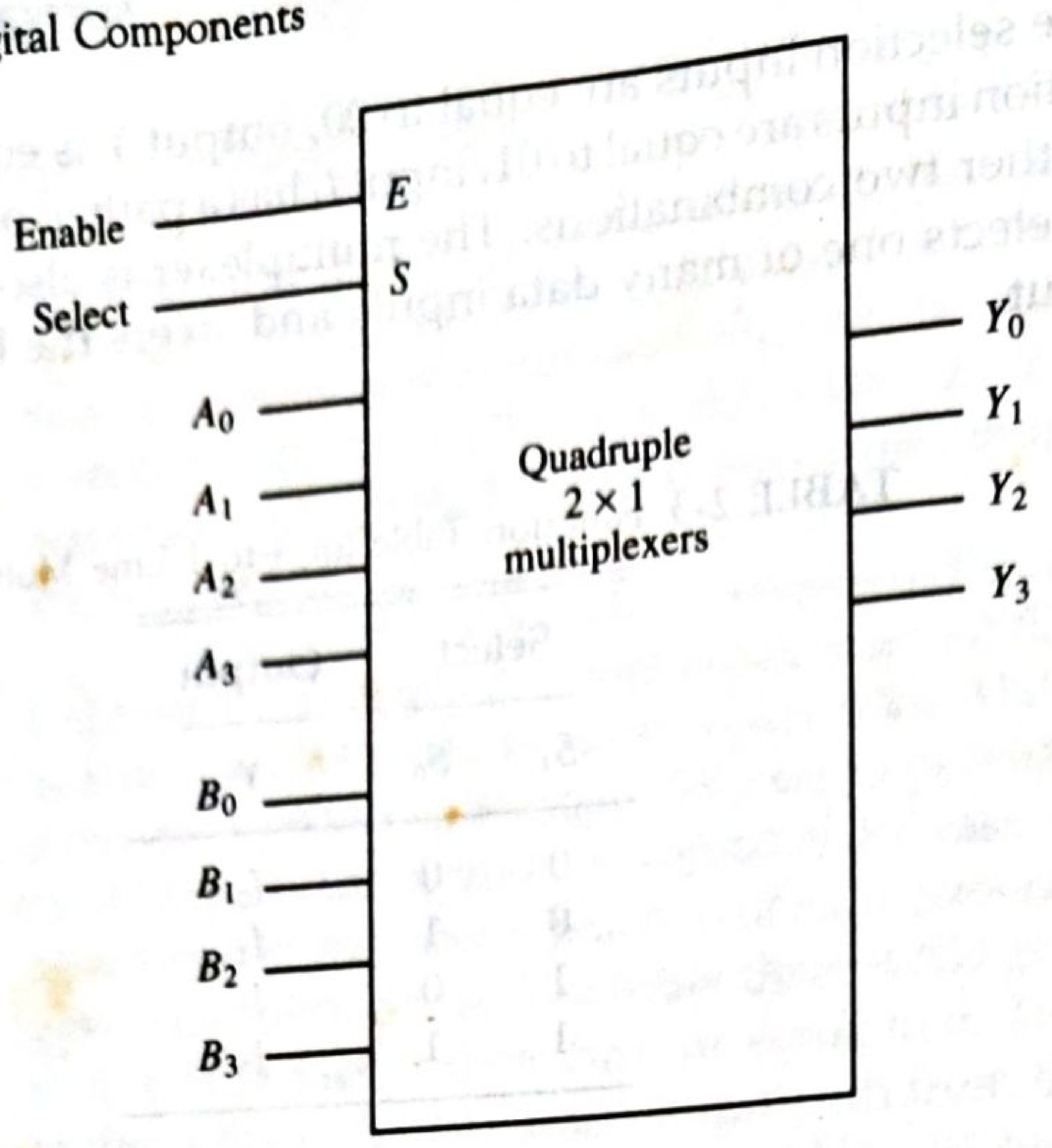


(a) Block diagram

E	S	Y
0	×	All 0's
1	0	A
1	1	B

(b) Function table

Figure 2-5 Quadruple 2-to-1 line multiplexers.

2-4 Registers

A register is a group of flip-flops with each flip-flop capable of storing one bit of information. An n-bit register has a group of n flip-flops and is capable of storing any binary information of n bits. In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks. In its broadest definition, a register consists of a group of flip-flops and gates that effect their transition. The flip-flops hold the binary information and the gates control when and how new information is transferred into the register.

Various types of registers are available commercially. The simplest register is one that consists only of flip-flops, with no external gates. Figure 2-6 shows such a register constructed with four D flip-flops. The common clock input triggers all flip-flops on the rising edge of each pulse, and the binary data available at the four inputs are transferred into the 4-bit register. The four outputs can be sampled at any time to obtain the binary information stored in the register. The *clear* input goes to a special terminal in each flip-flop. When this input goes to 0, all flip-flops are reset asynchronously. The clear input is useful for clearing the register to all 0's prior to its clocked operation. The clear input must be maintained at logic 1 during normal clocked operation. Note that the clock signal enables the D input but that the clear input is independent of the clock.

register load

The transfer of new information into a register is referred to as *loading* the register. If all the bits of the register are loaded simultaneously with a common

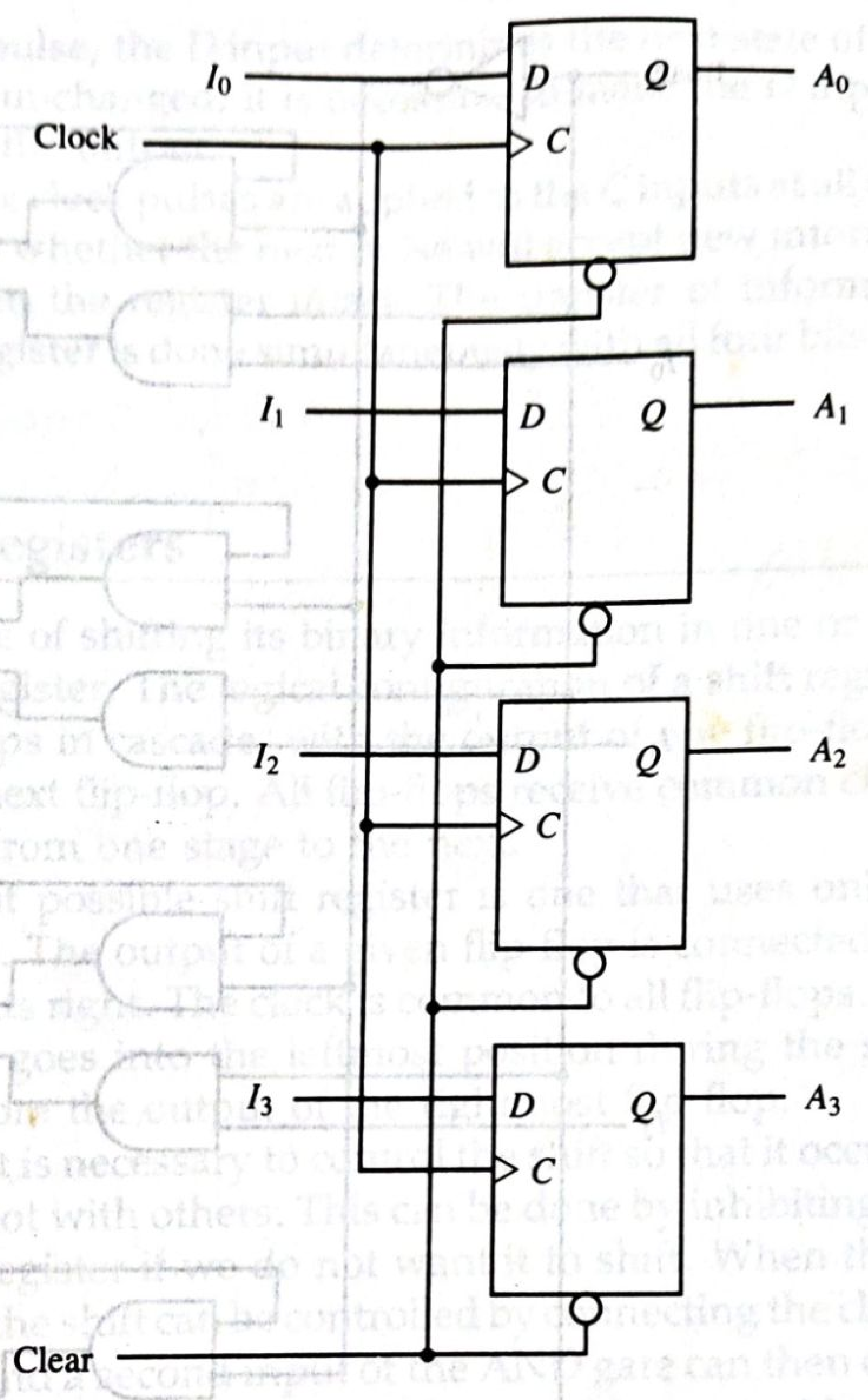


Figure 2-6 4-bit register.

clock pulse transition, we say that the loading is done in parallel. A clock transition applied to the C inputs of the register of Fig. 2-6 will load all four inputs I_0 through I_3 in parallel. In this configuration, the clock must be inhibited from the circuit if the content of the register must be left unchanged.

Register with Parallel Load

Most digital systems have a master clock generator that supplies a continuous train of clock pulses. The clock pulses are applied to all flip-flops and registers in the system. The master clock acts like a pump that supplies a constant beat to all parts of the system. A separate control signal must be used to decide which specific clock pulse will have an effect on a particular register.

A 4-bit register with a load control input that is directed through gates and into the D inputs is shown in Fig. 2-7. The C inputs receive clock pulses at all times. The buffer gate in the clock input reduces the power requirement

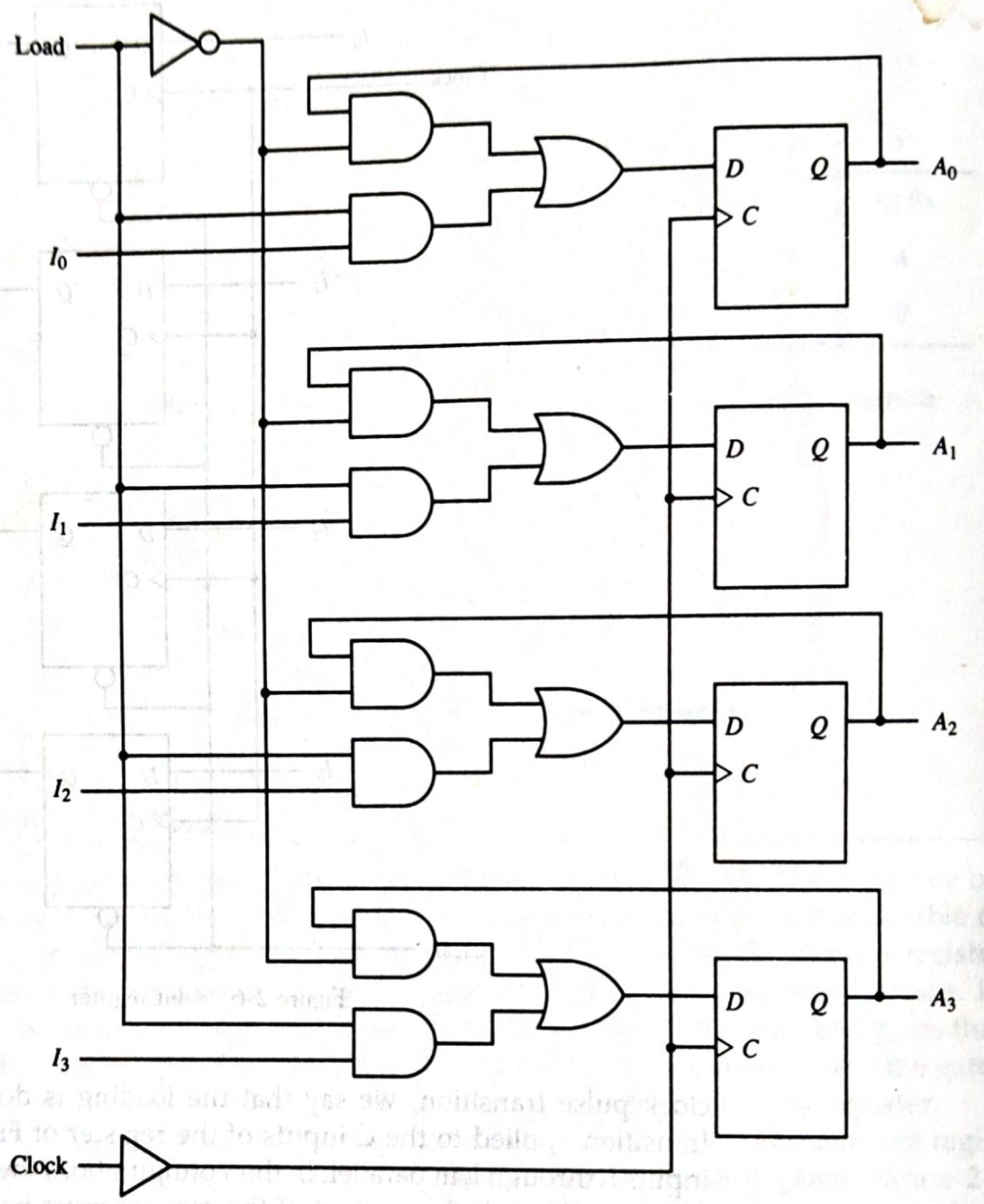


Figure 2-7 4-bit register with parallel load.

from the clock generator. Less power is required when the clock is connected to only one input gate instead of the power consumption that four inputs would have required if the buffer were not used.

load input The load input in the register determines the action to be taken with each clock pulse. When the load input is 1, the data in the four inputs are transferred into the register with the next positive transition of a clock pulse. When the load input is 0, the data inputs are inhibited and the *D* inputs of the flip-flops are connected to their outputs. The feedback connection from output to input is necessary because the *D* flip-flop does not have a "no change" condition.

With each clock pulse, the *D* input determines the next state of the output. To leave the output unchanged, it is necessary to make the *D* input equal to the present value of the output.

Note that the clock pulses are applied to the *C* inputs at all times. The load input determines whether the next pulse will accept new information or leave the information in the register intact. The transfer of information from the inputs into the register is done simultaneously with all four bits during a single pulse transition.

2-5 Shift Registers

A register capable of shifting its binary information in one or both directions is called a shift register. The logical configuration of a shift register consists of a chain of flip-flops in cascade, with the output of one flip-flop connected to the input of the next flip-flop. All flip-flops receive common clock pulses that initiate the shift from one stage to the next.

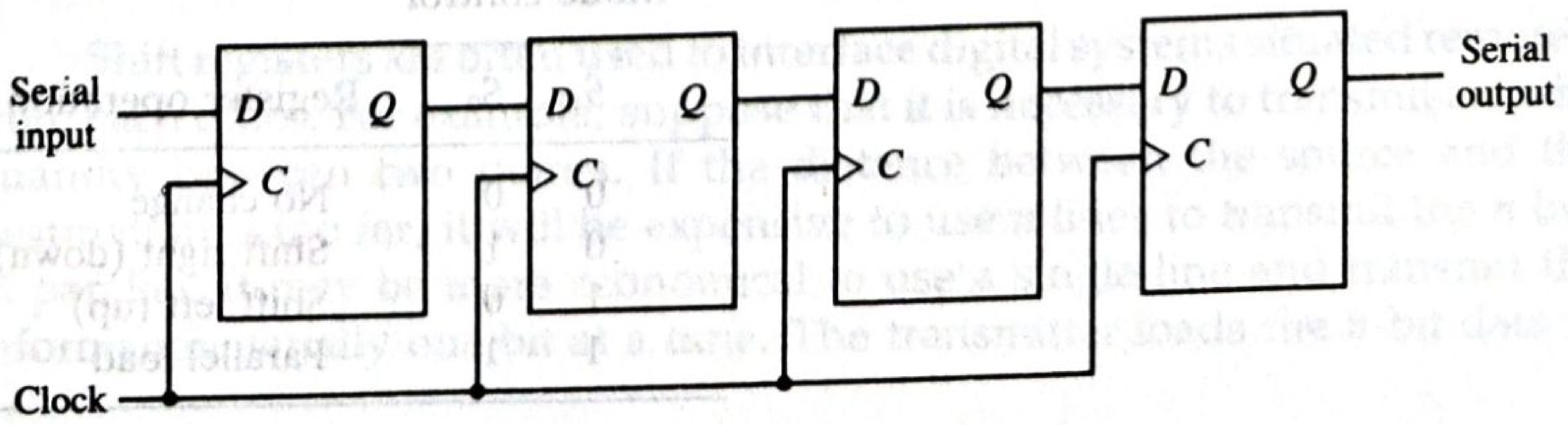
serial input The simplest possible shift register is one that uses only flip-flops, as shown in Fig. 2-8. The output of a given flip-flop is connected to the *D* input of the flip-flop at its right. The clock is common to all flip-flops. The *serial input* determines what goes into the leftmost position during the shift. The *serial output* is taken from the output of the rightmost flip-flop.

Sometimes it is necessary to control the shift so that it occurs with certain clock pulses but not with others. This can be done by inhibiting the clock from the input of the register if we do not want it to shift. When the shift register of Fig 2-8 is used, the shift can be controlled by connecting the clock to the input of an AND gate, and a second input of the AND gate can then control the shift by inhibiting the clock. However, it is also possible to provide extra circuits to control the shift operation through the *D* inputs of the flip-flops rather than the clock input.

Bidirectional Shift Register with Parallel Load

A register capable of shifting in one direction only is called a unidirectional shift register. A register that can shift in both directions is called a bidirectional shift register. Some shift registers provide the necessary input and output terminals

Figure 2-8 4-bit shift register.



until the memory word is available. To facilitate the presentation, we will assume that a wait period is not necessary in the basic computer.

To fully comprehend the operation of the computer, it is crucial that one understands the timing relationship between the clock transition and the timing signals. For example, the register transfer statement

$$T_0: AR \leftarrow PC$$

specifies a transfer of the content of PC into AR if timing signal T_0 is active. T_0 is active during an entire clock cycle interval. During this time the content of PC is placed onto the bus (with $S_2S_1S_0 = 010$) and the LD (load) input of AR is enabled. The actual transfer does not occur until the end of the clock cycle when the clock goes through a positive transition. This same positive clock transition increments the sequence counter SC from 0000 to 0001. The next clock cycle has T_1 active and T_0 inactive.

5-5 Instruction Cycle

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of subcycles or phases. In the basic computer each instruction cycle consists of the following phases:

1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

Fetch and Decode

Initially, the program counter PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal T_0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T_0, T_1, T_2, and so on. The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
T_2: $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$

Since only *AR* is connected to the address inputs of memory, it is necessary to transfer the address from *PC* to *AR* during the clock transition associated with timing signal T_0. The instruction read from memory is then placed in the instruction register *IR* with the clock transition associated with timing

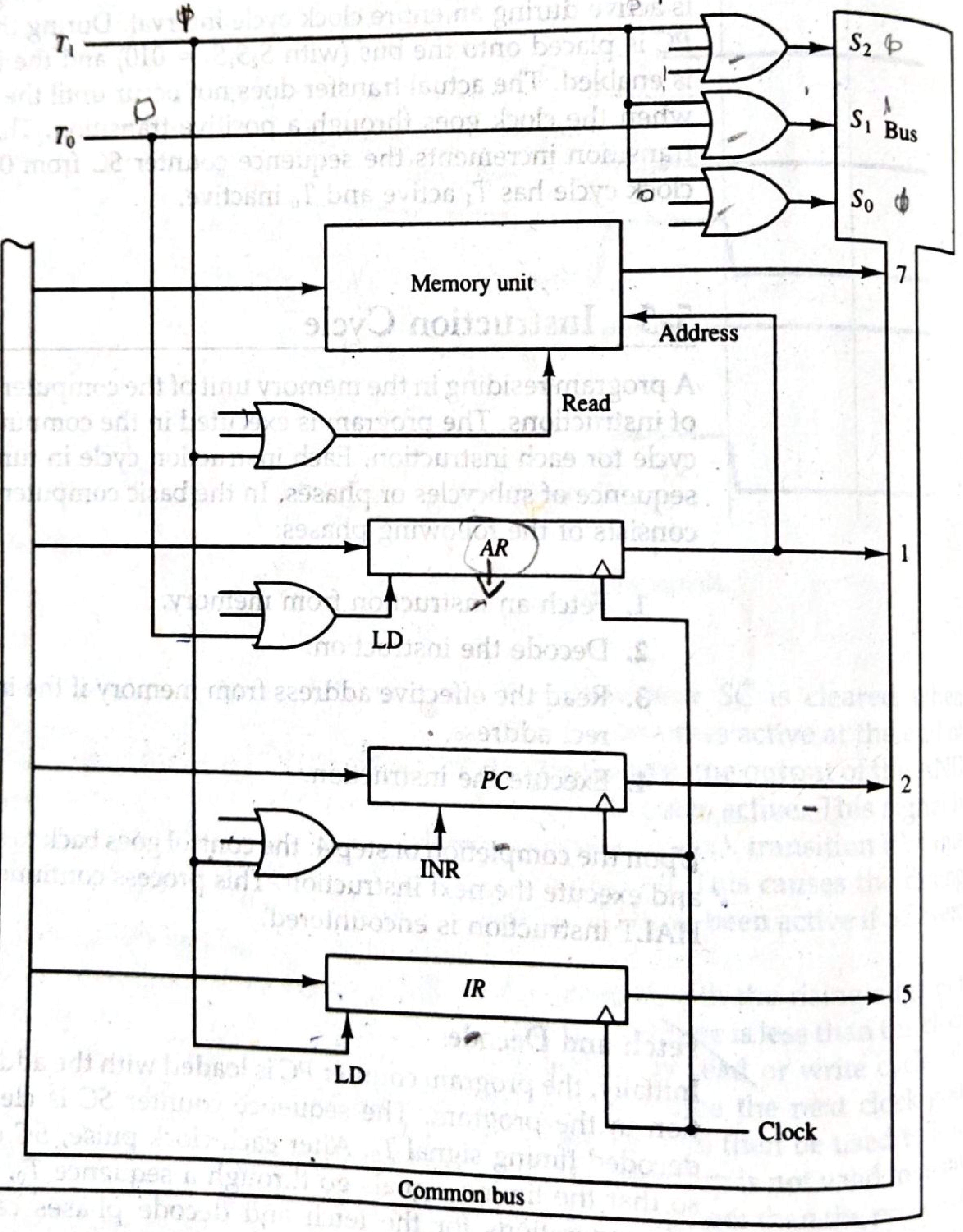


Figure 5-8 Register transfers for the fetch phase.

signal T_1. At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program. At time T_2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR. Note that SC is incremented after each clock pulse to produce the sequence T_0, T_1, and T_2.

Figure 5-8 shows how the first two register transfer statements are implemented in the bus system. To provide the data path for the transfer of PC to AR we must apply timing signal T_0 to achieve the following connection:

1. Place the content of PC onto the bus by making the bus selection inputs $S_2S_1S_0$ equal to 010.
2. Transfer the content of the bus to AR by enabling the LD input of AR.

The next clock transition initiates the transfer from PC to AR since $T_0 = 1$. In order to implement the second statement

$$T_1: \quad IR \leftarrow M[AR], \quad PC \leftarrow PC + 1$$

it is necessary to use timing signal T_1 to provide the following connections in the bus system.

1. Enable the read input of memory.
2. Place the content of memory onto the bus by making $S_2S_1S_0 = 111$.
3. Transfer the content of the bus to IR by enabling the LD input of IR.
4. Increment PC by enabling the INR input of PC.

The next clock transition initiates the read and increment operations since $T_1 = 1$.

Figure 5-8 duplicates a portion of the bus system and shows how T_0 and T_1 are connected to the control inputs of the registers, the memory, and the bus selection inputs. Multiple input OR gates are included in the diagram because there are other control functions that will initiate similar operations.

Determine the Type of Instruction

The timing signal that is active after the decoding is T_3. During time T_3, the control unit determines the type of instruction that was just read from memory. The flowchart of Fig. 5-9 presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding. The three possible instruction types available in the basic computer are specified in Fig. 5-5.

Decoder output D_7 is equal to 1 if the operation code is equal to binary 111. From Fig. 5-5 we determine that if $D_7 = 1$, the instruction must be a

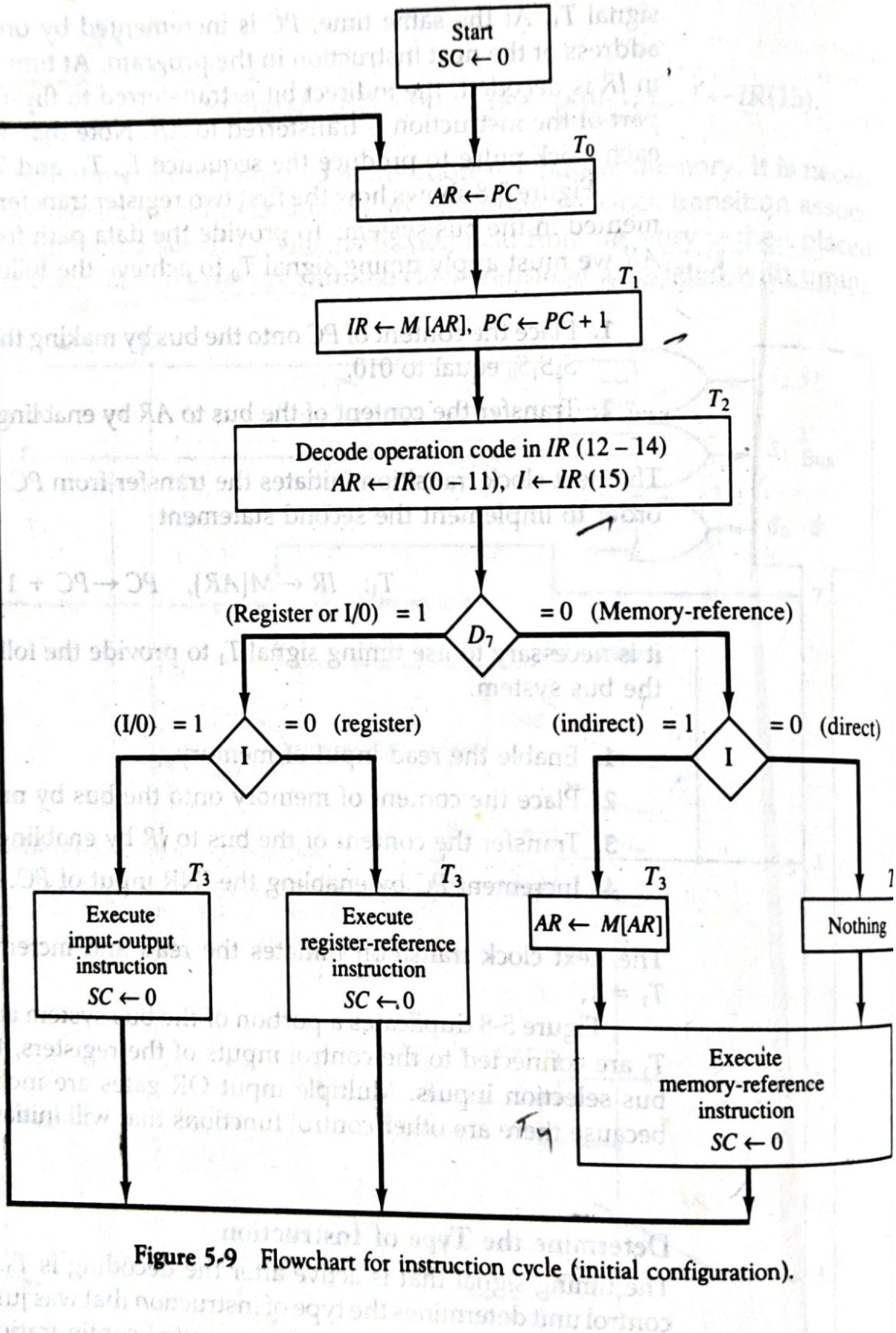

Start
SC ← 0
T0
AR ← PC
T1
IR ← M [AR], PC ← PC + 1
T2
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)
(Register or I/0) = 1
D7
= 0 (Memory-reference)
(I/0) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T3
T3
T3
Execute input-output instruction SC ← 0
Execute register-reference instruction SC ← 0
AR ← M[AR]
Nothing
Execute memory-reference instruction SC ← 0
T4

Figure 5-9 Flowchart for instruction cycle (initial configuration).

register-reference or input–output type. If $D_7 = 0$, the operation code must b one of the other seven values 000 through 110, specifying a memory-referenc instruction. Control then inspects the value of the first bit of the instructio which is now available in flip-flop I. If $D_7 = 0$ and $I = 1$, we have a memor reference instruction with an indirect address. It is then necessary to read th

indirect address

effective address from memory. The microoperation for the indirect address condition can be symbolized by the register transfer statement

$$AR \leftarrow M[AR]$$

Initially, AR holds the address part of the instruction. This address is used during the memory read operation. The word at the address given by AR is read from memory and placed on the common bus. The LD input of AR is then enabled to receive the indirect address that resided in the 12 least significant bits of the memory word.

The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T_3. This can be symbolized as follows:

001	$D_7'IT_3$:	$AR \leftarrow M[AR]$	011 01
	$D_7'I'T_3$:	Nothing	
	$D_7I'T_3$:	Execute a register-reference instruction	
	D_7IT_3:	Execute an input–output instruction	1 1 T

When a memory-reference instruction with $I = 0$ is encountered, it is not necessary to do anything since the effective address is already in AR. However, the sequence counter SC must be incremented when $D_7'T_3 = 1$, so that the execution of the memory-reference instruction can be continued with timing variable T_4. A register-reference or input–output instruction can be executed with the clock associated with timing signal T_3. After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase with $T_0 = 1$.

Note that the sequence counter SC is either incremented or cleared to 0 with every positive clock transition. We will adopt the convention that if SC is incremented, we will not write the statement $SC \leftarrow SC + 1$, but it will be implied that the control goes to the next timing signal in sequence. When SC is to be cleared, we will include the statement $SC \leftarrow 0$.

The register transfers needed for the execution of the register-reference instructions are presented in this section. The memory-reference instructions are explained in the next section. The input–output instructions are included in Sec. 5-7.

Register-Reference Instructions

Register-reference instructions are recognized by the control when $D_7 = 1$ and $I = 0$. These instructions use bits 0 through 11 of the instruction code to specify one of 12 instructions. These 12 bits are available in $IR(0–11)$. They were also transferred to AR during time T_2.

The control functions and microoperations for the register-reference in-