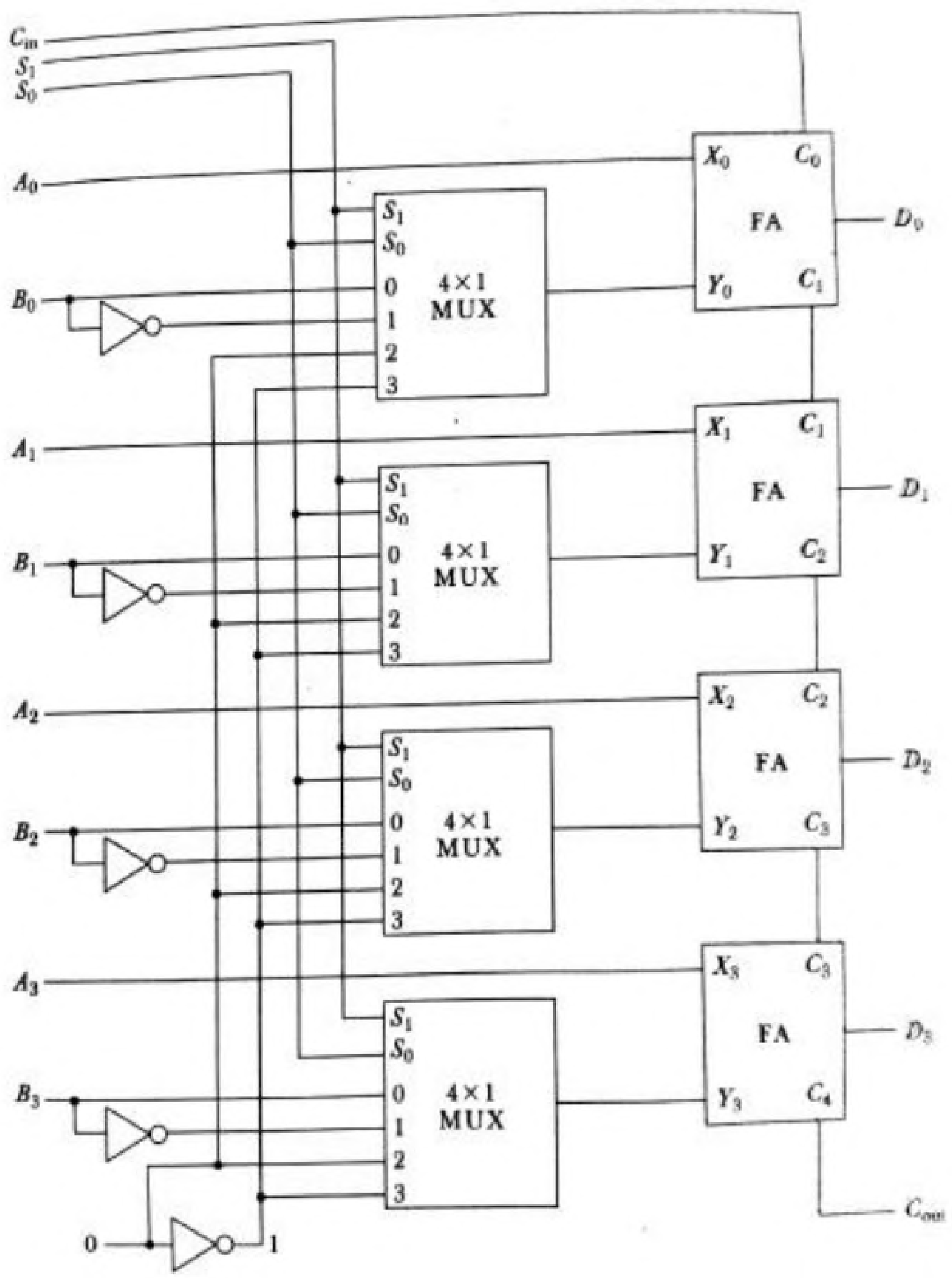
#2-1-COA

1. Draw and explain 4-bit arithmetic circuit with help of Functional table

Arithmetic Circuit

- The basic component of an arithmetic circuit is the parallel adder.
- By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.
- The diagram of a 4-bit arithmetic circuit is shown in Fig. 4-9. It has four full-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations.
- There are two 4-bit inputs A and B and a 4-bit output D.
- The four inputs from A go directly to the X inputs of the binary adder.
- Each of the four inputs from B is connected to the data inputs of the multiplexers.
- The multiplexers data inputs also receive the complement of B.
- The other two data inputs are connected to logic-0 and logic-1.
- The four multiplexers are controlled by two selection inputs S1 and S0. The input carry Cin goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.
- By controlling the value of Y with the two selection inputs S1 and S0 and making Cin equal to 0 or 1, it is possible to generate the eight arithmetic microoperations listed in Table 44.



C_{in}
S_1
S_0
A_0
B_0
A_1
B_1
A_2
B_2
A_3
B_3
0
1
S_1
S_0
0
1
2
3
4×1 MUX
X_0
Y_0
C_0
C_1
FA
D_0
X_1
Y_1
C_1
C_2
FA
D_1
X_2
Y_2
C_2
C_3
FA
D_2
X_3
Y_3
C_3
C_4
FA
D_3
C_{out}

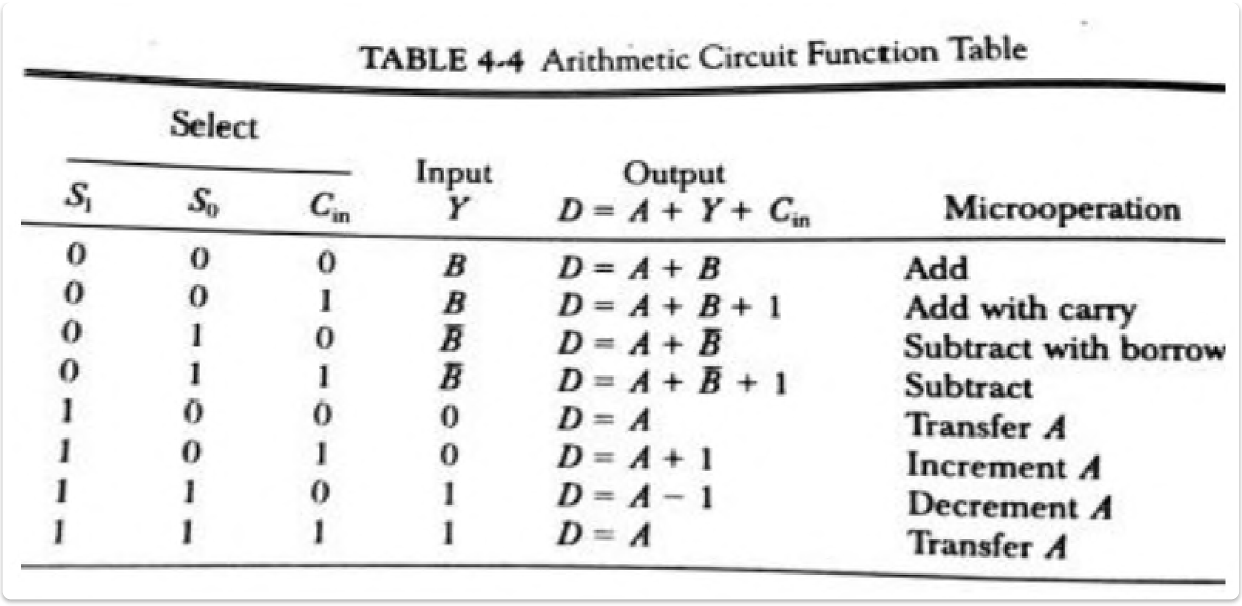


TABLE 4-4 Arithmetic Circuit Function Table

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\bar{B}$	$D = A + \bar{B}$	Subtract with borrow
0	1	1	$\bar{B}$	$D = A + \bar{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

Addition

- When S1S0= 00, the value of B is applied to the Y inputs of the adder.
  - If Cin = 0, the output D = A + B.
  - If Cin = 1, the output D = A + B + 1.
- Both cases perform the add microoperation with or without adding the input carry.

Subtraction

- When S1S0 = 01, the complement of B is applied to the Y inputs of the adder.
  - If Cin = 1, then D = A + B + 1. This produces A plus the 2's complement of B, which is equivalent to a subtraction of A -B.
  - When Cin = 0, then D = A + B. This is equivalent to a subtract with borrow, that is, A-B-1.

Increment

- When S1S0 = 10, the inputs from B are neglected, and instead, all 0's are inserted into the Y inputs.
  - The output becomes D = A + 0 + Cin. This gives D = A when Cin = 0 and D = A + 1 when Cin = 1.
  - In the first case, we have a direct transfer from input A to output D.
  - In the second case, the value of A is incremented by 1.

Decrement

- When S1S0= 11, all 1's are inserted into the Y inputs of the adder to produce the decrement operation D = A - 1 when Cin = 0.
- This is because a number with all 1's is equal to the 2's complement of 1 (the 2's complement of binary 0001 is 1111). Adding a number A to the 2's complement of 1 produces F = A + 2's complement of 1 = A — 1. When Cin = 1, then D = A -1 + 1=A, which causes a direct transfer from input A to output D.

2. Discuss in detail various types of shift micro operations

Shift Microoperations

- Shift microoperations are used for serial transfer of data.
- The contents of a register can be shifted to the left or the right.
- During a shift-left operation, the serial input transfers a bit into the rightmost position.
- During a shift-right operation, the serial input transfers a bit into the leftmost position.
- There are three types of shifts: logical, circular, and arithmetic.
- The symbolic notation for the shift microoperations is shown in Table 4-7.

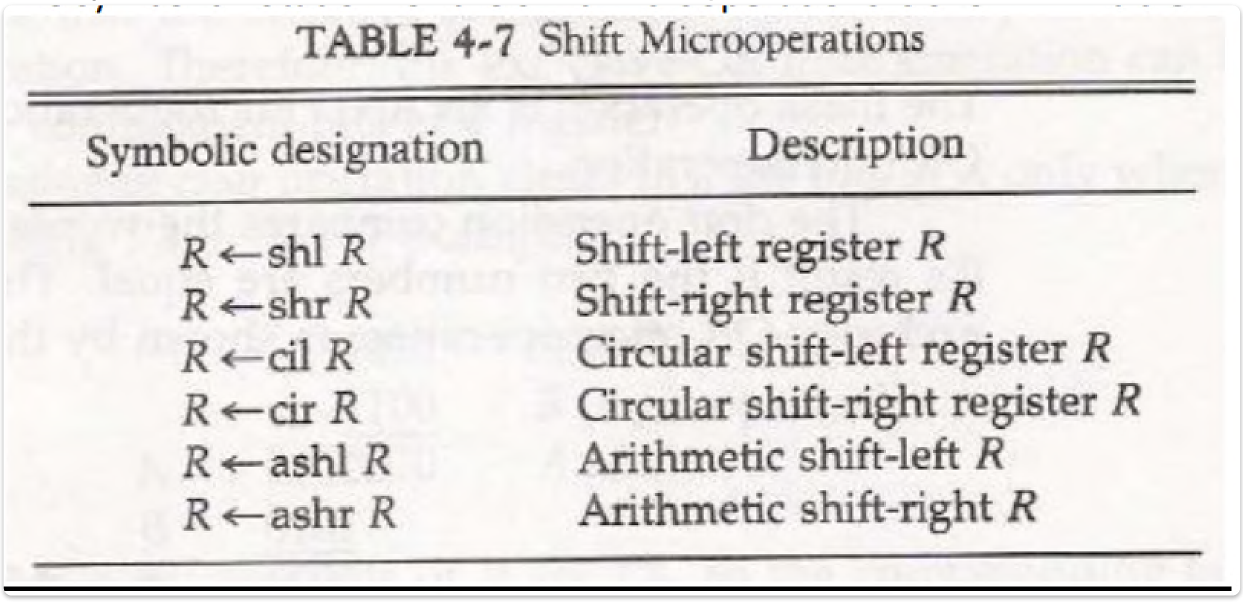


TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

Logical Shift

- A logical shift transfers 0 through the serial input.

- The symbols `shl` and `shr` represent logical shift-left and shift-right microoperations.
- The microoperations specify a 1-bit shift to the left of the content of register R and a 1-bit shift to the right of the content of register R, as shown in Table 4.7.
- The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.

Circular Shift

- The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information.
- This is accomplished by connecting the serial output of the shift register to its serial input.
- Symbols `cil` and `cir` are used for circular shift left and right, respectively.

Arithmetic Shift

- An arithmetic shift is a microoperation that shifts a signed binary number to the left or right.
- An arithmetic shift-left multiplies a signed binary number by 2.
- An arithmetic shift-right divides the number by 2.
- Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2.

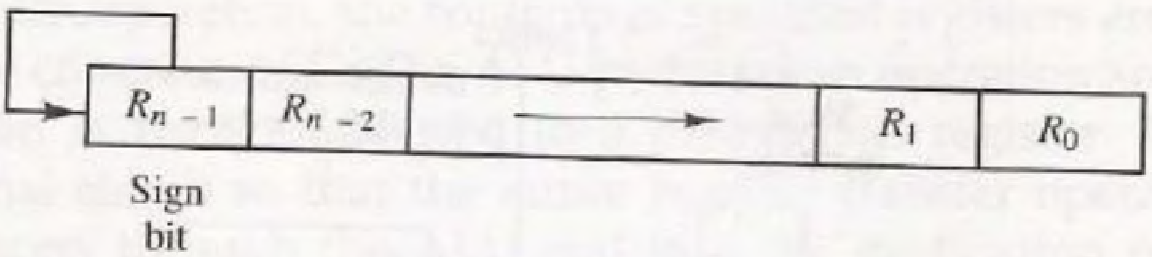
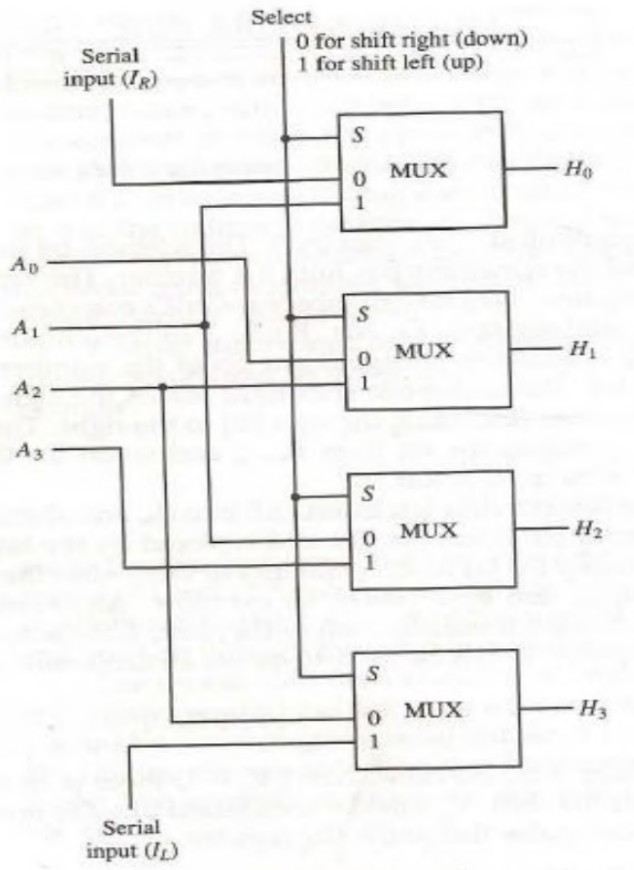


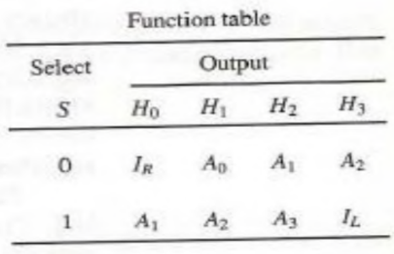
Figure 4-11 Arithmetic shift right.

Hardware Implementation

- A combinational circuit shifter can be constructed with multiplexers as shown in Fig. 4-12.

- The 4-bit shifter has four data inputs, A0 through A3, and four data outputs, H0 through H3.
- There are two serial inputs, one for shift left (IL) and the other for shift right (IR).
- When the selection input S=0, the input data are shifted right (down in the diagram).
- When S = 1, the input data are shifted left (up in the diagram).
- The function table in Fig. 4-12 shows which input goes to each output after the shift.
- A shifter with n data inputs and outputs requires n multiplexers.
- The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.





Function table

Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

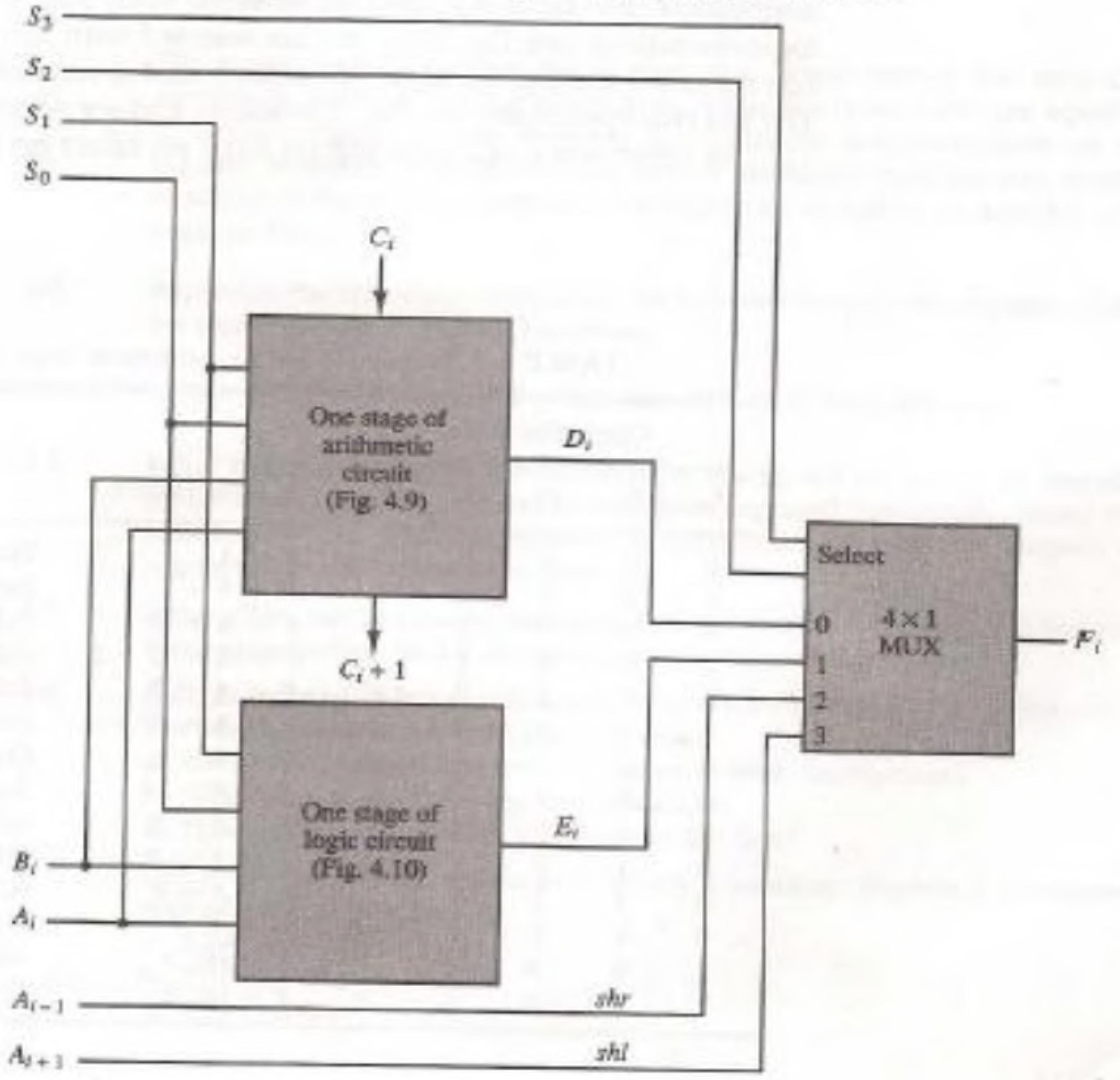
Figure 4-12 4-bit combinational circuit shifter.

Arithmetic Logic Shift Unit

- Instead of having individual registers performing the microoperations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU.

- The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period.
- The shift microoperations are often performed in a separate unit, but sometimes the shift unit is made part of the overall ALU.
- The arithmetic, logic, and shift circuits introduced in previous sections can be combined into one ALU with common selection variables. One stage of an arithmetic logic shift unit is shown in Fig. 4-13.
- A particular microoperation is selected with inputs S1 and S0. A 4 x 1 multiplexer at the output chooses between an arithmetic output in Di and a logic output in Ei.
- The data in the multiplexer are selected with inputs S3 and S2. The other two data inputs to the multiplexer receive inputs Ai-1 for the shift-right operation and Ai+1 for the shift-left operation.
- The circuit whose one stage is specified in Fig. 4-13 provides eight arithmetic operations, four logic operations, and two shift operations.
- Each operation is selected with the five variables S3, S2, S1, S0, and Cin.
- Table 4-8 lists the 14 operations of the ALU. The first eight are arithmetic operations and are selected with S3S2 = 00.
- The next four are logic and are selected with S3S2 = 01.
- The input carry has no effect during the logic operations and is marked with don't-care x's.
- The last two operations are shift operations and are selected with S3S2= 10 and 11.
- The other three selection inputs have no effect on the shift.

Figure 4-13 One stage of arithmetic logic shift unit.



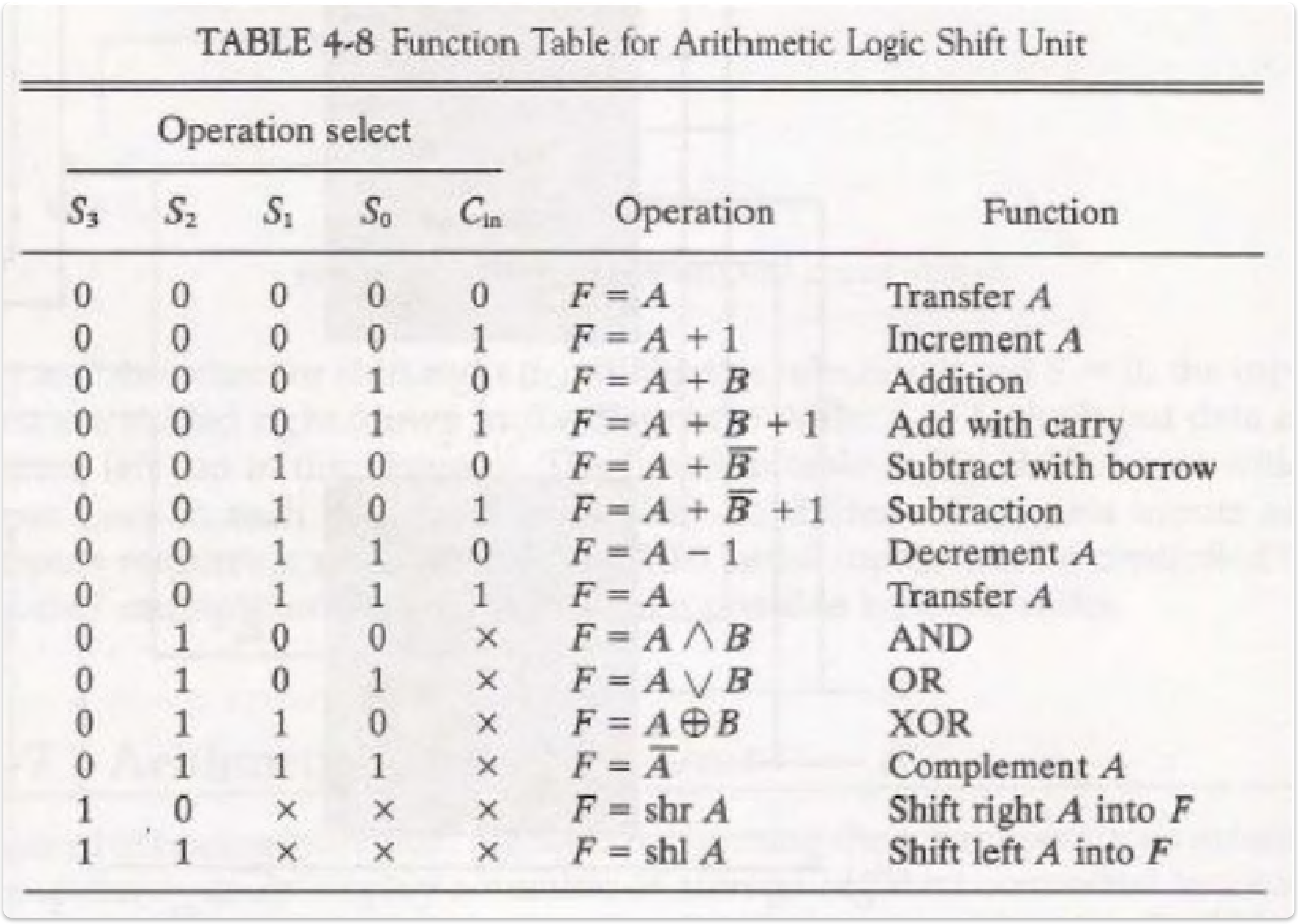


TABLE 4-8 Function Table for Arithmetic Logic Shift Unit

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F = \text{shr } A$	Shift right A into F
1	1	×	×	×	$F = \text{shl } A$	Shift left A into F

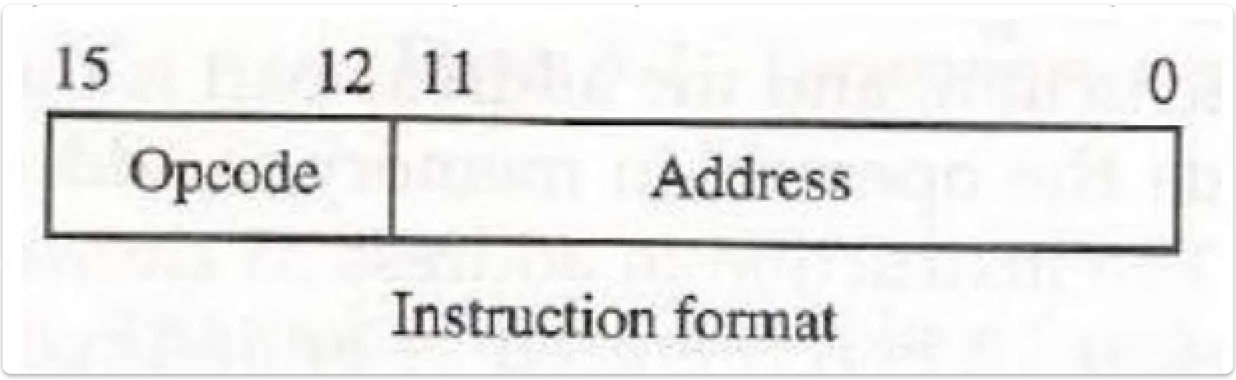
3. Define instruction code. Explain Direct and indirect addressing

Instruction Codes

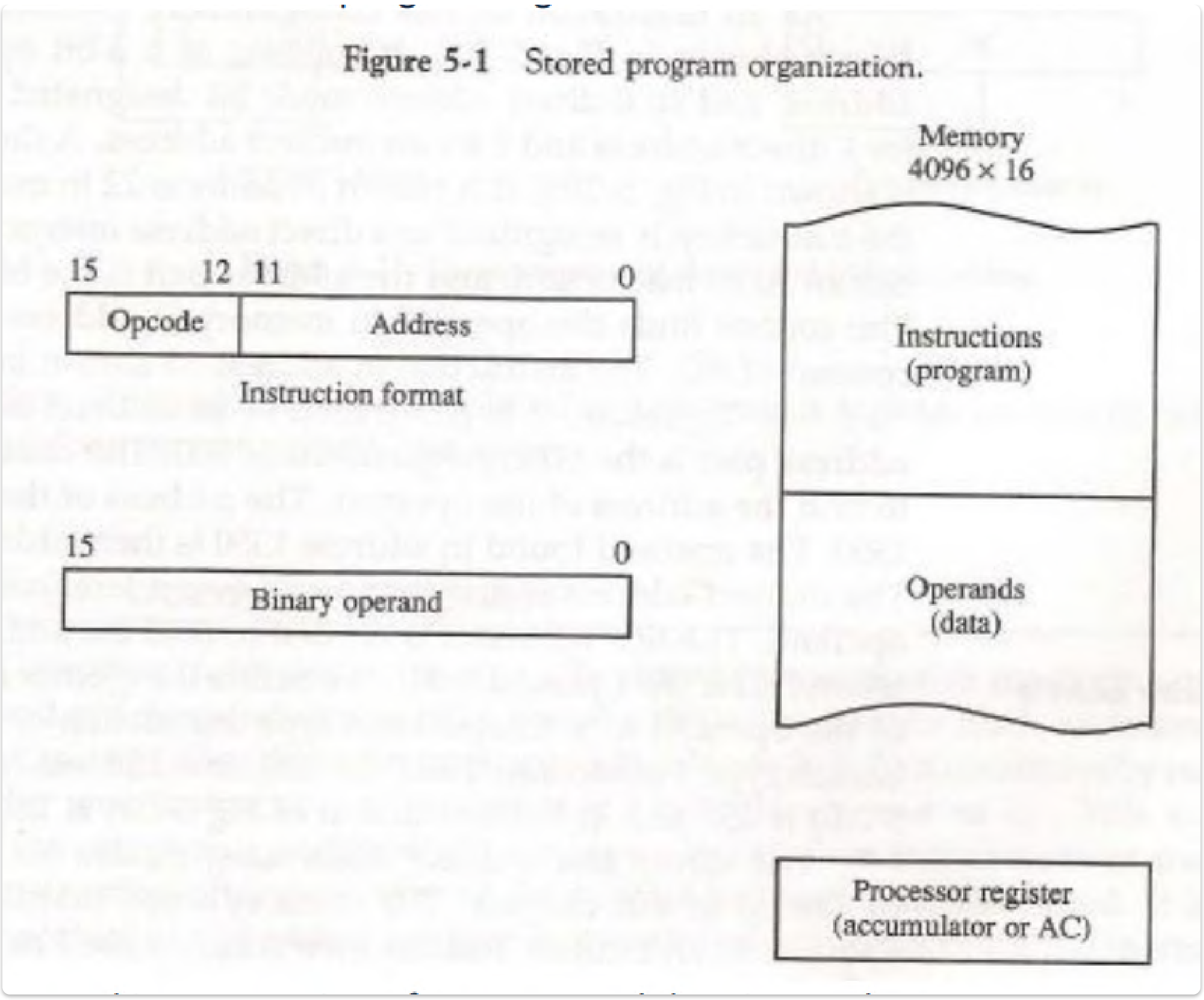
- The organization of the computer is defined by its internal registers, the timing and control structure, and the set of instructions that it uses.
- Internal organization of a computer is defined by the sequence of micro-operations it performs on data stored in its registers.
- The computer can be instructed about the specific sequence of operations it must perform.
- The user controls this process by means of a Program.
- Program: a set of instructions that specify the operations, operands, and the sequence by which processing has to occur.
- Instruction: a binary code that specifies a sequence of micro-operations for the computer.
- The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the

instruction and proceeds to execute it by issuing a sequence of micro-operations. – *Instruction Cycle*

- **Instruction Code:** a group of bits that instruct the computer to perform a specific operation.



- Instruction code is usually divided into two parts: *Opcode* and *address(operand)*
    - **Operation Code (opcode):**
        - A group of bits that define the operation.
        - Examples: add, subtract, multiply, shift, complement.
        - The number of bits required for opcode depends on the number of operations available in the computer.
        - n bit opcode >= 2^n (or less) operations.
    - **Address (operand):**
        - Specifies the location of operands (registers or memory words).
        - Memory words are specified by their address.
        - Registers are specified by their k-bit binary code.
        - k-bit address >= 2^k registers.

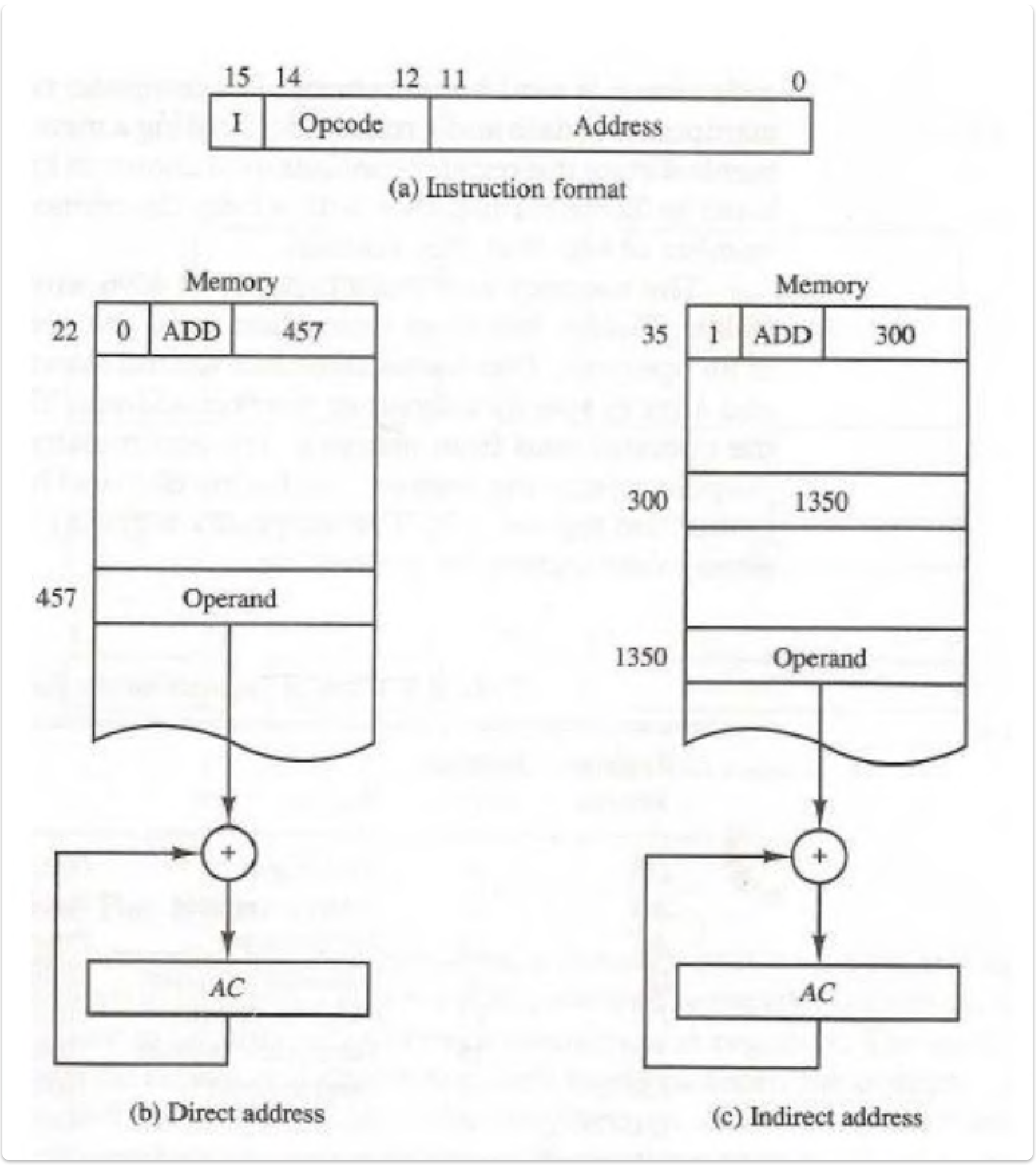


Addressing of Operand

- Sometimes it is convenient to use the address bits of an instruction code not as an address but as the actual operand.
- When the second part of an instruction code specifies an operand, the instruction is said to have an immediate operand.
- When the second part specifies the address of an operand, the instruction is said to have a direct address.
- When the second part of the instruction designates an address of a memory word in which the address of the operand is found, such instruction has an indirect address.
- One bit of the instruction code can be used to distinguish between a direct and an indirect address.
- The instruction code format shown in Fig. 5-2(a). It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit

designated by I. The mode bit is 0 for a direct address and 1 for an indirect address.

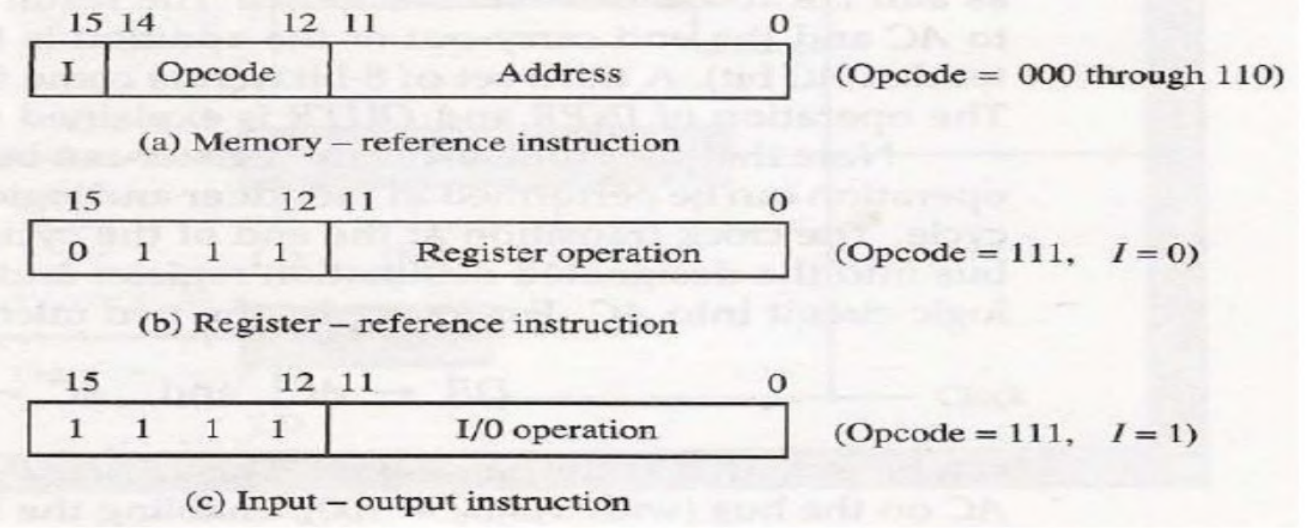
- A direct address instruction is shown in Fig. 5-2(b).
- It is placed in address 22 in memory. The I bit is 0, so the instruction is recognized as a direct address instruction. The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457.
- The control finds the operand in memory at address 457 and adds it to the content of AC.
- The instruction in address 35 shown in Fig. 5-2(c) has a mode bit I = 1.
- Therefore, it is recognized as an indirect address instruction.
- The address part is the binary equivalent of 300. The control goes to address 300 to find the address of the operand. The address of the operand in this case is 1350.
- The operand found in address 1350 is then added to the content of AC.
- The effective address is the address of the operand in a computation-type instruction or the target address in a branch-type instruction.
- Thus, the effective address in the instruction of Fig. 5-2(b) is 457, and in the instruction of Fig. 5-2(c), it is 1350.



4. Explain different types of computer instructions

Computer Instructions

Figure 5-5 Basic computer instruction formats.



- The basic computer has three instruction code formats, as shown in Fig. 5-5. Each format has 16 bits.
- The operation code (opcode) part of the instruction contains three bits, and the meaning of the remaining 13 bits depends on the operation code encountered.
- A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I.
- I is equal to 0 for a direct address and to 1 for an indirect address.
- The register-reference instructions are recognized by the operation code 1.11 with a 0 in the leftmost bit (bit 15) of the instruction.
- A register-reference instruction specifies an operation on the AC register. An operand from memory is not needed. Therefore, the other 12 bits are used to specify the operation to be executed.
- An input—output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction.
- The remaining 12 bits are used to specify the type of input—output operation.
- The instructions for the computer are listed in Table 5-2.
- The symbol designation is a three-letter word and represents an abbreviation intended for programmers and users.
- The hexadecimal code is equal to the equivalent hexadecimal number of the binary code used for the instruction.

5. How the computer provides control and timing signals

Timing and Control

- 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.
- Clock pulses do not change the state of a register unless the register is enabled by a control signal.
- 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.
- There are two major types of control organization:
  - Hardwired control
  - Microprogrammed control

	Hardwired control	**Microprogrammed control**
Control Logic	Implemented with gates, flip-flops, decoders, etc.	Control information stored in a control memory.
		The control memory is programmed to initiate the required
		sequence of microoperations.
Speed	Optimized for fast operation	Compared with hardwired control, operation is slow.
Modification	Requires changes in wiring for modifications	Changes or modifications can be done by updating the
		microprogram in control memory.

Hardwired Control Unit (Block Diagram - Fig. 5-6)

- Consists of two decoders, a sequence counter, and several control logic gates.

- An instruction read from memory is placed in the instruction register (IR), divided into three parts: The I bit, the operation code, and bits 0 through 11.
- The operation code in bits 12 through 14 is decoded with a 3 x 8 decoder, generating outputs D0 through D7.
- 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.
- Outputs of the counter are decoded into 16 timing signals T0 through T15.
- The sequence counter SC can be incremented or cleared synchronously.
- Example: SC is incremented to provide timing signals T0, T1, T2, T3, and T4 in sequence. At time T4, SC is cleared to 0 if decoder output D3 is active.

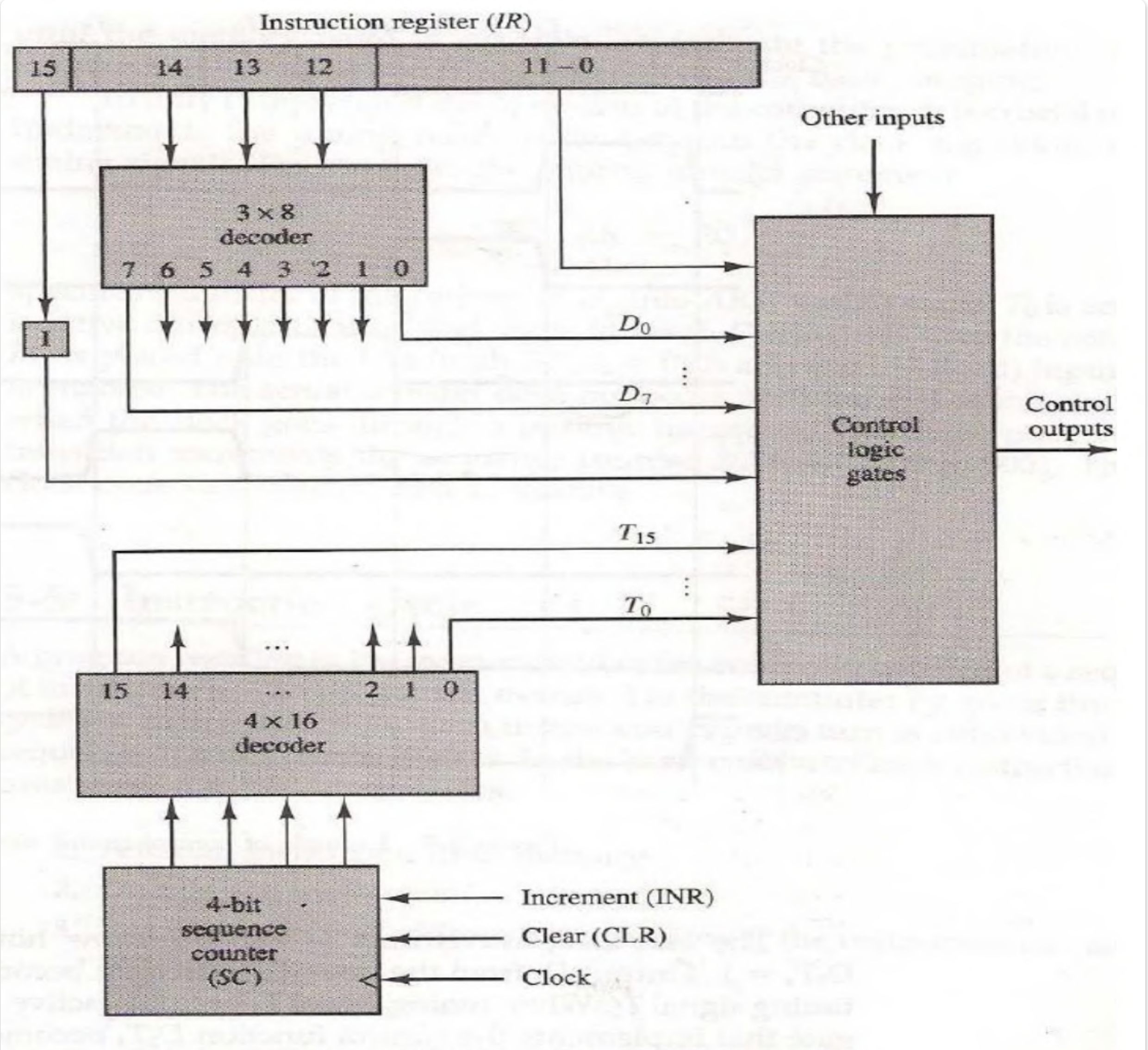


Figure 5-6 Control unit of basic computer.

Timing Diagram (Fig. 5-7)

- The sequence counter SC responds to the positive transition of the clock.
- The CLR input of SC is initially active. The first positive transition of the clock clears SC to 0, activating the timing signal T0 during one clock cycle.
- SC is incremented with every positive clock transition unless its CLR input is active, producing the sequence of timing signals T0, T1, T2, T3, T4, and so on.
- The last three waveforms show how SC is cleared when D3T4 = 1.
- Output D3 from the operation decoder becomes active at the end of timing signal T2.
- When timing signal T4 becomes active, the output of the AND gate implementing the control function D3T4 becomes active.
- This signal is applied to the CLR input of SC. On the next positive clock transition (marked T4), the counter is cleared to 0.
- This causes the timing signal T0 to become active instead of T5, which would have been active if SC were incremented instead of cleared.

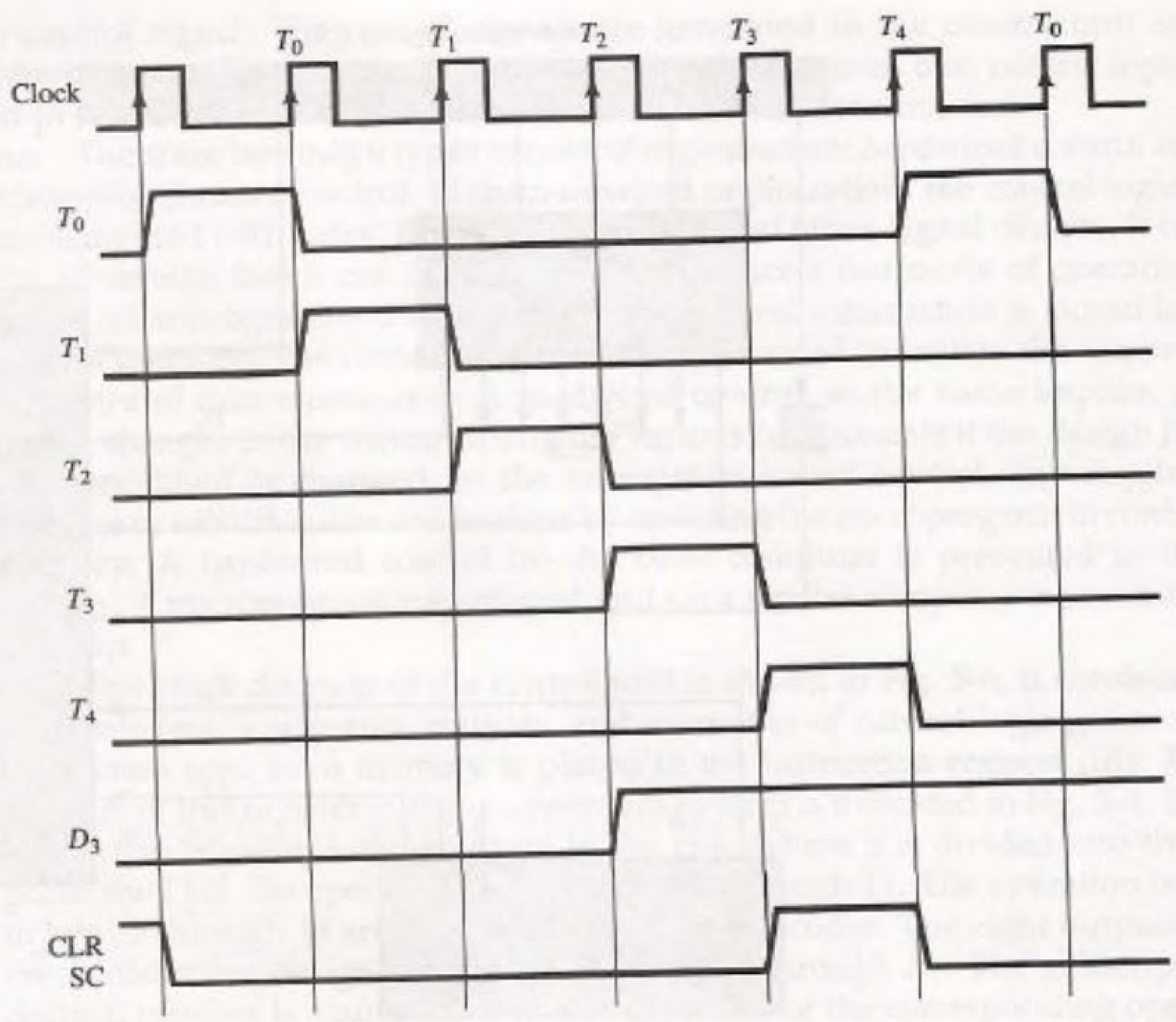


Figure 5-7 Example of control timing signals.