user. Programs and data must be transferred into memory and results of computations must be transferred back to the user.

The instructions listed in Table 5-2 constitute a minimum set that provides all the capabilities mentioned above. There is one arithmetic instruction, ADD, and two related instructions, complement *AC*(CMA) and increment *AC*(INC). With these three instructions we can add and subtract binary numbers when negative numbers are in signed-2's complement representation. The circulate instructions, CIR and CIL, can be used for arithmetic shifts as well as any other type of shifts desired. Multiplication and division can be performed using addition, subtraction, and shifting. There are three logic operations: AND, complement *AC*(CMA), and clear *AC*(CLA). The AND and complement provide a NAND operation. It can be shown that with the NAND operation it is possible to implement all the other logic operations with two variables (listed in Table 4-6). Moving information from memory to *AC* is accomplished with the load *AC*(LDA) instruction. Storing information from *AC* into memory is done with the store *AC*(STA) instruction. The branch instructions BUN, BSA, and ISZ, together with the four skip instructions, provide capabilities for program control and checking of status conditions. The input (INP) and output (OUT) instructions cause information to be transferred between the computer and external devices.

Although the set of instructions for the basic computer is complete, it is not efficient because frequently used operations are not performed rapidly. An efficient set of instructions will include such instructions as subtract, multiply, OR, and exclusive-OR. These operations must be programmed in the basic computer. The programs are presented in Chap. 6 together with other programming examples for the basic computer. By using a limited number of instructions it is possible to show the detailed logic design of the computer. A more complete set of instructions would have made the design too complex. In this way we can demonstrate the basic principles of computer organization and design without going into excessive complex details. In Chap. 8 we present a complete list of computer instructions that are included in most commercial computers.

The function of each instruction listed in Table 5-2 and the microoperations needed for their execution are presented in Secs. 5-5 through 5-7 We delay this discussion because we must first consider the control unit and understand its internal organization.

5-4 Timing and Control

clock pulses

The timing for all registers in the basic computer is controlled by a master clock generator. The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit. The clock pulses do not change the state of a register unless the register is enabled by a control signal. The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator.

hardwired control

There are two major types of control organization: hardwired control and microprogrammed control. In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the advantage that it can be optimized to produce a fast mode of operation. In the microprogrammed organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations. A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed. In the microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory. A hardwired control for the basic computer is presented in this section. A microprogrammed control unit for a similar computer is presented in Chap. 7.

microprogrammed control

control unit

The block diagram of the control unit is shown in Fig. 5-6. It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the instruction register (IR). The position of this register in the common bus system is indicated in Fig. 5-4. The instruction register is shown again in Fig. 5-6, where it is divided into three parts: the I bit, the operation code, and bits 0 through 11. The operation code in bits 12 through 14 are decoded with a 3×8 decoder. The eight outputs of the decoder are designated by the symbols D_0 through D_7. The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}. The internal logic of the control gates will be derived later when we consider the design of the computer in detail.

timing signals

The sequence counter SC can be incremented or cleared synchronously (see the counter of Fig. 2-11). Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4×16 decoder. Once in awhile, the counter is cleared to 0, causing the next active timing signal to be T_0. As an example, consider the case where SC is incremented to provide timing signals T_0, T_1, T_2, T_3, and T_4 in sequence. At time T_4, SC is cleared to 0 if decoder output D_3 is active. This is expressed symbolically by the statement

$$D_3T_4: \quad SC \leftarrow 0$$

The timing diagram of Fig. 5-7 shows the time relationship of the control signals. The sequence counter SC responds to the positive transition of the clock. Initially, the CLR input of SC is active. The first positive transition of the clock

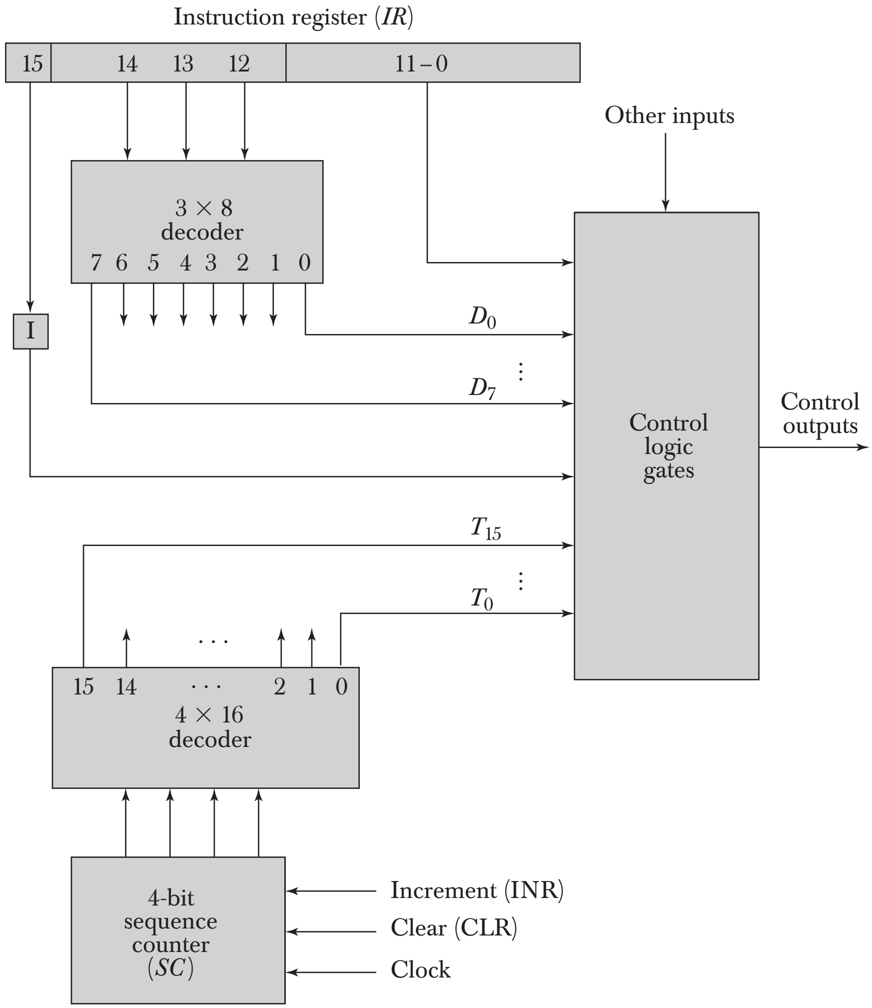

Figure 5-6 Control unit of basic computer.

clears SC to 0, which in turn activates the timing signal T_0 out of the decoder. T_0 is active during one clock cycle. The positive clock transition labeled T_0 in the diagram will trigger only those registers whose control inputs are connected to timing signal T_0. SC is incremented with every positive clock transition, unless its CLR input is active. This produces the sequence of timing signals T_0, T_1, T_2, T_3, T_4, and so on, as shown in the diagram. (Note the relationship between the timing signal and its corresponding positive clock transition.) If SC is not cleared, the timing signals will continue with T_5, T_6, up to T_{15} and back to T_0.

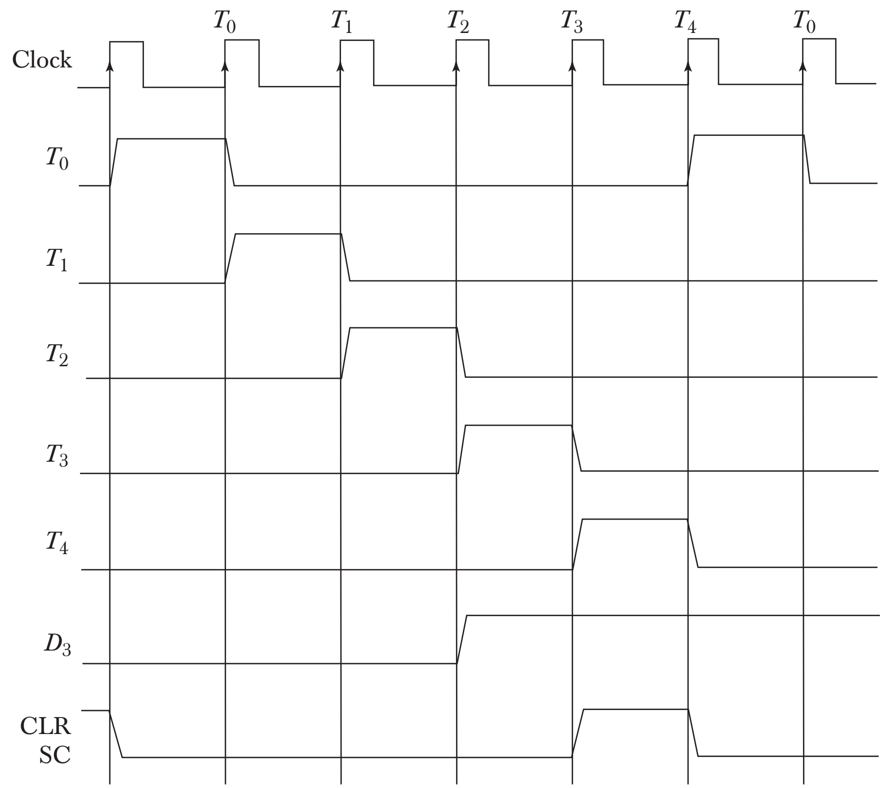

Figure 5-7 Example of control timing signals.

The last three waveforms in Fig. 5-7 show how SC is cleared when $D_3T_4 = 1$. Output D_3 from the operation decoder becomes active at the end of timing signal T_2. When timing signal T_4 becomes active, the output of the AND gate that implements the control function D_3T_4 becomes active. This signal is applied to the CLR input of SC. On the next positive clock transition (the one marked T_4 in the diagram) the counter is cleared to 0. This causes the timing signal T_0 to become active instead of T_5 that would have been active if SC were incremented instead of cleared.

A memory read or write cycle will be initiated with the rising edge of a timing signal. It will be assumed that a memory cycle time is less than the clock cycle time. According to this assumption, a memory read or write cycle initiated by a timing signal will be completed by the time the next clock goes through its positive transition. The clock transition will then be used to load the memory word into a register. This timing relationship is not valid in many computers because the memory cycle time is usually longer than the processor clock cycle. In such a case it is necessary to provide wait cycles in the