUS INPUT STORE line. This strobe activates a 74125 three-state buffer, which sends the FIFO's OR (output ready) bit onto the microprocessor bus, input bit 7 (NB7). The microprocessor tests this bit for the availability of keyboard data. If the bit is set to logic one, the microprocessor decides that keyboard data is ready, and then executes a DATA INPUT instruction. This instruction causes a strobe at the DATA INPUT STORE line, which in turn activates the OE (output enable) terminal of the FIFO. Now the FIFO's three-state outputs are enabled, and the keyboard data word is impressed onto the microprocessor input bus.

The seven lower-order bits of the keyboard data word are the ASCII-encoded character. The high-order bit is the B9 output from the encoder, a selective repeat bit. The accomplishment of the repeating character function is one of the interesting features of this design.

SEC. 21.5 THE REPEAT FUNCTION

A repeating strobe—as for underlining and backspacing—could have been provided by connecting the M5740 REPEAT terminal, through a REPEAT keyswitch to an external 10-Hz clock. The M5740AE internal gating would allow repeated characters only when its selective repeat bit (B9) is logic one, as it is for I, the backspace, the hyphen, etc. However, this repeat circuit would require an external 10-Hz clock and therefore additional hardware.

In this design, B9 is simply connected to the high-order microprocessor input bus. The REPEAT switch is connected to the next most significant microprocessor bit through a 74125 three-state buffer. The repeat function can now be implemented through a few extra instructions in the microprocessor's programs. This technique obeys the first axiom in designing with microprocessors: software is better than hardware. Adding instructions to ROM is generally cheaper than adding chips.

SEC. 21.6 THE REPEAT FUNCTION (cont'd)

Two additional keyswitches can be used to enter data directly onto the microprocessor input bus, using the two remaining 74125 three-state buffers and the same STATUS INPUT instruction. These switches can be used for mode control or a variety of other functions.

Figure 21.5.1 shows a typical subroutine for testing keyboard data, reading data into the microprocessor, and implementing the repeating character function.

<table>
<thead>
<tr>
<th>LABEL</th>
<th>INST.</th>
<th>ADDR.</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>KBD</td>
<td>INP</td>
<td>KBDDY</td>
<td>INPUT KEYBOARD DATA READY AND REPEAT SWITCH BIT.</td>
</tr>
<tr>
<td>RAL</td>
<td>ROTATE DATA READY BIT TO CARRY FLAG. REPEAT SWITCH BIT TO SIGN FLAG.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>RETURN TO CALLER IF NO KEYBOARD DATA IS READY.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INP</td>
<td>KDAA</td>
<td>INPUT KEYBOARD DATA.</td>
<td></td>
</tr>
<tr>
<td>JFC</td>
<td>CHAR</td>
<td>GO PROCESS CHARACTER IF REPEAT SWITCH IS ON (CARRY = 0).</td>
<td></td>
</tr>
<tr>
<td>MX1</td>
<td>177B</td>
<td>MASK OUT SELECTIVE REPEAT (SIGN) BIT.</td>
<td></td>
</tr>
</tbody>
</table>

Program to process characters—will depend on the nature of the machine: terminal, display, printer, etc. 7-bit ASCII character is in A reg. If the sign bit is set, the repeat key is depressed AND the character is repeatable. Program must: save character and sign flag (eg, in B register) and come back to repeat the character so long as the switch is set; and, develop the appropriate repeat interval.

Fig. 21.5.1—Typical 8008 Subroutine for Inputting Keyboard Data

Though the above program uses 8008 mnemonics, the 8080 version would be substantially identical. Translating: 8008 RFC = 8080 RFC; 8008 JNC = 8080 JNC.
### ANALOG INPUTS AND OUTPUTS

#### SEC. 22.1 AN ANALOG INTERFACE SYSTEM FOR MICROCOMPUTERS

The microcomputer is, of course, a digital machine. In many applications, the microcomputer is called upon to sense continually varying analog inputs, or to provide an analog output signal. This chapter contains a brief introduction to simple analog circuitry suitable for interfacing with a microcomputer.

The system to be described receives analog input signals from up to eight channels; has provisions for adjusting the gain and bias of each channel manually for optimum digital resolution; and converts the analog signals to digital bytes for processing by the microcomputer. After digital processing, the microcomputer can reconvert the data to an analog output for analog measurements, control, or testing. This design is based on an actual instrument, a Data Multiplexer, which monitors physiological instrumentation (electrocardiograph, pressure transducers, etc.) in a medical laboratory; preprocesses the information; and dumps data periodically on demand into a large central computer.

#### SEC. 22.2 ANALOG INPUT AMPLIFIERS

The single-channel analog input amplifier shown in Figure 22.2.1 accepts a differential or single-ended analog input signal. The first stage is a conventional differential amplifier; the non-inverting input is connected to the output ground connection of the source instrument. This amplifier responds to the difference between the + and - inputs, which tends to cancel out ground loop noise. The 100 K ohm 1% resistor across the input terminals assures a zero stage output with open input terminals. The stage gain is -1.

![Fig. 22.2.1—Analog Input Amplifier with Gain and Bias Trimmers](image)
**SEC. 22.2 ANALOG INPUT AMPLIFIERS (cont'd)**

The second stage has a gain control allowing manual adjustment in the range of -0.5 to -5. The inversion of the second stage restores the signal's original polarity. A second control biases the output voltage in the range of +5 to -5 volts. The output voltage range is limited to approximately +10 to -10 volts before clipping occurs.

The controls may be calibrated so that a given analog input voltage produces a standard digital result. Another mode of calibration is to adjust the controls so that the input voltage range corresponds closely to the range of the analog-to-digital (A/D) converter that follows. This leads to optimal digital resolution.

For example, assume that the A/D converter accepts voltages in the range of ±5 to ±5 volts (a 10-volt range). With the resistor values shown, this amplifier matches signals with peak-to-peak values of 2.0 to 20.0 volts to the converter. Signals with median values within plus or minus 5 volts of zero may be accommodated.

Note that once the analog gain settings are made for optimal resolution, further adjustments may be performed on the digital data using the microprocessor.

The capacitor shown causes the second op amp to function as a low-pass filter. Signals and noise above the frequency range of interest are kept out of succeeding stages. For this medical electronics application, a gradual rolloff is adequate. For some instrumentation, a higher-order active filter circuit may be required.

For an eight-channel system, eight of these dual op-amp circuits are required.

**SEC. 22.3 ONE-CHANNEL TRACKING A/D CONVERTER**

An effective A/D converter which is suitable for microcomputers needing only one analog input channel is shown in Fig. 22.3.1. Like many practical A/D designs, this circuit creates a trial eight-bit digital byte, and feeds it to a single-chip 8-bit converter (DAC). The analog result is compared to the analog input, and a digital error signal is used to readjust the trial digital byte.

This circuit is based on the MC1408L-8 DAC. (The 1408 is a device with military temperature ratings.) This DAC sinks between 0 and 2 mA of current, the amount being directly proportional to its eight-bit digital input.

---

**Fig. 22.3.1—Eight-Bit Tracking A/D Converter**
The circuit shown in Figure 22.3.1 is configured so as to produce an eight-bit digital result in unsigned-magnitude binary code, with the polarity the inverse of the analog input. That is, an input voltage near +5 volts produces a digital result near 00000000, and an input voltage near -5 volts produces a digital result near 11111111. When this circuit is in use as an 808 microprocessor, the result can be converted to two’s complement notation with two instructions: INP 7; XRI 177. The first instruction inputs the A/D converter to the microprocessor. The second instruction complements all seven of the low-order bits, leaving the high-order bit in its original state. The correspondence between analog inputs and digital values is shown in Figure 22.3.2.

<table>
<thead>
<tr>
<th>ANALOG INPUT</th>
<th>COUNT</th>
<th>A/D OUT</th>
<th>XRI 177</th>
<th>COMPUTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>864</td>
<td>11111111</td>
<td>10000000</td>
<td>-128</td>
</tr>
<tr>
<td>+5.08</td>
<td>+5.06</td>
<td>+5.10</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>+5.04</td>
<td>+5.02</td>
<td>+5.06</td>
<td>1</td>
<td>00000001</td>
</tr>
<tr>
<td>+5.00</td>
<td>+4.98</td>
<td>+5.02</td>
<td>2</td>
<td>00000010</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>+0.08</td>
<td>+0.06</td>
<td>+0.10</td>
<td>126</td>
<td>01111101</td>
</tr>
<tr>
<td>+0.04</td>
<td>+0.02</td>
<td>+0.06</td>
<td>126</td>
<td>01111110</td>
</tr>
<tr>
<td>0.00</td>
<td>-0.02</td>
<td>+0.02</td>
<td>127</td>
<td>01111111</td>
</tr>
<tr>
<td>-0.04</td>
<td>-0.06</td>
<td>-0.02</td>
<td>126</td>
<td>10000000</td>
</tr>
<tr>
<td>-0.08</td>
<td>-0.10</td>
<td>-0.06</td>
<td>126</td>
<td>10000001</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>-5.00</td>
<td>-5.02</td>
<td>-4.98</td>
<td>252</td>
<td>11111100</td>
</tr>
<tr>
<td>-5.04</td>
<td>-5.06</td>
<td>-5.02</td>
<td>253</td>
<td>11111101</td>
</tr>
<tr>
<td>-5.08</td>
<td>-5.10</td>
<td>-5.06</td>
<td>254</td>
<td>11111110</td>
</tr>
<tr>
<td>ERROR</td>
<td>-5.10</td>
<td></td>
<td>864</td>
<td>11111111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>256</td>
<td>00000000</td>
</tr>
</tbody>
</table>

Fig. 22.3.2—Analog Inputs, Digital Outputs for Tracking Design

Note that there are 256 possible levels of sink current through the DAC, corresponding to the possible combinations of its eight input bits, and that one of these levels is zero (all internal current switches turned off). This means that the maximum current sink possible is really 256/256 of 1.0 mA, or about 1.992 mA.

Consider the condition where there is a zero analog input. The voltage reference is adjusted for 2.00 V and the 200-ohm trimmer has been adjusted to provide a reference current of 0.096 mA flowing into the current node. Assume that the counters are properly tracking the input, and read 01111111. Of the DAC’s eight internal current switches, all are on except for the high-order bit. The DAC sinks 127/256 of 2 mA, or about 0.982 mA. The current imbalance forces the node voltage to go slightly positive, and the comparator input becomes positive. The comparator signal is used to control the up/down counters.

As with any device connected to a microprocessor, the counters should not change state while inputting to the CPU. In this design, for the 8080, the counter clock input is gated by the microprocessor main timing signals. The counters may count up only when STS is low and the comparator output is high— and may count down only when STS is low and the comparator output is low. The microprocessor input enable begins at T3 time, which occurs after ST2 and before ST3. This keeps the counter outputs stable during data transfer to the CPU.

Continuing with the example, the positive comparator output causes the counters to count up one step at the next ST3 time. Now the counters read 10000000. The DAC changes its current switches and now sinks 2.00 mA. Now the node voltage is slightly negative, and the comparator output goes negative. At ST2 time, the counters count down. Thus, as with any tracking A/D converter, the counters tend to oscillate around the true value.

However, in this design, the microprocessor sees a steady value. The CPU always reads during T3 time, after ST2. If the value is oscillating, the microprocessor sees only the lower of the two digital outputs from the counters. After the XRI 177 instruction has been executed, the CPU retains the more positive digital result—in this case, 00000000.

The 200-ohm current source trimmer, when adjusted properly, actually provides a bias current corresponding to one-half the least significant 0.125 (1/2 LSB) of DAC sink current. This bias, taken together with the counter gating explained above, causes the converter to output a unique digital value for any given analog input with an accuracy of + or - 1/2 LSB.
This type of A/D converter can change its output value by only one count every clock period. Thus a tracking design is most suitable for continuous signals. When the instrument is turned on, or an analog input voltage applied, up to 256 clock periods may be required for the converter to start tracking the input. This time is reduced by half at power-on initialization by using the microcomputer master reset signal to load the counters to the 01111111 count.

The delay between the falling edge of the start signal and the beginning of the clock, is a little over 500 nanoseconds, in a system running at standard 8086 speed. This is plenty of time for the clock to increment or decrement the counters. The STF is repeated during normal 8086 operations once every 20 microseconds, or more often, so that input signals into the kilohertz range may be sampled reliably. Note however that if the microprocessor program ever executes a halt (HLT) or wait instruction, the STF strobe will not occur, and the A/D converter using this circuit would temporarily cease tracking its input. If software constraints are unacceptable, a slightly more complex clocking scheme will solve the problem.

The MCl4068-A is rated to settle in 300 nanoseconds, typically, and the 311 comparator is also rated to operate within 500 ns.

Figure 22-3.1 shows that analog input voltages outside the converter's range will cause erroneous digital output voltage levels. There is a wraparound effect, whereby counting up from the highest counter output sets the counter at its lowest value. Now the DAC current is far from matching the analog input, but the digital feedback provided by the comparator will cause the counters to count up to try to balance the current node. As a result, when an overflow or underflow occurs, the counters must traverse all 256 counter values before the converters track again. Continuous inputs outside the voltage input limits shown will cause the converter to output a repetitive digital ramp signal. In a design where excessive input voltages are to be expected, the designer may need to include analog voltage clamps on the inputs, or a comparator which detects excessive deviations from 0 volts at the current node.

22.4.1 Need for Hold Circuitry

If the A/D converter of the preceding section is to be expanded to handle two or more channels, several changes in the design are required. It is usually desirable to use only one DAC, because of the relatively high expense of this circuit element. It is also desirable to use only one microcomputer input port. This implies a system of analog switching. See the block diagram in Figure 22.4.1.

22.4.2 Track/Hold Circuitry

The circuit in Figure 22.4.2 is a practical track and hold design for eight channels. When the IHS012 FET is turned on, through a low-gating voltage, the junction between the two 10 K ohm precision resistors is effective connected to the inverting input of the 2040 op amp. The op amp performs as a conventional inverting amplifier with a gain of -1. When the gating signal goes high, the FET is turned off, and the op amp tends to maintain the change on the 0.01
microfused capacitor—a low-leakage device. A switching transistor is used to provide a gating signal referenced to +12 V, a level needed for reliable turnoff of the FET switches.

Fig. 22.4.2—Eight Channel Track and Hold

When the TRACK/HOLD signal is HIGH, the eight channels are all in TRACK mode, and remain so until the signal goes LOW. This contrasts to a sample/hold circuit, where a positive-going strobe would cause the FETs to turn on momentarily on the positive-going transition of the strobe, then return to HOLD mode automatically. Comparing operation of the two circuits, the track/hold circuit is normally in TRACK mode, and the output follows the input. A control signal places the amplifiers in HOLD mode for the purpose of initiating an I/O conversion. Then the amplifiers are normally returned to TRACK mode. The average voltage excursion of the amplifier output is relatively little. With the sample/hold design, the device is normally in the HOLD position, and the output is allowed only when strobed; thus the average voltage excursion is greater.

Part (A) of Figure 22.4.3 shows how the low-order output bit of a conventional latching output port is used to gate the track/hold circuit of Figure 22.4.2. When an Output 7 instruction is executed, the low-order bit sets the track/hold mode (1 = track, 0 = hold), where it stays until the next Output 7 instruction. Part (B) shows an example of the pulse output technique of Chapter 9. The output is normally low (hold mode). When an Output 7 instruction is executed, with the low-order output bit set to 1, the one-shot is triggered. The one-shot is used to stretch the strobe so

Fig. 22.4.3—Track/Hold, Sample/Hold Gating

SEC. 22.6 ANALOG MULTIPLOLER

The eight-channel analog multiplexer circuit of Figure 22.5.1 uses a 7416 decoder to select one of eight input channels. Three bits from output port 7—D7-3, D7-2, and D7-1—are used to select the channel. The selected decoder output turns on the gate of a 5012 FET switch. The 7416 amp input circuitry includes a 10 K ohm resistor input resistor, connected to the selected analog input voltage. The feedback resistor is also a 10 K ohm resistor, with a switched FET switch (permanently turned on) in the feedback path for accurate gain and compensation. The output voltage is the inverse of the analog input voltage; gain is -1.

Note that in this and other schematics in this chapter, there is a separate ground return path for analog (A) and logic (L) signals. The noise caused by TTL switching transients will seriously upset low-current analog circuitry if present in the ground loops.

Fig. 22.5.1—Eight Channel Analog Multiplexer
SEC. 22.6 SUCCESSIVE APPROXIMATION A/D CONVERTER

The tracking A/D converter design discussed earlier in this chapter is not adequate in a multi-channel system, where the input signal is switched and therefore discontinuous. The successive approximation register (SAR) is capable of reaching an eight-bit resolution in eight clock periods, as opposed to a maximum of 256 for the eight-bit counter design. The attractiveness of the SAR is that it uses a nearly perfectly efficient algorithm for reaching an eight-bit result in a digital feedback system. For details, see the applications memo "A Successive Approximation Register" published by Advanced Micro Devices.

Figure 22.6.1 should be considered an inset to Figure 22.3.1, where the AMD 2503 SAR replaces the two binary up/down counters. The TTL outputs of the 2503 are connected to the microcomputer's input multiplexer, and to the 3N08-8 DAC. The comparator output is connected to the D (data) input of the SAR.

Fig. 22.6.1—SAR Replaces Counters in A/D Converter Circuit
The SAR is started with a conditional pulse output from the output 7 port (DT7-4 must be low). During the first clock period following the clock pulse, D7 is set low, with all other bits set high. This trial digital value is outputted to the DAC, and the comparator tells the SAR whether the analog input is greater or smaller than this trial value. At the beginning of the second clock cycle, D7 is set to the proper level, D6 is set low, and the remaining bits are set high. The trial comparison is repeated successively through D0.

The 2503 clock uses TTL input circuitry, imposing one standard TTL load on the signal source. It can operate at a frequency of more than 15 MHz, so that the conversion speed is effectively limited by the DAC and comparator. The 1408-8 DAC takes up to 300 nanoseconds, typically, and the 311 comparator takes about the same period of time. (For faster conversion, a 305 comparator has a delay of less than 50 ns.) Assuming one microsecond for each trial SAR value, the SAR can easily complete a conversion in less than the 20 microseconds standard 8085 instruction cycle time. Looking at the 8085 design example in Chapter 25, the designer could use the 1 MSF signal at the clock (C) input of the 7474 flip-flop in the main timing circuitry to drive the SAR clock.

The SAR provides a signal, conversion complete (CD), that goes low at the end of the approximation cycle. This terminal is intended for applications where it is necessary to test for the availability of analog information. When, as here, the SAR is being driven synchronously by the microcomputer, testing this signal is unnecessary, since the conversion time is known in advance.

Pin 15 of the SAR, the complement of the most significant bit, can be used to output to the CPU, and provides an alternate means of achieving two's complement notation.

A D/A converter circuit was presented earlier in this chapter—as part of the A/D designs. Figure 22.7.1 shows a basic D/A converter using the 1408L DAC and a 741 op amp. In order to preserve the complement notation, the most significant bit is inverted digitally before being presented to the DAC. The eight-bit digital word to be converted derives from microcomputer output port 15 (not shown), and the 42 volt reference derives from a regulated supply such as the 723 circuit presented earlier in this chapter.

In the medical electronics application for which this circuit was designed, the 741 op amp is adequately fast. Note that as the most significant bit changes at the input to the DAC, a relatively large current is switched reflecting the change of sign. This effectively limits the speed of the circuit still further. A more effective design for high-speed operation would omit the digital inverter at the DAC input (pin 5), and instead feed the current sink terminal for the most significant bit to the non-inverting input of a high-gain, low-error op amp. This option is unavailable when designing around the 1408L, with its single current sink terminal.
**SEC. 22.8, MULTIPLE ANALOG OUTPUTS**

Analog outputs may be multiplexed for the same reason that input multiplexers are used: to avoid the duplication of relatively expensive converters. Figure 22.8.1 shows a circuit in which eight sample/hold circuits receive an analog signal from a common DAC, like that shown in the previous section.

![Figure 22.8.1-Eight-Channel Sample/Hold Circuit Used for Output Multiplexing](image)

The analog output channel is selected by a 74145 decoder, driven by the three lower-order bits of an output port (here, Output 17). The next-higher-order bit—65,771.3—must be high to cause an analog output sample signal to be generated; this allows the four remaining Output 17 bits to be used for other control purposes, without interfering with the analog output multiplexing circuitry. A one-shot stretches the output pulse for long enough to allow the 308 op amp to charge or discharge the hold capacitor.

---

**SEC. 22.9, ANALOG RANGE SWITCHING**

The resolution of an A/D converter can be improved by using (cont’d)

![Figure 22.9.1-Analog Range Switching Circuit](image)

**Fig. 22.9.1-Analog Range Switching Circuit**

**Fig. 22.9.2-Digital Outputs Select Analog Ranges**

**Table 22.9.2**

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-5V</td>
</tr>
<tr>
<td>1</td>
<td>-4V</td>
</tr>
<tr>
<td>2</td>
<td>-3V</td>
</tr>
<tr>
<td>3</td>
<td>-2V</td>
</tr>
<tr>
<td>4</td>
<td>-1V</td>
</tr>
<tr>
<td>5</td>
<td>0V</td>
</tr>
<tr>
<td>6</td>
<td>+1V</td>
</tr>
<tr>
<td>7</td>
<td>+2V</td>
</tr>
<tr>
<td>8</td>
<td>+3V</td>
</tr>
<tr>
<td>9</td>
<td>+4V</td>
</tr>
<tr>
<td>10</td>
<td>+5V</td>
</tr>
</tbody>
</table>

---

**Microcomputer Design**
**SEC. 22.2 ANALOG RANGE SWITCHING (cont’d)**

Microcomputer-controlled gain and bias switching. Though this technique may be applied to analog output circuits, in general it is most useful for inputs. Figure 22.9.1 shows a circuit; Figure 22.9.2 shows how the output instruction controls the range of the circuit; and Figure 22.9.3 shows a typical program for use with the 8088.

---

**PROGRAM FOR DRIVING FIGURE 22.8.1**

<table>
<thead>
<tr>
<th>LAC</th>
<th>GET DIGITAL VALUE.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>016 OUTPUT ANALOG VOLTAGE.</td>
</tr>
<tr>
<td>LAC</td>
<td>017 SELECT ANO OP &amp; CHANNEL 7</td>
</tr>
<tr>
<td>OUT</td>
<td>017 DELIVER PULSE TO ATV-7 SAMPLE/HOLD</td>
</tr>
</tbody>
</table>

**PROGRAM TO SELECT ANALOG INPUT FOR GAIN = 2**

<table>
<thead>
<tr>
<th>LAC</th>
<th>100 PREPARE AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>017 ...SET GAIN OF 1</td>
</tr>
<tr>
<td>LAC</td>
<td>005 ANALOG INPUT #5</td>
</tr>
<tr>
<td>INF</td>
<td>007 CONVNT AND</td>
</tr>
<tr>
<td>XRC</td>
<td>177 ADJUST FOR TWO'S COMPLEMENT</td>
</tr>
<tr>
<td>ADA</td>
<td>CAREX = PCG OR NEX.</td>
</tr>
<tr>
<td>JTC</td>
<td>NEX SIGN = ABSOLUTE VALUE OF NO.</td>
</tr>
<tr>
<td>ADA</td>
<td>ACRX = LOWER RANGE.</td>
</tr>
<tr>
<td>JTC</td>
<td>A11 USE RANGE 11.</td>
</tr>
<tr>
<td>JTS</td>
<td>A1</td>
</tr>
<tr>
<td>JMP</td>
<td>A10</td>
</tr>
<tr>
<td>NEQ</td>
<td>XRC 377</td>
</tr>
<tr>
<td>ADA</td>
<td>A12</td>
</tr>
<tr>
<td>JTS</td>
<td>A12</td>
</tr>
<tr>
<td>A10</td>
<td>LAC 200 SEL. GAIN-SET &amp; RANGE 10 (CENTER).</td>
</tr>
<tr>
<td>JMP</td>
<td>CONT (SEE FIG. 22.8.2)</td>
</tr>
<tr>
<td>A1</td>
<td>LAC 200 SEL. GAIN-SET &amp; RANGE 11.</td>
</tr>
<tr>
<td>JMP</td>
<td>CONT</td>
</tr>
<tr>
<td>A12</td>
<td>LAC 240 SELECT GAIN-SET (BIT 3 = 0) AND RANGE 12.</td>
</tr>
<tr>
<td>CONT</td>
<td>OUT 017 SAVE GAIN &amp; BIAS SETTING.</td>
</tr>
<tr>
<td>LAC</td>
<td>005 INPUT WITH ADDING RESOLUTION.</td>
</tr>
<tr>
<td>INF</td>
<td>007</td>
</tr>
</tbody>
</table>

**COMMENTS:**

- USE RANGE 10 FOR VOLTAGE INPUT ± 1.25 V
- USE RANGE 11 FOR VOLTAGE INPUT ± 1.25 V TO +5 VOLTS
- USE RANGE 12 FOR VOLTAGE INPUT -5V TO -1.25 V

---

**SEC. 23 SOFTWARE TRICKS**

**SEC. 23.1 SOFTWARE TRICKS**

The usefulness of a microcomputer may be enhanced as much by good programming as by good hardware. Though the principal purpose of this book is to assist the designer with hardware, a number of simple programs are given in this chapter which can help the programmer perform the job quickly and with efficient use of ROM and RAM storage.

These subroutines have been written for the 8088. Some will help close the gap between the 8088 and the 8086, with its more versatile instruction set.

**SEC. 23.2 CLEAR A REGISTER INSTRUCTION**

XRA is an instruction which EXCLUSIVE ORs the content of the A register with itself, on a bit-by-bit basis. Since either 0 or 1 EXCLUSIVE-ORed with itself produces a result of zero, the whole A register will be cleared.

Since the EXCLUSIVE-OR instruction sets the flags, the status after the XRA instruction will be:

- A REGISTER 000
- CARRY FLAG 0
- SIGN FLAG 0
- ZERO FLAG 1
- PARITY FLAG 0

**SEC. 23.3 COMPLEMENT INSTRUCTION**

XRC 377 will complement every bit in the A register. That is, each bit which had been a one will become a zero, and vice versa. In the 8086, the single-byte instruction CMA (complement the A register) provides this function. Note that with either 8086 or 8088 the instruction XRC 017 complements the last four bits of the A register, leaving the other bits undisturbed.

**SEC. 23.4 MULTIPLE PRECISION ADD AND SUBTRACT**

Both the 8080 and 8086 were designed to facilitate multiple-byte binary addition and subtraction. The 8086 has a sixteen-bit add instruction, named DAW—a feature which the 8088 lacks. Since multiplication and division consist of additions and subtractions, it is not difficult to perform all four basic arithmetic functions.
The following program is a two-byte add routine. The first summand is a 16-bit number whose low-order bits are in the A register and whose high-order bits are in the B register. The second summand is in the C and D registers as shown in Figure 23.4.1. After the execution of the program shown in Figure 23.4.2, the sum will reside in the A and E registers.

**Figure 23.4.2** The Use of the Registers in Performing a Double-Precision Addition.

- **ADC** ADD LOW ORDER BYTES, CREATE CARRY.
- **LDA** SAVE RESULT.
- **LAB** ADD HIGH ORDER BYTES, SECOND SUMMAD, AND CARRY BIT FROM LOW ORDER BYTES.
- **ACD** ADD CARRY.

The heart of this program is the ACD instruction which allows adding together three numbers simultaneously. The first number is in the A register (8 bits), the second number is in the D register (8 bits), and the third number is in the carry register (1 bit). The result of this addition is 9 bits: 8 bits in the A register and a ninth high order bit in the carry register. This ninth bit of the result is shown as C next to the sum in Figure 23.4.1. The result of adding two numbers may be either 16 bits, or 17 bits, depending on the size of the numbers added. If the carry bit of the result is discarded and only 16-bit numbers are used, it is possible to represent numbers between -32,768 and +32,767.

**Note:** The carry is referred to either as a one-byte register or as a flag depending on how it is being used. Both references are to the same actual flip-flop in the 8008 CPU, however.

The program shown in Figure 23.4.3 is used to subtract two 16-bit numbers. The subtrahend is in the B and A registers where the first summand was in Figure 23.4.1. The subtrahend is placed in the D and C registers and the difference ends up in the A and E registers.
SEC. 23.6 CLEAR RAM MEMORY (cont'd)

LHI 040 POINT TOWARD PAGE WHERE RAM IS TO BE CLEARED.
XRA CLEAR A REGISTER.
LIA CLEAR L REGISTER.
LOOP LMA CLEAR MEMORY LOCATION.
INL POINT TO NEXT LOCATION.
JZ  LOOP LOOP UNTIL DONE.

Figure 23.6.1 Clear RAM to Zero Program

SEC. 23.7 INDEXED JUMP TABLE

An indexed jump instruction is one which causes the program to jump to one of several designated locations. Figure 23.7.1 shows a program which jumps to one of five locations, depending on the contents of the L register—which may take on values 0 through 4. Note that it is easiest to program indexed jump instructions if the jump table begins at the start of a new page in ROM.

PROGRAM INITIALIZATION, PERFORMED DURING POWER-ON INTERRUPT ONLY.

LHI 040
LIT 000
LIT 100
LIT 002
LHI 027 PAGE WHERE ROUTINES TO BE JUMPED TO ARE LOCATED

THE FOLLOWING JUMPS TO ROUTINES ACCORDING TO CONTENT OF L REGISTER.

LHI 020 POINT TOWARDS RAM TABLE.
LAM PICK UP L PORTION FROM TABLE.
LHI 040 POINT TOWARDS RAM.
LAM 001 SECOND BYTE OF JUMP INSTRUCTION.
LAM PLACE L PORTION IN JUMP INST.
JMP 040000 EXECUTE CONSTRUCTED JUMP INSTRUCTION IN RAM.

LOC 020000 START OF JUMP TABLE.
010 LOW PORTION OF ROUTINE 0 ADDRESS.
020 " " " 1
075 " " " 2
132 " " " 3
375 " " " 4

SEC. 23.7 INDEXED JUMP TABLE (cont'd)

LOC 027010 START OF ROUTINE 0
LOC 027030 START OF ROUTINE 1
LOC 027055 START OF ROUTINE 2
LOC 027132 START OF ROUTINE 3
LOC 027375 JUMP TO START OF ROUTINE 4
LOC 024000 START OF ROUTINE 4

Fig. 23.7.1--Indexed Jump Program Causes Jump to Location Indicated by L Register

SEC. 23.8 INDEXED LOOPS

An indexed loop is a method of performing a repetitive function, such as in table lookups, or in performing an operation on a list of variables. The DO LOOP in Fortran is a commonly-used statement. Figure 23.8.1 shows a loop which performs some desired operation seven times, before continuing with the next portion of the program.

... LIT 007 PERFORM 7 TIMES.
OPER XXX SOME OPERATION.
JZ  LIT COUNT DOWN.
JZ  OPER LOOP UNTIL DONE.
... CONTINUE WITH MAIN PROGRAM.

Fig. 23.8.1--Indexed Loop Using the L Register

SEC. 23.9 HARD HALT INSTRUCTION

The HLT (Halt) instruction stops a program from being executed only until an interrupt occurs.

The interrupt routine is executed, and then the main line program is free to continue, starting with the instruction following the HLT.
SEC. 23.9  HARD Halt Instruction (cont'd)

instruction.

A HARD Halt instruction can be used to prevent execution of the main line program altogether. The designer may specify a HARD Halt to be executed when machine conditions indicate that a dramatic system failure has occurred. Only a power down cycle will permit the main line program to continue.

A HARD Halt usually consists of a JMP * instruction, which continually jumps to its own location. Many programmers prefer the no-instruction sequence shown in the figure below, which hard-halts with a JMP instruction, but also goes through the HALT state. This will cause the main timing logic to develop a STOPPERR stroke, which can be used to light up a STOPPERR lamp—either on the front panel of a console, or on the circuit board in prototype systems.

| JMP A GO Halt AGAIN.

**Fig. 23.8.1—Two-Instruction HARD Halt Sequence**

If the microcomputer has a repetitive interrupt signal, in addition to any power-on interrupt present in the system—for example, from an interval timer, or from a peripheral device like a keyboard—the two-instruction HARD Halt sequence shown above will not prevent the execution of the associated interrupt subroutines. They will be executed, and then the program will re-enter the HARD Halt. To prevent the system from being interrupted, the interrupt port of the CPU must be disabled. With an 8088, a DI (disable interrupt) instruction at the beginning of the above routine will suffice. (See only a RESET will end the HARD Halt.) The 8086, a flip-flop can be added at the 8086 READY terminal; an OUTPUT X instruction is inserted at the beginning of the above instruction sequence, and the 8086 strobe which is created resets the flip-flop, disabling the interrupt. The MASTER RESET signal would re-enable the interrupt by clearing the flip-flop.

SEC. 23.10  PUSH ALL REGISTERS

This section describes a system with associated software subroutines which are used to PUSH and POP all of the 8086's internal data registers. (A routine for the 8088 appears in Sec. 23.11 of this chapter.) When the PUSH subroutine is called, the current contents of all registers and flags are saved sequentially in

---

SEC. 23.10  PUSH ALL REGISTERS (cont'd)

RAM. When the POP subroutine is called, all registers and flags are restored to the exact status extant at the time the PUSH subroutine was called.

The PUSH subroutine may be called a number of times, from any part of the program or its subroutines, just as long as the microprocessor's program counter (PC) stack is not overflowed. Each POP subroutine will restore the registers and flags as they were the very last time that the PUSH subroutine was called.

The hardware required includes two temporary data storage registers, referred to as TEM 7 and TEM 6 in the program. TEM 7 is loaded with an OUT 017 instruction and read with an INP 007 instruction; TEM 6 is loaded with an OUT 016 and read with an INP 006. Each register is to be implemented with the design for a "one-byte LIFO" shown in Chapter 11 above. Page 040 in RAM is designated as storage for use by the push/pop stack.

The push-on initialization program should include the sequence in Figure 23.10.1, which initializes the PUSH/POP pointer before any PUSH or POP subroutine is called.

LHI 040  POINT TO STACK POINTER (HIGH).
LHI 000  POINT TO STACK POINTER (LOW).
LHI 001  INITIALIZE TO LOC 004001.

**Fig. 23.10.1—Initialization Sequence Presets PUSH/POP Pointer**

The program in Figure 23.10.2 is the PUSH routine, which saves registers and flags, and the program in Figure 23.10.3 is the POP routine which restores status.

**PUSH ALL REGISTERS AND FLAGS**

| 007000/  | 155   | OUT 026 | A to TEM 5, |
| 007001/  | 306   | LAL    | GET L REGISTER, |
| 007002/  | 157   | OUT 027 | SAVE L IN TEM 7, |
| 007003/  | 305   | LAL    | GET H TO SAVE LATER, |
| 007004/  | 056   | LHI 040 | POINT AT STACK POINTER, |
| 007006/  | 056   | LHI 000 | POINT AT STACK POINTER, |
| 007010/  | 367   | LHI    | POINT TO STACK, |
| 007011/  | 370   | IHA    | PUSH H TO STACK, |
| 007012/  | 056   | LHI 000 | ASSUME ZERO FLAG IS SET, |

**Fig. 23.10.2—PUSH All Registers and Flags (cont'd on next page)**
SEC. 28.10  PUSH ALL REGISTERS (cont’d)

007034/  150  JSR  007021  IF TRUE GET CARRY BIT.
007037/  096  LSL  060  ASSUME SIGN BIT IS ZERO.
007040/  120  JFS  007026  IF TRUE GET CARRY BIT.
007043/  066  LSL  000  SIGN BIT WAS SET.
007046/  032  RAR  GET CARRY BIT.
007047/  170  JTP  007034  JUMP IF PARITY IS EVEN.
007050/  094  XEQ  0001  GET CARRY BIT.
007053/  060  INL  INCREMENT STACK POINTER.
007056/  370  LMA  PUSH B REGISTER.
007059/  060  INL  INCREMENT STACK POINTER.
007062/  371  LMB  PUSH B REGISTER.
007065/  060  INL  INCREMENT STACK POINTER.
007068/  372  LMC  PUSH C REGISTER.
007071/  060  INL  INCREMENT STACK POINTER.
007074/  373  LMD  PUSH D REGISTER.
007077/  060  INL  INCREMENT STACK POINTER.
007080/  374  LMED  PUSH D REGISTER.
007083/  060  INL  INCREMENT STACK POINTER.
007086/  117  INP  007  GET L REGISTER FROM 007.
007089/  370  LWA  PUSH L REGISTER.
007092/  060  INL  INCREMENT STACK POINTER.
007095/  115  INP  006  GET A REGISTER FROM 006.
007098/  370  LWM  PUSH M REGISTER.
007099/  060  INL  START OF NEXT PUS.
007099/  060  ASL  GET NEW POINTER.
007099/  066  LL0  POINT TO 000.
007099/  370  LWA  PUSH AWAY NEW POINTER.
007099/  115  INP  006  GET A REGISTER FROM 006.
007099/  007  RET  RETURN TO CALLER.

Fig. 28.10.5—PUSH All Registers and Flags

**POF ALL REGISTERS AND FLAGS**

007063/  056  LHR  040  POINT AT STACK POINTER.
007065/  066  LLL  000  POINT AT STACK POINTER.
007067/  397  LAM  GET STACK POINTER.
007070/  094  SUL  010  INCREMENT POINTER BY 8.
007072/  370  RMA  REPLACE POINTER.
007073/  094  ADD  007  START AT END.
007075/  360  LLA  POINT AT END (SAVED A REGI.
007076/  307  LAM  POP A REGISTER.
007077/  155  OUT  026  A TO TEM 6.
007080/  061  DCL  DECREMENT STACK POINTER.
007080/  397  LAM  POP L REGISTER.
007080/  157  OUT  027  L TO TEM 7.

Fig. 28.10.6 POP All Registers and Flags (cont’d on next page)

**THE FOLLOWING IS RAM MEMORY**

004000/  001  STACK POINTER.
004001/  120  H REGISTER SAVE.
004002/  060  F REGISTERS SAVE.
004003/  060  D REGISTER SAVE.
004004/  060  C REGISTER SAVE.
004005/  060  B REGISTER SAVE.
004006/  060  E REGISTER SAVE.
004007/  060  L REGISTER SAVE.
004010/  060  A REGISTER SAVE.
004011/  061  ABOVE USED ON FIRST PUSH LEVEL.
004012/  061  H REGISTER SAVE.
004013/  061  F REGISTERS SAVE.
004014/  061  D REGISTER SAVE.
004015/  061  C REGISTER SAVE.
004016/  061  B REGISTER SAVE.
004017/  061  E REGISTER SAVE.
004018/  061  L REGISTER SAVE.

Fig. 28.10.7—Address Assignments for PUSH/PUSH Storage of Registers and Flags in Memory (cont’d on next page)
SEC. 23.10  PUSH ALL REGISTERS (cont'd)

040020/
A REGISTER SAVE
ABOVE USED ON SECOND PUSH LEVEL
040021/
H
040022/
F
040023/
B
040024/
C
ET CETERA

Fig. 23.10.4--Address Assignments for PUSH/POP Storage of Registers and Flags in Memory

SEC. 23.11  8080 PUSH/POP INSTRUCTIONS

The 8080's PUSH and POP instructions represent one of the 8080's most significant improvements over the 8028. These single-byte instructions greatly facilitate saving CPU flag and register status.

Unlike the 8008, the 8080 does not have an internal program counter stack; instead it has a full 16-bit stack pointer, referencing any convenient area in external RAM memory, which becomes the CPU's program stack. This stack is addressable through various 8080 instructions, and can be used to store not only program counter values, but registers and flags as well.

For example, to store the B and C registers, a PUSH B instruction is executed. The following steps occur: the stack pointer is decremented by one; the B register is stored at this new location in RAM; SP is decremented again; and the C register is stored there. To restore the status of this register pair, a POP B instruction is executed, and the above process takes place in reverse. The B register is loaded from the RAM location designated by SP; SP is incremented; and the C register is loaded from RAM; SP is incremented again.

The D and E registers also make up a register pair for these instructions (PUSH D, POP D), as do the H and L (PUSH H, POP H). The A register is pushed and popped together with a processor status word (PSW), which contains the 8080 flag values, as follows:

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Z</td>
<td>A</td>
<td>P</td>
<td>CY</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 23.11.1--8080 Flags Stored in Processor Status Word (PSW)

SEC. 23.11  8080 PUSH/POP INSTRUCTIONS (cont'd)

In the above figure, S is the sign bit, Z is the zero bit, AC is the auxiliary carry bit (used by the decimal adjust instruction), P is parity, and CY is carry. PUSH PSW pushes the A register first, then the status word, and for consistency with the PUSH B, PUSH D, and PUSH H instructions, should perhaps be called PUSH A. POP PSW pops the status word first, then the A.

These instructions clearly simplify the storing and restoring of the 8080's flags and data registers. In order to push (that is, save) the program counter, to preserve the memory address from which the program instructions are currently being fetched, it is necessary merely to execute CALL (or RESTART) instructions, and to pop the program counter, to restore the program location, a RETURN instruction is executed.

In short, a CALL instruction can be defined simply as a PUSH PC; JMP instruction. A RESTART instruction is a specialized CALL instruction, referring to the first page in memory, and the same definition applies.

A RETURN instruction is simply a POP PC instruction.

An instruction sequence for saving 8080 flags and registers is shown in Figure 23.11.2.

| PUSH PSW | SAVE A REGISTER, FLAGS |
| PUSH B | SAVE B, C REGISTER |
| PUSH D | SAVE D, E REGISTER |
| PUSH H | SAVE H, L REGISTER |

Fig. 23.11.2--Saving 8080 Flags, Registers

To restore 8080 status, the sequence in Figure 23.11.3 is used:

| POP H | RESTORE L, H REGISTER |
| POP D | RESTORE E, D REGISTER |
| POP B | RESTORE C, B REGISTER |
| POP PSW | RESTORE STATUS, A REGISTER |

Fig. 23.11.3--Restoring 8080 Flags, Registers

The 8080's power-on initialization program should set up the stack pointer to an area of RAM suitable for housing the stack. The instruction LAT SP (load immediate the stack pointer) is convenient.

As the previous section (23.10) and Chapter 17 demonstrate, it
SEC. 24.1 TESTING MICROPROCESSORS

Complete testing of microprocessor chips is usually so complex that it is very difficult to design a general purpose tester which will check all of the characteristics of the device. Instead, a two-part testing program is recommended.

The first step is to use a testing circuit for burning in the microprocessor. The test circuit uses a simple program which is sufficient to reject the majority of defective devices.

The final test is performed in a circuit identical to that in which the microprocessor is expected to operate. This tests the chip with the programs it will actually be required to execute. Unless the equipment being produced will need updating in the field—by reprogramming the ROM—the cost of an elaborate test program is probably not justified.

0100000/015  RST 010 010041/ 276  CPL
010001/000  HLT 010042/ 013  RFE
010002/000  HLT 010043/ 066  LAI 125
010003/000  HLT 010045/ 278  CPN
010004/000  HLT 010046/ 013  RFE
010005/000  HLT 010047/ 264  XRA
010006/000  HLT 010058/ 274  CPE
010007/000  HLT 010059/ 013  RFE
010010/086  LAI 252 01005A/ 066  LAI 222
010112/318  LBA 01005B/ 273  CPD
010113/391  LCB 01005C/ 013  RFE
010114/332  LDC 01005D/ 066  LAI 111
010115/343  LED 010066/ 278  CPC
010116/354  LKE 010067/ 013  RFE
010117/365  LLH 010069/ 066  LAI 244
010228/086  LAI 125 01006A/ 271  CPB
010229/318  LBA 01006B/ 013  RFE
01023/391  LCB 01006C/ 066  LAI 243
01024/332  LDC 01007A/ 271  CPB
01025/343  LED 01007B/ 023  RPS
01026/354  LKE 01007C/ 033  RPS
01027/046  LEI 000 01007D/ 005  RST 000
01028/036  LDI 222 01007E/ 005  RST 000
01029/026  LDI 222 01007F/ 000  HLT
01032/016  LBI 252 010080/ 000  HLT
01035/016  LBI 244
01037/086  LAI 252

Fig. 24.1.1--Sample 8085 Test Program
SEC. 24.1 TESTING MICROPROCESSORS (cont'd)

The MIKE 1 circuit presented in Chapter 23 is quite suitable for 8086 testing. It uses only eight ICs in addition to the 8086, which would be inserted into a socket for testing.

A sample test program which may be employed for burning in 8086s with the MIKE 1 is shown in Figure 24.1.1. This program tests register-to-register transfers and flag conditions. More complete test programs are available with the MIKE 1.

SEC. 24.2 DESIGNING FOR EASY TESTING

In a typical microcomputer, most system failures occur as the result of the breakdown of devices connected directly or indirectly to the microprocessor bus. Since the microprocessor controls most system functions, these failures ultimately cause incorrect instructions to be delivered to the CPU, and show up as deviations in the execution of the CPU's program. Although these failures may appear to have been caused by the CPU itself, actually the microprocessor itself tends to be one of the most reliable devices in a microcomputer.

Essential troubleshooting equipment includes instrumentation which allows the engineer or technician to visualize the microprocessor's software as it is actually being executed. This is covered in sections below.

The designer can help the serviceman by designing extra signals into the microcomputer for testing purposes. For example, an 8086 or 8080 microprocessor generally includes an I/O strobe decoder, as described in Chapter 7 and illustrated in Chapters 25 and 26. Frequent signals from the decoder can be used to control output ports. These extra decoder outputs may be designated as test signals and brought out to test points on the PC board. Say that there is no output test port, but that the D37 strobe is available. The D37 strobe can be used as a test point by turning on the right line of the strobe decoder in the machine's programming, then probing this point to determine whether the program is executing that stage in the program.

Test points for some of the microcomputer's more important signals should be made readily accessible to the repairman. Some suggested connections are discussed in the following section.

SEC. 24.3 SIXTEEN-CHANNEL OSCILLOSCOPE DISPLAY

Without the proper equipment, it is often difficult to trace problems associated with microprocessors. This is true both in the design stages and in production.

The circuit described in this section allows sixteen channels of digital information to be displayed on a single-trace triggered-sweep oscilloscope. It provides a simple but very convenient method of debugging microcomputers equipped with interrupt capabilities.

Figure 24.3.1 shows a 16-channel display constructed with only three TTL and two CMOS integrated circuits. Shown connected to an 8086-based microcomputer, the design is applicable to an 8080. The circuits contains an oscillator which is used to provide continuous repetitive interrupt signals to the microcomputer under test. The signal which indicates the CPU has recognized an interrupt is connected to the oscilloscope's external signal input terminal. For an 8086, this signal is SST (as shown); for an 8080, it would be NMI. The scope's VERTICAL INPUT terminal is connected to the output of the D/A converter. The oscilloscope screen displays sixteen lines of digital information; the appearance is very much like the timing diagrams which appear in various sections of this book. The top trace shows the state of ST, the second of SS and so forth down the screen.

The display is of the alternating-sweep type; that is, sixteen repetitions of the program are needed to display the sixteen channels.

The reader might at first feel that a sixteen-channel display would be too complex or confusing. However, with a little practice in the use of the horizontal position control on the scope, one can line up any desired instruction with the center vertical line on the graph and read all of the microprocessor's state conditions quite easily.

Note also that the resistors in the D/A portion of the circuitry in Figure 24.3.1 are specified to display four groups of four lines each, thus making it easier to distinguish between traces on the screen. The 5% resistors shown produce satisfactory resolution on the screen. Theoretically, since this is a five-bit D/A converter, the resistors should have an accuracy of one part in 32 (3.14) or better, but even a selection of 10% resistors works well (values of 1 k, 2.2 k, 4.7 k, 10 k, 22 k). The four CMOS buffers in the D/A section may also be omitted—this omission leads to a somewhat inferior (though still useful) display appearance. The buffers between the 74150 and the CPU data bus are necessary, however, at least with simple microcomputers with unbuffered data busses, to avoid overloading the CPU.

The oscillator section generates a signal which interrupts the microcomputer every 4 milliseconds. A resistor is connected from the PLS 1 signal, developed by the 8080 main timing circuitry (Chapter 7), to the oscillator input in order to synchronize the interrupt clock with
the microcomputer and avoid the race condition which otherwise might occur. (See Chapter 10.) This is unnecessary with an 8080 CPU, when the microprocessor recognizes the interrupt, it generates an SRTI strobe, incrementing the 7493 counter and triggering the scope, causing the trace to begin moving from left to right. The counter's four output bits are used both to control which one of the sixteen input channels is selected, and as inputs to the D/A converter. The four counter outputs constitute the high-order D/A inputs, and the other outputs are weighted accordingly. Thus each time the counter increments, it selects a different input channel, and moves the trace up the screen one position.

The data on the selected channel moves through the 74150, is inverted to logic true state, and is connected as the fifth, low-order input bit to the D/A converter. As the trace moves from left to right, this data bit follows the input data on the selected channel, causing the trace to move up and down slightly and in effect drawing a logic timing diagram.

For an 8080, the suggested inputs to the 16-channel display include the eight 8080 output bus (TS) lines, S1, S2, S3, S4, S5, S6, S7, and S8, and two spares which may be connected by probes to signals under investigation. With an 8082, the six control signals might include PHT, SYNC, B8IN, W8, SYNCR, and EZO.

The display circuitry should be connected to the microcomputer output bus, as specified above—rather than directly to the CPU bidirectional data bus (8B). This is because the 74150's TS inputs will in most cases overload the 8B bus. (See Chapter 6 on bus structure, Ch. 8 on loading.) When the microcomputer has no output bus, the 16-channel display should include its own low-power buffers to isolate the 74150 from the CPU bus. The 4050 hex CMOS buffers shown in the next section would be suitable for this purpose.

**SEC. 24.4 OCTAL DATA DISPLAY**

The sixteen-channel oscilloscope display presented above is useful for visualizing the logic states of each bit on the data bus, and elsewhere in the microcomputer, as the program is executed. Sometimes, however, it is more convenient to have the logical states on the data bus decoded, and presented in digital form. The circuit shown in Figure 24.3.1 performs these functions. It is useful in debugging programs, finding errors in new hardware designs, and finding faults on microcomputer cards in production.
The potentiometer connected to the 555's THRESHOLD terminal is used to vary the delay period. By moving the control gradually—a taper-turn pot is recommended—the user can make the display move through the program, displaying data on the bus step by step, in octal code.

If more than one interrupt signal is used in the microcomputer under test, additional gating should be provided so that only one particular interrupt will trigger the timer. Alternately, an input strobe, output strobe, or other regularly-executed signal may trigger the timer.

For example, it may be desired to display the bus data only at T3 time. This allows the operator to visualize the instruction codes, input data, memory read data, and memory write data—all of which appear on the data bus of either an 8080 or 8085 microcomputer at this time. Further, the operator may also need to step through the program one instruction cycle at a time. See Figure 24.4.2, an addition to the previous diagram. With the RUN/STEP switch in the STEP position, the 74C68 continually enables the 9368 latch/decoder. Each time the STEP pushbutton is depressed, the CPU READY terminal is enabled by the 7474 flip-flop. The IP is reset at the beginning of the next machine cycle, and the CPU re-enters the WAIT state at the next T3 time. The new T3 bus data is now displayed continuously. With switch in the RUN position, the octal data display operates as described for Figure 24.4.1. In an 8080 system, the signal resetting the flip-flop is (as shown) 74F1 for an 8085, ADV may be used.
In microcomputers based on either the 8080 or the 8086, the CPU's READY terminal is often unused in the final design. This may result when the microprocessor need not wait for slow memory or an external peripheral. Rather than returning this terminal directly to +5 volts, however, it is often better to connect the terminal through a 10 K ohm resistor to +5. This allows a technician or engineer to bring the terminal low when testing the device. When the computer employs an array of pullup resistors on the CPU data bus, the READY pullup resistor can be included in the same resistor package. A test clip can then snap over the resistor pack (or the CPU) to allow the operator to step through each instruction cycle.

SEC. 24.6 USING THE SELF-TRANSFER INSTRUCTIONS FOR TESTING

The Output Any Register option in Chapter 11 above allows the designer to output any CPU index register to a special output port with only a single-byte instruction. The self-transfer instructions are decoded—in 8080 mnemonics, IAA, IBB, ICC, ILD, IEE, IEF, and IEL. With an 8086, these instructions are MOV A,AX; MOV B,B; etc.

A useful testing technique is based on this concept. Self-transfer instructions are inserted by the programmer into the microcomputer's permanently-stored software (in ROM at key points in the program). During normal operation, these instructions are NO-OP (no operation) instructions, and simply take up one instruction cycle of the microcomputer's time.

During testing, these instructions are decoded and visualized on a sixteen-channel oscilloscope display. A S124 comparator and a 74187 flip-flop (as shown in the Output Any Register circuit in Chapter 11) develop the SRT strobe pulse. This signal is used to locate the self-transfer instruction on the oscilloscope screen. One method is to connect one of the two extra display probes of the sixteen-channel display (described earlier in this chapter) to the SRT signal and look for the corresponding pulse on the screen, adjusting the oscilloscope's horizontal position and sensitivity controls for best visualization. The SRT strobe occurs at PCI-T4 time; at the same time, the designated index register appears on the data bus, and will be displayed on the eight lines at the bottom of the screen.

Another method is to use the SRT signal to strobe the oscilloscope display circuit (instead of SRT).

Note that an 8080 377 HALT instruction will be decoded using this technique. (The alternate 300 or 301 opcodes will not be detected.) The 8086 HALT instruction (code 166) will always be detected as well.

SEC. 26.1 A NINE-CHIP MICROCOMPUTER

For applications requiring very little temporary data storage capacity, the RAM may be eliminated altogether. The 8086's seven internal index registers are available for temporary storage. A practical circuit, which can be used as an 8086 tester, is shown in Figure 26.1.

Introduced as the MIKE 1, this micro has been replaced by the MIKE 2, available from Martin Research.

26.1.1 Two-Phase Clock The two-phase clock required by the 8086 is generated with three inverters and a capacitor, using a circuit presented in Chapter 5. A 7474 flip-flop and two 7402 NOR gates (shown here as negative AND gates) make up the counter and decoder stages.

26.1.2 strobe Decoding The strobes ST1, ST2, and so forth, are decoded by a 7405 as shown in Chapter 5. A set-reset flip-flop and an extra NAND gate generate the active-low T3A signal. T3A is not suppressed during PCI cycles, as usual, since there are no PCI cycles—there being no use for memory write (LWE) instructions in a RAM-less machine.

A single 8255 decoder generates both I/O strobe/enable signals (Chapter 7) and memory chip-enable signals. This is done by feeding CCI (sometimes called DWE) to the decoder as one of the variables. When CCI is low at T3A time, either a PCI (instruction fetch) or PCI (memory read) cycle is being executed—so a PRM should be enabled. When CCI is high at T3A time, the cycle is a PCI (command) cycle, and an I/O strobe/enable signal should occur.

26.1.3 Memory Addressing The IL register is of conventional design, and is used to address low-order memory. The CC/IL register does not develop either CCI or IM, neither being necessary in this minimally configured microcomputer.

26.1.4 Interrupts The only interrupt signal provided for is the master reset signal created at power-on. A 7474 flip-flop is connected to the 8086's INTERRUPT terminal. A pushbutton START switch at the flip-flop's SET terminal is depressed by the operator at power-on, interrupting the CPU. This initial interrupt starts the microprocessor (Chapter 16).

The IIR signal resets the CC/IL and IL registers to all-zero conditions. At the following PCI-T4 time, the CPU starts to execute an initialization subroutine stored in ROM, starting at memory location 000 000.
As the CPU recognizes the interrupt signal, it substitutes a TII state for the usual FII-TII state. The resultant STRI stroke resets the interrupt-request flip-flop.

25.1.6 Data Bus Loading The microcomputer employs a bus structure, without pullup resistors or buffers. The ICs which receive from the data bus—the 74073 and 74LS09—require less than the 1.5 mA which the 8088 can deliver. Only one IC drives the data bus, the 1702 PROM, which is a MOS device and outputs the required 3.5 volts in the logic high state without needing pullup resistors. Earlier TTL ICs (Chapter 8) require a pullup resistor. Should an input device be added to the NIKES, care must be taken that the IC is able to satisfy this requirement.

Note that the d1 and d2 inputs to the 8088 do employ pullup resistors. Since the SY (SYNC) terminal must drive the TIA decoder, this circuit employs a low-power Schottky device, the 74LS08, to avoid overloading the CPU.

25.1.8 Indicator Lamps To facilitate CPU testing, a yellow LED lamp is enabled whenever an Input 3 instruction is executed. So long as the CPU is successfully executing a program in which this instruction appears frequently, the yellow lamp stays lit. When the CPU hits a HALT instruction, the red LED lights.

25.2 USING THE CO-DH REGISTER AS A BUFFER

In small microcomputers it is possible to connect input/output ICs and memory chips directly to the CPU data bus—as shown in the previous section. As the microcomputer gets larger, some form of buffering is usually needed. A common method is to use noninverting CDMS buffers (like the 4050), or a pair of hex TTL inverters (like the 7404), followed by 74104s for higher-power bus loads.

The added cost and PCL board area which these buffers involve may be avoided by using the CO-DH register as a buffer. The DL register is loaded from the CO-DH register, rather than from the CPU data bus. The effective load on the BB bus is then only that imposed by the CO-DH register.

In the usual 8088 microcomputer timing sequence, the DL register is latched up at T1 time, and the CO-DH register at T2 time (Chapter 8). Figure 25.2.1 shows the revised timing scheme. At STI time, the CO-DH register is latched up with the contents of the BB bus. At the end of STI time, this information is passed on to the DL register. Then, at
SEC. 25.2 USING THE CC-DH REGISTER AS A BUFFER (cont'd)

Interrupt timing, the CC-DH register is strobed again, and now this register contains its usual information.

**ST1**

**ST2**

**STDH**

CC-DH contents: DL Info

CC-DH Info: DL Info

new DL Info: DL Info

new DH Info: DL Info

Fig. 25.2.2—Buffered CC Register Timing Diagram

One additional NAND gate is required to effect this scheme, in order to strobe the CC-DH register during both ST1 and ST2 times. The STDH strobe is active (high) both at ST1 and ST2 times. In Figure 25.2.1, the NAND gate which develops STIH is shown as a negative-OR gate.

In Figure 25.2.1, the circuit for MXR I (shown in Figure 25.1.1) is modified to show this data buffering technique. All eight CC-DH bits are developed, and low-power Schottky devices are unnecessary, since only one 74173 receives from the BB bus. T3A is generated within the 3205, thus freeing up the additional required gate. And, since SY is fed directly to the 3205, a 7400 replaces the 74LS00 needed before.

SEC. 25.3 A TWENTY-DOLLAR MICROCOMPUTER

The circuit for an 8008 microcomputer which can be built for less than $20.00 in QM quantities is shown in Figure 25.3.1. It uses a 74S286 PROM for program storage (Chapter 16) and develops only two bits of the CC-DH register: CCL and DHL. The two flip-flops that develop these bits serve as buffers for the respectively-numbered bits in the DL register. The remaining six bits of the DL register are obtained directly from the BB bus.

Another change which has been made in this circuit is that the STDH flip-flop has been eliminated. The T3A signal is generated, effectively, within the 3205 strobe decoder. The T3A signal is produced by
connecting its constituent signals to the decoder: T3A = ST ⋅ ST ⋅ SO ⋅ SY. The 3205 combines these signals with C11 and D11 to produce two I/O strobes. (The race problems which ensue are not detrimental to the operation of the microcomputer.)

Since the high-order DL register bits are not used for memory addressing, they may be used to drive current-limited LEDs. Unusual programming techniques are made possible. For example, the lamp connected to DLS will be lit by a RESTART 010 instruction, and extinguished by a RESTART 050 instruction. And, since the DLS bit is not used to distinguish between memory locations in the F/ROM, the program will fetch the instruction in location 010 of the F/ROM following either a RESTART 010 or a RESTART 050 instruction. A JMP or CALL instruction will turn the DLS lamp on or off.

The microcomputer shown in Figure 26.4.1 uses fewer chips than the previous circuits, but is more costly because it contains 2048 bytes of ROM storage. This circuit combines various features of the previous schematics.

Perhaps the most novel feature of this circuit is its economical two-phase clock. Four inverters from a 4448 hex CMOS inverter IC generate \( \#1 \) and \( \#2 \). Note that the values of the RC parts may easily be changed to create an asymmetrical (shaped) clock, should this be desired.
SEC. 26.1 A NINE-CHIP 8008-1 MICROCOMPUTER

This chapter describes a microcomputer design example called the MIKE 4. This microcomputer includes a full ASCII keyboard, and has 2048 bytes of ROM. The design also includes 256 bytes of RAM, and is organized in accordance with the RAM-PAGE option (described in Chapter 16) for easy memory addressing and more flexible interrupt handling. MIKE 4 has one eight-bit output port, and one six-bit input port (in addition to the keyboard). Five extra enable/strobe lines are developed, allowing the user to add input and output ports with little difficulty. The MIKE 4 is useful in applications requiring accurate and consistent timing, since it uses a crystal-controlled clock.

The remaining sections of this chapter describe the operation of the MIKE 4. Though unique features are described, a full description of every function is not provided here, to avoid duplication with the foregoing chapters of this book. Instead, the reader is referred to the relevant chapters.

SEC. 26.2 THE MICROPROCESSOR

The schematic for the MIKE 4 in Figure 26.2.1 shows an 8008-1 microprocessor. This selected device provides an instruction cycle time of 12.5 microseconds—the time needed to execute an instruction, such as ADD, which uses all five internal processor states (T1 through T5). A standard 808 may be substituted if the 12.5 MHz crystal is replaced with an 8.0 MHz crystal, without any other circuit changes. The instruction cycle time would then be 20.0 microseconds.

<table>
<thead>
<tr>
<th>CRYSTAL</th>
<th>61</th>
<th>SY</th>
<th>STATE</th>
<th>UINT.</th>
<th>CYCLE</th>
<th>FREQ.</th>
<th>FREQ.</th>
<th>FREQ.</th>
<th>TIME</th>
<th>TIME*</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(MHz)</td>
<td>(kHz)</td>
<td>(Hz)</td>
<td>(us)</td>
<td>(us)</td>
<td></td>
</tr>
<tr>
<td>8.0</td>
<td>500</td>
<td>250</td>
<td>4.0</td>
<td>20.0</td>
<td>Standard 808 speed.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.0</td>
<td>625</td>
<td>312.5</td>
<td>3.2</td>
<td>16.0</td>
<td>Many if not most 808s run this fast.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.8</td>
<td>800</td>
<td>400</td>
<td>2.5</td>
<td>12.5</td>
<td>8008-1 speed.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 26.2.1--Crystal Frequencies, Timing, 8008 Speeds
The 8008 bidirectional data bus, BB7 through BB0, is connected
directly to the ROM, DH and DL register data inputs, keyboard encoder
outputs, and the Input 8 input port. Note the use of low-power Schottky
devices for some of these functions, so that all the power on the 8008
data bus remains within specified limits. These parts allow the circuit
to function without data bus buffering. (See Chapter 6 on bus structures,
Chapter 8 on input load considerations.)

The clock oscillator produces a 12.8 MHz signal which is divided
by 16 by the 7493 binary counter. The two 74LS08ANDgates combine
the 800 Hz and 1.6 MHz signals from the counter to produce $a_1$ and $a_2$. The
1 K ohm pullup resistors make sure that these signals meet the 3.5-volt
CPU logic high input requirements. The CPU further divides $a_2$ by two,
producing the $SYNC$ (SY) signal, a 400-kHz (2.5-microsecond) state cycle
time. Thus, a five-cycle instruction takes 12.5 microseconds with the
8008-1, as stated above.

A 2050 (74LS248) 3-to-8 decoder connected to the CPU is used to
produce eight strobe/enable signals. The decoder is strobed by $a_2$ and
$SYNC$, which produces signals valid during the middle of each state--
T1, T2, T3, etc.--thus greatly simplifying the development of strobes and
enable signals used for activating inputs and outputs (Chapter 8).

The 2050 is chosen not only for its functional convenience but
because its low-power inputs do not overload the drive power of the CPU.

Also developed are several special-purpose signals. The STAG pulse
begins at the middle of T2 time (at ST2 time) and ends at the middle of
T3 time (at ST3 time). The 7474 flip-flop that develops STAG is reset
by $BST$, which suppresses the development of STAG during memory write
(FWM) instruction cycles.

The TIA signal is developed ANDing STAG and $SYNC$ from the CPU.
This produces a signal which goes high late in T2 time, and low in mid-
T3 time (end of $SYNC$). TIA is used primarily to enable input devices
to drive the CPU data bus, and is also used as an output strobe (Chapter 8).

When (that is, $BST$ modified) is produced by NANDing $MIN$ and $BBS$.
Note the conditions for $BST$ to go low: (1) $BST$ must be low and (2) $CCL$
must be low (3) at TIA time. $BST$ is used to enable ROM memory.

A 74273 eight-bit latch serves as the DL register. It is stroked
at ST2 time and cleared to all-zeros during interrupts by the I$ZH$ signal.

Two 74LS153s make up the DH register. Their low input currents
prevent overloading of the BB bus, which is already driving the DL regis-
ter (about 3.65 TIL load). These chips provide complementary outputs
which are needed in the microcomputer, and they are stroked by the rising
edge of STAG (in other words at the beginning of ST2 time). Like the
DL register, the DH register is cleared out during interrupts by I$NH$.

As always in the 8008 microcomputer, the DH and DL registers are
used to address memory (Chapter 12). In addition, the DH register
temporarily stores output data during output instructions (Chapters 7, 8).

This microcomputer has one page (256 bytes) of RAM memory, made up
of two 2112 256 x 4 RAMs. Memory is organized according to the RAM-PM2
option (discussed in detail in Chapter 12). Memory write instructions
automatically reference RAM memory without setting up the $H$ register, as
implied by the memory enabling signals discussed above. Only the low-order
memory address bits are used; note the connection of the 2112s
to the DL register bus.

A memory read instruction references RAM only if $BBS$ is high. That
is, all thirty-two high-order memory page addresses, 077 through 040,
point to this one page of RAM. This is of great convenience in pro-
gramming (as outlined in Chapter 12). For an illustration, see the
Make 4 memory map in Figure 26.6.1.

This design provides for one 8316 2248 x 8 ROM, occupying pages 007
through 060. The ROM is addressed during memory read cycles when $BST$
is low. The $BBS$ signal combines this condition, thus strobing the
ROM at the correct time for the CPU to receive input data on the BB
bus. This ROM is addressed when $BST$, $BST$, and $BST$ are all low. Other
ROMs may be added using other combinations of $BST$ and $BST$, but $BST$ must
always be low to avoid conflict with RAM addresses.

See Chapter 15 above for an alternate version of this Make 4
memory map, where the ROM-PM option allows access to ROM through input
instructions.
### SEC. 26.6 RAM AND ROM MEMORY (cont'd)

<table>
<thead>
<tr>
<th>PAGE NUMBER</th>
<th>USAGE</th>
<th>PAGE NUMBER</th>
<th>USAGE</th>
<th>PAGE NUMBER</th>
<th>USAGE</th>
<th>PAGE NUMBER</th>
<th>USAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>077</td>
<td>RAM</td>
<td>057</td>
<td>RAM</td>
<td>037</td>
<td>ROM #3 (OPT.)</td>
<td>017</td>
<td>ROM #1</td>
</tr>
<tr>
<td>078</td>
<td>RAM</td>
<td>056</td>
<td>RAM</td>
<td>036</td>
<td>ROM #1 (OPT.)</td>
<td>016</td>
<td>ROM #2</td>
</tr>
<tr>
<td>079</td>
<td>RAM</td>
<td>055</td>
<td>RAM</td>
<td>035</td>
<td>ROM #2 (OPT.)</td>
<td>015</td>
<td></td>
</tr>
<tr>
<td>074</td>
<td>RAM</td>
<td>054</td>
<td>RAM</td>
<td>034</td>
<td>ROM #2 (OPT.)</td>
<td>014</td>
<td></td>
</tr>
<tr>
<td>073</td>
<td>RAM</td>
<td>053</td>
<td>RAM</td>
<td>033</td>
<td>ROM #2 (OPT.)</td>
<td>013</td>
<td></td>
</tr>
<tr>
<td>072</td>
<td>RAM</td>
<td>052</td>
<td>RAM</td>
<td>032</td>
<td>ROM #2 (OPT.)</td>
<td>012</td>
<td></td>
</tr>
<tr>
<td>071</td>
<td>RAM</td>
<td>051</td>
<td>RAM</td>
<td>031</td>
<td>ROM #2 (OPT.)</td>
<td>011</td>
<td></td>
</tr>
<tr>
<td>070</td>
<td>RAM</td>
<td>050</td>
<td>RAM</td>
<td>030</td>
<td>ROM #2 (OPT.)</td>
<td>010</td>
<td></td>
</tr>
<tr>
<td>067</td>
<td>RAM</td>
<td>047</td>
<td>RAM</td>
<td>027</td>
<td>ROM #2 (OPT.)</td>
<td>007</td>
<td></td>
</tr>
<tr>
<td>066</td>
<td>RAM</td>
<td>046</td>
<td>RAM</td>
<td>026</td>
<td>ROM #2 (OPT.)</td>
<td>006</td>
<td></td>
</tr>
<tr>
<td>065</td>
<td>RAM</td>
<td>045</td>
<td>RAM</td>
<td>025</td>
<td>ROM #2 (OPT.)</td>
<td>005</td>
<td></td>
</tr>
<tr>
<td>064</td>
<td>RAM</td>
<td>044</td>
<td>RAM</td>
<td>024</td>
<td>ROM #2 (OPT.)</td>
<td>004</td>
<td></td>
</tr>
<tr>
<td>063</td>
<td>RAM</td>
<td>043</td>
<td>RAM</td>
<td>023</td>
<td>ROM #2 (OPT.)</td>
<td>003</td>
<td></td>
</tr>
<tr>
<td>062</td>
<td>RAM</td>
<td>042</td>
<td>RAM</td>
<td>022</td>
<td>ROM #2 (OPT.)</td>
<td>002</td>
<td></td>
</tr>
<tr>
<td>061</td>
<td>RAM</td>
<td>041</td>
<td>RAM</td>
<td>021</td>
<td>ROM #2 (OPT.)</td>
<td>001</td>
<td></td>
</tr>
<tr>
<td>060</td>
<td>RAM</td>
<td>040</td>
<td>RAM</td>
<td>020</td>
<td>ROM #2 (OPT.)</td>
<td>000</td>
<td></td>
</tr>
</tbody>
</table>

![Memory Map for MKE 4](image)

### SEC. 26.7 I/O STROBE/ENABLE DECODING

A second 1205 decoder is used to produce the eight signals I/O7 through I/O0. This decoder is enabled only at T4A time of an input/output (I/O) instruction (PCI cycle) when C2 = 0 and C1 = 1. The resultant enable signals are, in the schematic, used to activate the keyboard encoder (KBE), and activate the six-bit input port (INS). The input 7 instruction is used as a strobe for the output port, see below. The remaining enable and strobe signals, I/O7 through I/O0, can be used for input/output expansion. (However care must be taken not to overload the CPU data bus.)

### SEC. 26.8 INPUT PORT

A six-bit input port is available for accepting digital data from the outside world. Made of a 4502 three-state MOS buffer, this input is accessed by performing an Input 8 instruction. Note that the KBE has inverters inputs. Note also that all of the input devices on the CPU bidirectional data bus (B0 through B7) are MOS ICs. Noise should have trouble driving the 8008 to the required logic high input voltage of 3.5 volts. If a TTL three-state buffer (with non-inverting inputs) were used—such as the 74367—there would be no assurance that any given part would meet the 3.8-volt specification. An array of 22 K ohm pullup resistors would have to be added to the 80 bus to use either the 74367 or the new eight-bit TTL three-state buffer, the 742240.

### SEC. 26.9 KEYBOARD ENCODER

The MKE 4 includes a full ASCII keyboard. The Electronic Arrays EA 2000 keyboard encoder can handle up to 96 keys, with facilities for shift and control modes. The chip has a ten-by-ten scanning matrix; only the 99th position (X9, Y9) may not be used. Operation is similar to the National keyboard encoder described in more detail in Chapter 21 above.

In this design, when a key closure is detected, the encoder's DATA READY STORE signal goes high, causing an interrupt to be requested of the microprocessor. The CPU handles the processing of the ASCII-encoded keyboard word, as part of an interrupt subroutine. This subroutine contains an Input 8 instruction, which is used to get the keyboard data. The INS strobe enables the encoder's three-state outputs, transferring the keyboard data word directly onto the CPU data bus.

The EA2000 contains its own internal clock, whose speed is determined by the external RC components shown in the diagram.

### SEC. 26.10 INTERRUPT FUNCTION

An interrupt request generated by the keyboard encoder sets a 7474 flip-flop. At the rising edge of the next 8-bit signal, an associated 7474 flip-flop is also set. This provides the interrupt synchronization necessary to avoid a race condition (as discussed in Chapter 16).

When the CPU recognizes an interrupt, it substitutes a T11 cycle for the next T1 state. The microcomputer's state decoding stage decodes this state, producing the T11 strobe at T11 time. This signal sets the two-state set-reset flip-flop shown in the Interrupt Timing section of the MKE 4 schematic, causing the IDAM signal to go low.

The IDAM signal is used to reset the CD-IH and DL registers to an all-zero condition. In effect, the memory address is now 000 000. At the PCI-T3 time that follows, the CPU will fetch the instruction from memory location 000 000. This location contains a 527 010 which forces a call to a subroutine, starting at location 000 010, used for processing keyboard data.
NEW FROM MARTIN RESEARCH...

As the microprocessor revolution continues, microcomputers find their way into an ever-widening variety of applications. Our silver box—a versatile industrial control package—is a case in point, since it puts high-power computer technology to work in a low-cost controller.

Other new products:
- The new DEBUG version of the 8080-based MIKE 3, a low-cost development system with single-step and 32-channel oscilloscope display.
- New special-purpose boards in the modular micro series: a serial communications board; a battery-backed power supply; and a control interface module.

MODULAR MICRO SERIES...

The modular micros from Martin Research provide all the advantages of microprocessor technology. These versatile printed circuit modules communicate via a bus structure, compatible with:
- The MIKE 3 series, based on the industry standard 8080
- The MIKE 2, based on the 8080—suitable for small industrial controllers
- The new MIKE 8, based on the Z80 from Zilog/Mosel—combining the best features of the 8080 and 6800

MODULARITY

Our design philosophy: cost-effective modularity. The CPU board, housing the microprocessor, provides the basic control signals needed for the system. Then, each memory, I/O, or peripheral interface board contains all the logic unique to its special function. Nearly all the costs associated with a keyboard, for example, are on that keyboard module—and the user does not pay for it unless a keyboard is needed. Still, the bus structure is designed for complete flexibility, allowing the full potential of the microprocessor to be put to work as new functions are added.

THE MIKE 3...

The MIKE 3 typifies the modular micro approach. An optimal small system, yet fully expandable. The ATB13 is a three-board system, based on the Model 471 CPU board, with an 8080A microprocessor. (For complete description, see page 4.) The system comes with a Model 420 Console board, featuring a calculatory style keypad and six fully-decodable LED digit. The third board is a Model 423 PROM/RAM board, with 512 bytes (1K) of RAM and a separately MONITOR program in a PROM. The MONITOR allows the user to program the computer with the keyboard, visualizing the results on the LED digits. (The 423 board has capacity for up to 1K of RAM and 2K of PROM.) An ideal small system for prototyping and for educational use, the ATB13 lists for only $395.00, fully assembled and tested.

NEW DEBUG SYSTEM!

Martin Research takes pride in announcing the ATB14, a MIKE 3 computer featuring a powerful but low-cost diagnostic package. Designed for use in development, trouble-shoot-ing, and educational applications, the B14 is one of the most powerful basic development systems available today.

DEBUG PROM...

This software package supplements the MIKE 3 MONITOR by allowing the user to step through programs one instruction at a time, using the console keypad. Unlike the step-run switch on other computers, however— which simply deactivates the CPU based on instructions—the DE-
BUG PROM maintains full processor activity. After executing each instruction, the user can inspect the status of every data register in the computer—any desired memory location (including the program stack) and the 8080A's address registers that flag it. Or, the user can set a breakpoint at any desired location, even in a point-in-PRGM or ROM memory.

32-Channel Scope Display

The 32-channel Microprocessor Data Display allows the user to inspect all important computer signals on an external single trace, triggered sweep oscilloscope, 16 channels at a time. When a switch on the Model 472 Debug Board is pressed, the 16 address lines are displayed; otherwise, the data bus and selected control signals appear on the scope. Two probes on the 472 board may be used to trace any desired signals in the computer.

The ATB14 comes with the 472 Debug Board (with space for custom interface circuitry), 471 CPU board (with 8080A), console board, and 423 PROM/RAM board (MONITOR and DEBUG PROMs, and 1K of RAM). Fully assembled and tested, the ATB14 lists for $495.00. A similar MIKE 8 system, for the Z80, is available. Contact us for prices.
The FOUR KILOBYTE RANDOM ACCESS MEMORY is a member of the family of modular microcomputers from Martin Research. Consisting of 4,096 eight-bit bytes of memory storage (for a total of 32,768 bits), this 4K RAM is fully compatible either with an 8080 system—the MIKE 3—or with an 8008 system—the MIKE 2. Featuring static memory devices, the 405 includes full memory address decoding, with user-programmable addresses. An optional WAIT circuit on the board allows the user to accommodate slow RAM chips to a fast CPU. The board requires a single +5 volt supply, and uses standard 50-pin connectors for plug-in interfacing with the MIKE 3/MIKE 2 bus.

SPECIFICATIONS

Speed
450 nanoseconds access time is standard for 4K RAMs purchased from Martin Research. This speed is suitable both for our 8080 system (MIKE 3) and for the 8008 (MIKE 2).

Power
Requires +5 volt supply, regulated ±5%. Typical current requirement for full complement of 4K, using 32 type 2102A-4 RAM chips, 1.0A.

Interfacing
Connects directly to the modular micro bus with fifty-pin connectors. Memory address is user-selectable using jumpers on the board.

Mechanical
Standard size for Martin modular micro family: 5.5 by 7.0 inches (140 by 178 mm). Printed circuit boards are manufactured by commercial PC houses to meet professional standards, with plated-through holes, solder-mask protection of traces on the solder side of the board, and silk-screened part outlines for ease in assembly and repair.
The 420 Console Board consists of a calculator-type keyboard and a six-digit display. In a typical modular micro system, the keyboard is used to program the microcomputer, and the bright, large (0.3-inch) digits are used to display data and memory addresses.

The keyboard may also be used for any other data input function, and the digits for any other display purpose—under control of the microcomputer. The keyboard is simply an input port, and the display, a set of output ports. The digit drivers will display data in octal, decimal, or hexadecimal (0-9 and A-F) formats.

A significant advance over older console designs—where banks of switches and lights are hardwired into the system control circuitry—the 420 keyboard/display reflects state-of-the-art systems design, as used in the latest generation of small computers.

PHOTO: Closeup shows bright digits, 20 keys.

SPECIFICATIONS

Keyboard interface:
The numerals 0-9, and six special-function keys, are electronically interpreted in hexadecimal and make up microcomputer input port 006. Two more control keys interface as additional bits in input 006. The remaining two keys are related to the system interrupt function.

Display interface: The six LED digits are driven by microcomputer output ports 015, 016, and 017. The 420's decoders display the numerals 0 through 9, and the hexadecimal numbers from A through F, thus: A, b, c, d, e, F.

System interface: Board connects directly to the modular micro bus with standard fifty-pin connectors. Includes bus drivers for adequate drive power in expanded systems.

Power: Requires a single +5 volt supply, ±5%, 950 MA.
The 423 PROM/RAM BOARD is a basic memory component for computer systems in the modular micro family. The 423 has room both for 2K of programmable read-only memory (PROM), and for 1K of random access memory (RAM). Typically the PROM is used for storage of programs essential to system operation—such as the MIKE 3 or MIKE 2 Monitors. The RAM is for storing user's programs, for temporary data storage during program execution, and for storing CPU register status during subroutine and interrupt handling.

An optional WAIT circuit on the board allows the user to accommodate slow memory chips to a fast CPU. In addition, the memory address of each block of PROM and RAM is user-selectable. Both architecture and control structure of the 423 make it suitable for interfacing with a variety of modular micro systems.

SPECIFICATIONS

Memory Capacity
Up to 2K bytes of PROM: uses up to eight type 1702A, 256 by 8 reprogrammable memory devices. Up to 1K of RAM: up to eight type 2112 256 by 4 static RAM chips.

Speed
RAM, 800 ns select time; PROM, 900 ns select time (meets requirements of 8008-1 system without memory wait cycles). Faster memory chips available as options.

Interfacing
Connects directly to the modular micro bus with standard fifty-pin connectors. On-board bus drivers ensure adequate driving power for expanded systems. Typical page addresses: PROMs, pages 000 through 007; RAMs, pages 010 through 013. However, memory addresses are user-selectable using jumpers on the board.

Power
With 256 bytes each of PROM and RAM, requires +5 V, 380 MA; 0 V, 36 MA; regulated 75%. (0 V can be derived from a -12 V supply using optional on-board diodes.)
The Model 424 Serial Communications Board interfaces a microcomputer with a wide range of peripheral devices. A typical use is to interface a modular micro system to a teletypewriter, CRT terminal, or data set.

The 424 converts the eight-bit parallel data format found within the microcomputer, to a serial bit stream that can be transmitted over a single pair of wires. The 424 also receives serial data from an external device, converting the serial format back to eight-bit bytes for use by the computer. The 424 permits locating the microcomputer remotely from the peripheral device, both simplifying interconnections by reducing the number of conductors required, and enhancing data transmission reliability at any distance.

The 424 can handle two completely independent terminals, each capable of both sending and receiving data simultaneously (full-duplex mode). Data transmission need not be synchronous with microcomputer internal timing—thus simplifying interfacing—because the 424 makes use of IC UARTs (universal asynchronous receiver/transmitters). The speed of transmission (baud rate) is programmable, under control of microcomputer output instructions.

The microcomputer to which the 424 board is connected is effectively DATA TERMINAL EQUIPMENT (as contrasted with Data Communications Equipment) under the communications conventions which apply to serial data. The 424 is capable of meeting the principal requirements of RS-232-C, a widely-used serial data interchange standard ("EIA Interface"). The 424 may also be adapted for the 20-milliamp current loop (neutral current interface) found in the Model 33 Teletype, or to other teleprinters using 60 MA current loops.

Optical isolators may be inserted on the 424 board, when exceptional line distance or unusual ambient noise requires this optional feature.

A schematic for the 424-4 version appears on pages 3 through 8.

TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>PART</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC. 1</td>
<td>424 VERSIONS AVAILABLE</td>
<td>2</td>
</tr>
<tr>
<td>SEC. 2</td>
<td>SERIAL/PARALLEL INTERFACING</td>
<td>2</td>
</tr>
<tr>
<td>SEC. 3</td>
<td>SOFTWARE FOR THE 424.</td>
<td>14</td>
</tr>
</tbody>
</table>
SEC. 1 424 VERSIONS AVAILABLE

The 424 board includes provisions for a number of different options and configurations. Some of these options are available in standard versions from Martin Research, to fit certain common functions. Unless otherwise requested, all 424 boards are assembled and tested for operation in MIKE 3 systems (8080 processor) or MIKE 8 systems (Z-80 processor). For operation in a MIKE 2 system (8008 CPU), see Sec. 8 below.

DESCRIPTION AND COMMENTS

AT424-2 Interfaces to a single Model 33 Teletype, 20 MA current loop. Does not include a UART. Requires a routine in the micro's software for developing the appropriate printer timing. (See Sec. 7 below.)

AT424-3 Interfaces to a single Model 33 Teletype, 20 MA current loop. Includes a UART and programmable baud rate generator. Microcomputer communicates with the TTY via parallel data, i.e., memory read/write instructions.

AT424-4 Two-terminal interface; for one Model 33 TTY, 20 MA current loop, and one terminal with an EIA interface. Two UARTs; dual programmable baud rate generator. Can be modified for optoisolation (Sec. 8 below).

AT424-5 Two-terminal interface; for one PR11000 printer, and one terminal with EIA interface. Otherwise similar to AT424-4.

AT424-6 Two-terminal interface; for one PR11000 printer, and one terminal with EIA-style interface, including optoisolation on the input leads from the EIA terminal. Otherwise similar to AT424-4.

SEC. 2 SERIAL/PARALLEL INTERFACING

This section gives a general overview of the purpose of the 424 board, to interface between a modular micro system and peripheral devices, using serial data transfers.

2.1 Serial versus Parallel Data transfers within the micro-computer generally take place eight bits at once, in parallel. However, it is convenient when interfacing with communications devices, such as teleprinters, to use a serial data format, where characters are sent one bit at a time, sequentially. This format potentially saves on connection costs, and also readily lends itself to transmission via radio link or telephone line (after conversion of electrical pulses to audio tones, by a modulator/demodulator, or modem).

The principal function of the 424 board is to convert parallel data to serial, for transmission to peripheral devices external to the microcomputer, and to receive serial data from these peripherals and convert it back to parallel data.
From the point of view of the microcomputer, the peripheral devices connected to the 424 board look like input/output ports, and can be addressed with ordinary memory read and write instructions. In other words, the 424 I/O ports are memory-mapped: each port is addressed as a designated page in the microcomputer's memory. (See Sec. 3 below for software considerations.)

From the point of view of the system, the microcomputer is a data terminal which transmits and receives data serially. Start and stop bits begin and end each character, which may be from five to eight bits long. The number of start and stop bits per character, as well as the number of bits per second (baud rate), are programmed by the microcomputer.

2.2 Serial Data Conventions Many computer peripheral devices now used in conjunction with microcomputers were originally designed for use in communications systems. For example, a Teletype® Model 33 ASR (asynchronous send/receive) teletype-writer can be connected to a standard modem in order to transmit written messages over a telephone line. A standard has been developed by the Electronic Industries Association (EIA) on "Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange," number RS-232-C. This standard is widely used not only in the communications field, but by computer peripheral manufacturers, so that equipment can readily be interconnected. Devices meeting this specification are commonly described as having an "EIA Interface."

The Model 424 Serial Communications board is capable of meeting the principal requirements of RS-232-C, for connection to devices with an EIA interface. The 424 can also be modified for other serial data applications.

RS-232-C sets up specifications in three areas: serial data transfer format; connecting terminals; and electrical signals.

2.3 Serial Data Transfer Format In the example above--a teletypewriter connected to a modem, for transmission of data through a telephone line--the teletypewriter is considered a data terminal, and the modem, a data set. In a modular micro system, the microcomputer which includes the 424 board is the data terminal.

In accordance with the terminology of RS-232-C, the peripheral device connected to the 424 board would normally be a data set. When the serial data interchange is set up according to the full EIA standard, the data terminal and data set are connected with a 25-conductor cable. Approximately 20 of these signals are defined by the standard; most are specialized control signals. Figure 2.3.1 (following page) shows these signals, with a brief description of each. The far-left-hand column (EIA Conn.) shows the conductor number assigned by the EIA standard.
### 2.3.1-424 SERIAL DATA CONNECTIONS

<table>
<thead>
<tr>
<th>EIA Conn.</th>
<th>Ribbon Cable</th>
<th>Signal Description</th>
<th>Signal Function</th>
<th>(See Sec. 8)</th>
<th>Gev.</th>
<th>Data</th>
<th>Control</th>
<th>Timing</th>
<th>Return Path for:</th>
<th>EIA Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminal #1</td>
<td>Terminal #2</td>
<td>To Terminal</td>
<td>From Terminal</td>
<td>From Terminal</td>
<td>From Terminal</td>
<td>Signal</td>
<td>EIA Pin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J3</td>
<td>J4</td>
<td>P2/22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>26</td>
<td>1</td>
<td>Protective Ground</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>38</td>
<td>13</td>
<td>Signal Ground/Common Return</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>28</td>
<td>3</td>
<td>Transmitted Data (TD)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>5</td>
<td>Received Data (RD)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>7</td>
<td>Request to Send (RTS)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>34</td>
<td>9</td>
<td>Clear to Send (CTS)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>11</td>
<td>Data Set Ready (DSR)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>39</td>
<td>14</td>
<td>Data Terminal Ready (DTR)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>43</td>
<td>18</td>
<td>Ring Indicator (RI)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>15</td>
<td>Received Line Signal Detector (RLSD)</td>
<td>X</td>
<td>X</td>
<td>TSET 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>41</td>
<td>16</td>
<td>Signal Quality Detector (SQD)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>45</td>
<td>20</td>
<td>Data Signal Rate Selector (DSRD), Term.</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>45</td>
<td>20</td>
<td>Data Signal Rate Selector (DSRD), Set</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>47</td>
<td>22</td>
<td>Transmit Signal Element Timing (TSET), Term.</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>29</td>
<td>4</td>
<td>Transmit Signal Element Timing (TSET), Set</td>
<td>X</td>
<td>X</td>
<td>TSET 24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>33</td>
<td>8</td>
<td>Receiver Signal Element Timing (RSET)</td>
<td>X</td>
<td>X</td>
<td>TSET 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>27</td>
<td>2</td>
<td>Secondary Transmitted Data (STD)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>31</td>
<td>6</td>
<td>Secondary Received Data (SRD)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>37</td>
<td>12</td>
<td>Secondary Request to Send (SRTS)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>50</td>
<td>25</td>
<td>Secondary Clear to Send (SCTS)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>48</td>
<td>23</td>
<td>Secondary Rec’d Line Signal Detector (SRRLSD)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>42</td>
<td>17</td>
<td>(Reserved for testing)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>44</td>
<td>19</td>
<td>(Unassigned by RS-232-C)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>46</td>
<td>21</td>
<td>Teletype 20 MA Current Return</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>35</td>
<td>10</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>49</td>
<td>24</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 2.3.1-424 SERIAL DATA CONNECTIONS**

---

The first two connections shown in Fig. 2.3.1 are ground conductors; Protective Ground is normally connected to the equipment frame or chassis.

Transmitted Data (TD) is the conductor for serial data originated by the data terminal. It is received by the data set.

Received Data (RD) is the conductor for serial data received by the data terminal. It is transmitted by the data set.

It is possible to achieve a complete serial data interchange with only one shielded pair of wires—made up of one pair for TD and signal ground; another pair for RD and signal ground; and the shield connected to Protective Ground. This hookup would provide full duplex operation: the data terminal could transmit characters to the data set via the TD conductor, at the same time that it is receiving data from the data set via the RD conductor.

In many cases additional signals are necessary in order to properly synchronize data terminal and data set. For example, the data set may need to be able to signal the data terminal that it cannot receive data temporarily, since it is busy with some other task (e.g., dialing a new telephone number). Or, the data set may need to be able to originate a message, rather than leaving the data interchange under the sole control of the data terminal, as in the simplest system. The first four control signals shown in Fig. 2.3.1 (starting with Request to Send) are perhaps the most basic.

Data Set Ready (DSR) is a signal indicating that the data set is properly connected and ready to communicate.

Data Terminal Ready (DTR) is a signal indicating that the data terminal is ready to transmit or receive data.

Request to Send (RTS) is a signal from the data terminal indicating that it wishes to send information to the data set.

Clear to Send (CTS) is a signal from the data set indicating that it is ready to receive data, on the TD line.

Technically, under RS-232-C, the data terminal cannot transmit (via the TD line) unless all four of the above signals are active. In many of the applications contemplated by RS-232-C, moreover, even further control signals are required. For example, when a call is received by the data set, typically it sends the terminal a signal on the Ring Indicator line.

The EIA standard also provides for a secondary data channel, as shown in Fig. 2.3.1 (starting with Secondary Transmitted Data). This second data channel is typically used for remote terminal equipment; for back-up of the primary data channel to ensure reliability; or for transmission of data between data terminal and data set in a direction backwards to that of the primary channel.
The precise method of control interface is specified not only by RS-232-C but by other communications standards. A full discussion is far beyond the scope of this data sheet. In practice, however, few microcomputer applications make use of all of the control signals specified by RS-232-C. In fact, the precise method of "hand-shaking" varies widely among terminals and peripherals which nominally make use of an "EIA interface." The Model 424 Serial Communications Board includes sufficient optional features and connections to allow most practical serial interface problems to be solved. The user will usually need to refer to the specifications of his equipment to develop the exact interface required.

2.4 EIA Connectors

The 424 has provisions for two independent serial data channels, called Terminal #1 and Terminal #2. Each is independently controlled by the microcomputer. The 25 EIA conductors for each terminal (50 conductors total) are brought out at the rear of the board at P2, a fifty-pin header. (In the most simple 424 application, hookup wires may be soldered directly into the 424 board at P2, replacing the header.)

The header, P2, is meant to receive a 50-pin transition connector, J2, into which have been crimped two separate 25-wire ribbon cables of the 3M Delta® type. The other end of each of these two cables is crimped to a 25-pin female Delta-style EIA connector, which is in turn mounted to the wall of the microcomputer enclosure. These two EIA connectors are labeled J3 (for terminal #1) and J4 (for terminal #2).

Conductor assignments for P2/J2, J3, and J4 are shown in Figure 2.3.1 above.

The 424-4, 5, and 6 versions are provided with the fifty-pin header, P2. The 25-wire Delta ribbon cables must be added to the 424 assembly, as are the EIA connectors. The most convenient female EIA socket, for mounting on the microcomputer enclosure (J3 and J4), is the 3M Delta® connector, which can be crimped onto the 25-conductor 3M Delta® cable. An alternate socket, requiring soldering, is the Amphenol DB-25S.

Among suitable male EIA connectors, for plugging into J3 and J4, is Amphenol type DB-25P, plus type DB-51226-1 plastic hood for securing the connector onto an external 25-conductor round cable.

2.5 EIA Electrical Signals

Data and control signals in an EIA serial data system are normally generated from a ±12 V supply. A circuit is considered ON (spacing) when the signal voltage is greater than +3 volts, and OFF (marking) when the voltage is more negative than -3 volts. Voltages in the transition region, between +3 and -3 volts, are undefined. DC line resistance is 3000 to 7000 ohms, and shunt capacitance is 2500 picofarads maximum. The signal must pass through the transition region in less than 1 millisecond, but the maximum rate of voltage change is 30 volts per microsecond.

EIA interface chips, the 1488 and 1489 (with associated shunt capacitors), are used on the 424 board to meet EIA electrical requirements.
DS424
Page 14

SERIAL COMMUNICATIONS BOARD
Model 424

REV. A

A recommended connection scheme to avoid confusion is as follows:

Data terminals use 25-pin EIA female sockets at their chassis enclosures.
Data sets (e.g., modems) use 25-pin EIA male plugs at their chassis enclosures.

To connect a data terminal to a data set, use a 25-conductor extension cord,
consisting of a male EIA connector on one end, and a female at the other.
(There are no wire reversals within the extension cord.)

To connect a data terminal to a data terminal, use a 25-conductor adapter
cord, consisting of a male EIA connector at each end, with selected
pairs of wires swapped at one end, as indicated in Fig. 2.7.1.

2.8 Adapting for Optical Isolation and Current Loop

The EIA connection system needs modifi-
cation to adapt the serial interface for
a 20 MA current loop, for a Teletype® machine. (See Sec. 4. below.)

To permit optoisolation of incoming signals, a return current path must be provided for each
signal that is isolated. Typically, lesser-used conductors, such as the lead for SECONDARY
TRANSMITTED DATA, are used. Referring to Fig. 2.7.1 above, the connector numbers for the
italicized signals are typically used as return paths for the corresponding (Roman type) signals.
(For details, see Sec. 6. below.)

SEC. 3 SOFTWARE FOR THE 424

The 424 board is controlled by program instructions executed by the microprocessor. These
instructions perform two functions with respect to the 424: setting up the serial data format, and
interchanging data. This section presents typical software instructions for these purposes. First,
however, is a brief description of the input/output instructions used by the microcomputer in
interfacing to the Serial Communications board.

3.1 Memory-Mapped I/O Instructions

The 424 board has been set up to make use of
memory-mapped I/O instructions. In other words,
microcomputer instructions which write into memory serve as output instructions, and memory
read instructions are input instructions.

The 8080 microprocessor’s regular input and output instructions, IN and OUT, cannot readily
be used with the 424.

The 424 board has been assigned to three pages in memory: pages 305, 345, and 347 (in
octal; C5, E5, and E7 in hex). As usual in the modular micro series, a page refers to a block of
256 eight-bit memory words which have the same high-order memory address. (In other words,
**REV. A**

### TABLE 3.2.1 - SELECTING FUNCTIONS WITH MEMORY-MAPPED I/O INSTRUCTIONS

The I/N designation in the memory address column in Fig. 3.2.1 is actually a pair of octal digits (00 through 37). The actual value of these digits is used in programming baud rate or serial data format, as explained below.

#### 3.3 Selecting Baud Rate and Reseting UARTs

When a memory-mapped I/O instruction is executed, the references location 306 NN6 (for Terminal #2) or 306 NN2 (for Terminal #1) two functions are performed: first, the IC UART, which carries out serial to parallel data conversions, is initialized. Second, the baud rate (serial data transmission speed) is set up. The baud rate is programmed through four program bits, DL7 through DL4, and can range between 50 baud and 19,200 baud. See Fig. 3.3.1.

In Fig. 3.3.1, the 8080 instruction LDA is shown. This memory read instruction is normally used to load the 8080. A register from the contents of the designated location. In this case, the instruction is used simply to initialize the UART and set up the baud rate, as shown. Any data which may end up in the 8080 A register at the end of the instruction should be disregarded.

Figure 3.3.1 also shows the period, in microseconds, which corresponds to each selected baud rate. The 424 can be used as a programmable interval timer which interrupts the microcomputer repetitively at a set interval. For details see Sec. 9 below.

### TABLE 3.3.1 - SELECTING BAUD RATE AND INTERRUPT INTERVALS

*NOTE:* When baud rates of 134.4, 2000, or 19,200 are selected, a slight error exists in the output frequency: +0.016% (134.5187 Hz); +0.253% (2000.0625 Hz); +3.125% (19,800 Hz), respectively.

#### 3.4 Selecting Serial Data Format

A memory-mapped I/O instruction which references memory location 306 NN7 (for Terminal #2) or 306 NN3 (for Terminal #1) selects the serial data format necessary to interface with external data equipment. The I/N above ranges from 00 to 37. See Fig. 3.4.1.

Executing any memory read or write instruction which references a memory location shown in Fig. 3.4.1 will perform the designated function. The presence of the correct memory address, on the microcomputer address lines, is sufficient to select the serial data format.

The microcomputer data bus is inactive if the instruction is a memory write instruction; and though data may be transferred from the 424 into a CPU register, in the case of a memory read instruction, this data should be disregarded.
3.5 Synchronization In order to synchronize the microcomputer with the serial data hardware on the 424 board, selected memory read instructions can be executed which place status flags on the microcomputer's parallel data bus. Any memory read instruction referencing memory location 305 004 will input Terminal #2 status flags into the CPU. Reading from location 305 000 inputs status flags for Terminal #1.

The status flags appear on the microcomputer's data bus, as shown in Fig. 3.5.1.

### DATA BUS LINE | STATUS FLAG | COMMENTS
--- | --- | ---
DB7 | DATA READY | 1 = ENTIRE CHARACTER RECEIVED AND IS READY TO BE READ BY CPU.
DB6 | PARITY ERROR | 1 = THE PARITY OF RECEIVED CHAR. DOES NOT AGREE WITH SELECTED PARITY
DB5 | TRANSMITTER BUFFER EMPTY | 1 = TRANSMITTER READY TO BE LOADED WITH A NEW CHARACTER TO BE SENT
DB4 | FRAME ERROR | 1 = CHARACTER RECEIVED WHICH LACKED STOP BIT IN PROPER POSITION
DB3 | OVER-RUN ERROR | 1 = NEW CHARACTER RECEIVED BUT OLD CHAR. NOT YET INPUTTED TO CPU
DB0 | (SERIAL INPUT DATA) | SERIAL INPUT BIT, MAY BE USED FOR TEST PURPOSES BY THE COMPUTER.

![Figure 3.5.1-Status Flags for Synchronizing Serial Data to CPU](image)

Note the omission of DB2 and DB1 in Fig. 3.5.1; these data bus lines are not used for this purpose. DB0 carries the serial input data itself, before it has been converted to parallel format. While not a status flag, this bit can be checked by the CPU for test purposes.

3.6 Sending and Receiving Data When all of the initialization and hand-shaking discussed above has taken place through appropriate software instructions, the microcomputer can input and output serial data via the 424 board. Figure 3.6.1 includes typical software instructions for the 8080.

![Figure 3.6.1-8080 Instructions for Sending and Receiving Serial Data](image)
The programmer will recognize that there are a number of instructions for accomplishing the same results as those shown in the chart. For example, to send a character via Terminal #2, the following two 8080 instructions could be used: MVI B, #347, STAX B. A character previously stored in the A register will be transferred to the 424 board.

If the data format selected involves a serial data word of fewer than eight bits, the corresponding data word on the microcomputer data bus is right-justified. That is, the low-order serial data bit always corresponds to DBO, the low-order line on the microcomputer's data bus. In sending a five-bit serial data word, for example, the 424 will disregard the three high-order bits on the microcomputer's data bus. Similarly, in receiving data from the 424, when five-bit serial data is being handled, the microcomputer software should disregard the three high-ordered bits read in on the computer's parallel data bus. Any unused data lines may take on either logical state.

The above text serves as a basic introduction to the 424 board, and to the question of interfacing a microcomputer to a serial data system.

A complete data sheet is provided to purchasers of the 424 Serial Communications Board. In addition to the preceding, it provides the following sections:

**SEC. 4** GENERAL CIRCUIT DESCRIPTION

**SEC. 5** INTERFACING TO THE PR11000 PRINTER: THE 424-5

**SEC. 6** ADDING OPTICAL ISOLATION: THE 424-6

**SEC. 7** ONE-BIT SERIAL OUTPUT: THE 424-2

**SEC. 8** ADAPTATION FOR THE 8080 MICROPROCESSOR (MIKE 2)

**SEC. 9** USING THE 424 AS AN INTERRUPT TIMER

**SEC. 10** CIRCUIT CONNECTIONS FOR 424 OPTIONS

APPENDICES

APP. A MODEL 33 TELETYPE MODIFICATIONS FOR COMPUTER CONTROL

---

**MODEL 477 DEBUG BOARD**

with 32-Channel Scope Display

**DS477**

Page 1

REV. A

The 477 DEBUG BOARD is a special-purpose module in the modular micro series from Martin Research. Designed for use in development, troubleshooting, and educational applications, the 477 supports the following:

A 32-Channel Scope Display allows important microcomputer data and timing signals to be displayed on an external single-trace, triggered sweep oscilloscope. Timing diagrams of data, address, and control lines appear directly on the face of the scope.

A photograph of an oscilloscope tracing generated by the Model 477 Debug Board appears at the right. Connected to a Model 471 CPU board, based on the 8080 microprocessor, the 32 channels include eight control and eight data lines (top of picture) and the sixteen address lines (bottom of photo).

A Delayed Software Interrupt Port provides a software-generated interrupt request signal—used in software-controlled single-step debugging of programs.

Also included: sockets for optional pull-up resistors on data, address, and control lines—for use in multi-processor, DMA, and interrupt polling schemes. Breadboard space for seven 16-pin DIP ICs and one 14-pin DIP.

The Model 477 Debug Board is intended for use with the MIKE 3 computer, based on the 8080 microprocessor, and with the MIKE 8, based on the Z 80.

The 477 plugs into the system's 50-conductor ribbon cable bus, like other boards in the modular micro series.

Order information: catalog AT477 includes an assembled and tested 477 board (less optional pullup resistor pack). The user must add an additional 50-pin transition converter, part number 1001, to the system's ribbon cable if an open position is not already available. Prices: AT477, $89.00; 1001, $6.00. (U.S. prices, subject to change without notice; standard order terms apply.) The 477 is also available as part of the MIKE 3 DEBUG SYSTEM, model AT814.
WHAT'S IN THE SILVER BOX?

We don't know! ...until you tell us.

The silver box is our name for this versatile industrial control package. What's inside depends on your application. Based on a modular micro CPU board, the silver box may include a specialized keyboard and display panel; one or more memory boards; as required: software tailored to the job; and circuitry for interfacing with your system.

The silver box computer is linked with electrical devices which control a process, or sense a condition, or display information about the system. Examples: switches and relays; motors; AC, DC, and stepping; pressure transducers; thermometers; remote terminals; high-speed printers—the variety is endless.

Some current silver box applications suggest its versatility: a time accounting machine; an industrial batch controller; a vehicle dispatching terminal; and a multi-home security system.

Why Microprocessors in Industrial Control?

• Versatility—Once your basic product has been microcomputerized, adding new functions entails plugging in another circuit board or a new programmable memory element (with modified instructions)—not a whole new design.

• Low cost—Microprocessor-based control systems are not only much more powerful than older versions based on relays, motorized timers, or previous generations of electronic logic (such as TTL)—they are actually more economical in more and more applications today. Almost uniquely in today's inflationary economy, microcomputers are growing less and less expensive per control function performed.

OEMS... THE SILVER BOX CAN CONTROL YOUR PROCESS! Tell us your application, and we'll quote costs of a custom-engineered computer control system.
The 471 CPU is a complete 8-bit central processor on a single printed circuit board. Based on the 8080A microprocessor, the 471 generates the essential control signals for a complete MIKE 3 microcomputer system.

The 471 supports the full capabilities of the 8080A microprocessor, including high-speed
2.0 us instruction cycle—and memory addressing of up to 64K (65,536) bytes. The instruction set supports logical (Boolean), arithmetic, and bit-manipulating operations; the CPU can convert
data to decimal (BCD) notation with a single instruction. Logical operations set flags which may be tested by conditional branch instructions to control program flow. For added convenience in memory addressing, six general-purpose internal registers may be addressed not only singly, but
in pairs. There are also double-precision (16-bit) addition and increment/decrement instructions.

The 8080 stack pointer addresses external RAM memory, allowing for practically unlimited
nesting of program subroutines. Through single-byte PUSH and POP instructions, registers, flags, and the program counter can be saved in the external stack, and restored later—greatly facilitating
interrupt handling. The 471 includes circuitry for three interrupt levels, expandable to an eight
level prioritized interrupt system by adding an accessory circuit board. A digital timer restarts
the computer automatically should the CPU fail to recognize interrupt requests for a set period.

Power bus drivers are included for driving expanded systems. However, control by the
8080A of the data and address busses may be suspended by a HOLD request, thereby allowing
for direct memory access (DMA) or multi-processor operation.

The 471 includes a reliable crystal-controlled clock. Specialized control signals are developed
on the CPU board and are present on the system bus to simplify expansion, by minimizing
the extra interface circuitry needed on each new memory or input/output board. These signals not only include those required for 8080-type I/O instructions, but also support the 8080/6502
and 8086 I/O addressing modes. The 50-pin bus structure is compatible with these and other
eight-bit processors, allowing the 471 to be used interchangeably with other CPU boards in interfacing to memory and accessory modules in the modular micro series.

While supporting all the potential of the powerful 8080A microprocessor, the Model 471
CPU board is a good example of the flexibility of the modular micro approach. Optimal for small systems—yet fully expandable.

APPLICATIONS

Industrial The flexible modular micro approach makes the Model 471 ideal for prototyping computers based on the 8080A microprocessor. The 471 is also attractive to the OEM for pilot runs and low-volume production, and can easily be adapted for specialized purposes. Ideal for a whole new generation of intelligent machines, including industrial controllers, computer peripherals, terminals, business machines, and many others.

Educational In design and packaging, the modular micro concept is well-suited for educational purposes. Used by university engineering and computer science departments in formal course work, and by engineers and programmers to keep up with the state of the art.

modular micros  martin research
TECHNICAL DATA

SEC. 1.  CIRCUIT DESCRIPTION

1.1  Introduction

The 471 CPU board is based on the 8080 microprocessor. Normally the most complex element in a MIKE 3 microcomputer, the CPU generates nearly all of the control signals which appear on the system's bus structure.

To form a working microcomputer, the 471 CPU board must be interfaced with memory storage. For example, the Model 423 PROM/RAM board includes space for PROM memory from which to read permanently stored program instructions (such as the MIKE 3 Monitor), and RAM memory for temporary data storage and for the push-pull stack. In addition, a complete microcomputer usually includes input/output boards for interfacing with keyboards, displays, printers, and other peripheral devices.

This circuit description conforms to the breakdown of the schematic diagram (beginning on page 4 of this data sheet), as follows:

1.2  8080 CPU
1.3  Bus Drivers
1.4  Main Timing
1.5  System Control Logic
1.6  I/O Instruction Modes
1.7  Reset Circuitry
1.8  Interrupt Logic
1.9  Hold Circuitry
1.10  Wait Circuitry
1.11  Indicator Lights
1.12  Power Requirements

1.2  8080 CPU

The 8080 is a microprocessor within a single integrated circuit. Based on 17-volt N-channel MOS technology, its 40-pin package includes terminals for an eight-bit bidirectional data bus, D7-D0; a sixteen-bit address bus, A15-A0; power supplies of +12 V, +5 V, and -5 V; and a variety of control signals. The 8080 requires an externally-generated two-phase clock (PH1 and PH2) driven to +12 V; all other control signals are at TTL voltage levels, between 0 and +5 volts.

Complete functional and electrical characteristics of the 8080 are provided in the manufacturer's manuals and data sheets. Essential details are reprinted by Martin Research in data sheet DS2204. The reader should refer to these sources for an explanation of internal processor operations, and for detailed information on the 8080 instruction set. This data sheet focuses on the circuitry of the Model 471, a CPU board in the modular micro series from Martin Research.

NOTICE CONCERNING 8080 A VERSION - The 8080A is an improved version of the 8080, but is compatible with early 8080 devices. Changes implemented with the introduction of the 8080A include the following: a relaxation of the minimum clock input high voltage from +11.0 to +9.0 V; improved current sinking capabilities on output terminals, from 1.7 mA on the data bus (0.75 mA at other outputs) to 1.9 mA on all output terminals, measured at 0.45 V; and internal synchronization of the HOLD signal, avoiding the need for synchronizing circuitry outside the 8080.
SEC. 1.3 BUS DRIVERS (cont'd.)

In the modular micro series, the 16 address lines are designated DH7 through DH0 (corresponding to A15 through A8 on the 8080 package), and DL7 through DLO (A7 through A0).

The 8726s can sink up to 40 mA of current (about 25 standard TTL loads), and the 8797s, up to 48 mA (about 30 standard TTL loads)—more than enough for most microcomputer systems. In addition, both bus driver types are assured of delivering a logic high input voltage of at least +3.5 V to the 8080 (at rated 8080 leakage currents), meeting the 8080 VIH specification of +3.3 V.

1.4 Main Timing The 8080 requires a two-phase clock referenced to +12 V. The standard 8080 requires a clock cycle time, τc, ranging between 480 ns and 2.0 μs; the 471 runs with a 500 ns clock cycle time, for an instruction cycle time of 2.0 μs. An instruction cycle is defined for speed spec purposes as the time required for the 8080 to execute an instruction consisting of a single machine cycle consisting of four states, each state lasting one clock period. Actual instruction cycles vary in duration between four and 18 clock periods, depending on the instruction involved. The number of clock periods is further increased if the CPU is made to wait for slow memory.

The 471 CPU board uses a type 8224 clock generator/driver for supplying the 8080 with the required PH1 and PH2 signals. An 18.0 MHz crystal is connected to the 8224, whose internal oscillator produces an 18 MHz square wave (available at pin 12 for test purposes). An internal divide-by-nine counter produces two alternating positive pulses with a basic frequency of 2.0 MHz and an asymmetrical duty cycle (see Fig. 1.4.1). After passing through high-level drivers, these signals are connected directly to the 8080, as PH1 and PH2. A TTL version of PH2—PH2T—is also generated, and is available for system control purposes.

At press time, available high-speed versions of the 8080A (with maximum clock periods shown in nanoseconds) included Intel C8080A-2, Advanced Micro Devices Am98080A-2 (380); C8080A-1, Am98080A-1 (320); Am98080A-4 (250). The Intel 8224 can be used to generate 8080 clock signals up to a maximum frequency of 27 MHz (corresponding to a clock period down to 370 ns), which supports the 8080A-2. The Am8224 is specified to function with CPUs up to the 250 ns clock cycle speed. (These figures were those available at press time; new product releases were scheduled, and the reader should check for current specifications.)
SEC. 1A MAIN TIMING (cont'd.)

In order to permit the use of an overtone crystal which may be more economical at frequencies above 20 MHz—a tank circuit may be added, as shown by the optional parts connected to the 6224 in the 471 schematic. L1 and C12 are chosen for resonance at the overtone frequency, and C1 is an RF bypass capacitor.

The 8224 also supports three system control functions. First, it generates STB, a status strobe occurring at the beginning of each 8080 machine cycle. STB is used to strobe system control logic (Sec. 1.5), and is made up of SYN (an 8080 signal indicating the beginning of a machine cycle) and 61A, which occurs “1 unit” before 61 proper (referring to Fig. 1.4.1). Using 61A and incorporating this logic in the Schottky bipolar circuitry of the 8224 optimizes system speed. The 8224 circuitry also activates STB during reset cycles.

The 8224 synchronizes the system reset signal at its RES input with 620 of the clock (the rising edge of 62), to provide the synchronization required by the 8080. (For more on the reset function, see Sec. 1.7 below.)

Finally, the 8224 synchronizes the microcomputer’s BSY line with 620 of the clock, again to meet 8080 internal timing requirements. (For more on use of the BSY line for memory wait cycles, see Sec. 1.10.)

1.5 System Control Logic The 8080 is a synchronous device which follows certain intricate sequences in executing program instructions stored in memory. An instruction cycle is required for the execution of each instruction, with this time period subdivided into specialized cycles, as required in order to time-share the microprocessor's eight-bit data bus and sixteen-bit address bus. An instruction cycle consists of between one and four machine cycles, with the first always devoted to fetching the computer's program instruction from memory. The address bus selects the instruction location, and the data bus receives the memory word. In general, one additional machine cycle is necessary each time the instruction writes into or reads from memory or an I/O port. (For detailed information on processor cycles, refer to the 8080 data sheet, DS2204.)

Near the beginning of each machine cycle, the 8080 produces the signal SYNC, which is gated by the 8224 clock generator to produce the system strobe STB. At STB time, the 8080 data bus contains an eight-bit status word which defines the purpose of the machine cycle. The assignment of data bus bits is as follows:

- D7 M0EMR Memory to be read from
- D6 INP Input data to be received
- D5 M1 First byte of instruction to be fetched
- D4 OUT Output data to be transmitted

The system control logic on the 471 CPU board includes a 74LS174 hex latch and a 74LS74 D flip-flop. Both circuits are strobed by STB (STB inverted), latching up seven of the eight bits of the 8080 status word. (The M1 bit is not used.) Note that the inputs to the latches derive from the inverted data bus, DB7 through DB0; the data terminals of the latches have been bubbled to indicate the active-low logic involved. Three of the resultant outputs are inverted and gated with DBIN to produce MEMRA, MERA, and INTX. DBIN comes from the 8080 (through buffering inverters), and indicates that the 8080 data bus terminals are ready to receive from external circuitry. INTX is used to read in an interrupt jam instruction, during interrupt cycles (Sec. 1.8). MEMRA and MERA are buffered by the Address/Control Bus Driver stage (Sec. 1.3), and are relabeled MEMR and TFR. Distributed via the modular micro bus to circuitry outside the 471 CPU board, these signals are used for enabling memory to be read from, and input ports to be read from (respectively).

The 8080 creates an enable signal called WR, indicating that it has valid data on the data bus to be written either into memory or into an output port. To distinguish between write operations, two NAND gates combine WR (WR inverted) with the status word bit OUT, which is latched up in the 'LS74 and is available both true and complemented at the flip-flop outputs. STEW (sometimes called MEMWA) is active during memory write cycles, and IOWA during output write cycles. Both are buffered by the Address/Control Bus Drivers and appear on the modular micro bus as STEW and IOW. Note that these are enable signals, valid only at the moment when the 8080 is transmitting data on its data bus.

The signal MEMA is produced by ORing together MEMRA and STEWA, and after buffering, is available as MEM on the modular micro bus. It is a combined strobe/enable signal, active during cycles referring to memory (read or write), at the moment when data is transferred between CPU and memory via the data bus.

The signal PCWA indicates that a memory write cycle is in progress. Unlike the signals described above, it is not a strobe/enable signal—that is, it is not valid just at the moment when data is transferred on the data bus. Rather, PCWA starts near the beginning of a memory write cycle and does not end until the next machine cycle begins. A NAND gate is used to gate WO (a status bit indicating a memory or output write cycle) with OUT, in such a way that PCWA is active only when OUT is inactive. After buffering, the signal is called PCW, and is available on the modular micro bus. Typically, PCW is used to control the HI/II terminals of RAM memory. PCW anticipates both STEW and MEMA, so that the read/write mode on the RAM can be set up before the strobe/enable signal activates data transfers with the CPU.
SEC. 1.5 SYSTEM CONTROL LOGIC (cont'd.)

Another signal useful in interfacing with memory is the status strobe, STB, developed by the 8224 clock generator and discussed above. After buffering by the Address/Control Bus Driver stage, it is piped along the modular micro bus under the label AOV, for address valid. As the name implies, when AOV occurs the 8808A address lines have been loaded and are stable, indicating that the address bus may be used reliably in memory and I/O addressing. AOV occurs early enough in the machine cycle that, should the ready line be pulled down at this time, the CPU will enter a wait state. (See Sec. 1.10.)

1.6 I/O Instruction Modes

1.6.1 Standard 8080 I/O Mode

The IOR and IOW signals discussed above are used for strobing and enabling input and output ports, using the standard 8080 INP and OUT instructions. In either of these instructions the second byte is an eight-bit I/O port address, which appears on both the DH and the DL address lines of the microcomputer. The I/O circuitry must first be addressed by the DH or DL address lines; then the IOR signal enables an input port (or IOW strobes an output port) and the I/O data flows between the port and the 8080 accumulator (A register).

1.6.2 Memory-Mapped I/O

A second mode of I/O addressing is commonly used—simply defining selected addresses in memory as I/O ports. Output ports are addressed by memory write instructions—strobed with the STEW (MEMW) strobe. Input ports are addressed by memory read instructions—enabled with MEMR signals. This memory-mapped I/O mode is that used in the 8800/8502 microprocessor series, and is attractive because of its universality. That is, the approach is basically compatible with any microcomputer, and a peripheral device with this kind of I/O port addressing circuitry may likely be interfaced with few modifications to different microprocessors. This is especially true in the modular micro series, where the bus is compatible with various standard eight-bit processors. Another advantage is that the 8808A includes instructions for transferring data between memory and a number of CPU index registers, rather than just the A register (accumulator) as with I/O instructions proper. Since the computer's memory read/write signals are used in controlling memory-mapped I/O, no additional circuitry is required. Naturally the user must maintain a memory map, to make sure that memory and I/O port addresses do not overlap.

By predecoding some of the I/O address lines, producing a specialized I/O strobe/enable signal, the amount of I/O strobe decoding circuitry necessary on peripheral boards in the microcomputer can be reduced. (See Sec. 1.6.4.)

1.6.3 8080-Compatible I/O

A third mode of I/O addressing is supported by the 471 CPU board, which derives the DIN2 strobe/enable signal (as shown in the System Control Logic section of the schematic). This I/O addressing mode is used by the 8808 microprocessor, the 8808A's predecessor, widely used in industrial controllers. An 8088 microcomputer—the MIX82—is available from Martin Research, and some boards in the modular micro series use the 8088 mode of I/O addressing. The fact that the 471 includes these features...
SEC. 1.6 /I/O INSTRUCTION MODES (cont'd.)

makes it very attractive in updating 8008 systems, especially when it is not desired to change peripheral I/O boards.

In the 8008 I/O mode, the I/O port address appears on five lines of the high-order memory address bus, namely DH5 through DH1. Output data appears on the low-order memory address bus, DL7-DL0. Input data is transmitted to the CPU via the data bus. When executed on an 8008, there are special single-byte input and output instructions; but with the Model 471 8080 CPU board, these 32 I/O ports are addressed with certain memory referencing instructions.

Referring to the circuitry which creates DINA: this signal occurs when the 8008 data bus is ready to receive data (DBIN) or transmit data (WR); when the instruction is not a standard 8008 I/O instruction (OCT and IR must be high); and only when DH7 and DH6 are high, and DH5 is high. These conditions mean that any 8008 memory read/write instruction which references the odd-numbered pages in the upper one-fourth of memory can be used as an 8008-compatible I/O instruction. (See Figure 1.6.1.)

DINZ8 is a variant of DINA, differing only in that it is created when DH9 is low (rather than high). By using DIN2, another set of 32 I/O ports can be addressed. Again, see Fig. 1.6.1.

The Address/Control Bus Driver stage converts DINA to DIN, and DIN2 to DIN2, for use on the modular micro bus. Note that the user must choose between the extra 32 8008-compatible I/O ports (using the DIN2 signal), and the standard 8008 input ports (using the IOR signal), by wiring up either jumper 9 (J9) or J10. As the solid line indicates, factory-assembled boards are wired for IOR.

To decode the strobe and enable signals for an 8008-compatible I/O port, circuitry on a peripheral board combines the DIN or DIN2 signal on the modular micro bus with the five address bits DH5 through DH1. Using a single 74LS138 (2205/2206), eight such strobe/enable signals can be produced by connecting the five DH bits in various combinations to the three select inputs, and to one active-low and one active-high enable input. (This technique is elaborated in the data sheet for a Universal I/O Kit, KT8001, and in Chapter 7 of Microcomputer Design.)

Several kinds of 8008 instructions may be performed using the 471 CPU board in order to address 8008-compatible I/O ports. These are indicated in Figure 1.6.2.
SEC. 1.6 I/O INSTRUCTION MODES (cont'd.)

Finally, note that the use of these 8080-compatible I/O instructions is entirely optional. Any of the page addresses shown in Fig. 1.6.1 may instead be used for memory, activated by the normal 8080 memory signals (STEW, MEMR, etc.), and DTIN may be ignored. As with other forms of memory-mapped I/O--of which this is a specialized version--the only constraint is that a given memory page address not be used for more than one purpose (memory or I/O port).

1.6.4 Memory-Mapped I/O with Precoded Strobe

One advantage of the 8080-compatible I/O mode discussed above is that there are only five I/O address lines, as opposed to eight with the 8080 mode and eight with the memory-mapped mode (if a whole page in memory is used for each I/O port). If a single location in memory is assigned as an I/O port, this means that circuitry needed to produce I/O strobes, for activating input bus drivers and output latches on a peripheral circuit board, must combine eight or more I/O address lines, plus a control signal (IOW, IOM, STEW). This circuitry will be cumbersome, especially if more than one I/O strobe/enable signal is needed on that board.

The user of the 471 CPU board can combine the advantages of several I/O address modes by using DTIN (or DTINR) for memory-mapped I/O. To output data, the program executes a memory write instruction and data is written into the output port; to input data, the program executes a memory read instruction, and data is read from the input port into the CPU. All of the various memory-referencing instructions of the 8080 can be used. So far, this is a description of memory-mapped I/O in general. The only differences are these: the I/O ports use DTIN (or DTINR) instead of MEMR or STEW; and the high-order memory address which references these I/O ports must be one which will cause DTIN (or DTINR) to be produced. (See Fig. 1.6.1.)

With this I/O method, a single 74LS138 on a peripheral board will produce eight I/O strobe/enable signals, as described in Sec. 1.6.3 above.

1.7 Reset Circuitry

When power is first applied to the 8080, its internal registers may take on any random values. To initialize the machine, the 9090 RESET terminal must be brought to a logic high level for at least three clock cycles. This resets the program counter to zero, and the 8080 will proceed to execute stored program instructions, starting at location zero. The reset signal also resets the 8080 interrupt flip-flop, disabling interrupts (as does an interrupt itself). An initial interrupt is unnecessary for starting the computer. However, if it is desired to make the CPU wait for an initial interrupt before proceeding with its mainline program, the two instructions EI (enable interrupt), HL7 (halt) may be inserted starting at location zero. 8080 flags and registers other than the program counter and interrupt flip-flop are not affected by the reset pulse, but must be initialized as required through program instructions.

The Model 471 CPU board includes a Power On Reset circuit which senses the rising of the +5 volt power supply as power is applied, generating the signal: RES. The LM303/339 is capable of operation at supply voltages as low as +2.0 V. Until the +5 volt line reaches +3.9 V, the zener diode does not conduct, and the - input to the comparator receives the full power supply voltage.
SEC. 1.7 RESET CIRCUITRY (cont'd.)

request, then the counters are reset. But if 128 PH2T cycles have been clocked, the high-order 74393 output bit (pin B) goes high. The signal which results (after inversion), REE, causes a reset.

REE is ORed with the signal RESET, which originates from an optional interrupt control board (Sec. 1.8). A 7417 open-collector buffer is connected to the RES line, which may also be pulled low by the power-on reset comparator and the optional reset switch.

The automatic reset circuit may be defeated by removing the 74393 counter from its socket, and connecting a wire jumper between pins 8 and 7 (ground) on the 74393 socket.

1.8 Interrupt Logic

The 8080 fully supports external interrupt requests. A typical interrupt source is a peripheral input device (such as a keyboard) which inputs data relatively infrequently and unpredictably, but which must receive immediate priority when ready. Basically, the 8080 receives the interrupt request and issues an interrupt acknowledgment. External circuitry detects the acknowledge, and in a special instruction, causes the CPU to jump to the appropriate interrupt subroutine. When this ends, the CPU returns to the mainline program at exactly the point where it was interrupted.

In many systems there are multiple interrupt sources. It is necessary therefore to provide means for decoding the source of an interrupt request, in order to make the CPU execute the appropriate interrupt instructions. The 471 includes on-board provisions for three interrupt levels. In addition, the essential interrupt control signals are routed to a 20-pin connector at the side of the board, shown as P2 on the 471 schematic. An optional interrupt control board may be connected here for expansion to an eight-level prioritized interrupt system.

The interrupt cycle begins when some device within the modular micro system pulls the INTR line (pin 34) low. (Refer to page 7, D1.1.) INTR is connected through jumper J7 to a 74LS20 inverter stage. (J8 would replace J7 only if the optional interrupt board were installed.) The 74LS74 interrupt request flip-flop is clocked, causing the INT signal to go high. The INT signal is connected directly to the 8080, requesting an interrupt.

The 8080 contains an internal interrupt flip-flop, INTE (as mentioned above). Executing a D1 (disable interrupt) instruction, resetting the computer, or performing an interrupt, will reset the INTE flip-flop and prevent the CPU from acknowledging further interrupts. To enable interrupts, an EI instruction must be executed. This sets the INTE flip-flop and permits the 8080 to recognize interrupts. The 8080 CPU does not itself save interrupt requests which were placed when interrupts were disabled. However, the 471 CPU board does, in the interrupt request flip-flop (a 74LS74; page 7, D2 in the schematic).

When an interrupt is recognized, the 8080 enters a special interrupt cycle instruction. In the machine status word for the instruction fetch cycle appearing on the data bus at STB time, DS0 will be 0. The system control logic detects this interrupt acknowledgment, and at DBIN time, produces the signal INTX (Sec. 1.5 above). This enable signal is used to jam the special interrupt instruction onto the data bus. It also resets the 74LS74 interrupt request flip-flop.

INTX is connected to the inputs of two 7417 open-collector buffers, and to the enable terminals of an 8T97 tri-state buffer. The 7417s pull DB7 and DB0 to 0, and the 8T97 pulls DB3 through DB0 to 0. (DB3 is pulled low because pin 12 of the 8T97 is normally low-unless the optional interrupt board pulls down the NMI line.) DB7 and DB4 are held high by pullup resistors R4 and R3 (which would be replaced by jumpers J2 and J1 only if the optional interrupt board were installed).

These conditions mean that the inverted data bus bits DB7 through DB0 read 00110000. Reinverting, the data word seen by the CPU is 11001111, or 317 octal. This is the code for an 8080 RST 1 instruction, a single-byte call instruction which causes the CPU to jump to memory location 000010, pushing the program counter stack. Starting at this location, the user installs a program which contains the instructions necessary to carry out the purposes of the interrupt request. The interrupt routine ends with a RET (return) instruction, popping the PC stack and returning control to the mainline program.

The reader should note that, to avoid having the interrupt routine disturb the mainline program, any registers and flags used both by the interrupt and mainline programs should be saved as the interrupt routine begins (with PUSH instructions) and restored as ends (with POP instructions).

The device requesting an interrupt may pull down the DB5 and DB4 data bus lines at DBIN time, altering the interrupt jam instruction as shown in Figure 1.8.1.

<table>
<thead>
<tr>
<th>DB5</th>
<th>DB4</th>
<th>DATA</th>
<th>DATA</th>
<th>OCTAL</th>
<th>INSTRUCTION</th>
<th>RESTART ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>00110000</td>
<td>11001111</td>
<td>317</td>
<td>RST 1</td>
<td>000010</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>00100000</td>
<td>11011111</td>
<td>337</td>
<td>RST 3</td>
<td>000030</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>00010000</td>
<td>11101111</td>
<td>357</td>
<td>RST 5</td>
<td>000050</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>00000000</td>
<td>11111111</td>
<td>377</td>
<td>RST 7</td>
<td>000070</td>
</tr>
</tbody>
</table>

Figure 1.8.1--Interrupt Jam Instructions

This interrupt system effectively provides three interrupt levels, though there are four possible jam instructions (Fig. 1.8.1). RST 1 is a general-purpose interrupt, where the interrupting device does not affect DB5 or DB4. Typically this is a low-priority interrupt. A second interrupt port pulls down DB5, and a third, DB4. Other data bus lines could be pulled low but will have no
SEC. 1.8 INTERRUPT LOGIC (cont’d.)

effect unless the system is provided with the optional eight-level interrupt board. Since a RST 7 interrupt jam instruction would result when both the DB5 and DB4 interrupt ports requested interrupts at the same time, it cannot be used unambiguously for a third interrupt port. Note also that if a RST 3, 5, or 7 occurs, it masks the possibility that a RST 1 interrupt might also have been requested.

In order to synchronize circuitry on peripheral boards with the CPU interrupt logic, the INTX interrupt jam enable signal is connected through jumper J6 to INTS on the modular micro bus (page 7, location D3). The device requesting an interrupt uses this signal to enable circuitry which pulls down the DB5 and DB4 data bus lines. A suitable circuit using only two ICs appears in Figure 1.8.2. (A kit of the parts shown, plus wirewrap sockets, is available from Martin Research as KT9008.)

Figure 1.8.2--Interrupt Request Port

The device requesting an interrupt causes IRQ (interrupt request) to go high, setting the first 74LS74 flip-flop (pin 3). The Q output of this flip-flop (pin 6) resets the second flip-flop (pin 13), causing its Q output (pin 9) to go low. This signal is fed back to the first FF reset terminal (pin 1), resetting the first FF. However, the second FF remains reset. With pin 8 high, the 7438 open-collector buffer connected to the INTR line is activated, pulling INTR low and requesting an interrupt from the 471 CPU board.
SEC. 1.10 WAIT CIRCUITRY (cont’d.)

probably a safe figure, but the conservative designer may choose to specify memory with a 450 ns access time. 450 ns is Martin Research policy with regard to the Model 405 static 4K RAM board.

If slower memory is used, the 8080 must be made to wait for memory access. This is done by pulling the 8080 READY terminal low. When slow memory is addressed, special memory wait circuitry is triggered; pulling down the BSY (busy) line on the modular micro board. To synchronize wait requests with the 8080, BSY is connected to the 8224 clock generator, which drives the 8080 READY terminal (Sec. 1.4).

Any peripheral device with slow memory can pull BSY down and cause an extended wait period. The same provision may be used with slow I/O devices, during input and output instructions. Optional wait circuitry is a feature of several Martin Research circuit boards.

The 471 CPU board includes a wait circuit consisting of a 9601 or 74122 one-shot and a 7417 open-collector buffer. The one-shot is triggered by STB—a status strobe which comes sufficiently in advance of any read/write data transfer to guarantee that the CPU will enter the wait state. The G output (pin 6) goes low, activating the 7417 and pulling BSY low.

The 9601 and 74122 are not identical, but both work in this circuit. They have very similar timing characteristics. The values of R1 and C2 determine the length of the wait cycle. In factory-assembled boards, R1 is 10K, and C2 is 270 pf, producing the required one-shot period of about 915 ns. The goal is to cause two memory wait cycles, which adds two clock cycles (2 x 500 ns = 1.0 us) to the normal T3 state, adding to a total of 1.5 us. This compensates for memory with an access time as slow as 1.55 us.

For this CPU circuitry (8080 with 8224), to cause n wait states, the BSY line should be pulled low (starting at the leading edge of STB) for the period (n - 1/6) tcy, where tcy is the clock cycle time. This formula allows enough latitude to permit using 10% tolerance R and C components.

Greater system speed will result if memory wait cycles occur only when necessary, when slow memory devices are being addressed. It is more efficient to install the wait circuitry on any memory (or I/O) board that requires it, and to use the minimum effective delay in every case. Then the memory wait circuit on the 471 CPU board is deactivated, simply by removing the one-shot from its socket.

1.11 Indicator Lights The 471 CPU board has two light-emitting diodes which indicate machine status. A red LED glows when memory wait cycles are being performed; the LED is driven by the 8080 WAIT output terminal, through an inverter and open-collector buffer.
This data sheet consists of reprints of literature published by the 8008 manufacturer, Intel Corporation. The 8008/8008-1 microprocessor is the heart of the central processing board of the MIKE 2, a microcomputer available from Martin Research. For replacement purposes, the 8008 is stocked as Part No. 2202, and the 8008-1 as P/N 2203.

**SINGLE CHIP EIGHT-BIT PARALLEL CENTRAL PROCESSOR UNIT**

- Instruction Cycle Time—
  12.5 μs with 8008-1 or 20 μs with 8008
- Directly addresses 16K x 8 bits of memory (RAM, ROM, or S.R.)
- Interrupt Capability

The 8008 is a single chip MOS 8-bit parallel central processor unit

This CPU contains six 8-bit data registers, an 8-bit accumulator, two 8-bit temporary registers, four flag bits (carry, zero, sign, parity), and an 8-bit parallel binary arithmetic unit which implements addition, subtraction, and logical operations. A memory stack containing a 14-bit program counter and seven 14-bit words is used internally to store program and subroutine addresses. The 14-bit address permits the direct addressing of 16K words of memory (any mix of RAM, ROM or S.R.).

The instruction set of the 8008 consists of 48 instructions including data manipulation, binary arithmetic, and jump to subroutine.

The normal program flow of the 8008 may be interrupted through the use of the INTERRUPT control line. This allows the servicing of slow I/O peripheral devices while also executing the main program.

The READY command line synchronizes the 8008 to the memory cycle allowing any type or speed of semiconductor memory to be used.
2008 MICROPROCESSOR/8008-1 MICROPROCESSOR

8008 FUNCTIONAL PIN DESCRIPTION

---

Dp-Dy
BI-DIRECTIONAL DATA BUS. All address and data communication between the processor and the program memory, data memory, and I/O devices occur on these 8 lines. Cycle control information is also available.

INT
INTERRUPT input. A logic "1" level at this input causes the processor to enter the INTERRUPT mode.

READY
READY input. This command line is used to synchronize the 8008 to the memory cycle allowing any speed memory to be used.

SYNC
SYNC output. Synchronization signal generated by the processor. It indicates the beginning of a machine cycle.

T1, T2
Two phase clock inputs.

S0, S1, S2
MACHINE STATE OUTPUTS. The processor controls the use of the data bus and determines whether it will be sending or receiving data. State signals S0, S1, and S2, along with SYNC inform the peripheral circuitry of the state of the processor.

VCC +5V ±5%
VDD -9V ±5%

---

COPYRIGHT NOTICE

The information in this data sheet has been reprinted with the permission of the copyright holder, Intel Corporation. Copyright © 1973, 1974, 1975, Intel Corp.

---

8008 MICROPROCESSOR/8008-1 MICROPROCESSOR

PROCESSOR TIMING

The 8008 is a complete central processing unit intended for use in any arithmetic, control, or decision-making system. The internal organization is centered around an 8-bit internal data bus. All communication within the processor and with external components occurs on this bus in the form of 8-bit bytes of address, instruction or data. [Refer to the accompanying block diagram for the relationship of all of the internal elements of the processor to each other and to the data bus.]

A. State Control Coding

The processor controls the use of the data bus and determines whether it will be sending or receiving data. State signals S0, S1, and S2, along with SYNC inform the peripheral circuitry of the state of the processor. A table of the binary state codes and the designated state names is shown below.

<table>
<thead>
<tr>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>T1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>T1H</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>T2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>WAIT</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>T3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>STOPED</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>T4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>T5</td>
</tr>
</tbody>
</table>

B. Timing

Typically, a machine cycle consists of five states, two states in which an address is sent to memory (T1 and T2), one for the instruction or data fetch (T3), and two states for the execution of the instruction (T4 and T5). If the processor is used with slow memories, the READY line synchronizes the processor with the memories. When the memories are not available for either sending or receiving data, the processor goes into the WAIT state. The accompanying diagram illustrates the processor activity during a single cycle.
The receipt of an INTERRUPT is acknowledged by the T1. When the processor has been interrupted, this state replaces T1. A READY is acknowledged by T3. The STOPPED state acknowledges the receipt of a HALT instruction.

Many of the instructions for the 808 are multi-cycle and do not require the two execution states, T4 and T5. As a result, these states are omitted when they are not needed and the 808 operates asynchronously with respect to the cycle length. The external state transition is shown below. Note that the WAIT state and the STOPPED may be indefinite in length (each of these states will be 2h clock periods). The use of READY and INTERRUPT with regard to these states will be explained later.

Figure 2. CPU State Transition Diagram

C. Cycle Control Coding

As previously noted, instructions for the 808 require one, two, or three machine cycles for complete execution. The first cycle is always an instruction fetch cycle (PCI). The second and third cycles are for data reading (PCR), data writing (PCW), or I/O operations (PCO).

The cycle types are coded with two bits, D<sub>2</sub> and D<sub>1</sub>, and are only present on the data bus during T2.

<table>
<thead>
<tr>
<th>D&lt;sub&gt;2&lt;/sub&gt;</th>
<th>D&lt;sub&gt;1&lt;/sub&gt;</th>
<th>CYCLE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>PCI</td>
<td>Designates the address is for a memory read (first byte of instruction).</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>PCR</td>
<td>Designates the address is for a memory read data (additional bytes of instruction or data).</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>PCC</td>
<td>Designates the data as a command I/O operation.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>PCW</td>
<td>Designates the address is for a memory write data.</td>
</tr>
</tbody>
</table>

BASIC FUNCTIONAL BLOCKS

The four basic functional blocks of this Intel processor are the instruction register, memory, arithmetic-logic unit, and I/O buffers. They communicate with each other over the internal 8-bit data bus.

A. Instruction Register and Control

The instruction register is the heart of all processor control. Instructions are fetched from memory, stored in the instruction register, and decoded for control of both the memory and the ALU. Since instruction executions do not require the same number of states, the instruction decoder also controls the state transitions.

B. Memory

Two separate dynamic memories are used in the 808, the pushdown address stack and a scratch pad. These internal memories are automatically refreshed by each WAIT, T3, and STOPPED state. In the worst case the memories are completely refreshed every eighty clock periods.

1. Address Stack

The address stack contains eight 14-bit registers providing storage for eight lower and six higher order address bits in each register. One register is used as the program counter (storing the effective address) and the other seven permit address storage for nesting of subroutines up to seven levels.

The stack automatically stores the content of the program counter upon the execution of a CALL instruction and automatically restores the program counter upon the execution of a RETURN, CALLS may be nested and the registers of the stack are used as last in first out pushdown stack. A three-bit address pointer is used to designate the present location of the program counter. When the capacity of the stack is exceeded the address pointer recycles and the content of the lowest level register is destroyed. The program counter is incremented immediately after the lower order address bits are sent out. The higher order address bits are sent out at T2 and then incremented if a carry resulted from T1. The 14-bit program counter provides direct addressing of 16K bytes of memory. Through the use of an I/O instruction for bank switching, memory may be indefinitely expanded.

2. Scratch Pad Memory or Index Registers

The scratch pad contains the accumulator (A register) and six additional 8-bit registers (B, C, D, E, H, L). All arithmetic operations use the accumulator as one of the operands. All registers are independent and may be used for temporary storage. In the case of instructions which require operations with a register in external memory, scratch pad registers H & L provide indirect addressing capability, register L contains the eight lower order bits of address and register H contains the six higher order bits of address (in this case bit 6 and bit 7 are "don't cares").

C. Arithmetic/Logic Unit (ALU)

All arithmetic and logical operations (ADD, ADDC with carry, SUBTRACT, SUBTRACT with borrow, AND, EXCLUSIVE OR, OR, COMPARE, INCREMENT, DECREMENT) are carried out in the 8-bit parallel arithmetic unit which includes carry-look-ahead logic. Two temporary registers, register "a" and register "b", are used to store the accumulator and operand for ALU operations. In addition, they are used for temporary address and data storage during intra-processor transfers. Four control bits, carry-flip-flop (C), zero-flip-flop (Z), sign-flip-flop (S), and parity-flip-flop (P), are set as the result of each arithmetic and logical operation. These bits provide conditional branching capability through CALL, JUMP, or RETURN on condition instructions. In addition, the carry bit provides the ability to do multiple precision binary arithmetic.

D. I/O Buffer

This buffer is the only link between the processor and the rest of the system. Each of the eight buffers is bi-directional and is under control of the instruction register and state timing. Each of the buffers is lower power TTL compatible on the output and TTL compatible on the input.
**BASIC INSTRUCTION SET**

The following section presents the basic instruction set of the 8008.

### A. Data and Instruction Formats

Data in the 8008 is stored in the form of 8-bit binary integers. All data transfers to the system data bus will be in the same format.

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

#### DATA WORD

The program instructions may be one, two, or three bytes in length. Multiple byte instructions must be stored in successive words in program memory. The instruction formats depend on the particular operation executed.

#### TYPICAL INSTRUCTIONS

- **ADD**: Add registers, memory, immediate, or input/output
- **MUL**: Multiply registers, memory, immediate, or input/output
- **DIV**: Divide registers, memory, immediate, or input/output
- **XOR**: Exclusive OR of registers, memory, immediate, or input/output
- **OR**: OR of registers, memory, immediate, or input/output
- **AND**: AND of registers, memory, immediate, or input/output
- **CLR**: Clear registers, memory, immediate, or input/output
- **JMP**: Jump to an address
- **CALL**: Call subroutine
- **RST**: Return from subroutine

#### Example Instruction Set

**ADD**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**MUL**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**DIV**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**XOR**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**OR**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**AND**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**CLR**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**JMP**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**CALL**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**RST**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

#### B. Summary of Processor Instructions

**Index Register Instructions**

The load instructions do not affect the flag flip-flops. The increment and decrement instructions affect all flip-flops except the carry.

<table>
<thead>
<tr>
<th>INSTRUCTION CODE</th>
<th>MINIMUM STATES REQUIRED</th>
<th>DESCRIPTION OF OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX, LDX, LDI, LDI, LDI</td>
<td>1</td>
<td>Load index register with the content of index register</td>
</tr>
<tr>
<td>LDI, LDI</td>
<td>1</td>
<td>Load immediate memory with the content of index register</td>
</tr>
<tr>
<td>LDI, LDI</td>
<td>1</td>
<td>Load immediate memory with the content of index register</td>
</tr>
<tr>
<td>LDI, LDI</td>
<td>1</td>
<td>Load immediate memory with the content of index register</td>
</tr>
<tr>
<td>LDI, LDI</td>
<td>1</td>
<td>Load immediate memory with the content of index register</td>
</tr>
<tr>
<td>LDI, LDI</td>
<td>1</td>
<td>Increment the content of index register (r) or (d)</td>
</tr>
<tr>
<td>LDI, LDI</td>
<td>1</td>
<td>Decrement the content of index register (r) or (d)</td>
</tr>
</tbody>
</table>

**Accumulator Group Instructions**

The result of the ALU instructions affect all of the flag flip-flops. The rotate instructions affect only the carry flip-flop.

<table>
<thead>
<tr>
<th>INSTRUCTION CODE</th>
<th>MINIMUM STATES REQUIRED</th>
<th>DESCRIPTION OF OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD, ADD, ADD, ADD</td>
<td>1</td>
<td>Add the content of index register r, memory register M, or data D to the accumulator. An overflow carry sets the carry flip-flop.</td>
</tr>
<tr>
<td>ADD, ADD, ADD, ADD</td>
<td>1</td>
<td>Subtract the content of index register r, memory register M, or data D from the accumulator. An underflow borrow sets the carry flip-flop.</td>
</tr>
<tr>
<td>ADC, ADC, ADC, ADC</td>
<td>1</td>
<td>Add the content of index register r, memory register M, or data D to the accumulator. An overflow carry sets the carry flip-flop.</td>
</tr>
<tr>
<td>ADC, ADC, ADC, ADC</td>
<td>1</td>
<td>Subtract the content of index register r, memory register M, or data D from the accumulator. An underflow borrow sets the carry flip-flop.</td>
</tr>
<tr>
<td>SUI, SUI, SUI</td>
<td>1</td>
<td>Subtract the content of index register r, memory register M, or data D from the accumulator. An underflow borrow sets the carry flip-flop.</td>
</tr>
<tr>
<td>SUM, SUM, SUM, SUM</td>
<td>1</td>
<td>Subtract the content of index register r, memory register M, or data D from the accumulator. An underflow borrow sets the carry flip-flop.</td>
</tr>
</tbody>
</table>

### Program Counter and Stack Control Instructions

**BR**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**JMP**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**CALL**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**RET**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**INP**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**OUT**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

### Machine Language

**HALT**

\[ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 \]

**NOTES**

1. **SBS = Source Index Register**: These registers, r, are designated Accumulator--0000.
2. **DDO = Destination Index Register**: These registers, M, are designated Accumulator--0000.
3. **M, Memory register addresses**: The main memory is addressed by the contents of registers M, B, L.
4. **ADD**: The accumulator mode of instruction sets are designated to 11001111.
5. **X = Deny Card**: Flag flip-flops are defined by each card (0-overflow or undefined, 0 result is zero, 10 (negative) or result is non-zero).
Internal Processor Operation

Internally the processor operates through five different states:

- **T1 (NORMAL)**: Send out lower eight bits of address and increment program counter.
- **T2 (INTERRUPT)**: Send out lower eight bits of address and suppress incrementing of program counter and acknowledge interrupt.
- **T3 (WAIT)**: Wait for READY signal to become true. Refresh internal dynamic memories while waiting.
- **T3 (NORMAL)**: Fetch and decode instruction; fetch data from memory; output data to memory. Refresh internal memories.
- **T4 and T5 (STOPPED)**: Remain stopped until INTERRUPT occurs. Refresh internal memories. Execute instruction and appropriately transfer data within processor. Context of data bus transfer is available at I/O bus for convenience in testing. Some cycles do not require these states. In those cases, the states are skipped and the processor goes directly to T1.

The 8008 is driven by two non-overlapping clocks. Two clock periods are required for each state of the processor. \( \phi_1 \) is generally used to precharge all data lines and memories and \( \phi_2 \) controls all data transfers within the processor. A SYNC signal (divide by two of \( \phi_2 \)) is sent out by the 8008. This signal distinguishes between the two clock periods of each state.

The figure below shows state transitions relative to the internal operation of the processor. As noted in the previous table, the processor skips unnecessary execution steps during any cycle. The state counter within the 8008 operates is a five bit feedback shift register with the feedback path controlled by the instruction being executed. When the processor is either waiting or stopped, it is internally cycling through the T3 state. This state is the only time in the cycle when the internal dynamic memories can be refreshed.

The following pages show the processor activity during each state of the execution of each instruction.
PROCESSOR CONTROL SIGNALS

A. Interrupt Signal (INT)

1) INTERRUPT REQUEST

If the interrupt line is enabled (Logic "1"), the CPU recognizes an interrupt request at the next instruction fetch (PCI) cycle by outputting $S_2 S_1 S_0 = 011$ at T11 time. The lower and higher order address bytes of the program counter are sent out, but the program counter is not advanced. A successive instruction fetch cycle can be used to insert an arbitrary instruction into the instruction register in the CPU. (If a multi-cycle or multi-byte instruction is inserted, an interrupt need only be inserted for the first cycle.)

When the processor is interrupted, the system INTERRUPT signal must be synchronized with the leading edge of the $\phi_3$ or $\phi_2$ clock. To assure proper operation of the system, the interrupt line to the CPU must not be allowed to change within 200ns of the falling edge of $\phi_1$.

2) START-UP OF THE 808

When power ($V_{DD}$) and clocks ($\phi_1$, $\phi_2$) are first turned on, a flip-flop internal to the 808 is set by sensing the state of $V_{DD}$. This internal signal forces a HALT (00000000) into the instruction register and the 808 is then in the STOPPED state. The following sixteen clock periods after entering the STOPPED state are required to clear logic "0" of memories (accumulator, scratch pad, program counter, and stack). During this time the interrupt line has been at logic "0". Any time after the memories are cleared, the 808 is ready for normal operation.

To reset the flip-flop and also escape from the stopped state, the interrupt line must go to a logic "1". It should be returned to logic "0" by decoding the state T1 at some time later than $\phi_1$. Note that whenever the 808 is in a T1 state, the program counter is not incremented. As a result, the same address is sent out on two successive cycles.

Three possible sequences for starting the 808 are shown on the following page. The RESTART instruction is effectively a one cycle call instruction, and it is convenient to use this instruction to call an initiation subroutine. Note that it is not necessary to start the 808 with a RESTART instruction.

The selection of initiation technique to use depends on the sophistication of the system using the 808. If the interrupt feature is used only for the start-up of the 808 use the ROM directly, no additional external logic associated with instructions from source other than the ROM program need be considered. If the interrupt feature is used to jam instructions into the 808, it would then be necessary to use it to jam the initial instruction.

The timing for the interrupt and the start-up timing is shown on an accompanying sheet. The jamming of an instruction and the suppression of the program counter update are handled the same for all interrupts.
EXAMPLE 1:
Shown below are two start-up alternatives where an instruction is not forced into the 8008 during the interrupt cycle. The normal program flow starts the 8008.

a. 8008 ADDRESS OUT INSTRUCTION IN ROM
   00000000 00000000
   00000000 00000000
   00000000 00000000
   00000000 00000000
   00000000 00000000
   00000000 00000000
   00000000 00000000
   INSTR1
   INSTR2

b. 8008 ADDRESS OUT INSTRUCTION IN ROM
   00000000 00000000
   00000000 00000000
   00000000 00000000
   00000000 00000000
   00000000 00000000
   00000000 00000000
   00000000 00000000
   RST (RST +00 XYZ 101)
   INSTR1
   INSTR2

A Jump To The Main Program

EXAMPLE 2:
A RESTART instruction is jammed in and first instruction in ROM initially ignored.

8008 ADDRESS OUT INSTRUCTION IN ROM
   00000000 00000000
   00000000 00000000
   00000000 00000000
   00000000 00000000
   00000000 00000000
   00000000 00000000
   00000000 00000000
   INSTR1
   INSTR2
   RETURN
   (INSTR1), executed now. Start-up Routine
   Main Program

Note that during the interrupt cycle the flow of the instruction to the 8008 either from ROM or another source must be controlled by hardware external to 8008.

START-UP OF THE 8008

B. Ready (RDY)
The 8008 is designed to operate with any type or speed of semiconductor memory. This flexibility is provided by the READY command line. A high-speed memory will always be ready with data (the READY line to Vcc) almost immediately after the second byte of the address has been sent out. As a result the 8008 will never be required to wait for the memory. On the other hand, with slow ROMs, RAMs or shift registers, the data will not be immediately available; the 8008 must wait until the READY command indicates that the valid memory data is available. As a result any type or any combination of memory types may be used. The READY command line synchronizes the 8008 to the memory cycle. When a program is being developed, the READY signal provides a means of stepping through the program, one cycle at a time.
**ABSOLUTE MAXIMUM RATINGS**

Ambient Temperature Under Bias

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>LIMITS</th>
<th>TYP.</th>
<th>MAX.</th>
<th>TEST CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_{14} )</td>
<td>AVERAGE SUPPLY CURRENT - OUTPUTS LOADED*</td>
<td>30</td>
<td>80</td>
<td>mA</td>
<td>( T_a = 25^\circ C )</td>
</tr>
<tr>
<td>( I_{15} )</td>
<td>INPUT LEAKAGE CURRENT</td>
<td>10</td>
<td>( \mu A )</td>
<td>( V_{CC} = 0 ) V</td>
<td></td>
</tr>
<tr>
<td>( I_{16} )</td>
<td>INPUT LOW VOLTAGE (INCLUDING CLOCKS)</td>
<td>( V_{IL} )</td>
<td>( 4.5 ) V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( V_{15} )</td>
<td>INPUT HIGH VOLTAGE (INCLUDING CLOCKS)</td>
<td>( V_{IH} = 1.5 )</td>
<td>( 6.3 ) V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( I_{17} )</td>
<td>OUTPUT LOW VOLTAGE</td>
<td>2.4</td>
<td>V</td>
<td>( I_{OL} = 0.44 \mu A \</td>
<td></td>
</tr>
<tr>
<td>( I_{18} )</td>
<td>OUTPUT HIGH VOLTAGE</td>
<td>( V_{OH} = 1.5 )</td>
<td>( 6.3 ) V</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**D.C. AND OPERATING CHARACTERISTICS**

\( \gamma = 0^\circ C \) to 70\(^\circ C \), \( V_{CC} = \pm 5V \pm 5\% \), \( V_{DD} = \pm 5V \pm 5\% \), unless otherwise specified. Logix "1" is defined as the more negative level \( (V_{IL}, V_{IH}) \), Logix "0" is defined as the more negative level \( (V_{OL}, V_{OH}) \).

**A.C. CHARACTERISTICS**

\( \gamma = 0^\circ C \) to 70\(^\circ C \), \( V_{CC} = \pm 5V \pm 5\% \), \( V_{DD} = \pm 5V \pm 5\% \), All measurements are referenced to 1.5V levels.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>8008</th>
<th>LIMITS</th>
<th>8008-1</th>
<th>LIMITS</th>
<th>UNIT</th>
<th>TEST CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{14} )</td>
<td>CLOCK PERIOD</td>
<td>2</td>
<td>3</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td>( t_{p14} = 50ns )</td>
</tr>
<tr>
<td>( t_{14f} )</td>
<td>CLOCK RISE AND FALL TIMES</td>
<td>50</td>
<td>50</td>
<td>ns</td>
<td>ns</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>( t_{19} )</td>
<td>PULSE WIDTH OF ( \phi_4 )</td>
<td>70</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>PULSE WIDTH OF ( \phi_5 )</td>
<td>70</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>CLOCK DELAY FROM FALLING EDGE OF ( \phi_5 ) TO FALLING EDGE OF ( \phi_2 )</td>
<td>0.1</td>
<td>0.1</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>CLOCK DELAY FROM FALLING EDGE OF ( \phi_2 ) TO ( \phi_1 )</td>
<td>0.2</td>
<td>0.2</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>CLOCK DELAY FROM ( \phi_2 ) TO ( \phi_2 )</td>
<td>0.2</td>
<td>0.2</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>CLOCK DELAY FROM ( \phi_2 ) TO ( \phi_2 )</td>
<td>0.2</td>
<td>0.2</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>CLOCK DELAY FROM ( \phi_2 ) TO ( \phi_2 )</td>
<td>0.2</td>
<td>0.2</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>CLOCK DELAY FROM ( \phi_2 ) TO ( \phi_2 )</td>
<td>0.2</td>
<td>0.2</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>OUTPUT DELAY</td>
<td>1.0</td>
<td>1.0</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>HOLE TIME FOR DATA BUS OUT</td>
<td>0.1</td>
<td>0.1</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>HOLE TIME FOR DATA IN</td>
<td>1.0</td>
<td>1.0</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>SYNC OUT DELAY</td>
<td>0.70</td>
<td>0.70</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>STATE OUT DELAY (ALL STATES EXCEPT ( T1 ) AND ( T11 ))</td>
<td>1.1</td>
<td>1.1</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>STATE OUT DELAY (STAGES ( T1 ) AND ( T11 ))</td>
<td>1.0</td>
<td>1.0</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>PULSE WIDTH OF READY DURING ( \phi_1 ) TO ENTER ( T3 ) STATE</td>
<td>0.35</td>
<td>0.35</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>READY DELAY TO ENTER WAIT STATE</td>
<td>20</td>
<td>20</td>
<td>( \mu s )</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**TYPICAL D.C. CHARACTERISTICS**

**TYPICAL A.C. CHARACTERISTICS**

**CAPACITANCE**

\( f = 1MHz, \gamma = 25^\circ C \); Unmeasured Pins Grounded
ARCHITECTURE OF THE 8080 CPU

The 8080 CPU consists of the following functional units:

- Register array and address logic
- Arithmetic and logic unit (ALU)
- Instruction register and control section
- Bidirectional, 3-state data bus buffer

Registers:

The register section consists of a static RAM array organized into six 16-bit registers:

- Program counter (PC)
- Stack pointer (SP)
- Six 8-bit general purpose registers arranged in pairs, referred to as B,C; D,E; and H,L
- A temporary register pair called W,Z

The program counter maintains the memory address of the current program instruction and is incremented automatically during every instruction fetch. The stack pointer maintains the address of the next available stack location in memory. The stack pointer can be initialized to use any portion of read-write memory as a stack. The stack pointer is incremented when data is "pushed" onto the stack and decremented when data is "popped" off the stack (i.e., the stack grows "downward").

The six general purpose registers can be used either as single registers (8-bit) or as register pairs (16-bit). The temporary register pair, W,Z, is not program addressable and is only used for the internal execution of instructions.

Eight-bit data bytes can be transferred between the internal bus and the register array via the register-select multiplexer. Eight-bit transfers can proceed between the register array and the address latch or the incrementer/decrementer circuit. The address latch receives data from any of the three register pairs and drives the 16-bit address output buffers (A0-A15), as well as the incrementer/decrementer circuit. The incrementer/decrementer circuit receives data from the address latch and sends it to the register array. The 16-bit data can be incremented or decremented or simply transferred between registers.
The processor cycle

An instruction cycle is defined as the time required to fetch and execute an instruction. During the fetch, a selected instruction (one, two or three bytes) is extracted from memory and deposited in the CPU's instruction register. During the execution phase, the instruction is decoded and translated into specific processing activities.

Every instruction cycle consists of one, two, three, four or five machine cycles. A machine cycle is required each time the CPU accesses memory or an I/O port. The fetch portion of an instruction cycle requires one machine cycle for each byte to be fetched. The duration of the execution portion of the instruction cycle depends on the kind of instruction that has been fetched. Some instructions do not require any machine cycles other than those necessary to fetch the instruction; other instructions, however, require additional machine cycles to write or read data to/from memory or I/O devices. The DAD instruction, consisting of an instruction that requires two additional machine cycles to complete an internal register-to-addr add.

Each machine cycle consists of three, four or five states. A state is the smallest unit of processing activity and is defined as the interval between two successive positive-going transitions of the 51 driven clock pulse. The BO80 is driven by a two-phase clock oscillator. All processing activities are referred to the period of this clock. The two non-overlapping clock phases, labeled $\phi_1$ and $\phi_2$, are furnished by external circuitry. It is the $\phi_1$ clock pulse which divides each machine cycle into states. Timing logic within the BO80 uses the clock inputs to produce a SYNC pulse, which identifies the beginning of every machine cycle. The SYNC pulse is triggered by the low-to-high transition of $\phi_1$.

Instruction Register and Control:

During an instruction fetch, the first byte of an instruction (containing the OP code) is transferred from the internal to the BO80's instruction register.

The contents of the instruction register are, in turn, available to the instruction decoder. The output of the decoder, combined with various timing signals, provides the control signals for the register array, ALU and data buffer blocks. In addition, the outputs from the instruction decoder and external control signals feed the timing and state control section which generates the state and cycle timing signals.

Data Bus Buffer:

This 8-bit bidirectional 3-state buffer is used to isolate the CPU's internal bus from the external data bus (D3 to D7). In the output mode, the internal bus content is loaded into the buffer latch and, in turn, drives the data bus output buffers. The output buffers are switched off during input or non-transfer operations.

During the input mode, data from the external data bus is actually placed in the CPU's internal bus. The internal bus is pre-charged at the beginning of each internal state, except for the transfer state T3 described later.

The processor cycle consists of several steps, each of which is associated with a specific machine cycle. These machine cycles are used to implement the ADD and DAD ADD instructions.

$\phi_1$ and $\phi_2$ Sync Timing

There are three exceptions to the defined duration of a state. They are the WAIT state, the hold (HILDA) state and the halt (HLTA) state, described later in this chapter. Because the WAIT, HILTA and HILDA states depend upon external events, they are not their own time-independent length. Even these exceptional states, however, must be synchronized with the pulses of the driving clock. Thus, the duration of all states is integral multiples of the clock period.

To summarize, each clock period marks a state: three to five states constitute a machine cycle; and one or five machine cycles comprise an instruction cycle. A full instruction cycle requires anywhere from four to eight states for its completion, depending on the kind of instruction involved.

Machine Cycle Identification:

With the exception of the DAD instruction, there is just one consideration that determines how many machine cycles an instruction comprises: the number of times that the processor must reference a memory address or an I/O addressable peripheral device, in order to fetch and execute an instruction. Like many processors, the BO80 is so constructed that it can transmit only one address per machine cycle. Thus, if the fetch and execution of an instruction requires two memory references, then the instruction cycle comprises two machine cycles. If five such references are called for, then the instruction cycle contains five machine cycles.

Every instruction cycle has at least one reference to memory, 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 therefore a FETCH. Beyond that, there are no fixed rules. It depends on the kind of instruction that is fetched.

Consider some examples. The add-register (ADD) instruction is an instruction that requires only a single machine cycle (FETCH) for its completion. In this one-byte instruction, the address of one of the CPU's general purpose registers is added to the existing contents of the accumulator. Because it is necessary to execute the command is contained in the eight bits of the instruction code, only one memory reference is necessary. Three states are required to execute the instruction from memory, and one additional state is used to accomplish the desired addition. The entire instruction cycle that requires only one machine cycle that consists of four states, or four periods of the external clock.

Suppose now, however, that we wish to add the contents of a specific memory location to the existing contents of the accumulator (ADD M0). Although this is quite similar in principle to the example just cited, several additional steps will be used. An extra machine cycle will be used, in order to address the desired memory location.

The actual sequence is as follows. First, the processor extracts the memory the one-byte instruction word addressed by its program counter. This takes three states. The address of the memory location that the fetch machine cycle is deposited in the CPU's instruction register and used to direct activities during the remainder of the instruction cycle. Next, the processor sends out an address, the contents of its H and L registers. The eight-bit data word is returned during the MEMORY READ cycle, which is placed in a temporary register inside the BO80 CPU. By these next three clock periods (states) have elapsed. In the seventh and final state, the contents of the temporary register is added to those of the accumulator. Two machine cycles, consisting of seven states in all, complete the "ADD M0" instruction cycle.

At the opposite extreme is the WAIT, L register (SHLD) instruction, which requires more machine cycles. During an "SHLD" instruction cycle, the contents of the processor's H and L registers are deposited in two sequentially adjacent memory locations; the destination is indicated by the value of the EX instruction. The address bits of the memory locations are immediately following the operation code byte. The sequence of events occurring is as follows:

(1) FETCH machine cycle, consisting of four states. During the first three states of this machine cycle, the processor fetches the instruction indicated by its program counter. The program counter is then incremented. The fourth state is used for internal state control.

(2) A MEMORY READ machine cycle, consisting of three states. During this machine cycle, the byte indicated by the program counter is taken from memory and placed in the processor's Z register. The program counter is incremented again.

(3) Another MEMORY READ machine cycle, consisting of three states, in which the byte indicated by the processor's program counter is read from memory and placed in the W register. The program counter is incremented again, in anticipation of the next instruction fetch.

(4) A MEMORY WRITE machine cycle, of three states, in which the contents of the W register are transferred to the memory location pointed to by the program counter of the W and Z registers. The state following the transfer is used to increment the W register counter so that it indicates the next memory location to receive data.

(5) A MEMORY WRITE machine cycle, of three states, in which the contents of the W register are transferred to the new memory location pointed to by the new W register pair.

In summary, the "SHLD" machine cycle consists contains five machine cycles and takes 15 states to execute.

Most instructions fall somewhere between the extremes illustrated by the "ADD M0" and the "SHLD" instructions. The input (INP) and the output (OUT) instructions, for example, require three machine cycles: a FETCH, to obtain the instruction; a MEMORY READ, to obtain the address of the object peripheral; and an INPUT or an OUTPUT cycle, to complete the transfer.
While no one instruction cycle will consist of more than five machine cycles, the following ten different types of machine cycles may occur within an instruction cycle:

1. FETCH (MT1)
2. MEMORY READ
3. MEMORY WRITE
4. STACK READ
5. STACK WRITE
6. INPUT
7. OUTPUT
8. INTERRUPT
9. HALT
10. HALT+INTERRUPT

The machine cycles that actually do occur in a particular instruction cycle depend upon the kind of instruction, with the overriding stipulation that the first machine cycle in any instruction cycle is always a FETCH.

The processor identifies the machine cycle in progress by transmitting an eight-bit status word during the first state of every machine cycle. Updated status information is presented on the 8808's data lines (D0-D7) during the SYNC interval. This data should be saved in latches, and used to develop control signals for external circuitry. Table 2-1 shows how the processor's true status information is distributed on the processor's data bus.

Status signals are provided principally for the control of external circuitry. Simplicity of interface, rather than machine cycle identification, dictates the logical definition of individual status bits. You will, therefore, observe that certain processor machine cycles are uniquely identified by a single status bit, but that others are not. The M1 status bit (D6), for example, unambiguously identifies a FETCH machine cycle. A STACK READ, on the other hand, is indicated by the coincidence of STACK and MEMR signals. Machine cycle identification data is also valuable in the test and debug-phasing of system development. Table 2-1 lists the status bit outputs for each type of machine cycle.

State Transition Sequence:

Every machine cycle within an instruction cycle consists of three to five active states (referred to as T1, T2, T3, T4, T5, or T6). The actual number of states depends upon the instruction being executed, and on the particular machine cycle within the instruction cycle itself. The state transition diagram in Figure 2-4 shows how the 8808 proceeds from state to state in the course of a machine cycle. The same diagram also shows how the READY, HOLD, and INTERRUPT lines are sampled during the machine cycle, and how the conditions on these lines may modify the basic transition sequence. In the present discussion, we are concerned only with the basic sequence and with the READY function. The HOLD and INTERRUPT functions will be discussed later.

The 8808 CPU does not directly indicate its internal state by transmitting a "state control" output during each state; instead, the 8808 supplies direct control output (INTE, HLDA, DBIN, WAIT) for use by external circuitry.

Recall that the 8808 passes through at least three states in every machine cycle, with each state defined by successive low-to-high transitions of the 68C clock. Figure 2-5 shows the timing relationships in a typical FETCH machine cycle. Events that occur in each state are referenced to transitions of the 68C clock and clock pulse.

The SYNC signal identifies the first state (T1) in every machine cycle. As shown in Figure 2-6, the SYNC signal is related to the leading edge of the 68C clock. There is a delay (T6) between the low-to-high transition of the 68C clock and the positive-going edge of the SYNC pulse. There is also a corresponding delay (also T6) between the next pulse and the trailing edge of the SYNC signal. Status information is displayed on D0-D7 during the same T6 to T6 interval.

Switching of the status signals is likewise controlled by T6.

The rising edge of T6 during T3 also leads the processor's address lines (A0-A15). These lines become stable within a brief delay (T12) of the 68C clocking pulse, and they remain stable until the next 68C pulse after state T5. This gives the processor ample time to read the data returned from memory.

Once the processor has sent an address to memory, there is an opportunity for the memory to request a WAIT. This it does by pulling the processor's READY line low, prior to the "Ready setup" interval (T7) which occurs during the 68C pulse within state T7 or T8. As long as the READY line remains low, the processor will idly, giving the memory time to respond to the addressed data request. Refer to Figure 2-6.

The processor responds to a WAIT request by entering an alternative state (T13) at the end of T7, rather than proceeding directly to the T8 state. Entry into the T9 state is indicated by a WAIT signal from the processor, acknowledgment of the memory's request. A low-to-high transition on the WAIT line is triggered by the rising edge of the 68C clock and occurs within a brief delay (T5) of the actual entry into the T9 state.

A wait period may be of indefinite duration. The processor remains in the waiting condition until its READY line again goes high. A READY indication must precede the falling edge of the 68C clock by a specified interval (T12) in order to guarantee an exit from the T9 state. The cycle may then proceed, beginning with the rising edge of the 68C clock. A WAIT interval will therefore consist of an integral number of T9 states and will always be a multiple of the clock period.

Table 2-1. 8808 Status Bit Definitions

<table>
<thead>
<tr>
<th>Bit</th>
<th>Definition</th>
<th>Symbol</th>
<th>Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Acknowledge signal</td>
<td>INTA</td>
<td>D6</td>
</tr>
<tr>
<td>2</td>
<td>WRITE memory or OUTPUT data</td>
<td>WD</td>
<td>D7</td>
</tr>
<tr>
<td>3</td>
<td>Indicates that operation in current machine cycle will be a WRITE memory or OUTPUT data</td>
<td>D8</td>
<td>D8</td>
</tr>
<tr>
<td>4</td>
<td>Indicates that the address bus holds the pushdown stack address from the Stack Pointer</td>
<td>D9</td>
<td>D9</td>
</tr>
<tr>
<td>5</td>
<td>Indicates that the address bus contains the address of an input device and the data bus should be ignored</td>
<td>D10</td>
<td>D10</td>
</tr>
<tr>
<td>6</td>
<td>Provides a signal to indicate that the CPU is in the fetch cycle for the first byte of an instruction</td>
<td>D11</td>
<td>D11</td>
</tr>
<tr>
<td>7</td>
<td>Indicates that the address bus contains the address of an output device and the input data should be ignored</td>
<td>D12</td>
<td>D12</td>
</tr>
<tr>
<td>8</td>
<td>Designates that the data bus will be used for memory read data</td>
<td>D13</td>
<td>D13</td>
</tr>
</tbody>
</table>

Table 2.1 8808 Status Bit Definitions
The events that take place during the T2 state are determined by the kind of machine cycle in progress. In a FETCH machine cycle, the processor interprets the data on its data bus as an instruction. During a MEMORY READ or a STACK READ, data on this bus is interpreted as a data word. The processor outputs data on this bus during a MEMORY WRITE machine cycle. During I/O operations, the processor may either transmit or receive data, depending on whether an OUTPUT or an INPUT operation is involved.

Figure 2.6 illustrates the timing that is characteristic of a data input operation. As shown, the low-to-high transition of $\phi_2$ during T2 clears status information from the processor's data lines, preparing these lines for the receipt of incoming data. The data presented to the processor must have stabilized prior to both the $\phi_3$-data set-up (t251), that precedes the falling edge of the $\phi_3$ pulse defining state T3, and the $\phi_3$-data set-up (t3532), that precedes the rising edge of $\phi_3$ in state T3. This same data must remain stable during the "data hold" interval (t404) that occurs following the rising edge of $\phi_3$ pulse. Data placed on these lines by memory or by other external devices will be sampled during T3.

During the input of data to the processor, the 8080 generates a DBIN signal which should be used externally to enable the transfer. Machine cycles in which DBIN is available include FETCH, MEMORY READ, STACK READ, and INTERRUPT. DBIN is initiated by the rising edge of $\phi_2$ during state T2 and terminated by the corresponding edge of $\phi_3$ during T3. Any $\phi_3$ phases intervening between T2 and T3 will therefore extend DBIN by one or more clock periods.

Figure 2.7 shows the timing of a machine cycle in which the processor outputs data. Output data may be destined either for memory or for peripherals. The rising edge of $\phi_1$ within state T2 clears status information from the CPU's data lines, and latches the data which is to be output to external devices. This substitution takes place within the

```
/* Insert diagram here */

NOTE: Refer to Status Word Chart

Figure 2.5: Basic 8080 Instruction Cycle

Figure 2.4: CPU State Transition Diagram
```
"Data output delay" interval (tqg) following the dh clock's leading edge. Data on the bus remains stable throughout the remainder of the machine cycle, until replaced by updated status information in the subsequent T4 state. Observe that a READY signal is necessary for completion of an OUTPUT machine cycle. Unless such an indication is present, the processor enters the T4g state, following the T3g state. Data on the output lines remains stable in the interim, and the processing cycle will not proceed until the READY line again goes high.

The 8080 CPU generates a WR output for the synchronization of external transfers, during those machine cycles, in which the processor outputs data. These include MEMORY WRITE, STACK WRITE, and OUTPUT. The negative-going leading edge of WR is referenced to the rising edge of the first dh clock pulse following T2g, and occurs within a brief delay (tqg) of that event. WR remains low until re-triggered by the leading edge of dh during the state following T4g. Note that any T4g states intervening between T2g and T4g of the output machine cycle will necessarily extend WR, in much the same way that DBIN is affected during data input operations.

All processor machine cycles consist of at least three states: T3g, T4g, and T5g as just described. If the processor has to wait for a response from the peripheral or memory with which it is communicating, then the machine cycle may also contain one or more T4g states. During the three basic states, data is transferred to or from the processor.

After the T3g state, however, it becomes difficult to generalize. T4g and T5g states are available, if the execution of a particular instruction requires them. But not all machine cycles make use of these states. It depends upon the kind of instruction being executed, and on the particular machine cycle within the instruction cycle. The processor will terminate any machine cycle as soon as its processing activities are completed, rather than proceeding through the T4g and T5g states every time. Thus the 8080 may exit a machine cycle following the T3g, the T4g, or the T5g state and proceed directly to the T3g state of the next machine cycle.

<table>
<thead>
<tr>
<th>STATE</th>
<th>ASSOCIATED ACTIVITIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>A memory address or I/O device number is placed on the Address Bus (A15:0); status information is placed on Data Bus (D7:0).</td>
</tr>
<tr>
<td>T2</td>
<td>The CPU samples the READY and HOLD inputs and checks for halt instruction.</td>
</tr>
<tr>
<td>TW</td>
<td>Processor enters wait state if READY is low or if HALT instruction has been executed. (optional)</td>
</tr>
<tr>
<td>T3</td>
<td>An instruction byte (FETCH machine cycle), data bytes (MEMORY READ, STACK READ) or interrupt instruction (INTERRUPT machine cycle) is input to the CPU from the Data Bus; or a data byte (MEMORY WRITE, STACK WRITE or OUTPUT machine cycle) is output onto the data bus.</td>
</tr>
<tr>
<td>T4</td>
<td>States T4 and T5 are available if the execution of a particular instruction requires them; if not, the CPU may skip one or both of them. T4g and T5g are only used for internal processor operations. (optional)</td>
</tr>
</tbody>
</table>

Table 2-2: State Definitions
INTERRUPT SEQUENCES

The 8080A has the built-in capacity to handle external interrupt requests. A peripheral device can initiate an interrupt simply by driving the processor's interrupt (INT) line high.

The interrupt (INT) input is asynchronous, and a request may therefore originate at any time during any instruction cycle. Internal logic re-clocks the external request, so that a proper correspondence with the driving clock is established. As Figure 2.6 shows, an interrupt request (INT) arriving during the time that the interrupt enable line (INTE) is high, acts in coincidence with the 8080A clock to set the internal interrupt latch. This event takes place during the last state of the interrupt cycle in which the request occurs, thus ensuring that any instruction in progress is completed before the interrupt can be processed.

The INTERRUPT machine cycle which follows the arrival of an enabled interrupt request resembles an ordinary FETCH machine cycle in most respects. The M₀ status bit is transmitted as usual during the SYNC interval. It is accompanied, however, by an INTA status bit (D₃) which acknowledges the external request. The contents of the program counter are latched onto the CPU's address lines during T₁, but the counter itself is not incremented during the INTERRUPT machine cycle, so it otherwise would be.

In this way, the pre-interrupt status of the program counter is preserved, so that data in the counter may be restored by the interrupted program after the interrupt request has been processed. The interrupt cycle is otherwise indistinguishable from an ordinary FETCH machine cycle. The processor itself takes no further special action. It is the responsibility of the peripheral logic to see that the eight-bit interrupt instruction is "jammed" onto the processor's data bus during state T₃. In a typical system, this means that the data-in bus from memory must be temporarily disconnected from the processor's main data bus, so that the interrupting device can command the main bus without interference.

The 8080A's instruction set provides a special one-byte CALL which facilitates the processing of interrupts (the ordinary program CALL takes three bytes). This is the RESTART instruction (RST). A variable three-bit field embedded in the eight-bit field of the RST enables the interrupting device to direct a CALL to one of eight fixed memory locations. The decimal addresses of these dedicated locations are: 0, 8, 16, 24, 32, 40, 48, and 56. Any of these addresses may be used to store the first instructions of a routine designed to service the requirements of an interrupting device. Since the (RST) is a call, completion of the instruction also stores the old program counter contents on the STACK.

Figure 2.6: Interrupt Timing

Figure 2.8: HOLD Operation (Read Mode)

Figure 2.10: HOLD Operation (Write Mode)
HOLD SEQUENCES

The 8080A CPU contains provisions for Direct Memory Access (DMA) operations. By applying a HOLD to the appropriate control pin on the processor, an external device can cause the CPU to suspend its normal operations and relinquish control of the address and data busses. The processor responds to a request of this kind by floating its address to other devices sharing the busses. At the same time, the processor acknowledges the HOLD by placing a high on its HOLD output pin. During an acknowledged HOLD, the address and data busses are under control of the peripheral which originated the request, enabling it to conduct memory transfers without processor intervention.

Like the interrupt, the HOLD input is synchronized internally. A HOLD signal must be stable prior to the "Hold set-up" interval \( T_3 \), which precedes the rising edge of \( T_4 \).

Figure 2-9 and 2-10 illustrate the timing involved in HOLD operations. Note the delay between the asynchronous HOLD REQUEST and the recognized HOLD. As shown in the diagram, a coincidence of the READY, the HOLD, and the \( d_0 \) clock is the internal hold latch. Setting the latch enables the subsequent rising edge of the \( d_0 \) clock pulse to trigger the HOLD output.

Acknowledgement of the HOLD REQUEST produces slightly the actual holding of the processor's address and data lines. The processor acknowledges a HOLD at the beginning of \( T_3 \), if a read or an input machine cycle is in progress (see Figure 2-9). Otherwise, acknowledgement is deferred until the beginning of the state following \( T_2 \) (see Figure 2-10). In both cases, however, the HOLD goes high within a specified delay \( (T_{PP}) \) of the rising edge of the selected \( d_0 \) clock pulse. Address and data lines are floated within a brief delay after the rising edge of the next \( d_0 \) clock pulse. This relationship is also shown in the diagrams.

To allow proper sequencing, the processor has suspend ed its operations once the address and data busses are floated. Internally, however, certain functions may continue. If a HOLD REQUEST is acknowledged at \( T_2 \), and if the processor is in the middle of a machine cycle which requires four or more states to complete, the CPU proceeds through \( T_3 \) and \( T_4 \) before coming to a rest. Next, until the end of the machine cycle is reached, processing activity continues. Internal processing is thus permitted to overlap the external DMA transfer, improving both the efficiency and the speed of the entire system.

The processor exits the holding state through a sequence similar to that by which it entered. A HOLD REQUEST is terminated asynchronously when the external device has completed its data transfer. The ILDA output returns to a low level following the leading edge of the next \( d_0 \) clock pulse. Normal processing resumes with the machine cycle following the last cycle that was executed.

HALT SEQUENCES

When a halt instruction (HLT) is executed, the CPU enters the halt state \( H \) after state \( T_2 \) of the next machine cycle, as shown in Figure 2-11. There are only three ways in which the 8080 can exit the halt state:

- A high on the RESET line will always reset the CPU to state \( T_1 \). RESET clears the program counter.
- A HOLD input will cause the 8080 to enter the halt state, as previously described. When the HOLD line goes low, the 8080 returns to the halt state on the rising edge of the next \( d_0 \) clock pulse.
- An interrupt (i.e., INT goes high while INTE is enabled) will cause the 8080 to exit the Halt state and assert state \( T_2 \) on the rising edge of the next \( d_0 \) clock pulse. NOTE: The interrupt enable (INTE) flag must be set when the halt state is entered; otherwise, the 8080 will only be able to exit via a RES set.

Figure 2-12 illustrates halt sequencing in flow chart form.

START-UP OF THE 8080 CPU

When power is applied initially to the 8080, the processor begins operating immediately. The contents of its program counter, stack pointer, and the other working registers are naturally subjected to random factors and cannot be specified. For this reason, it will be necessary to begin the power-up sequence with RESET.

An external RESET signal of three clock period duration (minimum) restarts the processor's internal program counter to zero. Program execution then begins with memory location zero, following a RESET. Systems which require the processor to wait for an explicit start-up signal will store a halt instruction (81H, HLT) in the first two locations. A manual or an automatic INTERRUPT will be used for starting. In other systems, the processor may begin executing its stored program immediately. Note, however, that the RESET has no effect on status flags, or on any of the processor's working registers (accumulator, registers, or stack pointer). The contents of these registers remain indeterminate, until initialized explicitly by the program.

Figure 2-11. HALT Timing

Figure 2-12. HALT Sequence Flow Chart.
NOTES:
1. The first memory cycle (M1) is always an instruction fetch; the first (or only) byte, containing the op code, is fetched during this cycle.
2. If the READY input from memory is not high during T2 of each memory cycle, the processor will enter a wait state (TW) until READY is sampled as high.
3. States T4 and T5 are present, as required, for operations which are completely internal to the CPU. The contents of the internal bus during T4 and T5 are available at the data bus; this is designed for testing purposes only. An "X" denotes that the state is present, but is only used for such internal operations as instruction decoding.
4. Only register pairs 8B and BI or rp = D (registers D and E) may be specified.
5. These states are skipped.
6. Memory read sub-cycle: an instruction or data word will be read.
7. Memory write sub-cycle.
8. The READY signal is not required during the second and third sub-cycles (M2 and M3). The HOLD signal is accepted during M2 and M3. The S/RNC signal is generated during M2 and M3. During the execution of DAT, M2 and M3 are required for an internal register-pair add; memory is not referenced.
9. The results of these arithmetic, logical, or rotate instructions are not moved into the accumulator (A) until state T2 of the next instruction cycle. That is, A is loaded while the next instruction is being fetched; this overlapping of operations allows for faster processing.
10. If the value of the least significant 4 bits of the accumulator is greater than 8, or if the carry bit is set, 6 is added to the accumulator. If the value of the most significant 4 bits of the accumulator is now greater than 8, or if the carry bit is set, 6 is added to the most significant 4 bits of the accumulator. This represents the first sub-cycle (the instruction fetch) of the next instruction cycle.
12. If the condition was met, the contents of the register pair M2 are output on the address lines (A0-A2) instead of the contents of the program counter (PCL).
13. If the condition was not met, sub-cycles M4 and M5 are skipped; the processor instead proceeds immediately to the instruction fetch (M1) of the next instruction cycle.
14. If the condition was not met, sub-cycles M2 and M3 are skipped; the processor instead proceeds immediately to the instruction fetch (M1) of the next instruction cycle.
15. Stack read sub-cycle.
16. Stack write sub-cycle.
17. CONDITION C00 Z - not zero Z = 0 -000 00000000 00000000
    N - zero Z = 1 001 00000001 00000001
    C - carry (CY = 0) 010 00000100 00000100
    P - parity odd (P = 1) 011 00000101 00000101
    PE - parity even (P = 1) 101 00001010 00001010
    P - plus (S = 0) 110 00001100 00001100
    M - minus (S = 1) 111 00001110 00001110
18. I/O sub-cycle: the I/O pair's 8-bit select code is duplicated on address lines D7 (A7) and B15 (A15).
19. Output sub-cycle.
20. The processor will remain idle in the hold state until an interrupt, a reset, or a hold is accepted. When a hold request is accepted, the CPU enters the hold mode; after the hold mode is terminated, the processor returns to the halt state. After a reset is accepted, the processor begins execution at memory location zero. After an interrupt is accepted, the processor executes the instruction forced onto the data bus (usually a reset instruction).

<table>
<thead>
<tr>
<th>SS5 or DDD</th>
<th>Value</th>
<th>rp</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>111</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>010</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>001</td>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>001</td>
<td>E</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>011</td>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>100</td>
<td>L</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2-13. Reset.

Figure 2-14. Relation between HOLD and INT in the HALT State.
The 8080 INSTRUCTION SET

The 8080 can address data up to 65,536 bytes of memory, which may consist of both read-only memory (ROM) elements and random access memory (RAM) elements (read/write memory).

In the 8080, data is stored in the form of 8-bit binary images.

When a register or data word contains a binary number, it is necessary to establish the order in which the bits of the number are written. In the Intel 8080, bit 0 is referred to as the Least Significant Bit (LSB), and bit 7 (if an 8-bit number) is referred to as the Most Significant Bit (MSB).

The 8080 program instructions may be one, two, or three bytes in length. Multiple byte instructions must be stored in successive memory locations; the address of the first byte is always used as the address of the instructions. The exact instruction format will depend on the particular operation to be executed.

Single Byte Instructions

```
<table>
<thead>
<tr>
<th>D1</th>
<th>D2</th>
</tr>
</thead>
</table>

```

Two-Byte Instructions

```
<table>
<thead>
<tr>
<th>D1</th>
<th>D2</th>
</tr>
</thead>
</table>

```

Three-Byte Instructions

```
<table>
<thead>
<tr>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
</table>

```

Addressing Modes:

Often the data that is to be operated on is stored in memory. When multi-byte numeric data is used, the data, like instructions, is stored in successive memory locations, with the least significant byte first, followed by increasingly significant bytes. The 8080 has four different modes for addressing data stored in memory or in registers:

- Direct - Bytes 2 and 3 of the instruction contain the exact memory address of the data item (the low-order bits of the address are in byte 2, the high-order bits in byte 3).
- Register - The instruction specifies the register or register-pair in which the data is located.
- Register Indirect - The instruction specifies a register-pair which contains the memory address where the data is located (the high-order bits of the address are the first register of the pair, the low-order bits in the second).
- Immediate - The instruction contains the data it sets. This is either an 8-bit quantity or a 16-bit quantity (least-significant byte first, most significant byte second).

Unless directed by an interrupt or branch instruction, the execution of a program is continu- ously increasing memory locations. A branch instruction can specify the address of the next instruction to be executed in one of two ways:

- Direct - The branch instruction contains the address of the next instruction to be executed. (Except for the "RET" instruction, byte 2 contains the low-order address and byte 3 the high-order address.)
- Register Indirect - The branch instruction contains the register-pair which contains the memory address of the next instruction to be executed. (The high-order bits of the address are in the first register of the pair, the low-order bits in the second.)

The RET instruction is a special one-byte call instruction (usually used during interrupt sequences). RET includes a three-byte field, program control is transferred to the instruction whose address is eight times the contents of this three-byte field.

Condition Flags:

- Zero: If the result of an instruction has the value 0, this flag is set; otherwise it is reset.
- Sign: If the most significant bit of the result of the operation has the value 1, this flag is set; otherwise it is reset.
- Parity: If the modulus 2 sum of the bits of the result of the operation is 0, i.e., if the result has even parity, this flag is set; otherwise it is reset (i.e., if the result has odd parity).
- Carry: If the instruction resulted in a carry (from addition), or a borrow from subtraction or a comparison) out of the high-order bit, this flag is set; otherwise it is reset.

Auxiliary Carry: If the instruction caused a carry out of bit 3 and into bit 4, the auxiliary carry value, the auxiliary carry is set; otherwise it is reset. This flag is affected by single precision additions, subtractions, increments, decrements, comparisons, and logical operations, but is principally used with additions and increments providing a DAA (Decimal Adjust) Accumulator instruction.

Symbols and Abbreviations:

The following symbols and abbreviations are used in the subsequent description of the 8080 instructions:

- **DOUBLE MEANING**
  - **accumulator** Register A
  - **add** 16-bit address quantity
  - **data** 16-bit address quantity
  - **byte** 2 The second byte of the instruction
  - **byte 3** The third byte of the instruction
  - **port** 8-bit address of an I/O device
  - **r11/2** One of the registers A,B,C,D,E,H,L
  - **DOO.SSS** The bit pattern designating one of the regist- ers, A,B,C,D,E,H,L (DOO=destination, SSS=source).
  - **DOO or SSS REGISTER NAME**
    - **001** A
    - **002** B
    - **003** C
    - **010** D
    - **011** E
    - **100** H
    - **101** L
  - **rp** One of the register pairs

The following pages provide a detailed description of the instruction set of the 8080.
## Summary of Processor Instructions

### By Alphabetical Order

<table>
<thead>
<tr>
<th>Instruction Code</th>
<th>Instruction Description</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Move immediate memory</td>
<td>MOV IM</td>
<td>Move Immediate Memory</td>
</tr>
<tr>
<td>001</td>
<td>Move immediate register</td>
<td>MOV IR</td>
<td>Move Immediate Register</td>
</tr>
<tr>
<td>010</td>
<td>Move register to memory</td>
<td>MOV RM</td>
<td>Move Register to Memory</td>
</tr>
<tr>
<td>011</td>
<td>Move register to register</td>
<td>MOV RR</td>
<td>Move Register to Register</td>
</tr>
<tr>
<td>100</td>
<td>Move memory to register</td>
<td>MOV MR</td>
<td>Move Memory to Register</td>
</tr>
<tr>
<td>101</td>
<td>NOP</td>
<td>NOP</td>
<td>No Operation</td>
</tr>
<tr>
<td>110</td>
<td>ORI</td>
<td>ORI</td>
<td>OR Immediate</td>
</tr>
<tr>
<td>111</td>
<td>ORR</td>
<td>ORR</td>
<td>OR Register</td>
</tr>
<tr>
<td>1110</td>
<td>ROR</td>
<td>ROR</td>
<td>Rotate Right</td>
</tr>
<tr>
<td>1111</td>
<td>RSH</td>
<td>RSH</td>
<td>Shift Right</td>
</tr>
<tr>
<td>11110</td>
<td>RLSE</td>
<td>RLSE</td>
<td>Rotate Left and Store Excess</td>
</tr>
<tr>
<td>11111</td>
<td>RLSEI</td>
<td>RLSEI</td>
<td>Rotate Left and Store Excess Immediate</td>
</tr>
<tr>
<td>111110</td>
<td>RORI</td>
<td>RORI</td>
<td>Rotate Right and Immediate</td>
</tr>
<tr>
<td>111111</td>
<td>RORR</td>
<td>RORR</td>
<td>Rotate Right and Register</td>
</tr>
<tr>
<td>1111110</td>
<td>RSHI</td>
<td>RSHI</td>
<td>Shift Right and Immediate</td>
</tr>
<tr>
<td>1111111</td>
<td>RSHR</td>
<td>RSHR</td>
<td>Shift Right and Register</td>
</tr>
</tbody>
</table>

### High-Level Summary

- **MOV IM**: Move Immediate Memory
- **MOV IR**: Move Immediate Register
- **MOV RM**: Move Register to Memory
- **MOV RR**: Move Register to Register
- **MOV MR**: Move Memory to Register
- **NOP**: No Operation
- **ORI**: OR Immediate
- **ORR**: OR Register
- **ROR**: Rotate Right
- **RSH**: Shift Right
- **RLSE**: Rotate Left and Store Excess
- **RLSEI**: Rotate Left and Store Excess Immediate
- **RORI**: Rotate Right and Immediate
- **RORR**: Rotate Right and Register
- **RSHI**: Shift Right and Immediate
- **RSHR**: Shift Right and Register

### Notes

1. DDD or SSS - 000 B - 001 C - 010 D - 011 E - 100 H - 101 L - 110 Memory - 111 A.
2. Two possible cycle times, (1/11) indicates instruction cycles dependent on condition flags.
We appreciate your comments on MICROCOMPUTER DESIGN:

Name__________________________

Title__________________________ Phone__________________________

Company/Institution__________________________

Address__________________________

City__________________________ State__________________________ ZIP__________________________

Please rush the following:

___ A catalog on the modular micros from Martin Research, with prices.

___ Another copy of this book, MICROCOMPUTER DESIGN. $25.00 is enclosed.

___ Brochure(s) on this book, including current prices: ___ brochures.

___ Quantity discount schedule for MICROCOMPUTER DESIGN.

___ Educational discount schedule (for officials and representatives of educational institutions only).
modular micros!

We are proud to announce the modular micros from Martin Research. These versatile printed circuit modules can be configured to produce a variety of microcomputer systems, from the simple to the complex.

**BEST VALUE!**

Not just a collection of parts, with the wirewrapping left to you. ...but complete with commercial-grade printed circuit boards, made to top PC houses, with plated-through holes. Minimizes construction time, and greatly reduces troubleshooting.

**MOST ADAPTABLE!**

Our universal bus structure is compatible with all standard eight-bit microprocessors! This is a major advantage for the user who wishes to upgrade his system in the future. And, it is ideal for the OEM building prototypes with competing processors. Memory and accessory boards remain compatible when the CPU is changed.

**MOST FLEXIBLE!**

The system architecture, designed for optimum flexibility, allows any board to be inserted in any position on the bus. A flat cable and snap-on fifty-pin connectors make for easy expansion.

**SUPERIOR SYSTEMS DESIGN!**

Programs are entered on a calculator-style keyboard, and six bright LED digits display data and memory addresses. A Monitor program in a PROM makes program entry easy. This is a significant advance over console designs with cumbersome arrays of toggle switches and lamps.

**FAST MEMORY AVAILABLE!**

Our 4K static RAM board uses RAM chips fast enough—460 ns—to avoid making the CPU wait for memory access in most applications. Now in stock, this board is price-competitive with others' memory boards with far slower specifications. Warning: some manufacturers using slow RAM do not inform the buyer of the fact. The result is that the CPU must wait for memory, resulting in a computer much slower than the specs might imply.

**BEST FOR YOUR APPLICATION!**

With microcomputer systems starting at under $400, the modular micros are ideal for:

- The computer enthusiast, in his home;
- The professional engineer or programmer, to keep up with the state of the art;
- The manufacturer, for prototyping microcomputer systems, or for incorporation in his product.

(OEMs: write for further information.)
## REGISTER INSTRUCTIONS

<table>
<thead>
<tr>
<th>EX:</th>
<th>LOAD ACCUMULATOR FROM MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=0</td>
<td>D=3</td>
</tr>
<tr>
<td>E=2</td>
<td>H=5</td>
</tr>
</tbody>
</table>

### LOAD REGISTER IMMEDIATE
- **LIT**: 006 016 026 036 046 056 066 076
- **INCR**: 016 026 036 046 056 066 076 060
- **DECR**: 016 026 036 046 056 066 076 000

### ALU INSTRUCTIONS

<table>
<thead>
<tr>
<th>MNE/MONIC</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>H</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
</tr>
<tr>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
<td>ADD F TO A</td>
</tr>
<tr>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
</tr>
<tr>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
<td>SUB F FROM A</td>
</tr>
<tr>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
</tr>
<tr>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
<td>AND F WITH A</td>
</tr>
<tr>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
</tr>
<tr>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
<td>EXCL OR F WITH A</td>
</tr>
<tr>
<td>COMplement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
</tr>
<tr>
<td>COMplement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
<td>Complement F WITH A</td>
</tr>
</tbody>
</table>

### PC/STACK INSTRUCTIONS

<table>
<thead>
<tr>
<th>MNE/MONIC</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>H</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>RST 00H</td>
<td>RST 00H</td>
<td>RST 00H</td>
<td>RST 00H</td>
<td>RST 00H</td>
<td>RST 00H</td>
<td>RST 00H</td>
<td>RST 00H</td>
<td>RST 00H</td>
</tr>
</tbody>
</table>

### UNCONDITIONAL INSTRUCTIONS

<table>
<thead>
<tr>
<th>MNE/MONIC</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>H</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>JUMP 1x4</td>
<td>JUMP 1x4</td>
<td>JUMP 1x4</td>
<td>JUMP 1x4</td>
<td>JUMP 1x4</td>
<td>JUMP 1x4</td>
<td>JUMP 1x4</td>
<td>JUMP 1x4</td>
<td>JUMP 1x4</td>
</tr>
<tr>
<td>CALL 1x4</td>
<td>CALL 1x4</td>
<td>CALL 1x4</td>
<td>CALL 1x4</td>
<td>CALL 1x4</td>
<td>CALL 1x4</td>
<td>CALL 1x4</td>
<td>CALL 1x4</td>
<td>CALL 1x4</td>
</tr>
<tr>
<td>RETURN 2x7</td>
<td>RETURN 2x7</td>
<td>RETURN 2x7</td>
<td>RETURN 2x7</td>
<td>RETURN 2x7</td>
<td>RETURN 2x7</td>
<td>RETURN 2x7</td>
<td>RETURN 2x7</td>
<td>RETURN 2x7</td>
</tr>
</tbody>
</table>

### FALSE CONDITION

- **JF**: 100
- **JF**: 100
- **JC**: 100
- **JE**: 100
- **JL**: 100
- **JE**: 100
- **JL**: 100
- **JL**: 100

### TRUE CONDITION

- **JL**: 120
- **JE**: 120
- **JE**: 120
- **JE**: 120
- **JE**: 120
- **JE**: 120
- **JE**: 120
- **JE**: 120

### PC/STACK CODES

- **HLY 000**: 010
- **HLY 001**: 011
- **HLY 011**: 020
- **HLY 021**: 021
- **HALT**: 070
- **HALT**: 071
- **HALT**: 072
- **HALT**: 077

---

*ADD*: 200
*SUB*: 220
*AND*: 240
*EXOR*: 260
*COMPLEMENT*: 280
*ADD F TO A*: 210
*SUB F FROM A*: 220
*AND F WITH A*: 240
*EXCL OR F WITH A*: 260
*COMPARE F WITH A*: 280

---

*FLAGS AFFECTED*: C Z S P
*INCREMENT/DECREMENT ALU INSTRUCTIONS*: / / / /
## 8080 INSTRUCTIONS

### Summary of Processor Instructions in Alphabetic Order

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Opcode Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU r</td>
<td>OR register with A</td>
<td>2 6 r</td>
</tr>
<tr>
<td>ORI r</td>
<td>OR immediate with A</td>
<td>3 6 r</td>
</tr>
<tr>
<td>OUT port</td>
<td>Output</td>
<td>3 2 3</td>
</tr>
<tr>
<td>PCHL</td>
<td>HL to program counter</td>
<td>3 5 1</td>
</tr>
<tr>
<td>POP r</td>
<td>Pop register pair onto stack</td>
<td>3 6 1 r</td>
</tr>
<tr>
<td>POP PSW</td>
<td>Pop A and flags off stack</td>
<td>3 6 3</td>
</tr>
<tr>
<td>PUSH r</td>
<td>Push register pair onto stack</td>
<td>3 5 3 (only B, D, H)</td>
</tr>
<tr>
<td>PUSH PSW</td>
<td>Push A and flags onto stack</td>
<td>3 6 3</td>
</tr>
<tr>
<td>RAL</td>
<td>Rotate A left through carry</td>
<td>0 2 7</td>
</tr>
<tr>
<td>RAR</td>
<td>Rotate A right through carry</td>
<td>0 5 7</td>
</tr>
<tr>
<td>RCV</td>
<td>Return on condition</td>
<td>3 7 0</td>
</tr>
<tr>
<td>RCL</td>
<td>Rotate left</td>
<td>0 0 7</td>
</tr>
<tr>
<td>RRC</td>
<td>Rotate A right</td>
<td>0 1 7</td>
</tr>
<tr>
<td>RST m</td>
<td>Restart</td>
<td>3 7 1</td>
</tr>
<tr>
<td>SUB P</td>
<td>Subtract reg. from A</td>
<td>2 3 p</td>
</tr>
<tr>
<td>SUBB P</td>
<td>Subtract imm. from A</td>
<td>2 3 p</td>
</tr>
<tr>
<td>SMLD</td>
<td>Store HL direct</td>
<td>0 4 2</td>
</tr>
<tr>
<td>SPHL</td>
<td>HL to stack pointer</td>
<td>3 7 1</td>
</tr>
<tr>
<td>STA a</td>
<td>Store A direct</td>
<td>0 6 2</td>
</tr>
<tr>
<td>STAX P</td>
<td>Store A indirect only B, D</td>
<td>0 6 2</td>
</tr>
<tr>
<td>STD</td>
<td>Set carry</td>
<td>0 6 7</td>
</tr>
<tr>
<td>SUB r</td>
<td>Subtract register from A</td>
<td>2 2 r</td>
</tr>
<tr>
<td>SWT data</td>
<td>Subtract immediate from A</td>
<td>3 2 6</td>
</tr>
<tr>
<td>XCH</td>
<td>Exchange HC, HL register pairs</td>
<td>3 5 3</td>
</tr>
<tr>
<td>XMA r</td>
<td>Exclusive OR register with A</td>
<td>2 5 3</td>
</tr>
<tr>
<td>XMI data</td>
<td>Exclusive OR immediate with A</td>
<td>3 5 6</td>
</tr>
<tr>
<td>XNIE</td>
<td>Exchange top of stack with HL</td>
<td>3 4 3</td>
</tr>
</tbody>
</table>

### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Memory address</td>
<td>16</td>
</tr>
<tr>
<td>c</td>
<td>Condition</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>Destination</td>
<td>3</td>
</tr>
<tr>
<td>date</td>
<td>Data</td>
<td>8/16</td>
</tr>
<tr>
<td>port</td>
<td>I/O port</td>
<td>8</td>
</tr>
<tr>
<td>r</td>
<td>Register</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>Source</td>
<td>3</td>
</tr>
</tbody>
</table>

### Instruction Code Values

<table>
<thead>
<tr>
<th>OCTAL</th>
<th>REGISTER</th>
<th>REGISTER PAIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B</td>
<td>BC</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>CC</td>
</tr>
<tr>
<td>2</td>
<td>D</td>
<td>DE</td>
</tr>
<tr>
<td>3</td>
<td>E</td>
<td>EE</td>
</tr>
<tr>
<td>4</td>
<td>H</td>
<td>HL</td>
</tr>
<tr>
<td>5</td>
<td>L</td>
<td>LF</td>
</tr>
<tr>
<td>6</td>
<td>M</td>
<td>SF</td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td>SP</td>
</tr>
</tbody>
</table>

### Conditions

- **Set zero** (Z) | 0
- **Negative (N)** | 1
- **Positive (P)** | 2
- **Carry (C)** | 3
- **Parity odd (O)** | 4
- **Parity even (P)** | 5
- **Overflow (V)** | 6
- **Greater than (G)** | 7

### Instruction Code Flags

<table>
<thead>
<tr>
<th>Code</th>
<th>Description Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

---

**The 8080 is an 8-bit microprocessor, and its instruction code tends to break down into a 2/3/3-bit grouping. This is why it was convenient in this chart to list 8080 instructions in octal format, where digits follow this same pattern.**

**Hexadecimal notation (4/4-bit grouping) is often used instead of octal, but since hex does not correspond directly to the way in which the 8080 is microcoded, hex instruction codes are not readily listed in condensed graphic form, and are more difficult for the programmer to learn.**

---

*Copyright © 1976, Martin Research*
**Register Transfers**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>H</th>
<th>L</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
</tr>
</tbody>
</table>

**Double-Precision Transfers**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>H</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
</tr>
</tbody>
</table>

**Indirect Transfers**

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>INDEX</th>
<th>DATA</th>
<th>RAW TRANSFER</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>X</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

**Direct Load & Store**

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>INDEX</th>
<th>DATA</th>
<th>RAW TRANSFER</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>X</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

**Program Control**

<table>
<thead>
<tr>
<th>UN</th>
<th>CN</th>
<th>BE</th>
<th>IM</th>
<th>NOT</th>
<th>ZE</th>
<th>ZO</th>
<th>CUD</th>
<th>TAN</th>
<th>VTY</th>
<th>NUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
</tbody>
</table>

**Bit Manipulation**

<table>
<thead>
<tr>
<th>A</th>
<th>K</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>H</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
</tr>
</tbody>
</table>

**ALU Instructions**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>H</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
</tr>
</tbody>
</table>

**Machine Control**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>H</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
</tr>
</tbody>
</table>

**Flags**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>H</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
</tr>
</tbody>
</table>

---

**Copyright © 1976**

**Martin Research**

**All Rights Reserved**