


BY RANDY CARLSTROM

DESIGNING WITH THE

8080 MICROPROCESSOR

Part 1: The Basic System

With the widely used 8080 as a model, the basic features of a central processing system are explored



IN ADDITION to its obvious application as the central processing unit (CPU) of a computer system, the microprocessor has found its way into a variety of products ranging from kitchen equipment to sophisticated laboratory data-acquisition systems. The key to this widespread utility is flexibility, which in turn comes from the microprocessor's unique ability to alter its internal logic in response to an external program. Since the response to inputs from the program is extremely rapid—on the order of a few microseconds—the processor can change its electrical configuration practically instantaneously, usually fast enough to convince a human correspondent that it is performing several activities simultaneously.

Given the speed and flexibility of microprocessors, and the fact that they are available at very reasonable prices, it is often economical to use a single processor rather than a great many simpler chips to synthesize logic functions, act as a controller, or the like. To accomplish this, however, it is necessary to understand the architecture of the processor, its needs in terms of support circuitry, how to program it, and how to interface it with the "outside world." Development of the necessary understanding is the goal of this multipart series.

Microprocessors vary in design, with

each design programmable only via its own set of instructions. The unit that will be covered in detail in this series is the 8080. Since this CPU is the grandfather of a growing family of processors, including the Z80, 8048, and 8085, all with a common internal programming language, most of the information will apply to the entire family as well. Instructions not used by the 8080 will not, however, be covered.

The Basic System. Like many processors and logic elements, the 8080 requires a small number of support ICs in order to function. An 8080 along with its support chips is called a *CPU module*.

The program that determines the internal states taken on by the CPU is supplied to it in the form of electrical signals. To generate these signals as required and in the proper order, the program must be stored in some form of "memory" device. These devices represent the binary digits (1, 0) by means of "on-off" switching devices or analogous circuit elements. The binary code in which program instructions are expressed is called *machine language*. Each microprocessor (or microprocessor family) has its own machine language.

Binary instructions or data that are not subject to change can be stored permanently in ROM (read-only memory).

Elements that are variable must be stored in RAM (random-access memory), which can be written, erased, and rewritten by the CPU.

To affect or control devices that interact with the outside world, the processor must deliver signals to them. It does this by means of an I/O (input/output) port. As the name implies, an I/O port can also deliver signals to the CPU from devices that sense external parameters.

Electrical signals representing data, instructions, and addresses (the locations of particular items in memory) pass between the CPU, memory devices, and I/O ports via a set of dedicated lines known collectively as *buses*. A typical bus (Fig. 1) also supplies dc operating power to the elements of the system.

Bus System. There are many versions of the bus system currently used, with the S-100 and SS50 being two of the most common. Although different mechanically, they all contain three major elements: the address bus, the data bus, and the control bus. (Figure 1 does not show the power supply lines and common ground usually carried on the bus system.)

In most systems, there are 16 lines in the address bus, thus enabling 2^{16} or 65,536 (64K) unique addresses. In an 8-bit system, there are 8 lines on the data

8080 microprocessor

bus, allowing 2^8 or 256 data combinations. The control bus carries all system synchronization signals including the "clock" that keeps all CPU module events in step.

Memory. A computer memory is formed from a large array of semiconductor elements, each capable of storing a single binary 1 or 0, organized into groups of *bits* (short for "binary digits") often called *words*. The number of bits in each word is determined by the size of the CPU *registers* (storage locations in-

ternal to the microprocessor) and the number of data lines. A typical RAM arrangement is shown in Fig. 2. A memory word of eight bits is often referred to as a *byte*. Each byte represents one of 2^8 or 256 unique values (0-255). As the 8080 microprocessor uses this memory structure, it is considered a byte-oriented device.

Each memory location contains one word of memory bits, and is identified by a unique number, or *address*, assigned to it. The CPU gains access to the contents of any memory location by

means of its address. A memory word may represent the encoded form of an instruction, or may be data to be processed by the CPU.

The CPU has control of memory in the sense that it can read data and instructions from memory and write data back into memory. Only when the CPU receives a direct memory access (DMA) signal via the control bus does it relinquish control of memory. DMA allows a high-speed device such as a magnetic disk to gain access to memory and control it. As noted earlier, memory that can be read and written or altered is termed read/write memory, or random-access memory (RAM). Memory that can be read, but not altered by writing, is termed read-only memory (ROM).

Input/Output. To the 8080, the outside world may consist of up to 256 input and 256 output devices. These are usually referred to as peripherals, and may include keyboards, printers, displays, etc. Each peripheral communicates with the CPU by exchange of data bytes sent via its associated I/O port and the data bus (Fig. 3). Each peripheral is assigned an address from 0 to 255, much as each memory location is assigned an address. The portion of the I/O system that actually conditions data for input and output is known as the *interface* and generally there is one interface for each peripheral. The use of a port for input or output is done under program control.

Communication between the computer and a peripheral is done in one of two formats—serial or parallel. In parallel data transfer, all eight bits of the data byte are handled simultaneously. This permits rapid movement of data. In serial transfer, data is handled bit-by-bit instead of a byte at a time. This is slower, but has the advantage of using simple hardware (for example, a two-conductor cable or a telephone circuit instead of a multiconductor bus). When two computers exchange data via, say, an intercom line, the parallel data from buses of both computers is converted to serial form and transmitted bit-by-bit down the cable. The IC that performs the conversion from parallel to bit-serial form (and vice versa) belongs to a family of components known as UARTs (Universal Asynchronous Receiver-Transmitter). If used, the UART is part of the computer's I/O interface since it is used for conditioning data for input and output.

There are two basic types of serial communication—RS232 and what is called the 20-mA current loop. Basically, RS232 is a voltage circuit where a logic 0 is a positive voltage, and a logic 1 is a negative voltage. The newest version

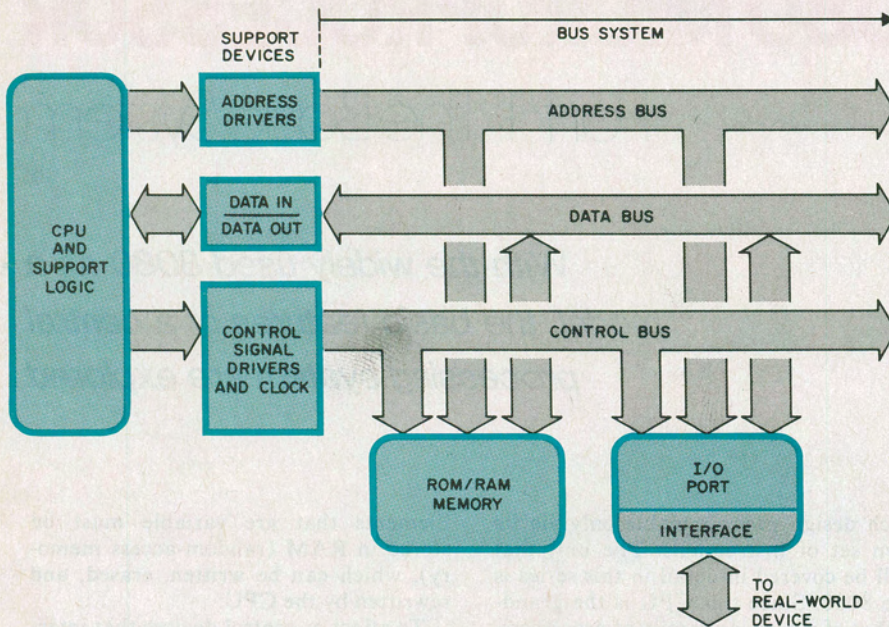


Fig. 1. A typical bus system contains three major elements: address bus, data bus, and control bus.

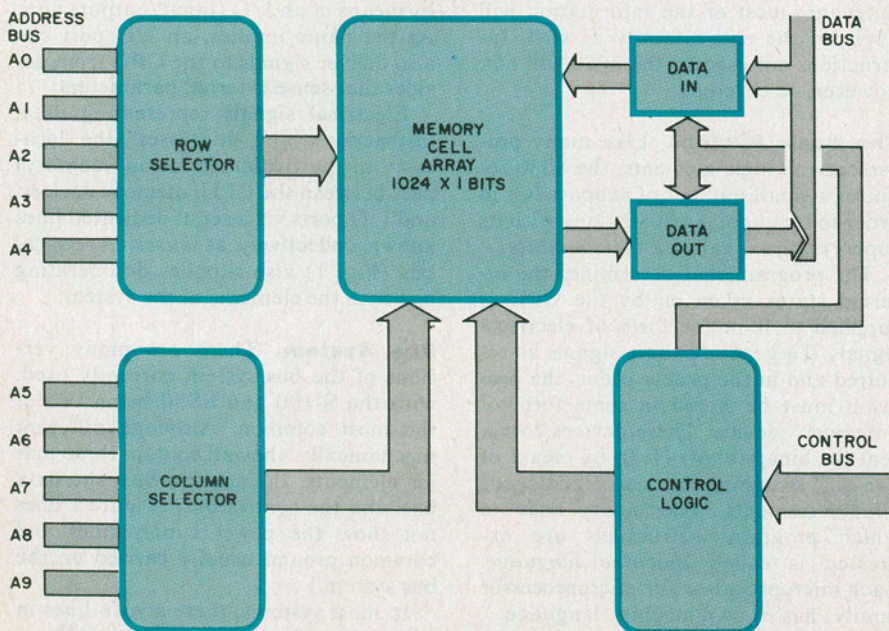


Fig. 2. Arrangement of a 2102 random-access memory. Eight of these are needed for 1024 by 8 bits.

of this voltage interface is RS422—which uses balanced transmission lines and differential current sensing to eliminate noise. The other commonly used serial port is the 20-mA current loop in which a flow of 20 mA in the series circuit produces a logic 1 while an absence of current denotes a logic 0. Both of these serial ports are controlled by a baud rate generator that “clocks” the operational speed of the port. Most peripherals use either the RS232 or 20-mA loops for communication.

Program Interrupt I/O improves the efficiency of CPU operation while data is being transferred to or from a peripheral that is many times slower than the CPU itself. Consider a computer processing large amounts of data, portions of which are to be output to a printer. When the peripheral is ready for data, it

signals the CPU through a program interrupt. When the CPU acknowledges the interrupt, it completes the current instruction being executed in the main program and then automatically branches to a routine that will output the next data byte. After the byte is output to the printer, the CPU returns to where it left off in the main program. The 8080 is capable of handling up to eight interrupts from eight I/O devices using a special instruction of its instruction set. Data input is similarly handled.

Three-State Logic. There can be many peripherals connected to, and communicating along, the same bus lines. Thus, unless some form of “traffic control” is used, confusion can reign. Keeping order is the purpose of the three-state devices, shown in Fig. 4.

Simply, a three-state device can be thought of as an electronic switch connected between each bus line and its associated logic. When the switch is closed, the associated logic can accept or deliver signals to the bus. But, when the switch is open, the bus does not “see” the logic—in effect, the logic does not exist for the bus.

Programming. A program for a computer or processor consists of a sequence of operational instructions stored in memory. Each instruction enables a single elementary operation such as the movement of a data byte, an arithmetic or logical operation on a data byte, or a change in instruction execution sequence. The set of all instructions common to a given CPU is referred to as its *instruction set*. The size of the instruction set is a measure of the CPU’s capabilities. Another such measure is the length of the binary words the CPU can work with. Generally speaking, the larger the instruction set, or word size, the more powerful the CPU. The 8080 (an 8-bit CPU with 72 instructions) is thus more powerful than the 4040 (a 4-bit CPU with 60 instructions). Some microprocessor instruction sets may approach 200 instructions in length.

A program is stored in memory (RAM or ROM) as a sequence of bytes that represent the instructions. The memory address of the next instruction to be executed is held in an internal register of the CPU called the *Program Counter*. Early in the execution phase of each instruction, the program counter is automatically advanced to the address of the next sequential instruction in memory. Thus, program execution proceeds sequentially (i.e. memory location 213 is executed after location 212 is executed, etc.) unless a transfer-of-control, or **BRANCH** instruction (8080 **JUMP**, **CALL**, or **RETURN**) is executed, which causes the program counter to be set to a specified memory address. Program execution would then continue sequentially from this new memory location. The **JUMP** instruction specifies the address to be jumped to, which can be anywhere in memory. During execution of a **JUMP**, the CPU replaces the contents of the Program Counter with the address contained in the **JUMP** instruction.

Subroutines. A special type of jump occurs when the stored program **CALLS**, or accesses, a subroutine (a program within a program). Usually, a subroutine is a set of instructions that must be executed repeatedly in the course of running the main program. Algorithms that calculate mathematical functions and routines to input or output data to a peripheral device are often programmed as subroutines. The subroutine type of

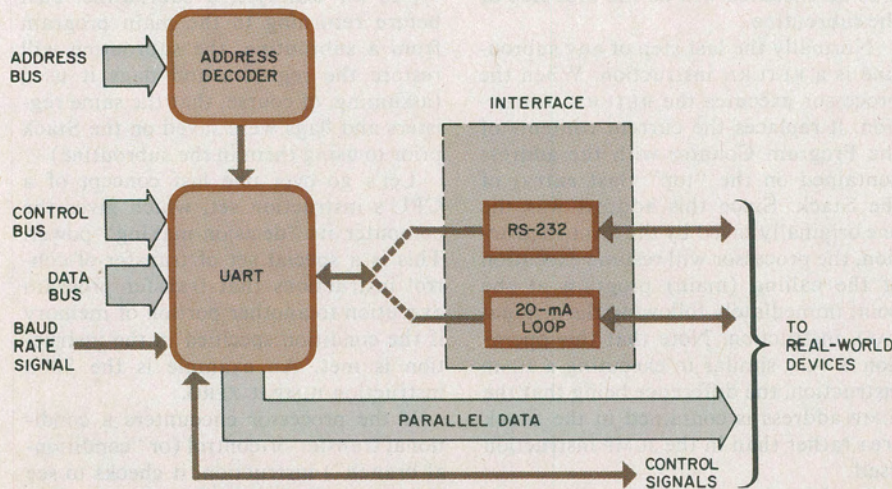


Fig. 3. Each peripheral communicates with the CPU through an associated I/O port and data bus.

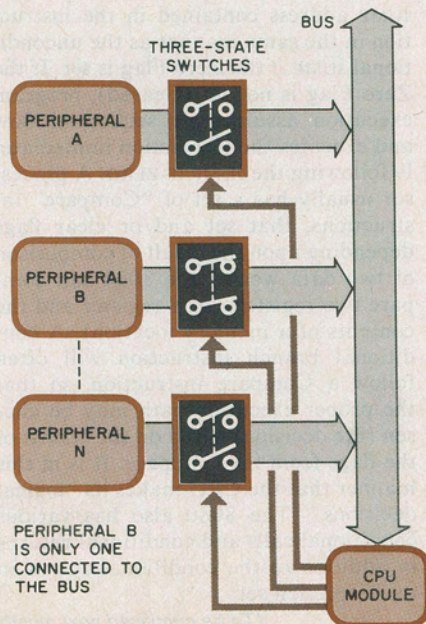


Fig. 4. A three-state device is an electronic switch connected between each bus line and associated logic.

jump requires the CPU to store the contents of the program counter at the time the jump occurs (when the CALL instruction is executed). This enables the processor to resume execution of the main program after the last instruction of the subroutine has been executed.

The processor has a special method of handling subroutines to insure an orderly return to the main program. When the CPU receives a CALL instruction from memory, it advances the Program Counter to the address of the next sequential instruction, and saves the Counter's contents in a special memory area known as the *Stack*. The latter holds the memory address of the instruction to be executed after the subroutine is completed. The processor then loads the address specified in the CALL instruction into its Program Counter. Consequently, the next instruction that is to be executed will be the first step of the subroutine.

Normally the last step of any subroutine is a RETURN instruction. When the processor executes the RETURN instruction, it replaces the current contents of the Program Counter with the address contained on the "top" (last entry) of the Stack. Since this address was the one originally saved by the CALL instruction, the processor will resume execution of the calling (main) program at the point immediately following the original CALL instruction. Note that this operation is very similar to executing a JUMP instruction, the difference being that the JUMP address is contained in the Stack area rather than in the JUMP instruction itself.

A subroutine may CALL another subroutine. This is called "nesting subroutines." If the microprocessor being used has a Stack for storing RETURN addresses, the maximum depth of nesting subroutines is determined solely by the depth of the Stack itself. So if the Stack has space for saving five return addresses, then five levels of subroutines can be accommodated.

Microprocessors have different methods of maintaining their Stack. Some store the RETURN addresses within registers in the processor, but this limits the levels of subroutine nesting. Others, such as the 8080, use a reserved area of RAM for the Stack and maintain a *Stack Pointer* (an internal register of the CPU) which contains the address of the most recent Stack entry; *i.e.*, the Stack Pointer always "points" to the top of the Stack. This type of Stack may be looked upon as a last-in-first-out (LIFO) memory, and allows virtually unlimited subroutine nesting.

Flags. The CPU has a set of flags, or internal flip-flops that are set or cleared (*i.e.*, set to a logic 1 or 0, respectively)

depending upon the results of certain instructions as they are executed. Two flags of the 8080 are: The "Zero Flag," which is set if the accumulator is 0 (actually 00000000 binary), and the "Carry Flag," which may be set when an arithmetic instruction causes the accumulator to overflow (*i.e.*, carry or borrow from an addition or subtraction). In most microprocessors there are other flags besides these. The 8080 has a total of five.

Most processors have instructions available that will store the accumulator and other general-purpose registers and flags on the Stack temporarily. Likewise, there are instructions available to reload the general-purpose registers and flags with data contained on the top of the Stack. This allows the contents of the registers and flags to be saved so that they may be used in another activity, as for example, a subroutine. Just before returning to the main program from a subroutine, the subroutine will restore the registers and flags it used (assuming, of course, that the same registers and flags were saved on the Stack prior to using them in the subroutine).

Let's go over one last concept of a CPU's instruction set, which gives the computer its "decision-making" power. This is a special set of transfer-of-control instructions that transfer program execution to another portion of memory if the condition specified in the instruction is met. An example is the 8080 instruction JUMP-IF-ZERO.

If the processor encounters a conditional transfer-of-control (or "conditional branch") instruction, it checks to see if the specified condition is met. The "condition" is always related to one of the flags. In the case of JUMP-IF-ZERO, program execution is transferred to the JUMP address contained in the instruction in the same manner as the unconditional JUMP if the Zero Flag is set. If the Zero Flag is not set (cleared), program execution assumes its sequential flow and executes the instruction immediately following the JUMP-IF-ZERO. A processor usually has a set of "Compare" instructions, that set and/or clear flags depending upon the result of comparison of two data words (the 8080 can compare two registers, or a register and the contents of a memory location). A conditional branch instruction will often follow a Compare instruction, so that the proper execution path may be chosen (the decision) based on the results of the flags from the Compare. It is in this manner that the CPU makes its "logical decisions." The 8080 also has various conditional calls and conditional returns in addition to the conditional JUMPS in its instruction set. ◇

(To be continued next month)