CHAPTER
SEVEN
SYSTEMS AND SYSTEM COMPONENTS

When we looked at the traffic light controller, the bank teller, and the tram controller examples in previous chapters, we were concerned with describing an algorithm for operation of a small system—the algorithm, in fact, that described the *systematic behavior* of the example in each case.

Systems are all about us today—some quite simple and many very complex. Many (e.g., a bank teller) were designed by one or a few people. Such a simple system is relatively easy to describe. Proper operation of such a system is easily tested. Other systems are "designed" by a much broader "team" over a longer time period. It is often very difficult to tell whether the system is operating well, and if not, what to do about it (e.g., the Federal Government or the petroleum industry).

Systems are aggregate collections of components arranged in a precise, or at least particular, order to perform a specific function. Systems may be improved by improving the performance of any component area, by rearranging component blocks, or by the addition or deletion of component blocks.

Good systems are nearly always designed by thinking about the problem requirements, organizing several alternative block diagrams of system component arrangements which may solve the problem, and then selecting the most promising block diagram.

Fundamental changes in system arrangement as well as component block selection and realization are occurring (e.g., compare a slide rule with an electronic calculator or an analog watch with a digital watch). These changes are a result of dramatic performance increases in integrated-circuit components. The performance of integrated circuits will continue to improve; so we can expect continuing changes in system design.

MODELS AND BLOCK DIAGRAMS

One important concept is the use of *models* to represent real behavior. Models are valuable for several reasons:

1. Models allow simplification of the system behavior from many variables to just a relevant few.
2. Models permit analogy to other more readily understood systems.
3. Models are changeable if found to be in error or insufficiently illustrative.

We use models to provide insight into complex situations and to simplify and idealize problems so that we can describe them in mathematical form or by a graphical plot. It is important to keep this modeling activity firmly in mind—to be constantly aware that models are, at best, shallow imitations of reality—and not to delude ourselves into equating the two.

Models can be developed in several forms. The Bohr model of the atom is a *physical model.* Newton's laws are *mathematical models.* Other models may be *graphical.* For many engineering problems, *block diagrams* are common models representing the component parts of a system. Block diagrams hide most of the details of a system, so that fundamental functions are clear.

Designing with Block Diagrams

Much effective design is carried out at the system block-diagram level. Engineers are often concerned with reducing cost or increasing performance. Therefore, system improvements are often conceptualized as a reduction in the cost/performance ratio. Usually, changes in the block diagram will cause more dramatic improvements in the cost/performance ratio than changes within a subsystem. Considerable effort should be spent optimizing the system block diagram before concentrating on the subsystems. In fact, a wrong solution may be obtained by focusing on subsystem improvement too early.

The arrangement and function of subsystems is often called the system's *architecture.* The architecture of a system is often hierarchical, as shown in Fig. 7-1. A very generalized system block diagram is shown in Fig. 7-1*a*. This block diagram may not seem very useful, and, indeed, it is seldom drawn. However, the first step of any system's design is *represented* by this diagram. That first step is the description of the inputs and outputs of the system and the general algorithm to be implemented. Many questions must be accurately answered before continuing with the system's design. Are additional outputs required? Are there sufficient inputs to calculate the required outputs? How will the outputs be calculated? Are there too many inputs? Some inputs may be redundant or completely irrelevant. Once you're satisfied that inputs, outputs, and overall system operation are correctly defined, divide the block in Fig. 7-1*a* into appropriate subsystems. For many instrumentation systems, this subdivision would look like Fig. 7-1*b*. The arrangement of this simple block diagram may also have

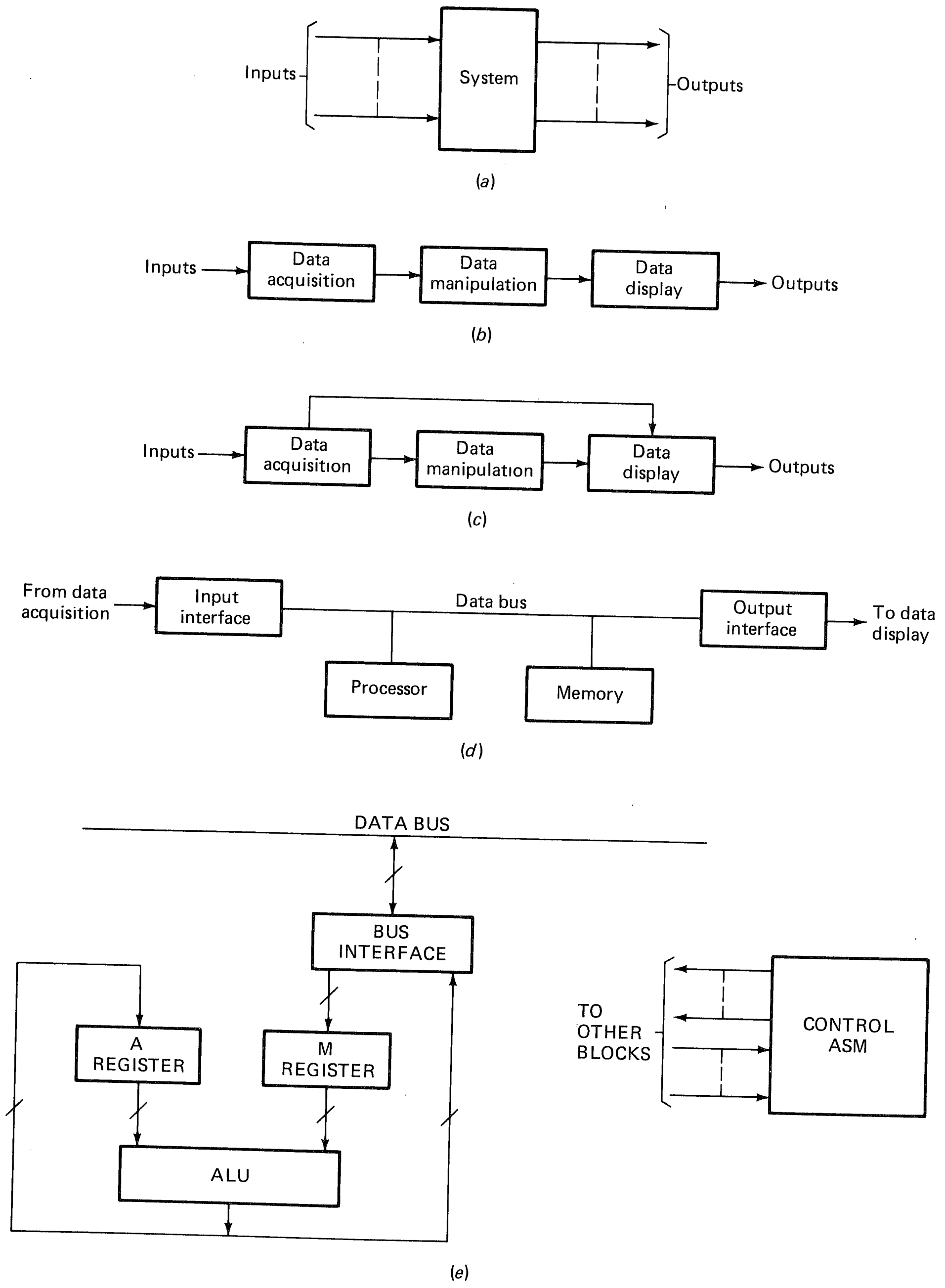

Figure 7-1 (*a*) Generalized system. (*b*) Typical instrumentation system. (*c*) Modified instrumentation system. (*d*) Data manipulation block. (*e*) Processor block.

important consequences. For example, suppose that a large amount of data needs only to be displayed and not manipulated. As drawn, the block diagram of Fig. 7-1*b* would require *all* data to travel through the data manipulation block. The data manipulation block must transfer many data that do not need manipulation from input to output subsystem. The manipulation block may have to be much higher-performance than really necessary in order to transfer all this data. Optimization of the manipulation block will not improve this situation. A change in the block diagram is necessary. If many data were not manipulated, we'd want to modify the block diagram to Fig. 7-1*c*. An additional data path has been added to allow data that does not need manipulation to be transferred directly from input to output subsystem.

Next, each of the blocks of Fig. 7-1*c* must be implemented. Often blocks are sufficiently complex to require further subdivision. The data manipulation block is subdivided in Fig. 7-1*d*. This block consists of a processing unit and memory as well as connections called *interfaces* to other blocks. An important decision we made in drawing Fig. 7-1*d* was to use a single common data bus. Only one data transfer may take place between two of the four subsystems at any one time. Not allowing simultaneous data transfers limits performance but also reduces cost, as multiple data busses would be expensive.

Finally, the processing unit of Fig. 7-1*d* is sufficiently complex to require subdivision into the subsystems of Fig. 7-1*e*. An ALU performs processing operations under the direction of an ASM. Two registers *A* and *M* allow temporary data storage in the processing unit.

It may seem that the decisions we made in this example were easy to make. In real problems the "best" arrangement of subsystems is not obvious. Considerable study and iteration may be required to develop a suitable solution.

SYSTEM DESIGN

By now, it should be obvious that system design is complex. Widely divergent choices are available to the designer. *Intuition and experience play a far greater role in the process than is generally recognized.* However, there are some steps that are beneficial to study in preparation for effective system design:

1. It is valuable to be able to describe desired functions in terms of flowcharts. These are implementation-independent algorithms.
2. It is equally valuable to be able to develop algorithm solutions and to realize them in operational circuitry. We have done this in simple ways in previous chapters.
3. It is important to be familiar with alternative architectures. The remainder of this book will give some examples of microprocessor architectures. Experience is important here. Familiarity with alternatives chosen by other designers is the first step to creativity.
4. It is important to understand major component blocks that are generally used (e.g., transducers, *A*-to-*D* converters, and arithmetic processors). These are discussed throughout this book.

5. It is important to realize that block diagrams are only models of reality. Attention must be paid to differences between idealized blocks and real circuitry. We have already discussed some realities such as fan out, delay, races, hazards, and noise.

LARGE DIGITAL SYSTEMS

As digital integrated circuits become less costly and more complex, more and more functions are added to formerly simple systems. The block diagram technique described previously will allow system architecture to be planned. The classical ASM technique, by itself, will be insufficient to implement these systems. The processing unit in Fig. 7-1*d* could have been implemented as a ROM and flip-flop ASM. Dozens of flip-flops and millions of bits of ROM would have been required. A distinguishing characteristic of this processing unit (and many others) is that it processes several bits grouped together as a *word* simultaneously. This data word might contain 4, 8, 16, or more bits of data. The implementation we postulated in Fig. 7-1*e* was oriented toward this data word. Data paths guided data words among bus interface, registers, and arithmetic logic unit. Because data and control were separated, the ASM controlling the flow of these data words was a reasonable size. This controlling ASM did not have to have all the data bus bits as inputs because the internal data paths and the ALU actually moved and manipulated the data. This controlling ASM did not need additional state flip-flops to remember data states because the *A* and *M* registers stored data states. As you might guess, the generality and performance of this processing unit were reduced by using a specific arrangement of ALU, registers, and data paths, rather than a very large classical ASM. Only those operations allowed by the configuration of data paths chosen and by the limited ALU functions available can be performed. Dozens of clock cycles may be required to perform an operation requiring one clock cycle in a classical ASM. Yet, the economic saving is so great that division of data flow and processing circuits from the controlling ASM is almost universal. Data flow and processing components are mass-produced and inexpensive. More importantly, engineers easily familiarize themselves with the limited and standardized functions of these components. This familiarity reduces the cost of designing complex systems.

The system blocks associated with data flow and processing fall into three general categories:

Processor. Processors manipulate and move data. They often contain a small amount of memory (even if only a register or two) to hold intermediate results of their manipulations. Traditionally, the control ASM has been considered part of the processing unit. The controller has more interaction with the processor than with other subsystems.

Memory. Data memory holds operands and intermediate and final results. Operands are variables used by the processor to calculate the results stored in memory.

Input-Output. A digital system must input operands from outside sources and output results of its calculations to be useful. Input-output subsystems provide connections, or *interfaces*, with people or machines not part of the digital system.

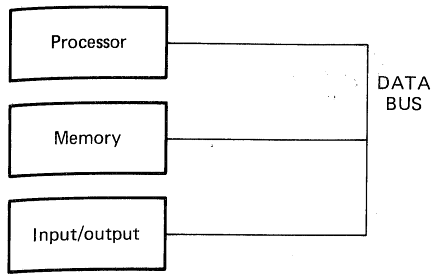

Figure 7-2 A generalized architecture.

These three subsystems are so common that we shall want to connect them together in a generalized architecture. They could be interconnected in a large variety of ways. One simple and common technique is shown in Fig. 7-2. A single common data bus allows data to move among the three subsystems. Although limited to one data transfer at a time, the economics of this configuration make it widely used for many applications for which its performance is sufficient.

The task of specifying or *programming* the ROM in the controlling ASM (part of the processor in Fig. 7-2) is time-consuming, prone to error, and expensive. An alternative is the stored-program technique.

Stored-Program Control

The ASM in the processor of Fig. 7-2 controls the operation of the entire system via a control program stored in its ROM. Since Fig. 7-2 is supposed to be a generalized architecture, it is essential to be able to change the ROM program to perform different functions for different applications. Programming the ASM's ROM is expensive and time-consuming. The *programmer* must be intimately familiar with the circuitry of this digital system. Each line of ROM contents must specify the control signals of the system in gruesome detail. A hierarchical approach called *stored-program control* has been developed to remove the necessity for intimate familiarity with the circuitry and reduce the detailed specifications required.

In a stored-program system, the ROM in the controlling ASM contains a standardized program used for all applications. The ASM ROM guides the system to read and interpret instructions from the programmer. The programmer's instructions are stored, not in the ASM's ROM, but in the memory subsystem of Fig. 7-2. Furthermore, each of these instructions initiates a complex sequence of actions in the control ASM. Now, a single instruction stored in the memory subsystem may cause a complex sequence of actions to be performed by the controlling ASM. For example, stored-program instructions specify entire data manipulations to be performed, rather than control signals to be activated. The standard program in the controlling ASM's ROM sequences all the control signals to perform the specified data manipulation. Commonly implemented instructions are data movement, arithmetic, boolean functions, and data shifting.

Again, performance has suffered for economy. The programmer is limited to a small specific set of data manipulations that are understood and interpreted by the controlling ASM. However, the digital system of Fig. 7-2 is now completely standardized with the exception of the application-dependent instructions to be stored in the memory subsystem. Because this system can be applied to solve many problems by changing the instruction sequence, it can be and is mass-produced very inexpensively. Such a system is called a *microcomputer*, and the application of microcomputers is the topic of the remainder of this book. Microcomputers are advantageous, not only because they are mass-produced and inexpensive, but also because they reduce the development cost of complex digital systems.

Microcomputer Operation

Let's examine the operation of a very simple microcomputer. Part of the microcomputer is shown in Fig. 7-3. A memory subsystem contains both a ROM for storing instructions and an RWM for storing data. The subsystems are interconnected with a single bus. This bus consists of two subbusses: the data and the address bus. Addresses are sent to the memory from the processor. Data may be sent from the processor to the memory or from the memory to the processor. Instructions are also transferred on the data bus from the memory to the processor.

This simple processor system consists of two major parts. The *A* register and the arithmetic logic unit (ALU) manipulate data transferred between processor and memory via the data bus. The rest of the processor sequences the operation of the instructions stored in the memory system's ROM. The *PC* or program counter register provides the address of the instruction to be performed. The *IR* or instruction register holds the instruction after it is extracted from memory. The control ASM sequences all the control signals needed to perform these operations and those signals needed to perform the data operations specified by the instruction in the *IR*. The control ASM also provides memory control signals. An I/O subsystem, if it had been shown, would have been connected similarly to memory with I/O control signals from the processor's control ASM.

Let's trace the operation of this microcomputer by using the partial ASM chart of Fig. 7-4. When power is first applied, initialization circuitry starts the ASM in state *INIT.* State *INIT* has output *HCLPC*, which clears the program counter to 000H. We have stored the first instruction to be performed at address 000H in the ROM. The ASM sequences to state *FETCH.* The *PC* is sent to the memory via the address bus (*IHPCON*). A 16-bit instruction, located at address 000H, is put on the data bus (*ILRD*), and this instruction is stored in the instruction register (*HIRLD*). The instruction fetch cycle simply retrieves the instruction specified by the program counter from memory.

Instructions in our simple computer look like Fig. 7-5. The most significant 4 bits of the instruction are called the operation code or opcode and specify 1 of 16 possible operations to be performed. The least significant 12 bits contain the address of

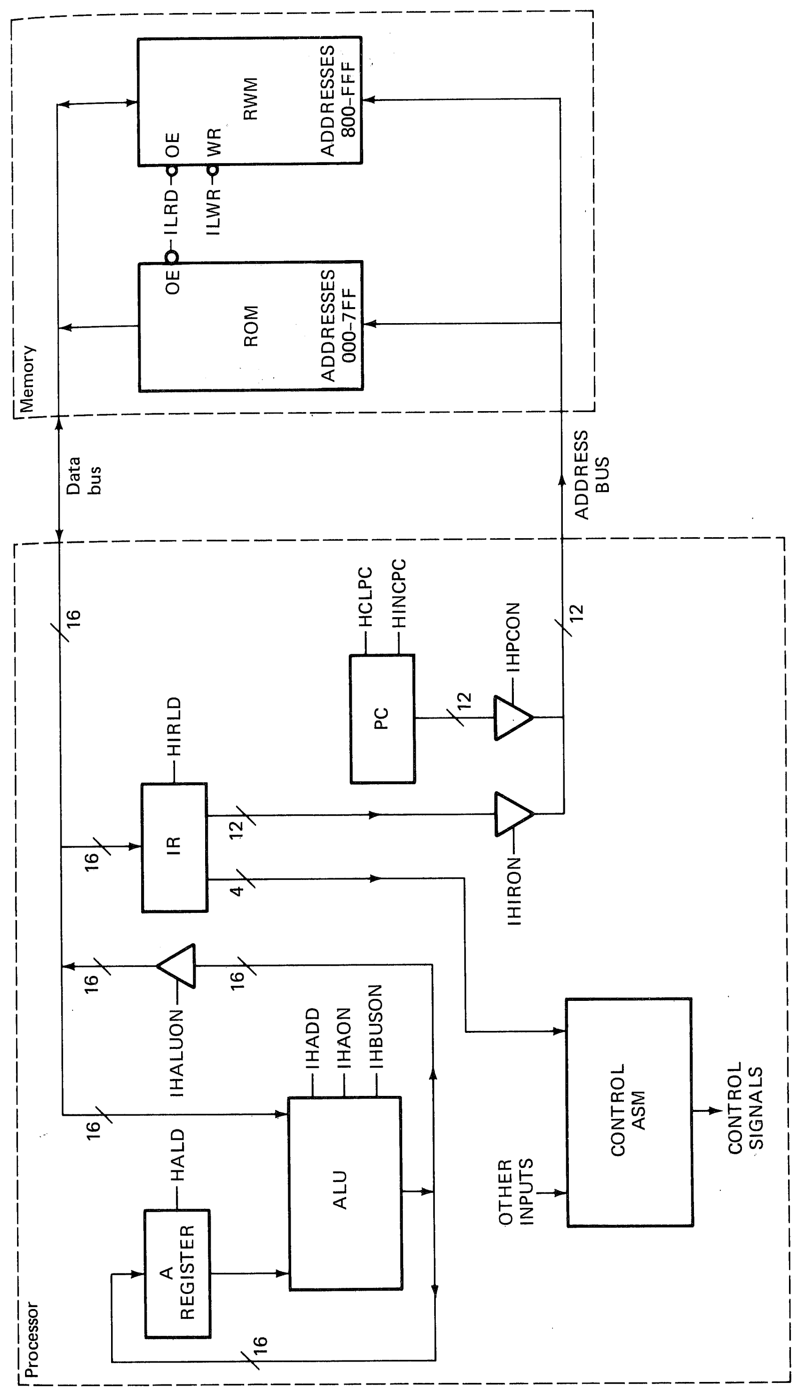

Figure 7-3 Block diagram of a simplified microcomputer.

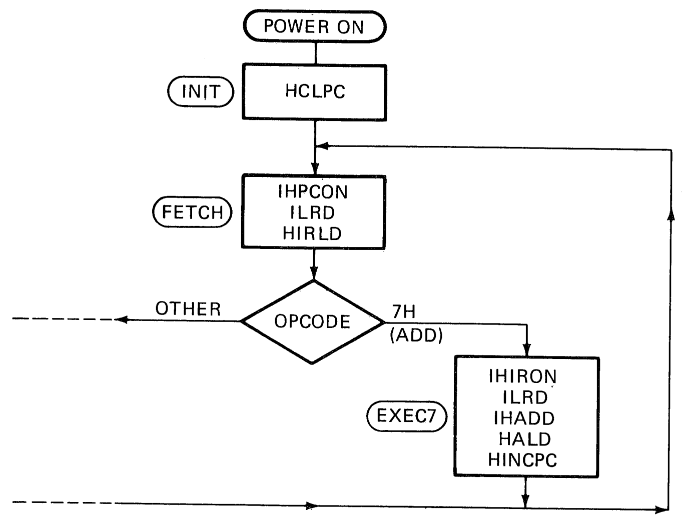

Figure 7-4 Part of the ASM chart for the computer of Fig. 7-3.

a data operand in memory. In this example we've fetched the instruction 79FAH from memory. Operation code 7H means add the contents of a memory location to the *A* register. The address of the datum to be added is 9FAH.

Return to the ASM chart and block diagram to notice that the most significant 4 bits of the *IR* are inputs to the ASM. Since, in our example, these bits are 7H, the ASM chart takes the branch to state *EXEC7.* This state executes the required data operation. The least significant 12 bits of the *IR* are sent to the address bus (*IHIRON*). The datum stored in RWM at address 9FAH is put on the data bus (*ILRD*). This datum becomes one input to the ALU. The memory datum and the *A* register are summed (*IHADD*) and the result stored in the *A* register (*HALD*). Independently, the program counter is incremented (*HINCPC*). After each instruction is executed, another is fetched from memory. Now, when the ASM returns to the *FETCH* state, the *PC* contains 001H; so the next instruction, stored in memory at address 001H, is fetched to be executed. You can see that a fairly complex ASM sequence is necessary, even in this simple computer, to fetch and execute instructions.

Fortunately, the programmer need know nothing about the details of this internal ASM sequence. The designer of the instruction sequence need only remember the overall characteristics of each instruction. For example, he or she must know that opcode 7H means add a memory datum to the *A* register. Even the engineer designing the memory and I/O systems for this computer must only know enough of the computer's internal operation to understand the sequencing of the external memory and

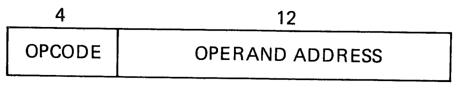

Figure 7-5 Instruction format for the computer of Fig. 7-3.

I/O control signals. Our interest in the rest of this book will be in the external and macroscopic description of microcomputers. We will be more interested in using microcomputers than in designing the individual processor circuits.

PROBLEMS

7-1 Because of its low cost, engineers often try to use the microcomputer of Fig. 7-2 in systems in which its performance would ordinarily be insufficient. Changes are made in its architecture to increase its performance. A common change is to allow data transfers between memory and input-output *simultaneously* with transfers between processor and memory. Modify the block diagram of Fig. 7-2 to permit such simultaneous transfers. Make as few changes as possible.

7-2 Figure P7-1 shows a simple but large automated railroad. Trains stop at 200 stations along the line and may pass each other at passing sidings. Two alternative architectures are to be examined: one large single controller and many smaller controllers. Draw block diagrams of each architecture and discuss merits and drawbacks of each.

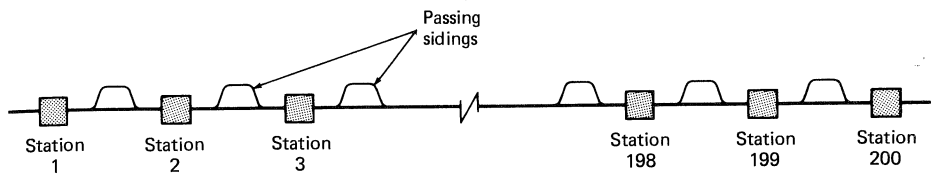

Figure P7-1 Railroad.

7-3 Many times, a single processing unit will not be sufficiently fast for a particular application. An alternative to a single fast processor is several slower processors. One such multiple processor architecture, called a pipeline, is shown in Fig. P7-2. Each processor performs only a part of the data manipulation and transfers the partially processed data to the next processor in the pipe. Assume that a single processor with the same performance as one of the three in the diagram could perform the *entire* data manipulation in time *T*. Thus, a new input datum could be processed at intervals of *T* seconds. How should the data manipulation task be divided among the three processors in the pipeline for maximum performance? How often could a new datum be processed by the pipeline processor? How long does it take to process a single datum from input to output? What would happen if the task could not be subdivided among the processors optimally?

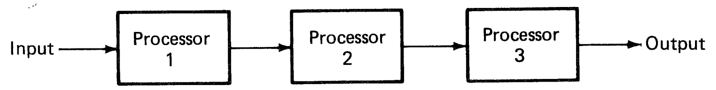

Figure P7-2 Pipeline processor.

7-4 Figure P7-3 is one of many architectures of parallel processors. Each processor implements an entire data manipulation in time *T*, but each manipulates only every third datum. Thus, processor 1 manipulates data input at times 1, 4, 7, 10, etc.; processor 2 at times 2, 5, 8, 11, etc.; processor 3 at times 3, 6, 9, 12, etc. Compare the performance of this parallel processor with the pipeline

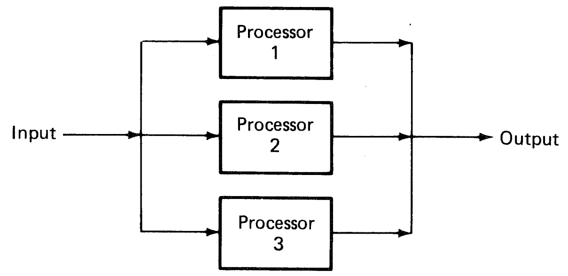

Figure P7-3 Parallel processor.

processor of Fig. P7-2. Are there algorithms for which one or the other architecture may not work? Explain.

7-5 A digital system must be designed to independently control the temperatures of 100 vats of colored dyes. Each vat has a temperature sensor, which will be an input to the digital system, and a heater controlled by an output of the digital system. A single processor has sufficient speed to control the temperature of up to 10 vats. What multiple-processor architecture would you use for this temperature controller and why? You are *not* constrained to use only the architectures previously discussed.

7-6 Consider the microcomputer of Fig. 7-3. Expand the ASM chart of Fig. 7-4 to perform the following functions.

(*a*) Implement a store the *A* register into memory instruction. This instruction has opcode 3H.

(*b*) Change the ASM chart so that even and odd opcodes perform identical functions except for addressing. Opcodes 6H and 7H would both add the contents of a memory location to the *A* register, while opcodes 2H and 3H would both store the datum from the *A* register into memory. The odd opcodes would address memory like the example in the text. The even opcodes assume that the address in the instruction is actually the address of a memory location which contains the address of the operand. Signal *HIRLD* still loads an entire 16-bit instruction into the *IR*. You can use a new signal *HIRALD*, which loads only the 12 address bits into the *IR* without changing the 4 most significant bits.

(*c*) So far, we have no way to implement a decision with our simple microcomputer. Normally, the *PC* is incremented after each instruction. A decision is implemented by changing the *PC* to the address contained in the decision instruction if a condition is true. Implement an instruction that changes the contents of the *PC* to the address contained in the instruction only if the *A* register is not 0. If the *A* register is 0, increment the *PC* as usual. You can use a signal named *YZERO*, which is 1 if the output of the ALU is 0. A signal named *HPCLD* allows you to load the program counter, rather than just incrementing it. Besides designing the ASM chart, show any changes required in the block diagram.

CHAPTER

EIGHT

MICROCOMPUTERS AND PROGRAMMING

The word microcomputer implies a very small computer. Computers in which the central processing unit is fabricated as a single integrated circuit are called microcomputers. The first microcomputers were physically small, had little computation ability, had limited amounts of memory, and had few interfaces to external devices. Their scope of application was also limited. Even these early microcomputers were inexpensive compared with other available computers and were widely used.

Advances in technology have drastically changed the microcomputer since the first crude devices were introduced. Computational power has been increased dramatically. Microcomputers now have large amounts of memory and connect to many external devices. Even with this added capability, microcomputers have become much less expensive than they once were.

In the process of being improved, the microcomputer has become much more difficult to identify. Important identifying characteristics are physical size, cost, and type of application. The term microcomputer is almost always reserved for computers in which the main processing unit is fabricated as a single integrated circuit or, occasionally, a few integrated circuits. Being fabricated as ICs, microcomputers are inexpensive. Even more parts of the computer, such as memory, are being added to this single integrated circuit.

Microcomputers are often dedicated to a single problem or a group of related problems. Formerly, expensive computers had to be shared among several users to justify their cost. The microcomputer's low cost makes it possible to dedicate a computer to a single task or small group of tasks that were formerly implemented with specialized ASMs. Thus, we find microcomputers in sewing machines, microwave ovens, and automobiles. More complex tasks, such as control of an industrial plant, have been partitioned into smaller tasks assigned to multiple microcomputers.