

Basic Computer Organization (Instruction Codes, Registers)

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.
- The Internal organization of a digital system is defined by the **sequence of micro-operations** it performs on data stored in its registers.
- The user of a computer can control the process by means of a **program**.
- A **program** is a set of instructions that specify the operations, operands, and the processing sequence.

Instruction Codes

- A **computer instruction** is a **binary code** that specifies a **sequence of micro-operations** for the computer. Each computer has its unique instruction set.
- Instruction codes and data are stored in memory.
- The computer reads each instruction from memory and places it in a control register.
- The **control unit** interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations.

Instruction Codes

- An **Instruction code** is a group of bits that instructs the computer to perform a specific operation.
- The most basic part of an instruction code is its **operation code** part.
- The operation code of an instruction is a group of bits that defines **certain operations** such as add, subtract, shift, and complement.

Instruction Codes

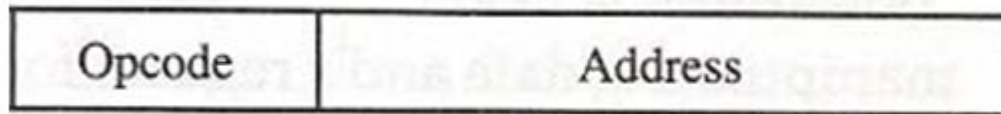
- The number of bits required for the operation code depends on the total number of operations available in the computer.
- 2^n (or little less) distinct operations
n bit operation code

Instruction Codes

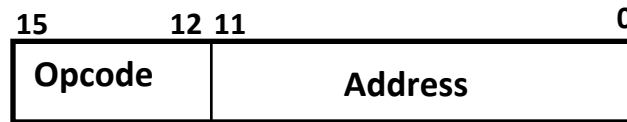
- An operation must be performed on some data stored in processor **registers** or in **memory**.
- An **instruction code** must therefore specify not only the **operation**, but also the **location of the operands** (in registers or in the memory), and where the **result** will be stored (registers/memory)

Instruction Codes

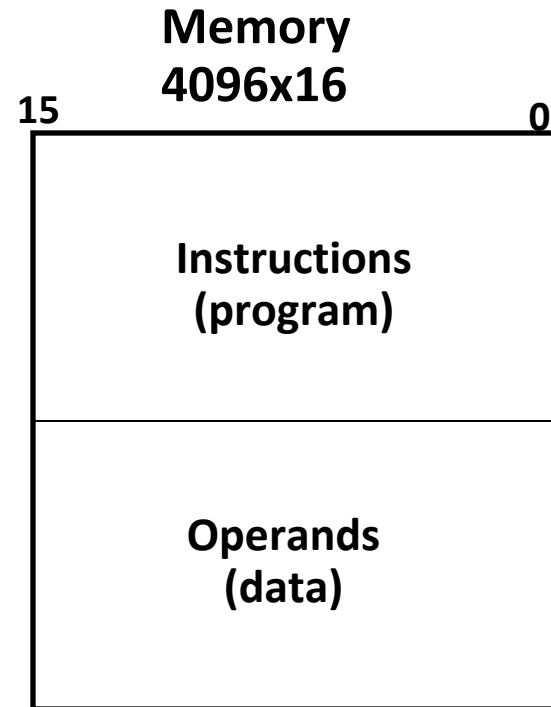
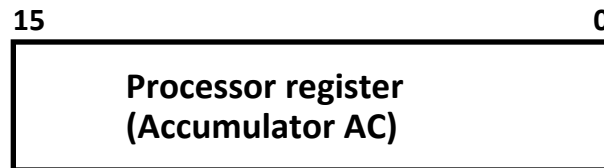
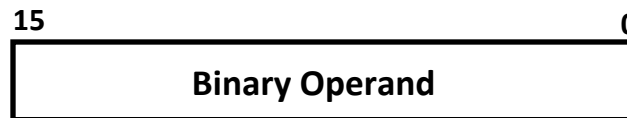
- An instruction code is usually divided into **operation code, operand address** and **addressing mode**.
- The simplest way to organize a computer is to have one processor register (Accumulator AC) and an instruction code format with two parts (op code, address)



Stored Program Organization



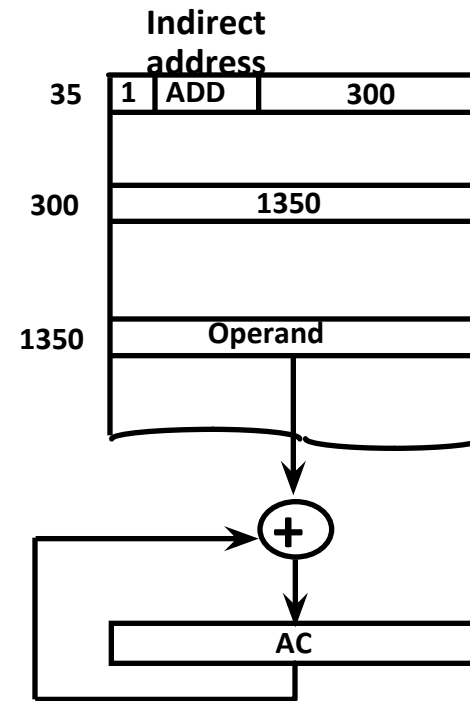
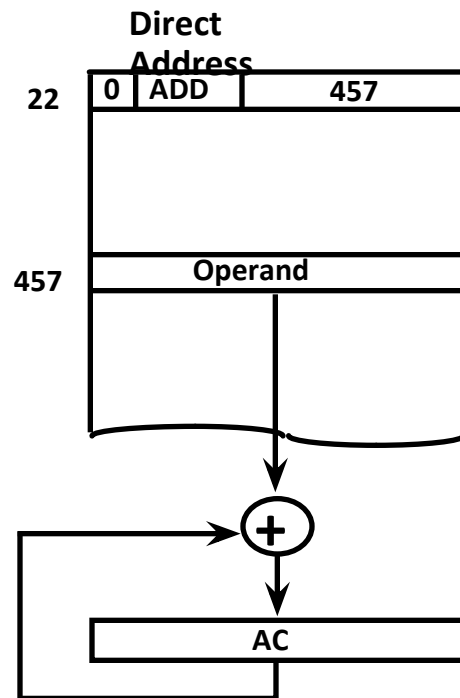
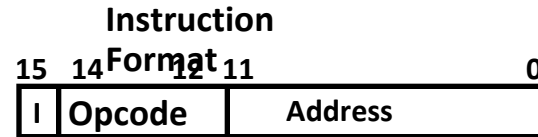
Instruction Format



Indirect Address

- There are three **Addressing Modes** used for address portion of the instruction code:
 - Immediate: the operand is given in the address portion.
 - Direct: the address points to the operand stored in the memory.
 - Indirect: the address points to the pointer (another address) stored in the memory that references the operand in memory.
- One bit of the instruction code can be used to distinguish between direct & indirect addresses

Indirect Address



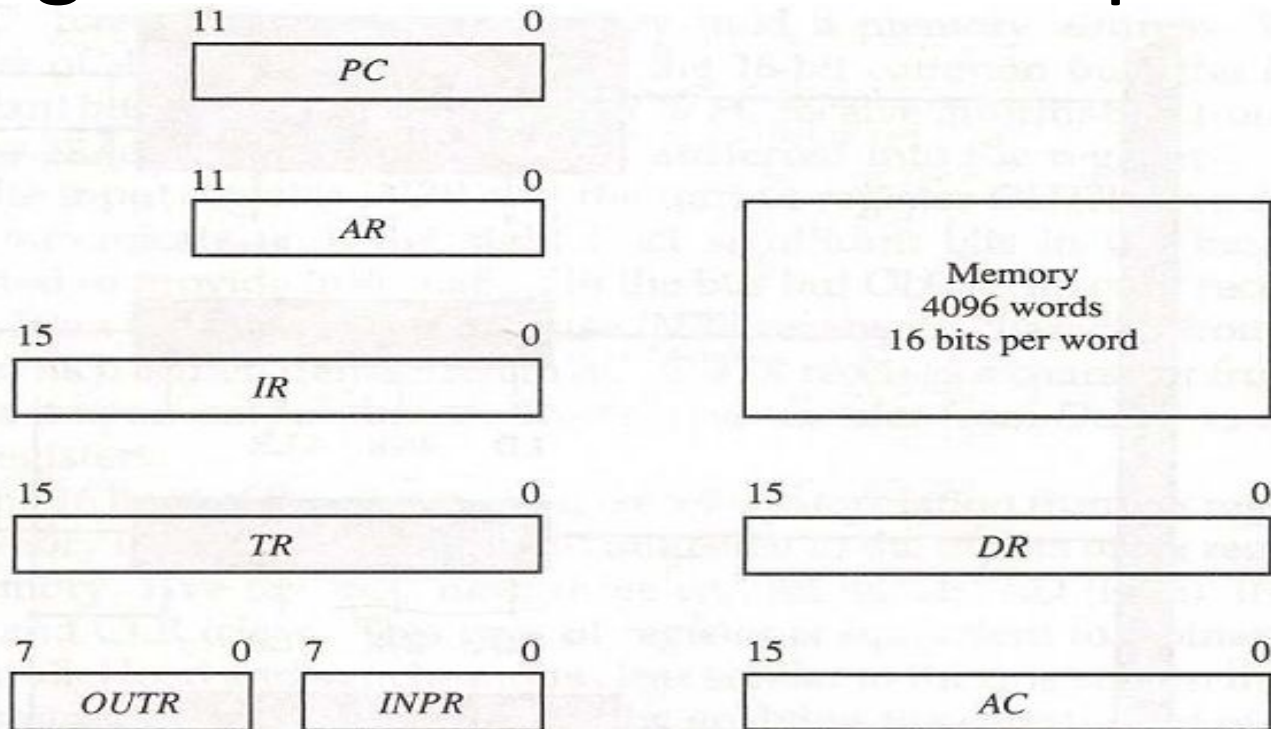
Indirect Address

- Effective address: the **address of the operand** in a computation-type instruction or the target address in a branch-type instruction
- The pointer can be placed in a processor register instead of memory.

Computer Registers

- Computer instructions are normally stored in consecutive memory locations and executed sequentially one at a time.
- The control reads an instruction from a specific address in memory and executes it.
- This type of sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed.
- It is also necessary to provide a register in the control unit for storing the instruction code after it is read from memory.
- The computer needs processor registers for manipulating data and a register for holding a memory address.

Registers for the Basic Computer

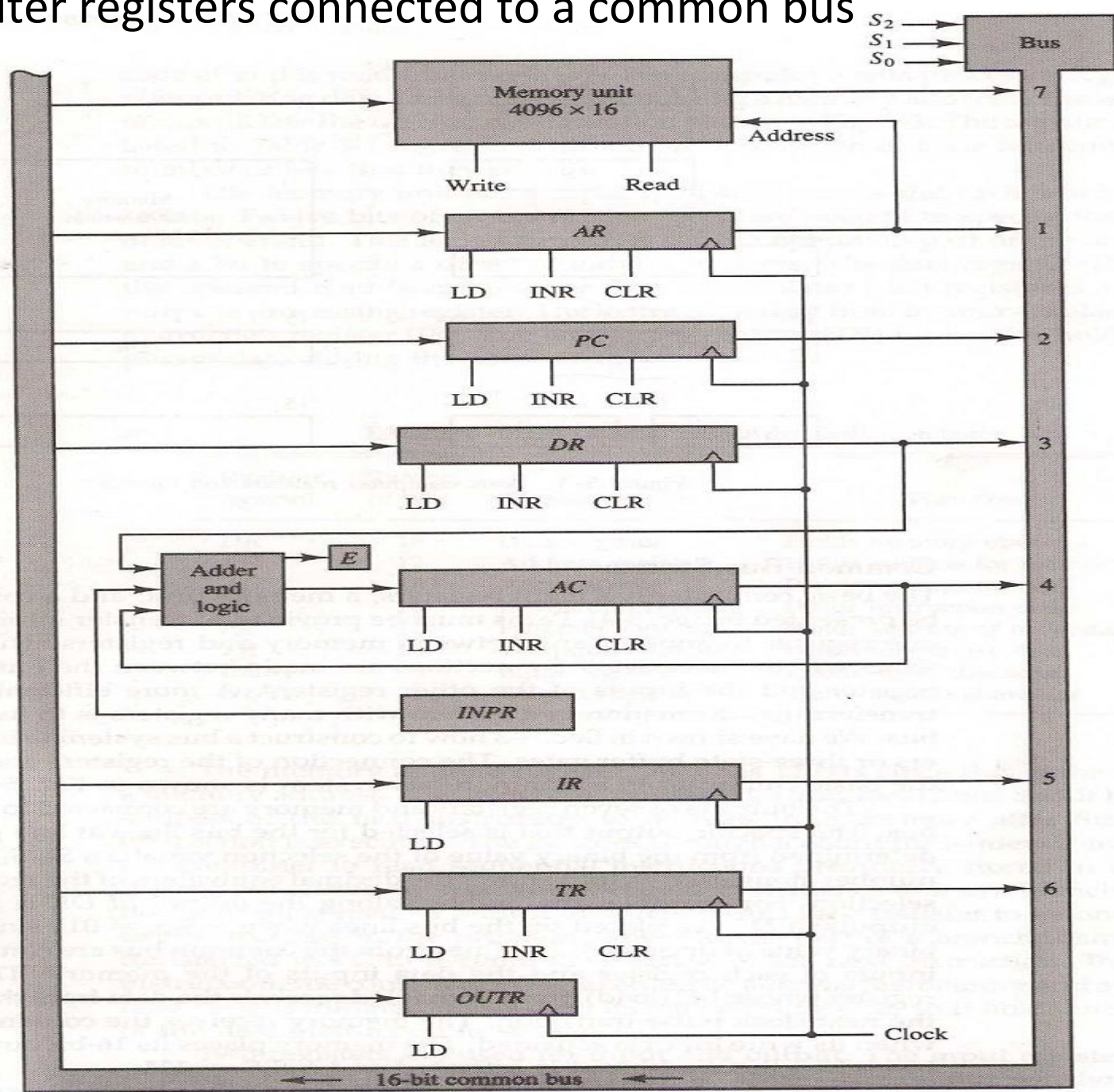


Register symbol	Number of bits	Register name	Function
<i>DR</i>	16	Data register	Holds memory operand
<i>AR</i>	12	Address register	Holds address for memory
<i>AC</i>	16	Accumulator	Processor register
<i>IR</i>	16	Instruction register	Holds instruction code
<i>PC</i>	12	Program counter	Holds address of instruction
<i>TR</i>	16	Temporary register	Holds temporary data
<i>INPR</i>	8	Input register	Holds input character
<i>OUTR</i>	8	Output register	Holds output character

Common Bus System

- Path must be provided to transfer information from one register to another register and between memory and registers.
- The number of wires will be excessive if connections are made between the outputs of each register and the inputs of the other registers.
- A more efficient scheme for transferring information in a system with many registers is to use a common bus system.

Basic computer registers connected to a common bus



Register Transfer, Bus and Memory Transfer

Register Transfer Language

- Digital Computer System: The various **modules** are interconnected with common data and control paths.
- The modules are constructed from digital components as registers, decoders, arithmetic elements and control logic.
- **Digital Module:** Registers + Operations performed on the data stored in registers.
- **Micro-operations:** The operations executed on data stored in registers.

Register Transfer Language

The result of the operation may be:

- replace the previous binary information of a register or
- transferred to another register
- Micro-operations: Examples:- shift, count, clear and load.
- A **counter** with parallel load is capable of performing the micro-operations **increment** and **load**.
- A **bidirectional shift** register is capable of performing the shift **right** and shift **left** micro-operations.

Register Transfer Language

- The internal hardware organization of a digital computer is defined by specifying:
 - The set of registers it contains and their function.
 - The sequence of micro-operations performed on the binary information stored in the registers.
 - The control that initiates the sequence of micro-operations
- Registers + Micro-operations Hardware + Control Functions = Digital Computer

Register Transfer Language

- Register Transfer Language (RTL) : a symbolic notation to describe the micro-operation transfers among registers.
 - Define symbols for various types of micro-operations
 - Describe the hardware that implements these micro-operations

Register Transfer

- Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register.
 - MAR: Memory Address Register (holds an address for a memory unit)
 - PC: Program Counter
 - IR: Instruction Register
 - SR: Status Register
 - R1: Processor Register

Register Transfer

- The individual flip-flops in an **n-bit register** are numbered in sequence from 0 to n-1 (from the right position toward the left position).
- Most common way to represent a register is by a rectangular box.
- A 16 bit register can be partitioned into two parts, low bytes and high byte.

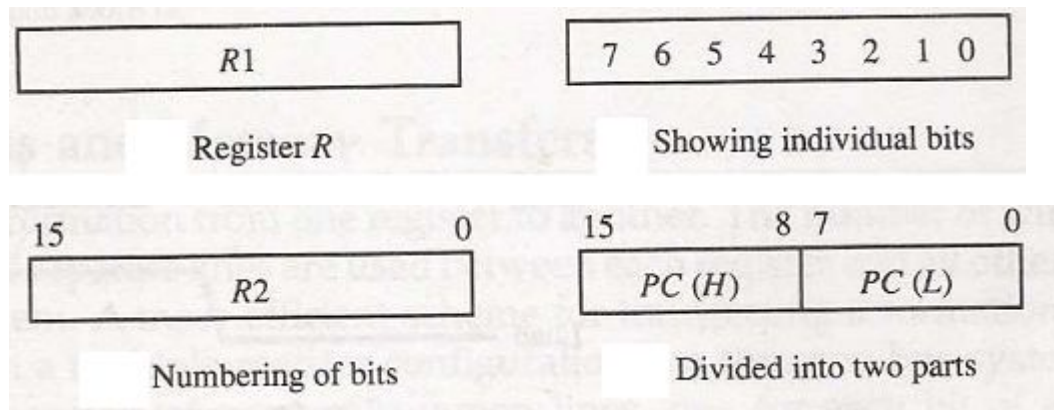


Fig: A block diagram of a register

Register Transfer

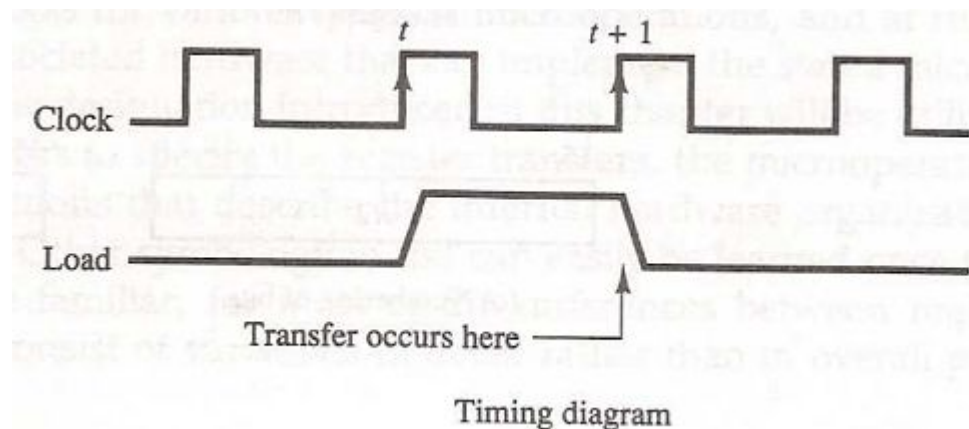
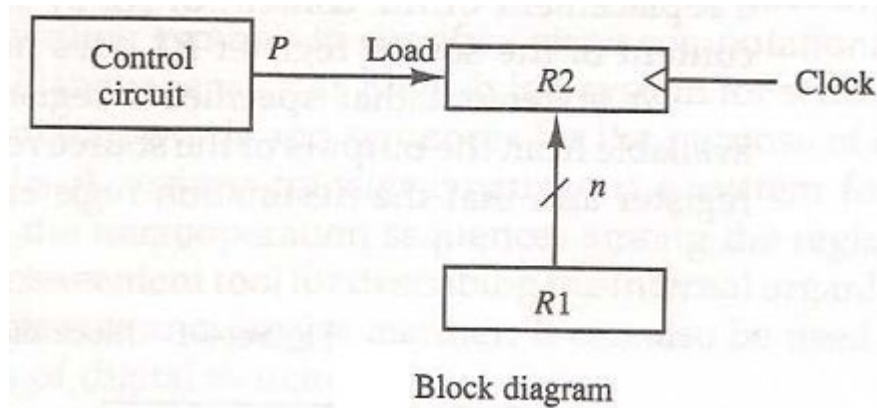
- Information transfer from one register to another is described by a **replacement operator**: $R2 \leftarrow R1$
- This statement denotes a transfer of the content of register R1 into register R2.
- The transfer happens in one clock cycle.
- The content of the R1 (source) does not change.
- The content of the R2 (destination) will be lost and replaced by the new data transferred from R1.
- We are assuming that the circuits are available from the outputs of the source register to the inputs of the destination register, and that the destination register has a parallel load capability.

Register Transfer

- Conditional transfer occurs only under a control condition
- Representation of a (conditional) transfer
P: $R2 \leftarrow R1$
- A binary condition (P equals to 0 or 1) determines when the transfer occurs.
- The content of R1 is transferred into R2 only if **P is 1**.

Register Transfer

Hardware implementation of a controlled transfer: $P: R2 \leftarrow R1$



Register Transfer

Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters & numerals	Denotes a register	MAR, R2
Parenthesis ()	Denotes a part of a register	R2(0-7), R2(L)
Arrow \leftarrow	Denotes transfer of information	$R2 \leftarrow R1$
Comma ,	Separates two micro-operations	$R2 \leftarrow R1, R1 \leftarrow R2$

Bus and Memory Transfers

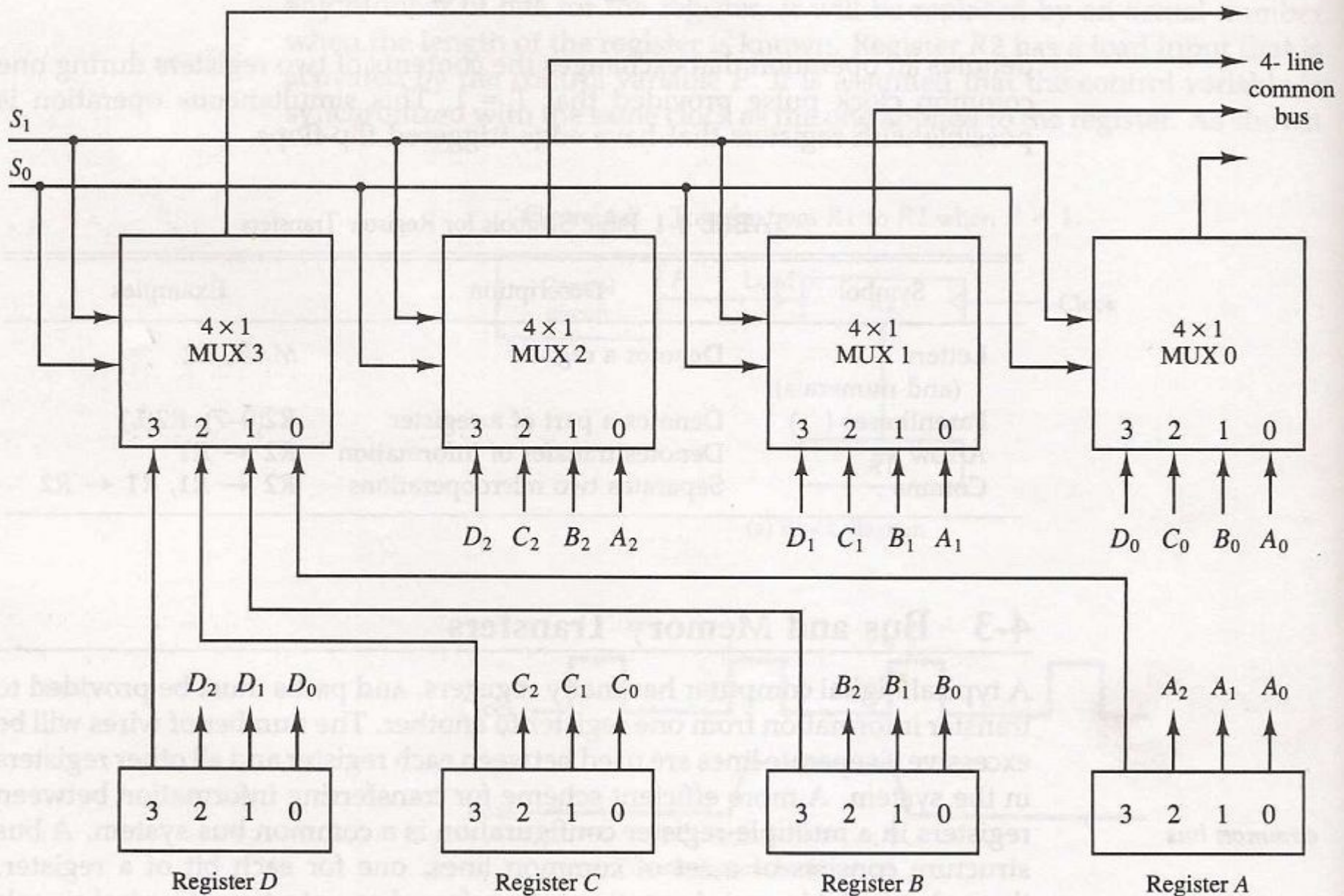
- A typical digital computer has many registers and paths must be provided to transfer information from one register to another.
- The number of wires will be excessive if separate lines are used between each register and all other registers in the system.
- **Common Bus System** is a scheme for transferring information between registers in a multiple-register configuration.
- **Bus**: set of common lines, one for each bit of a register, through which binary information is transferred one at a time.
- **Control signals** determine which register is selected by the bus during each particular register transfer.

Bus and Memory Transfers

- One way of constructing a common bus system is with multiplexers.
- The multiplexers select the source register whose binary information is then placed on the bus.

Function Table for Bus

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D



Bus and Memory Transfers

Construction of a bus system for four registers
(Example)

- Each register has four bits, numbered 0 through 3.
- The bus consists of four 4×1 multiplexers and two selection inputs, S_1 and S_0 .
- Both the selection lines are connected to the selection inputs of all four multiplexers.
- The selection lines choose the four bits of one register and transfer them into the four-line common bus.

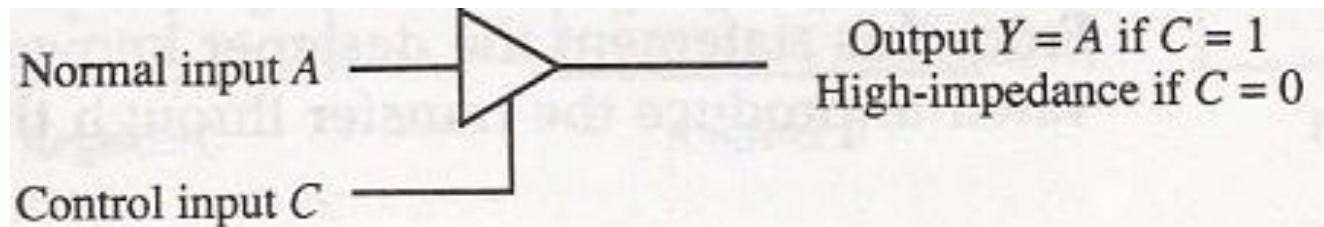
Bus and Memory Transfers

- A common bus system can be constructed with three-state gates instead of multiplexers.

Bus and Memory Transfers:

Three-State Bus Buffers

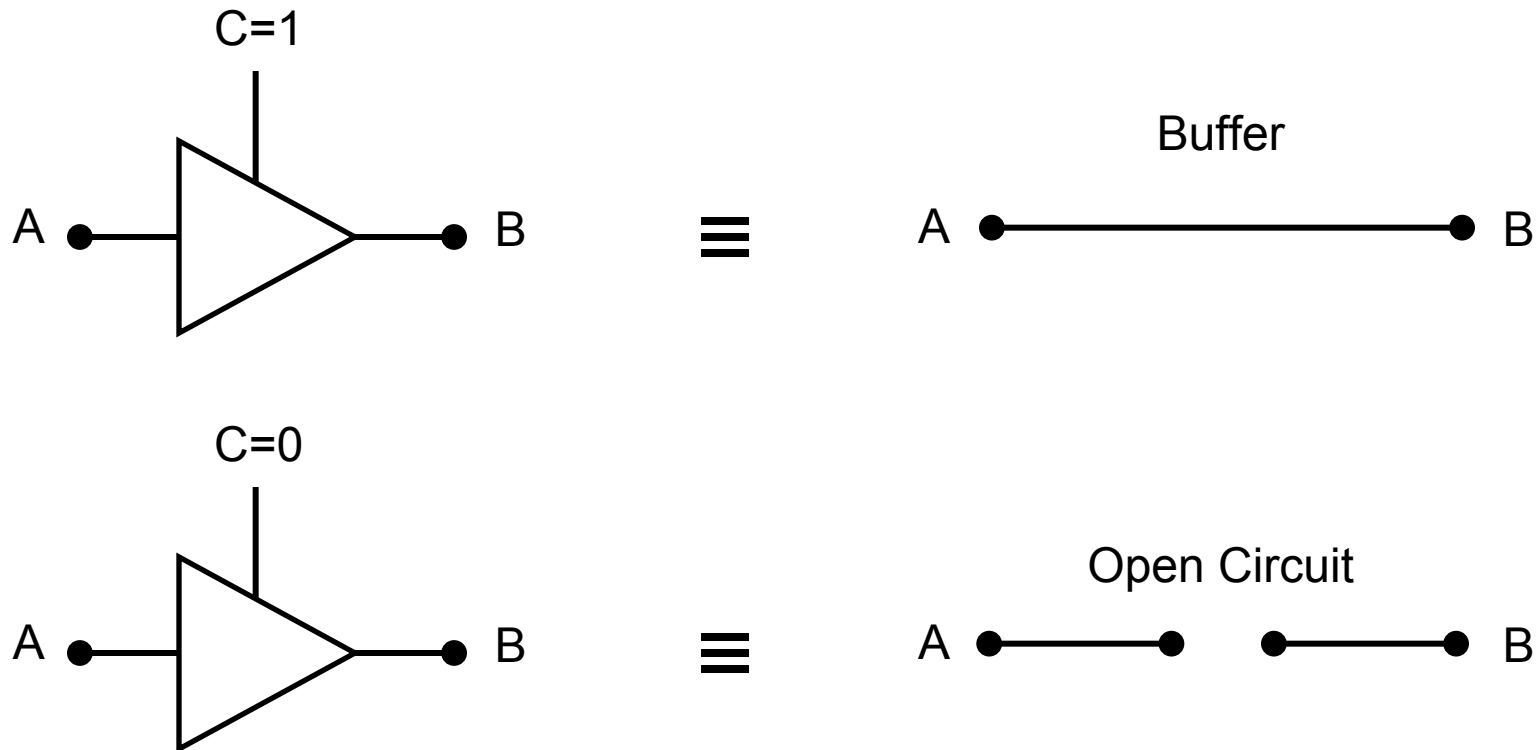
- A bus system can be constructed with three-state buffer gates instead of multiplexers
- A three-state buffer is a digital circuit that exhibits three states:
 - logic-0,
 - logic-1, and
 - high-impedance (Hi-Z)



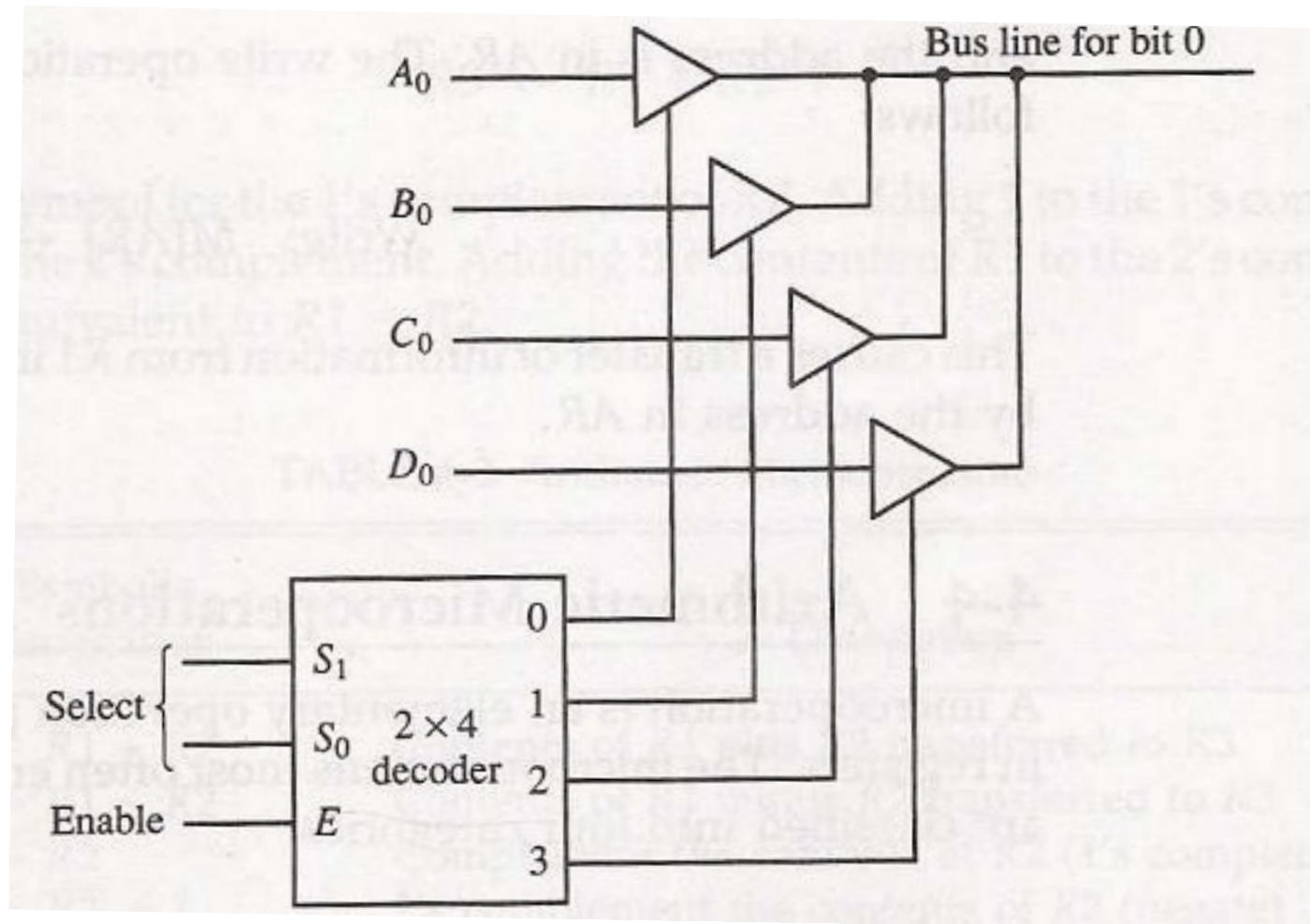
Bus and Memory Transfers:

Three-State Bus Buffers

- Tri-state buffer has both a normal input and a control input.



Bus and Memory Transfers: Three-State Bus Buffers



Bus and Memory Transfers:

Three-State Bus Buffers

- The outputs of four buffers are connected together to form a single bus line.
- When the enable input of decoder is 0, all of its four outputs are 0, and bus line is in a high-impedance state because all four buffers are disabled.
- When the enable input is active, one of the three-state buffers will be active, depending on the select inputs of the decoder.

Bus and Memory Transfers:

Memory Transfer

- Memory read : Transfer from memory
- Memory write : Transfer to memory
- Data being read or wrote is called a memory word (called M)
- It is necessary to specify the address of M when writing /reading memory
- This is done by enclosing the address in square brackets following the letter M
- Example: M[0016] : the memory contents at address 0x0016

Bus and Memory Transfers:

Memory Transfer

- Assume that the address of a memory unit is stored in a register called the Address Register AR
- Lets represent a Data Register with DR, then:
 - Read: $DR \leftarrow M[AR]$
 - Write: $M[AR] \leftarrow DR$

Arithmetic Micro-operations

Arithmetic Microoperations

- A **microoperation** is an elementary operation performed with the data stored in **registers**.
- The microoperations most often encountered in digital computers are classified into four categories:
 - Register transfer microoperations (on binary information)
 - Arithmetic microoperations (on numeric data stored in the registers)
 - Logic microoperations (bit manipulations on non-numeric data)
 - Shift microoperations

Arithmetic Microoperations

- The basic arithmetic microoperations are: addition, subtraction, increment, and decrement.

Arithmetic Microoperations

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R3$
$R3 \leftarrow R1 - R2$	Contents of $R1$ minus $R2$ transferred to $R3$
$R2 \leftarrow \overline{R2}$	Complement the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of $R2$ (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus the 2's complement of $R2$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ by one
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ by one

Arithmetic Microoperations

- The arithmetic microoperations change the information content.
- Addition Microoperation:

$$R3 \leftarrow R1 + R2$$

- Subtraction Microoperation:

$$R3 \leftarrow R1 - R2 \text{ or :}$$

$$R3 \leftarrow R1 + \overline{R2} + 1$$

1's complement

Arithmetic Microoperations

- One's Complement Microoperation:

$$R2 \leftarrow \overline{R2}$$

- Two's Complement Microoperation:

$$R2 \leftarrow \overline{R2} + 1$$

- Increment Microoperation:

$$R2 \leftarrow R2 + 1$$

- Decrement Microoperation:

$$R2 \leftarrow R2 - 1$$

Arithmetic Microoperations:

Binary Adder

- A binary adder is constructed with full-adder circuits connected in cascade, with the output carry from one full-adder connected to the input carry of the next full-adder.

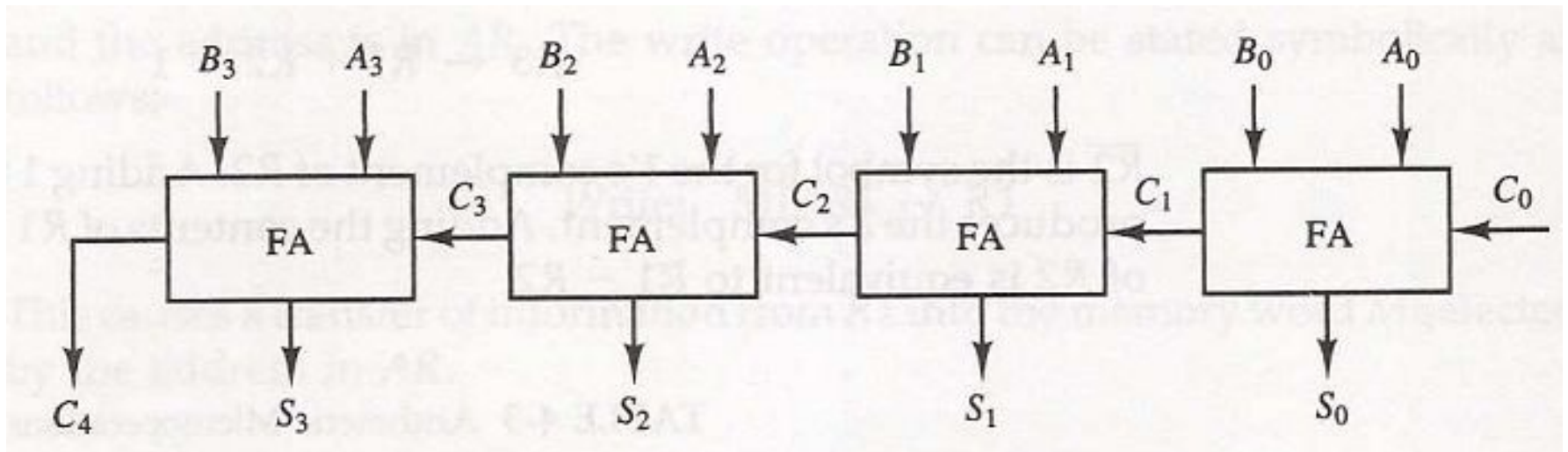


Fig: 4-bit binary adder

Arithmetic Microoperations:

Binary Adder

- 4 data bits for the A inputs come from one register.
- 4 data bits for the B inputs come from another register.
- The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the low-order bit.
- The input carry to the binary adder is C_0 and the output carry is C_4 .
- The output carry from each full-adder is connected to the input carry of the next-high-order full-adder.

Arithmetic Microoperations:

Binary Adder

- 4-bit binary adder is constructed with interconnection of four full-adders (FA).
- An n -bit binary adder requires n full-adders.
- The n data bits for the A inputs come from one register.
- The n data bits for the B inputs come from another register.
- The sum can be transferred to a third register or one of the source registers, replacing its previous content.

Arithmetic Microoperations:

Binary Adder-Subtractor

- The subtraction $A-B$ can be done by taking the 2's complement of B and adding it to A . The 2's complement can be obtained by taking the 1's complement and adding one to the least significant pair of bits.
- The 1's complement can be implemented with inverters and a one can be added to the sum through the input carry.
- The addition and subtraction operation can be combined into one common circuit by including an exclusive-OR gate with each full-adder.

Arithmetic Microoperations:

Binary Adder-Subtractor

- The mode input M controls the operation.
- When $M=0$ the circuit is an Adder and when $M=1$ the circuit becomes a Subtractor.

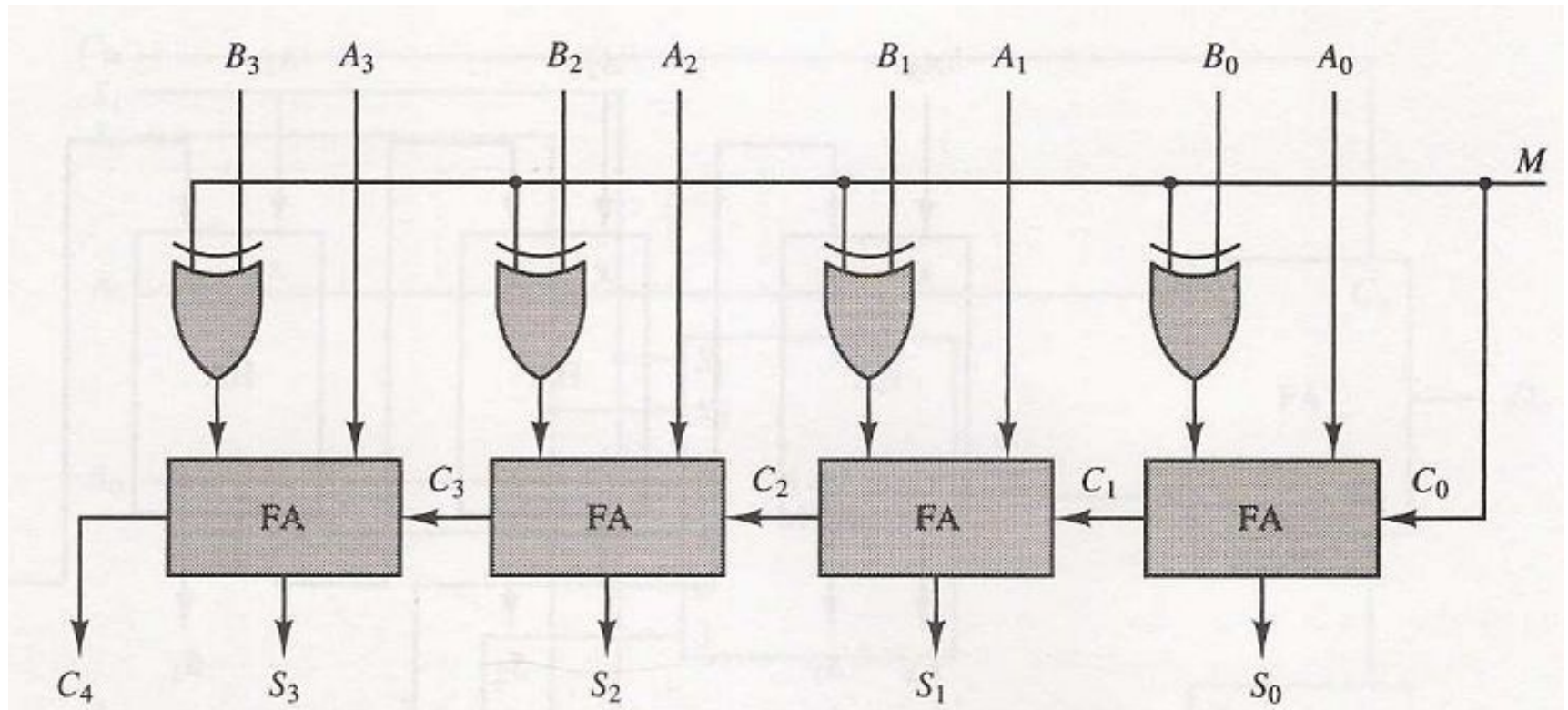


Fig: 4-bit Adder-Subtractor

Arithmetic Microoperations:

Binary Adder-Subtractor

- Each exclusive-OR gate receives input M and one of the input of B.
- When **M=0**, we have $B \oplus 0 = B$.
- The full-adder receive the value of B, the input carry C_0 is 0, and the circuit performs A plus B.
- When **M=1**, we have $B \oplus 1 = B'$ and $C_0 = 1$.
- The inputs are all complemented and a 1 is added through the input carry C_0 .
- The circuit performs the operation A plus the 2's complement of B.

Arithmetic Microoperations: Binary Adder-Subtractor

- For unsigned numbers, this gives $A - B$ if $A \geq B$
provided that there is overflow or
- For signed numbers, the result is $A - B$ provided that
there is no overflow.
the 2's complement of the result if $A < B$
(example: $3 - 5 = -2 = 1110$)

Arithmetic Microoperations:

Binary Incrementer

- The increment microoperation adds one to a number in a register.
- Binary Incrementer can also be implemented using a counter.
- A incrementer can be accomplished by half-adders (HA) connected in cascade.

Arithmetic Microoperations: Binary Incrementer

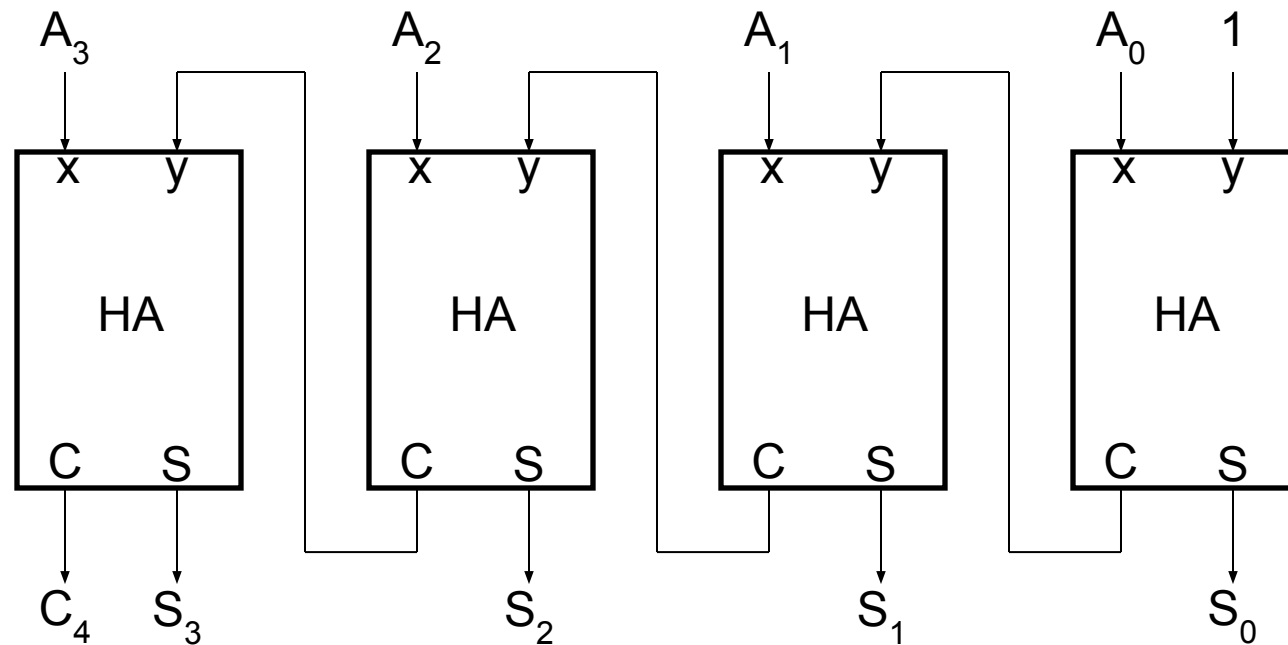


Fig: 4-bit Binary Incrementer

Arithmetic Microoperations:

Binary Incrementer

- One of the input to the least significant half-adder (HA) is connected to logic-1.
- The output carry from one half-adder is connected to one of the input of the next-higher-order half-adder.
- The circuit receives the four bits from A_0 through A_3 , adds one to it, and generates the incremented output in S_0 through S_3 .
- An n-bit binary incrementer can be extended by including n half-adders.

4-4 Arithmetic Microoperations: Binary decrementer

- A binary decrementer can be implemented by adding 1111 to the desired register each time.

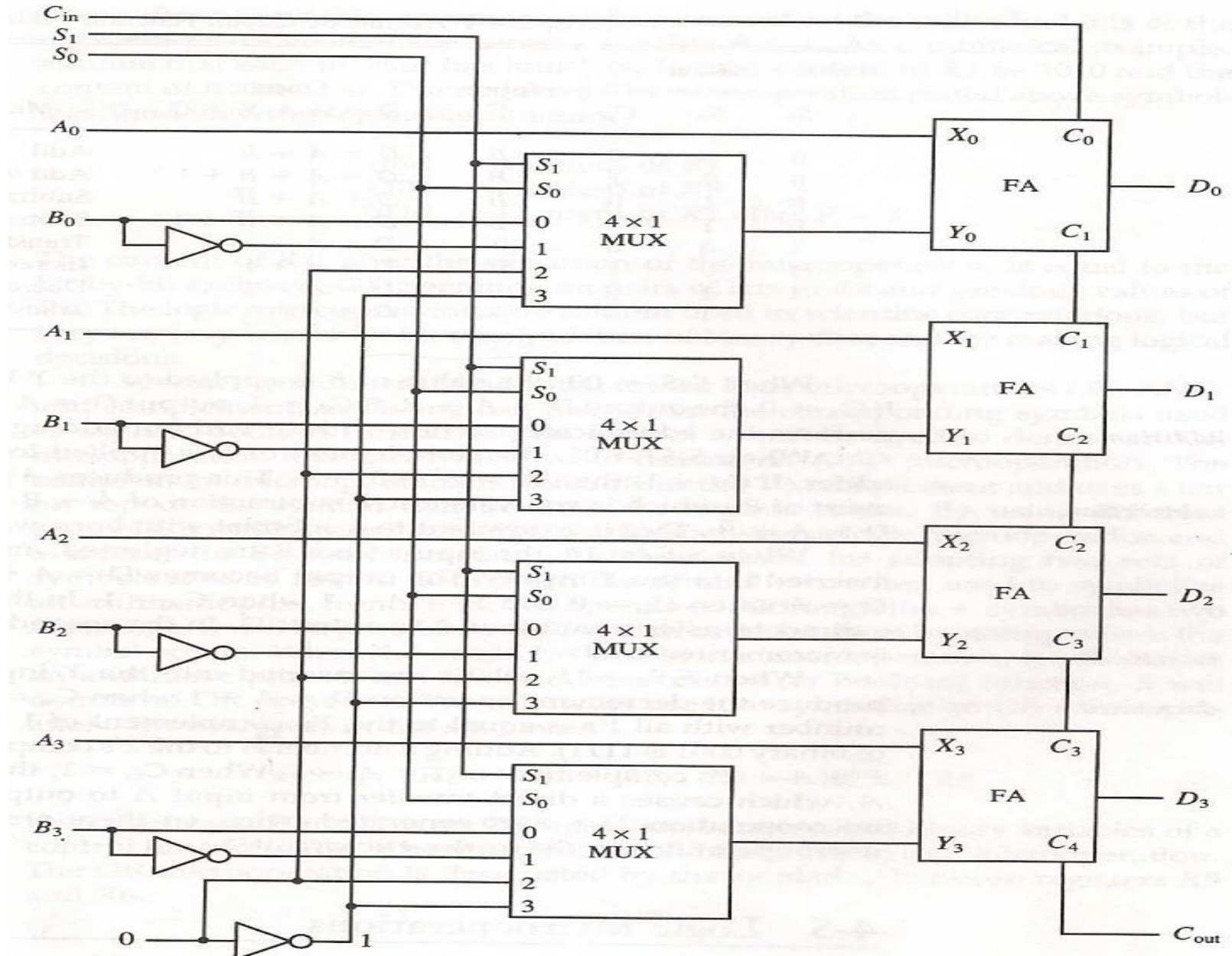
Arithmetic Microoperations:

Arithmetic Circuit

- Arithmetic circuit performs seven distinct arithmetic operations and the basic component of it is the parallel adder.
- By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.
- The output of the binary adder is calculated from the following arithmetic sum:

$$D = A + Y + C_{in}$$

Arithmetic Microoperations: Arithmetic Circuit



Arithmetic Microoperations:

Arithmetic Circuit

- 4-bit Arithmetic circuit 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 are connected to the data inputs of the multiplexers.
- The multiplexers data inputs also receive the complement of B.

Arithmetic Microoperations:

Arithmetic Circuit

- The other two data inputs are connected to logic-0 and logic-1.
- The four multiplexers are controlled by two selection inputs S_1 and S_0 . The input carry C_{in} , 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 S_1 and S_0 and making C_{in} equal to 0 or 1, it is possible to generate the eight arithmetic microoperations

Arithmetic Microoperations:

Arithmetic Circuit

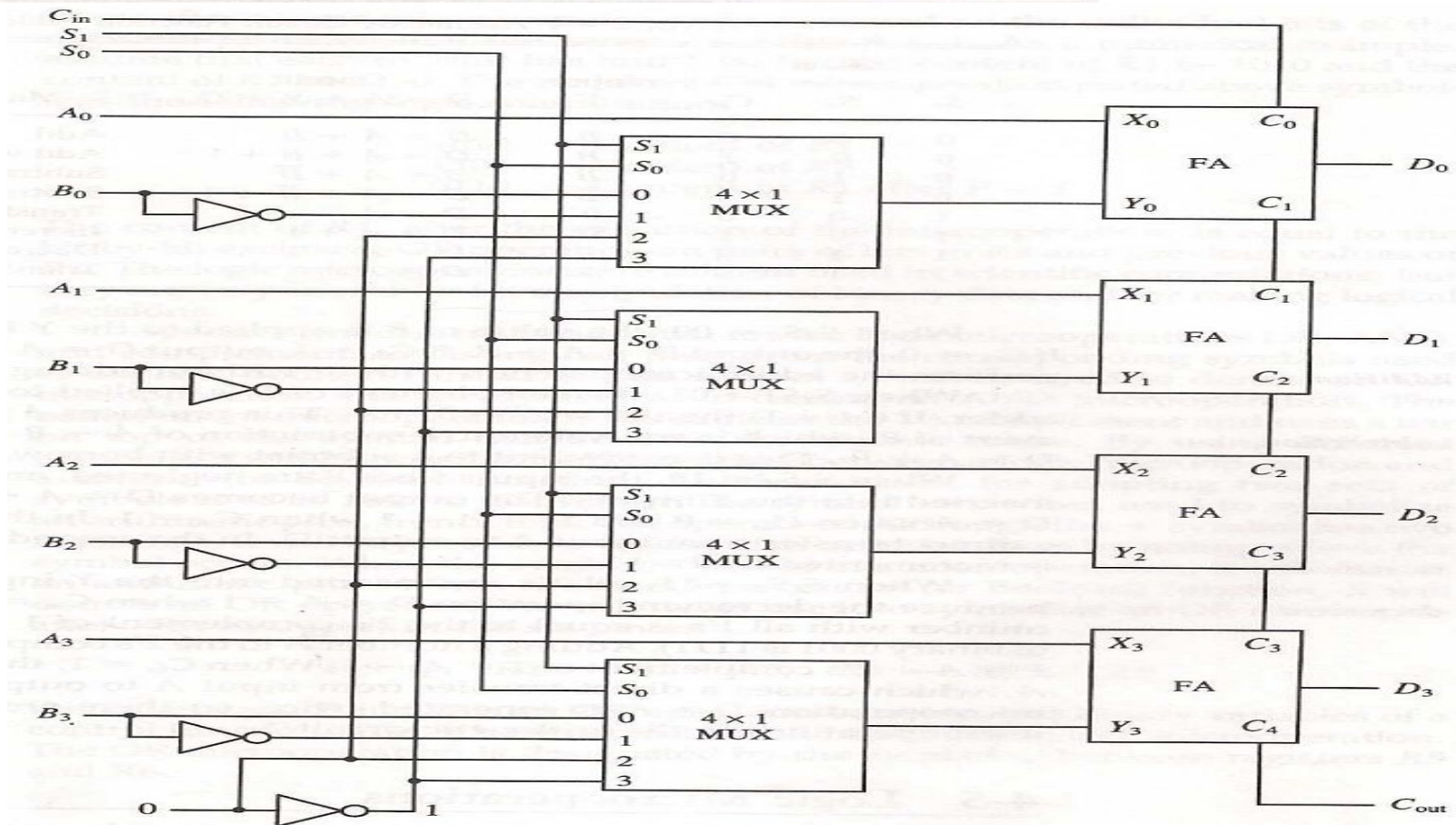
Arithmetic Circuit Function Table

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

Arithmetic Circuit

Arithmetic Circuit Function Table

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



Logic Microoperations

Logic Microoperations

- Logic microoperations specify binary operations for strings of bits stored in registers.
- These operations consider each bit of the register separately and treat them as binary variables.
- For example, the exclusive-OR microoperation with the contents of two registers R1 and R2 is symbolized by the statement

$$P: R1 \leftarrow R1 \oplus R2$$

- The exclusive-OR microoperation symbolizes the logic computation

1010	Content of R1
1100	Content of R2
<u>0110</u>	Content of R1 after $P = 1$

Special symbols

- For logic microoperations some special symbols are adopted.
- The reason for adopting the special symbols is to be able to distinguish the symbols used in arithmetic microoperations.

Four basic microoperations

OR Microoperation

- Symbol: \vee

- Gate: 

- Example: $100110 \vee 010110 = 110110$

Four basic microoperations

AND Microoperation

- Symbol: \wedge

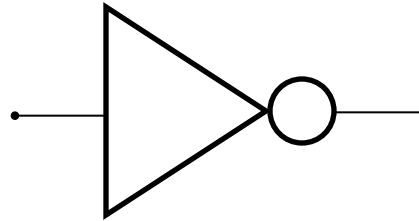
- Gate: 

- Example: $100110 \wedge 010110 = 000110$

Four basic microoperations

Complement (NOT) Microoperation

- Symbol: $\overline{}$



- Gate:

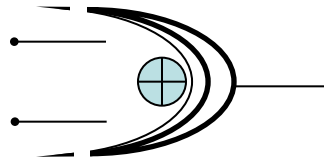
- Example: $1010110 = 0101001$

Four basic microoperations

XOR (Exclusive-OR) Microoperation

- Symbol: \oplus

- Gate:



- Example: $100110 \oplus 010110 = 110000$

List of Logic Microoperations

- There are 16 different logic operations that can be performed with two binary variables.
- The Boolean functions listed in the first column represents a relationship between two binary variables x and y .
- The logic micro-operations listed in the second column represent a relationship between the binary content of two registers A and B .

Sixteen Logic Microoperations

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow \text{all 1's}$	Set to all 1's

List of Logic Microoperations

- The 16 different logic operations can be determined from all possible truth tables obtained with two binary variables.
- Each of the 16 columns F_0 through F_{15} represents a truth table of one possible Boolean function for the two variables x and y .

Truth Tables for 16 Functions of Two Variables

[illegible]

List of Logic Microoperations

Sixteen Logic Microoperations

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow \text{all 1's}$	Set to all 1's

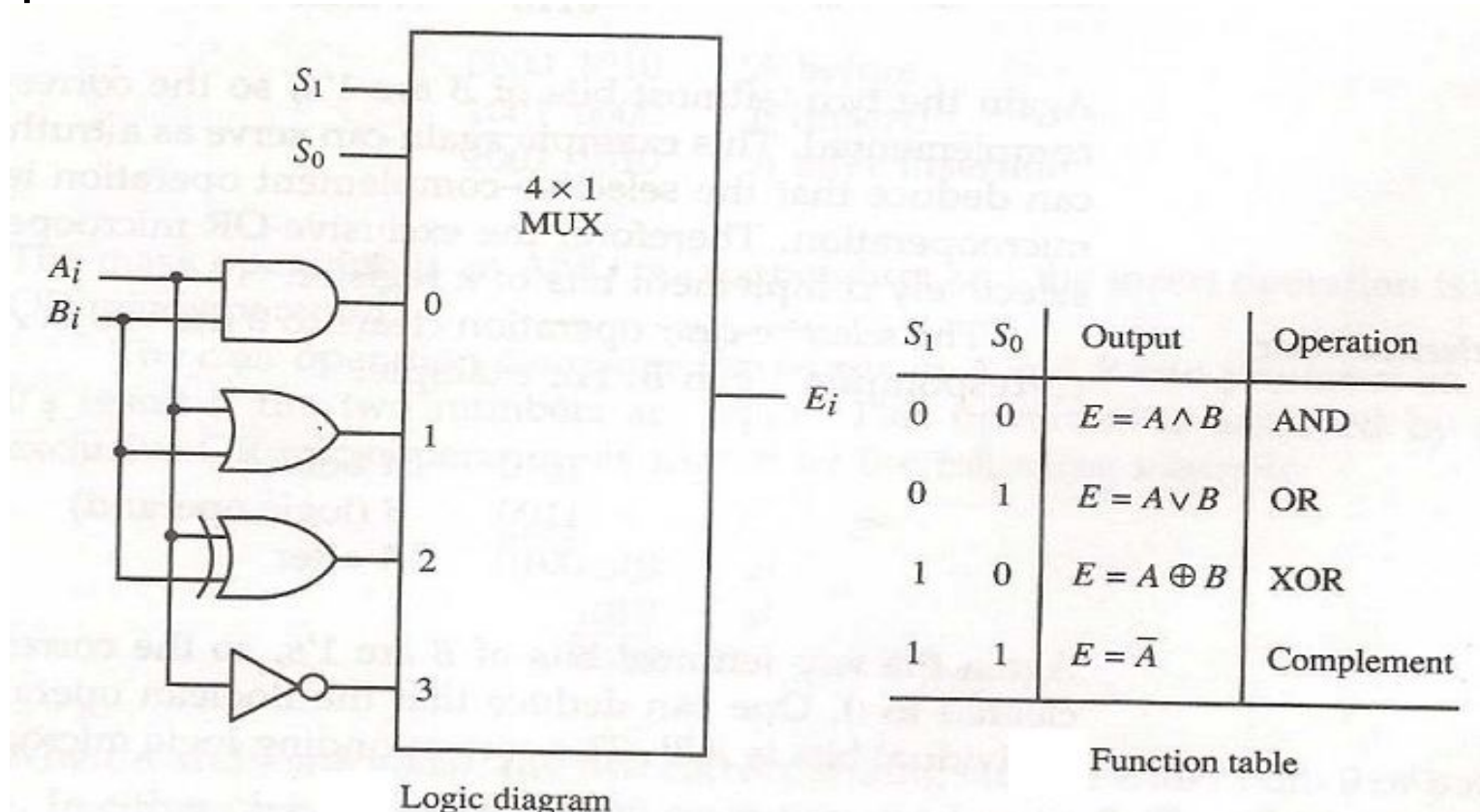
- Most computers use only four:- AND, OR, XOR, and complement operations to derive 16 logic microoperations.

Truth Tables for 16 Functions of Two Variables

[illegible]

Logic Circuit

- It consists of four gates and a multiplexer. Each of the four logic operations is generated through a gate that performs the required logic.
- The two selection inputs S_1 and S_0 choose one of the data inputs of the multiplexer and direct its value to the output.



Some Applications

- Logic micro-operations are very useful for manipulating individual bits or a portion of a word stored in a register.
- They can be used to change bit values, delete a group of bits or insert new bits values into a register.

Selective-set operation

- The selective-set operation sets to 1 the bits in register A where there are corresponding 1's in register B. It does not affect bit positions that have 0's in B.

1010	A before
<u>1100</u>	B (logic operand)
1110	A after

- The OR microoperation can be used to selectively set bits of a register.

Selective-complement operation

- The selective-complement operation complements bits in A where there are corresponding 1's in B. It does not affect bit positions that have 0's in B. For example:

1010	A before
1100	B (logic operand)
<u>1100</u>	
0110	A after

- The exclusive-OR microoperation can be used to selectively complement bits of a register.

Selective-clear operation

- The selective-clear operation clears to 0 the bits in A only where there are corresponding 1's in B . For example:

1010	A before
<u>1100</u>	B (logic operand)
0010	A after

- The corresponding logic microoperation is

$$A \leftarrow A \wedge \bar{B}$$

Mask operation

- The mask operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding 0's in B. The mask operation is an AND micro operation as seen from the following numerical example:

1010	A before
1100	B (logic operand)
<u>1100</u>	
1000	A after masking

Insert operation

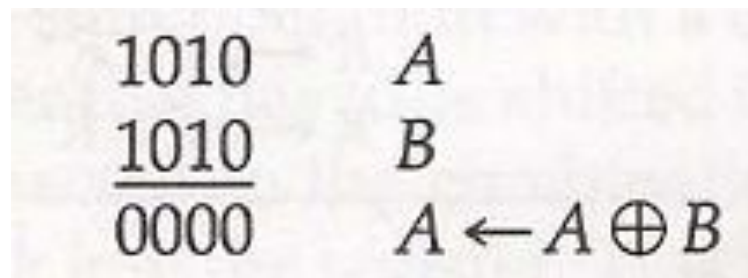
- The insert operation inserts a new value into a group of bits. This is done by first masking the bits and then ORing them with the required value.
- For example, suppose that an A register contains eight bits, 0110 1010. To replace the four leftmost bits by the value 1001 we first mask the four unwanted bits:

0110 1010	A before
<u>0000 1111</u>	B (mask)
0000 1010	A after masking

0000 1010	A before
<u>1001 0000</u>	B (insert)
1001 1010	A after insertion

Clear operation

- The clear operation compares the words in A and B and produces an all 0's result if the two numbers are **equal**.



1010 A
1010 B
1010
0000 $A \leftarrow A \oplus B$

- This operation is achieved by an exclusive-OR microoperation as shown by the following example

Shift Microoperations

Microoperations

- A **microoperation** is an elementary operation performed with the data stored in **registers**.
- The microoperations most often encountered in digital computers are classified into four categories:
 - Register transfer microoperations (on binary information)
 - Arithmetic microoperations (on numeric data stored in the registers)
 - Logic microoperations (bit manipulations on non-numeric data)
 - Shift microoperations

Shift Microoperations

- Shift microoperations are used for serial transfer of data.
- The Shift microoperations that 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.
- 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.

Types of Shift Microoperations

There are three types of shifts Microoperations :

- 1) Logical
- 2) Circular
- 3) Arithmetic

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 is one that transfers 0 through the serial input.
- The symbols **shl** and **shr** for logical shift-left and shift-right microoperations.
- 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.
- We will use the symbols **cil** and **cir** for the circular shift left and right, respectively.

Arithmetic Shift

- An arithmetic shift microoperation 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.
- Negative numbers are stored in 2's complement form.

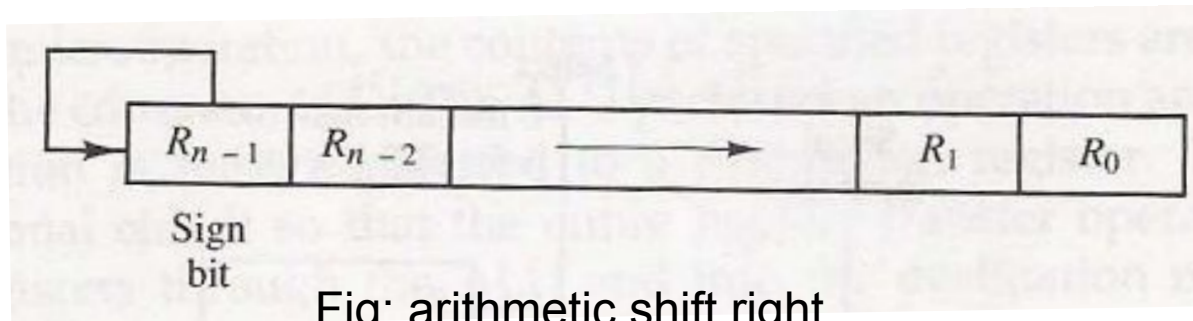


Fig: arithmetic shift right

Hardware Implementation

- The 4-bit shifter has four data inputs, A_0 through A_3 , and four data outputs, H_0 through H_3 .
- There are two serial inputs, one for shift left (I_L) and the other for shift right (I_R).

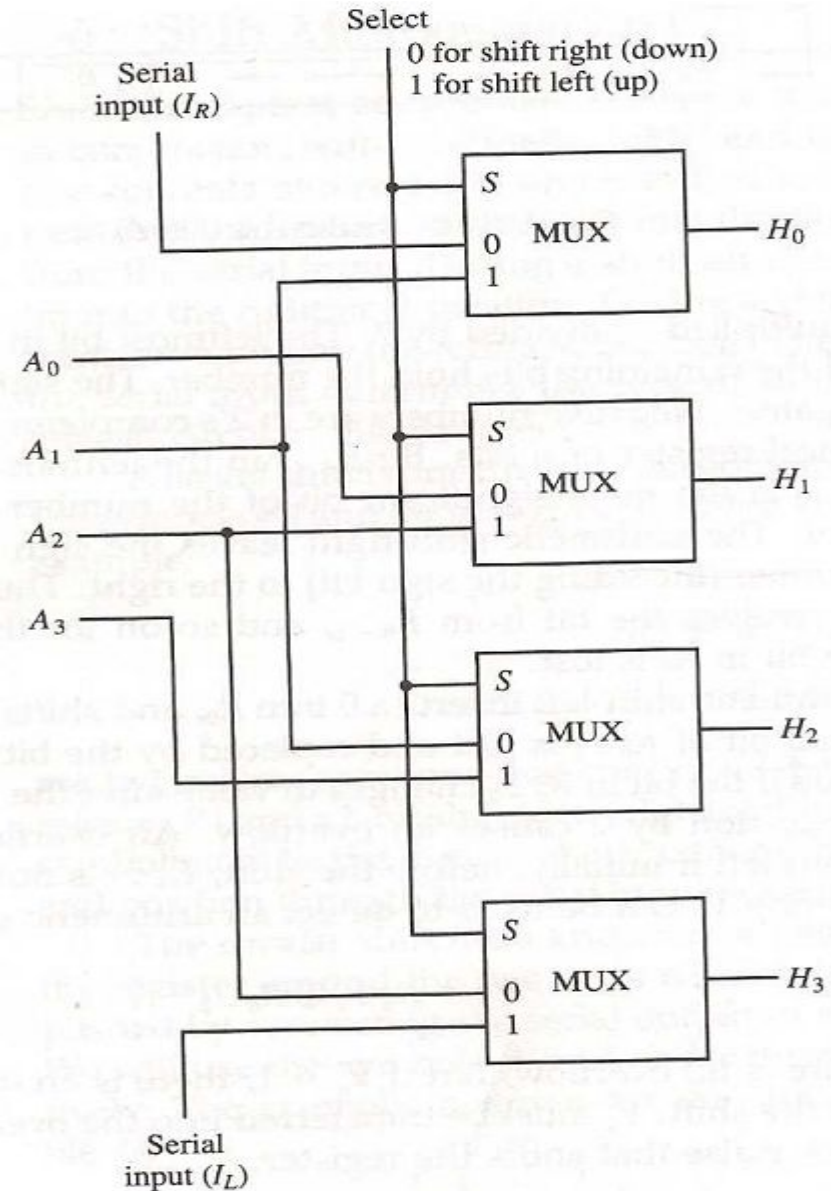


Fig: 4-bit combinational circuit shifter

Hardware Implementation

- 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).
- A shifter with **n** data inputs and outputs requires **n** multiplexers.

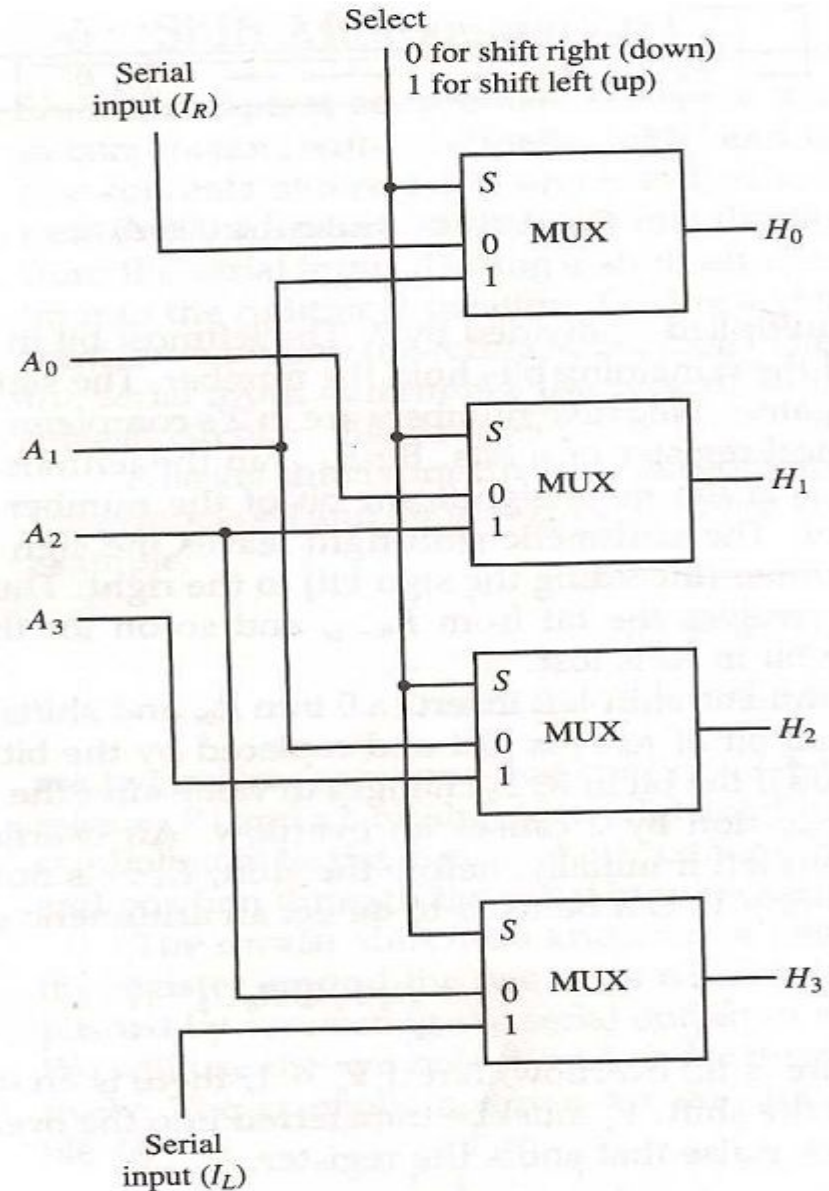
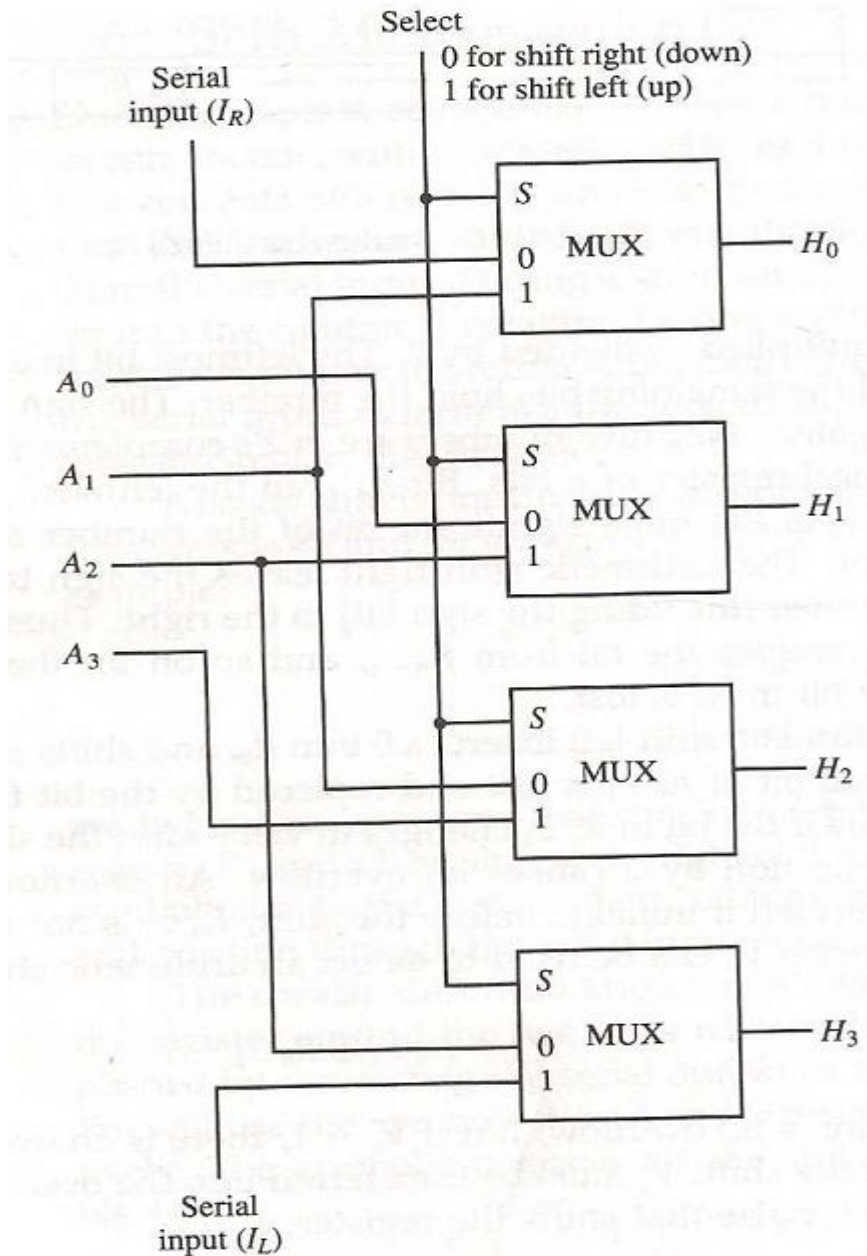


Fig: 4-bit combinational circuit shifter

4-bit Combinational Circuit Shifter



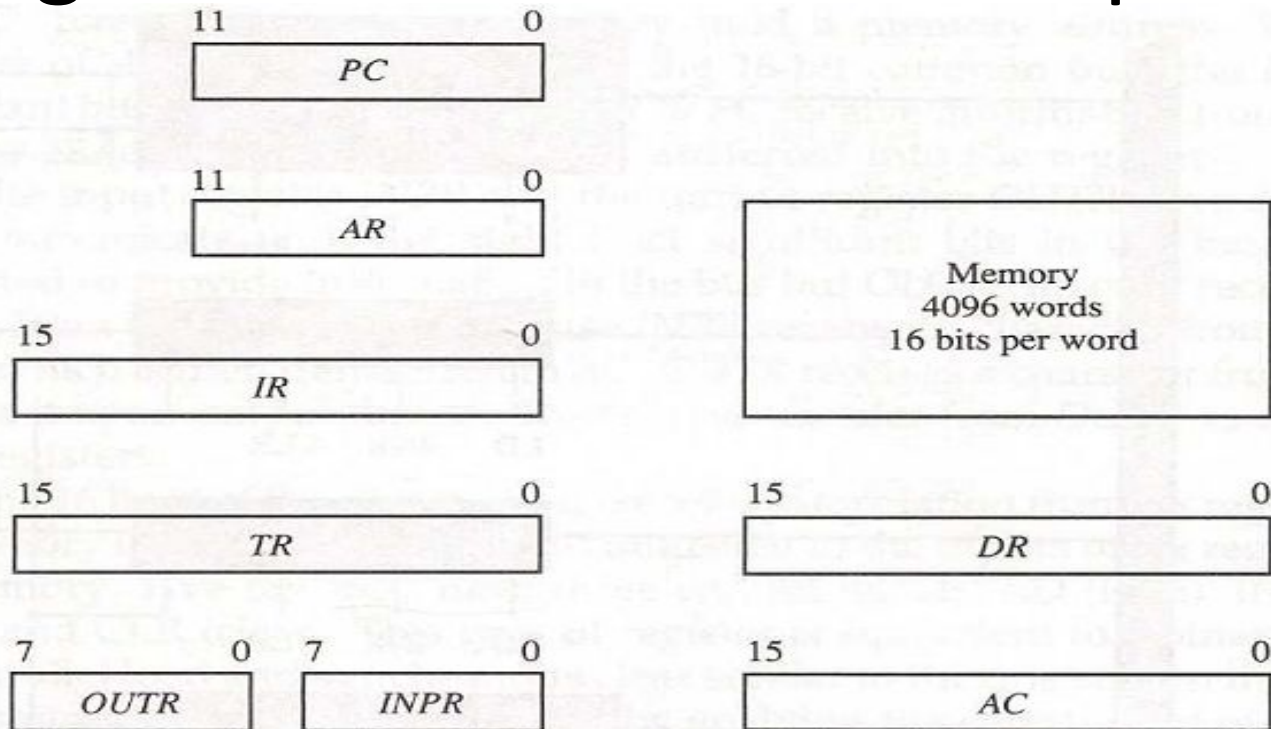
Function table				
Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

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 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 can be combined into one ALU with common selection variables.

Common Bus System & Computer Instructions

Registers for the Basic Computer

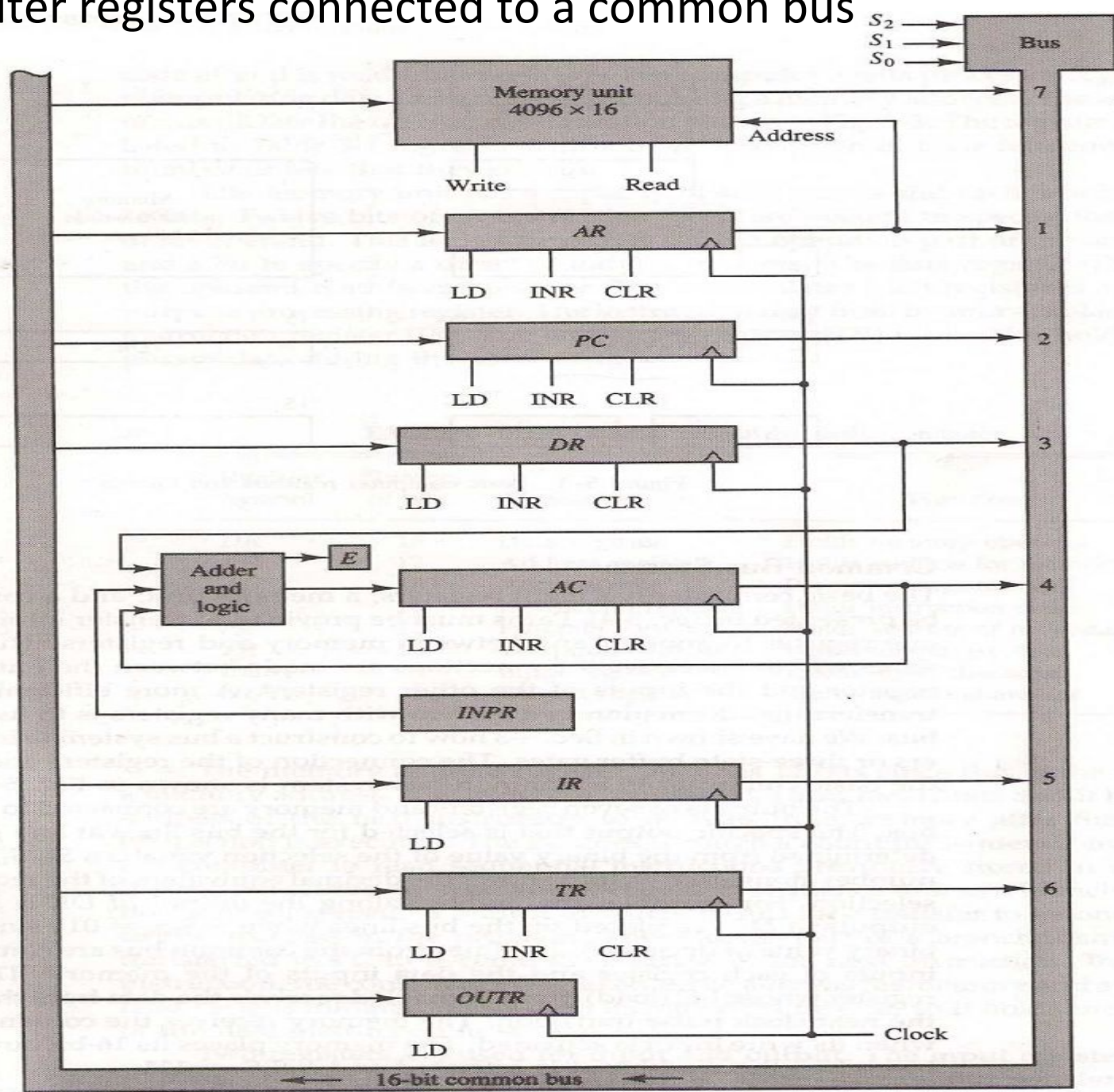


Register symbol	Number of bits	Register name	Function
<i>DR</i>	16	Data register	Holds memory operand
<i>AR</i>	12	Address register	Holds address for memory
<i>AC</i>	16	Accumulator	Processor register
<i>IR</i>	16	Instruction register	Holds instruction code
<i>PC</i>	12	Program counter	Holds address of instruction
<i>TR</i>	16	Temporary register	Holds temporary data
<i>INPR</i>	8	Input register	Holds input character
<i>OUTR</i>	8	Output register	Holds output character

Common Bus System

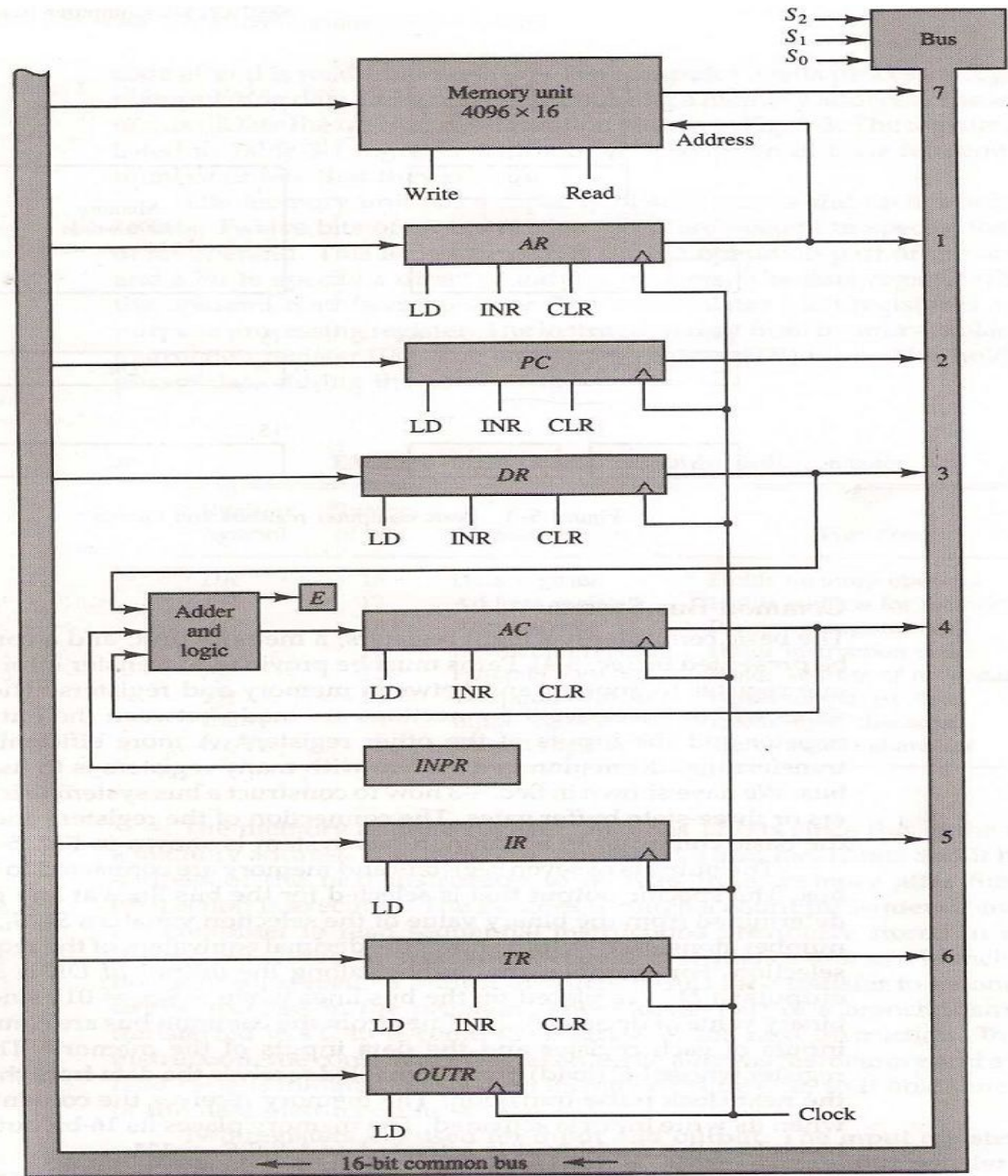
- Path must be provided to transfer information from one register to another register and between memory and registers.
- The number of wires will be excessive if connections are made between the outputs of each register and the inputs of the other registers.
- A more efficient scheme for transferring information in a system with many registers is to use a common bus system.

Basic computer registers connected to a common bus



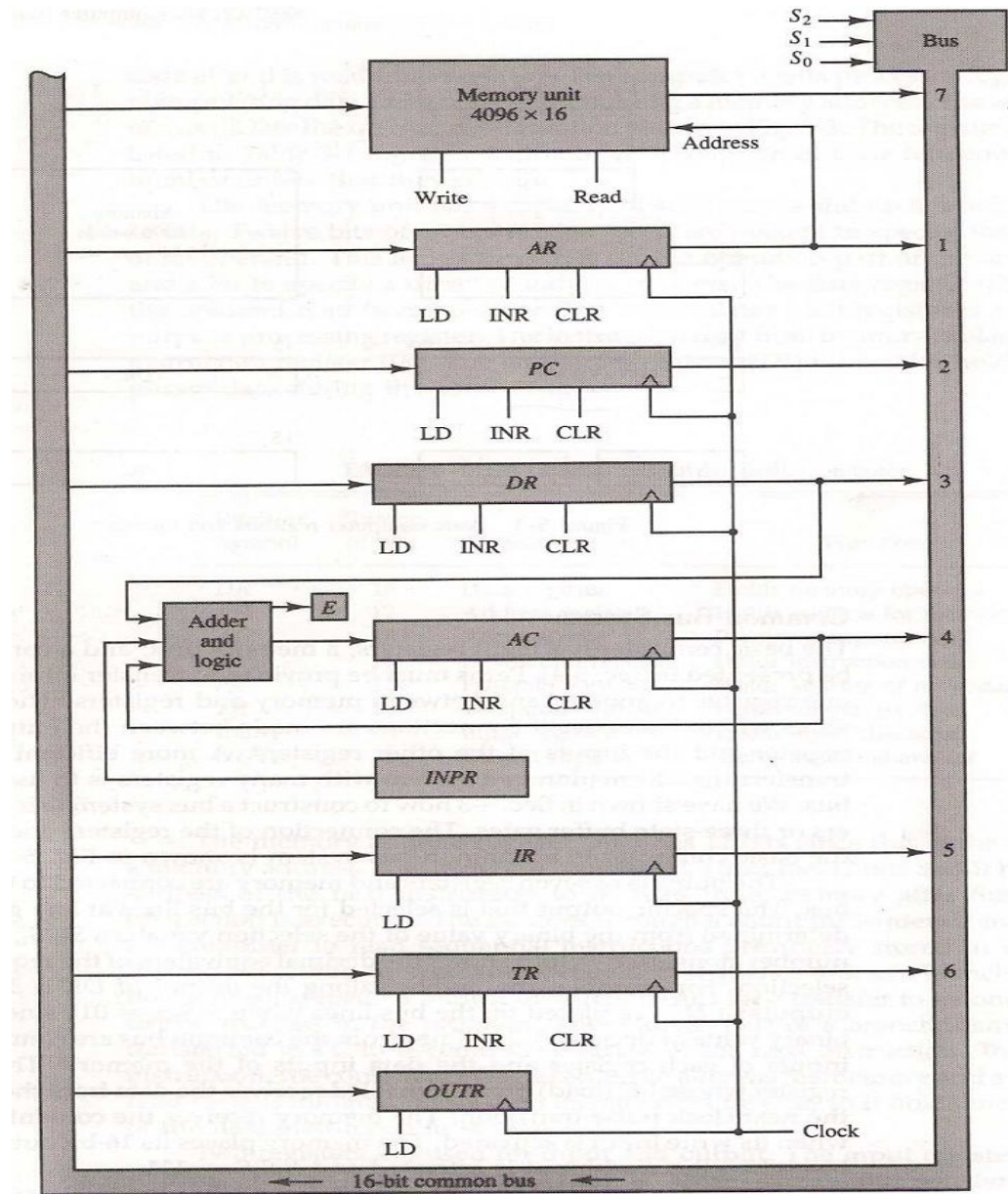
Common Bus System

- $S_2S_1S_0$: Selects the register/memory that would use the bus.
- LD (load): When enabled, the particular register receives the data from the bus during the next clock pulse transition.
- The memory receives the contents of the bus when its write input is activated.



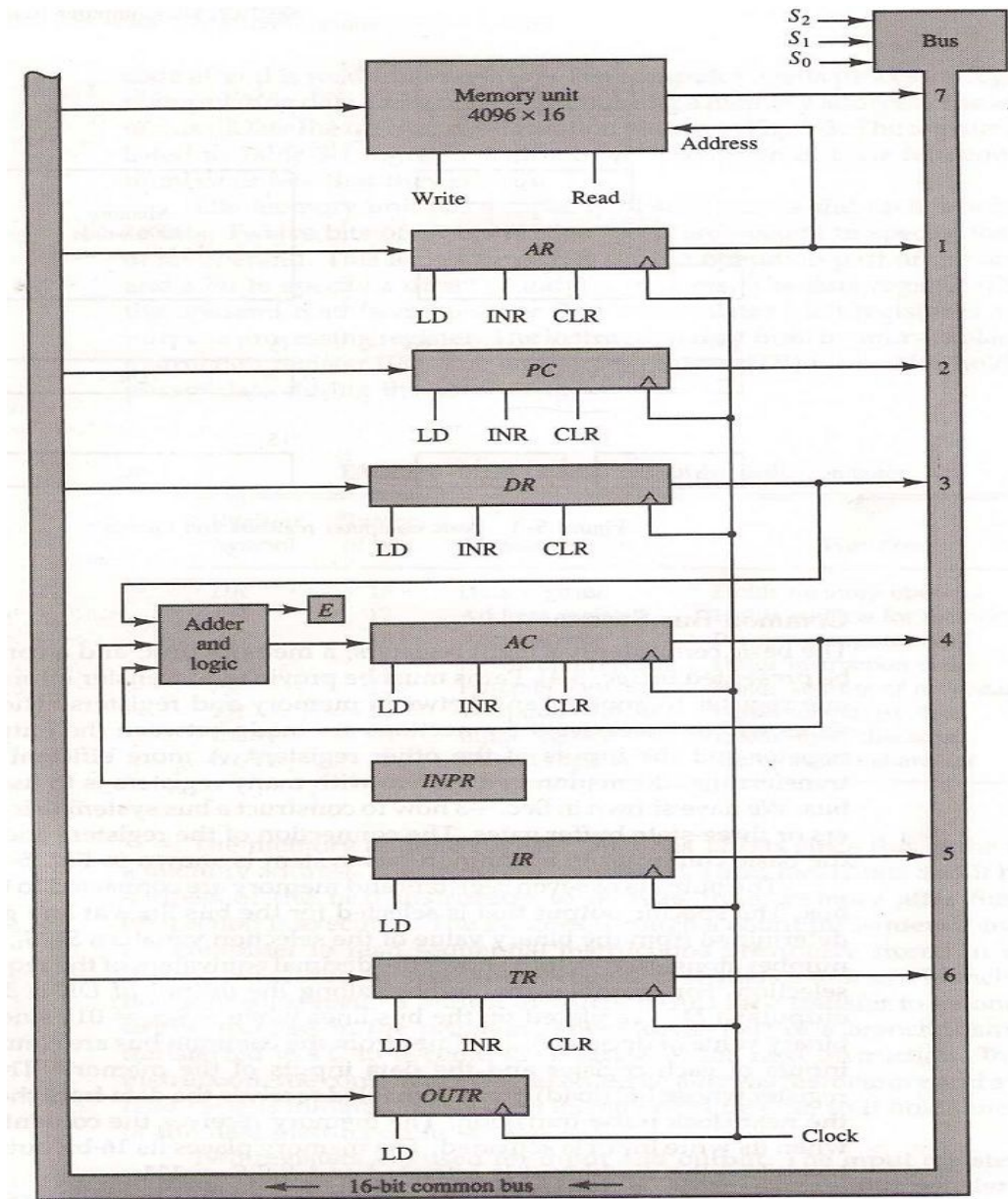
Common Bus System

- The memory places its 16-bit output onto the bus when the read input is activated and $S_2S_1S_0=111$.
- DR, AC, IR, and TR have 16 bits each.
- AR and PC: have 12 bits each since they hold a memory address



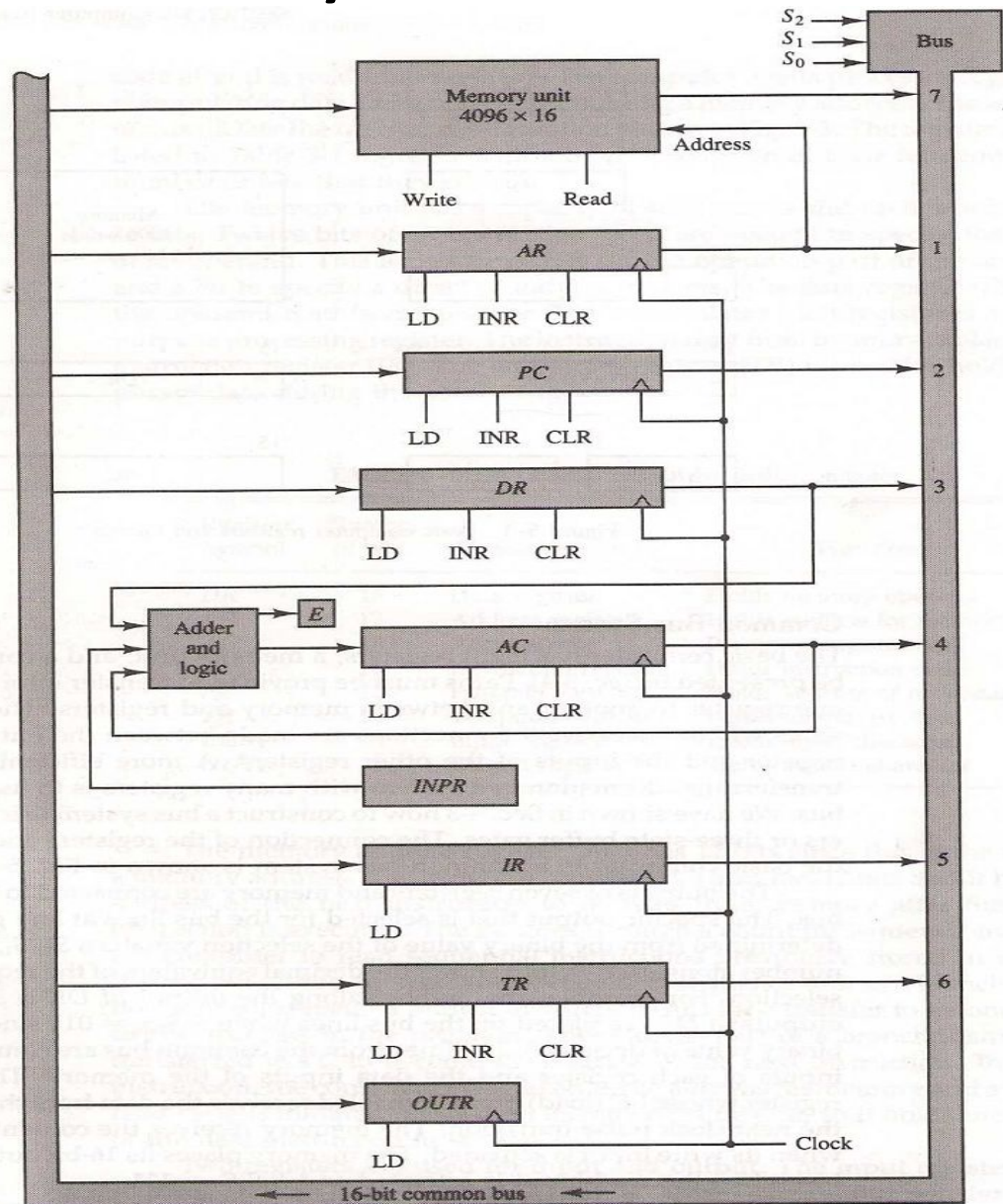
Common Bus System

- When the contents of AR or PC are applied to the 16-bit common bus, the four most significant bits are set to zeros.
- When AR or PC receives information from the bus, only the 12 least significant bits are transferred into the register.



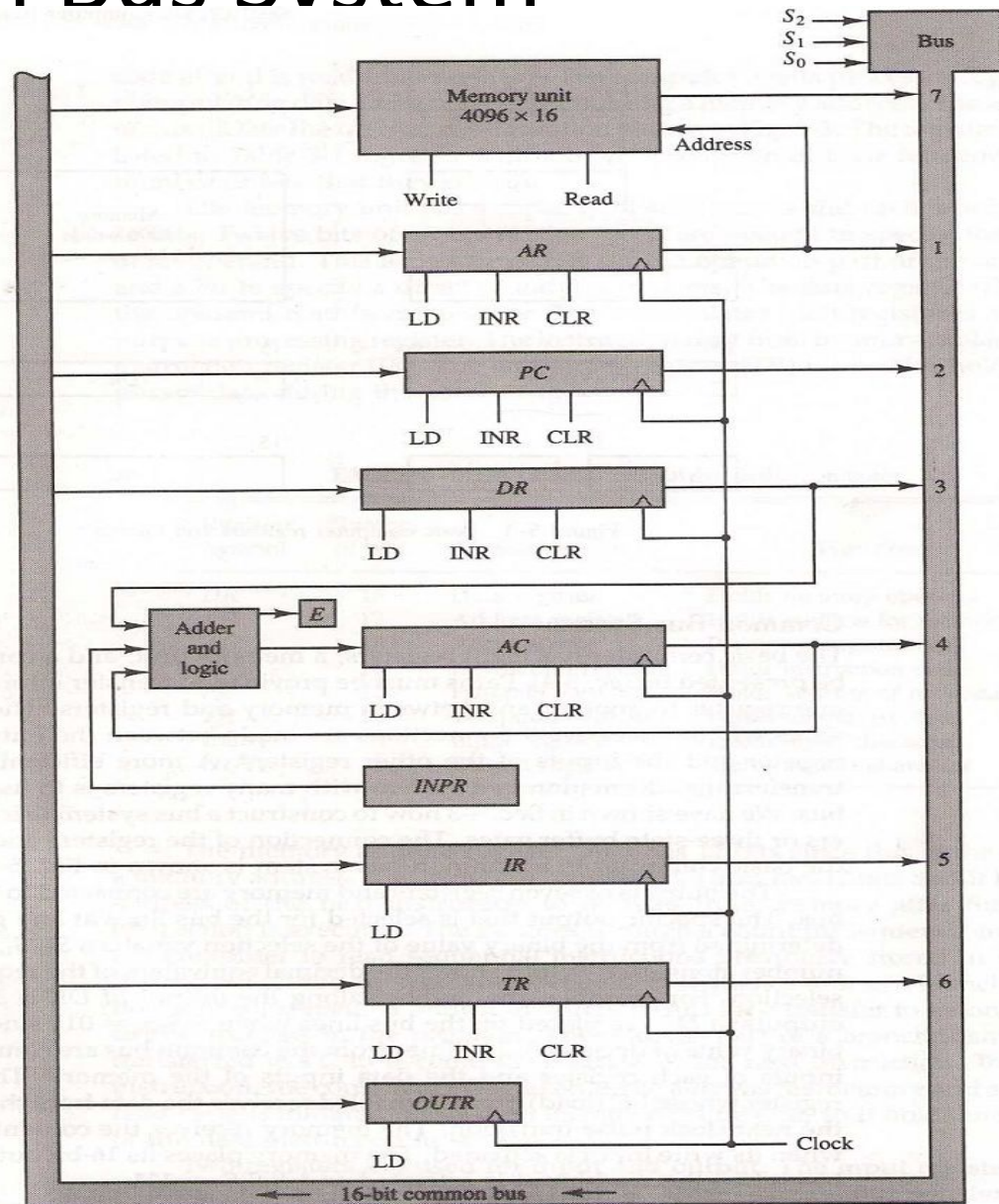
Common Bus System

- INPR and OUTF: communicate with the eight least significant bits in the bus.
- INPR: Receives a character from the input device (keyboard,...etc) which is then transferred to AC.
- OUTF: Receives a character from AC and delivers it to an output device (say a Monitor).



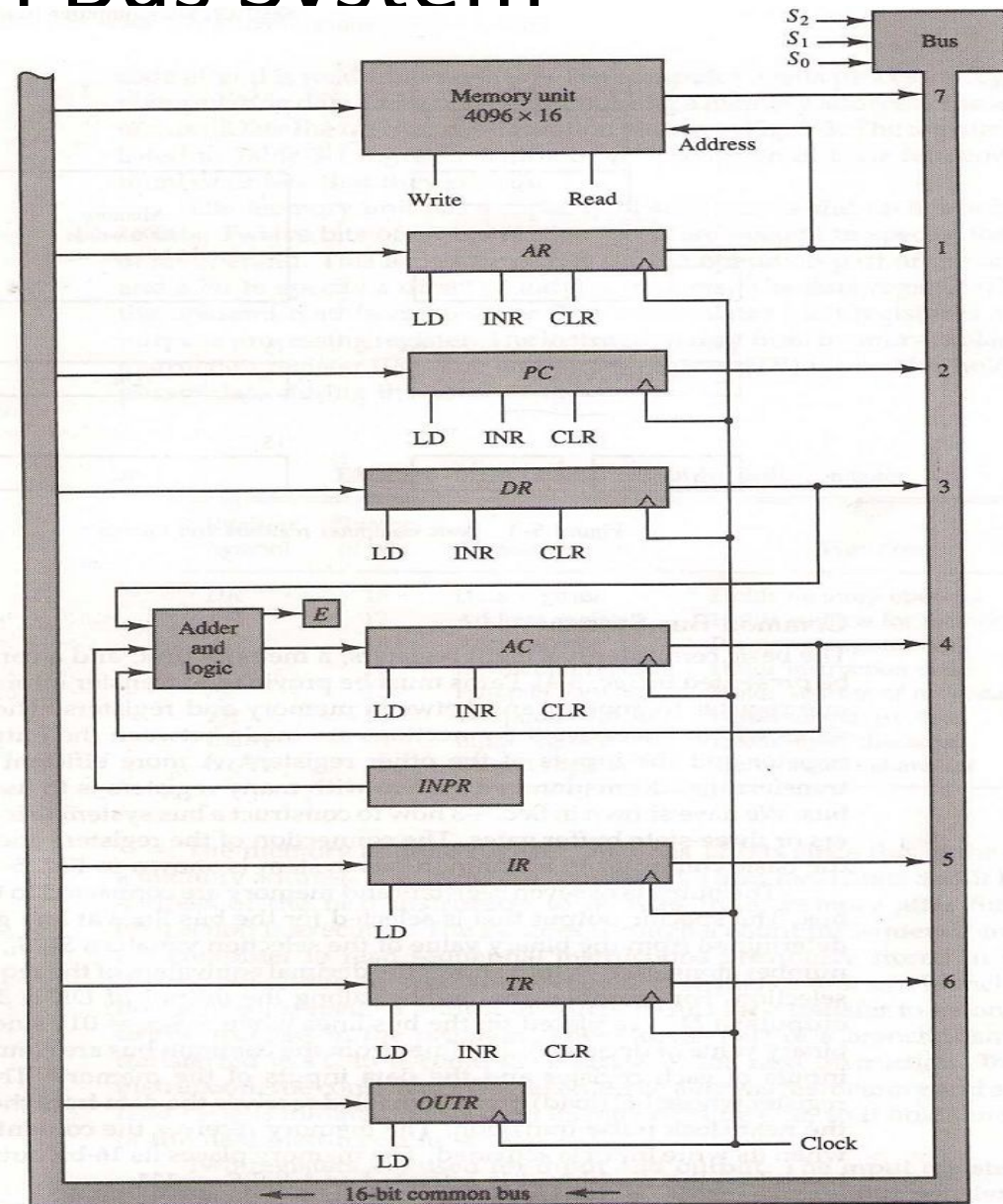
Common Bus System

- The common bus receive information from six registers and the memory unit. The bus lines are connected to the inputs of six registers and the memory.
- Five registers have three control inputs: LD (load), INR (increment), and CLR (clear). This type of register is equivalent to a binary counter with parallel load and synchronous clear.



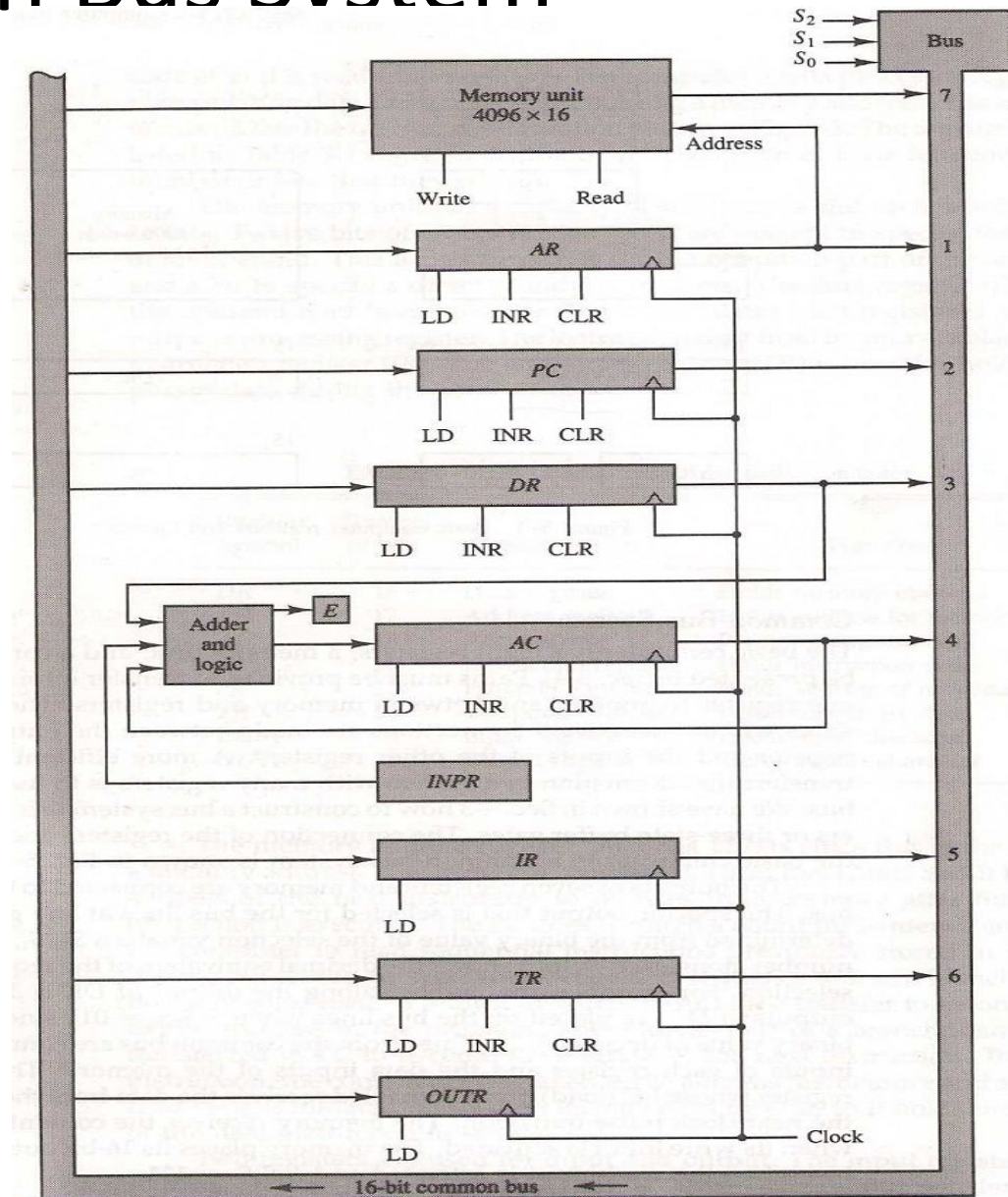
Common Bus System

- The increment operation is achieved by enabling the count input of the counter.
- Two registers have only a LD input.



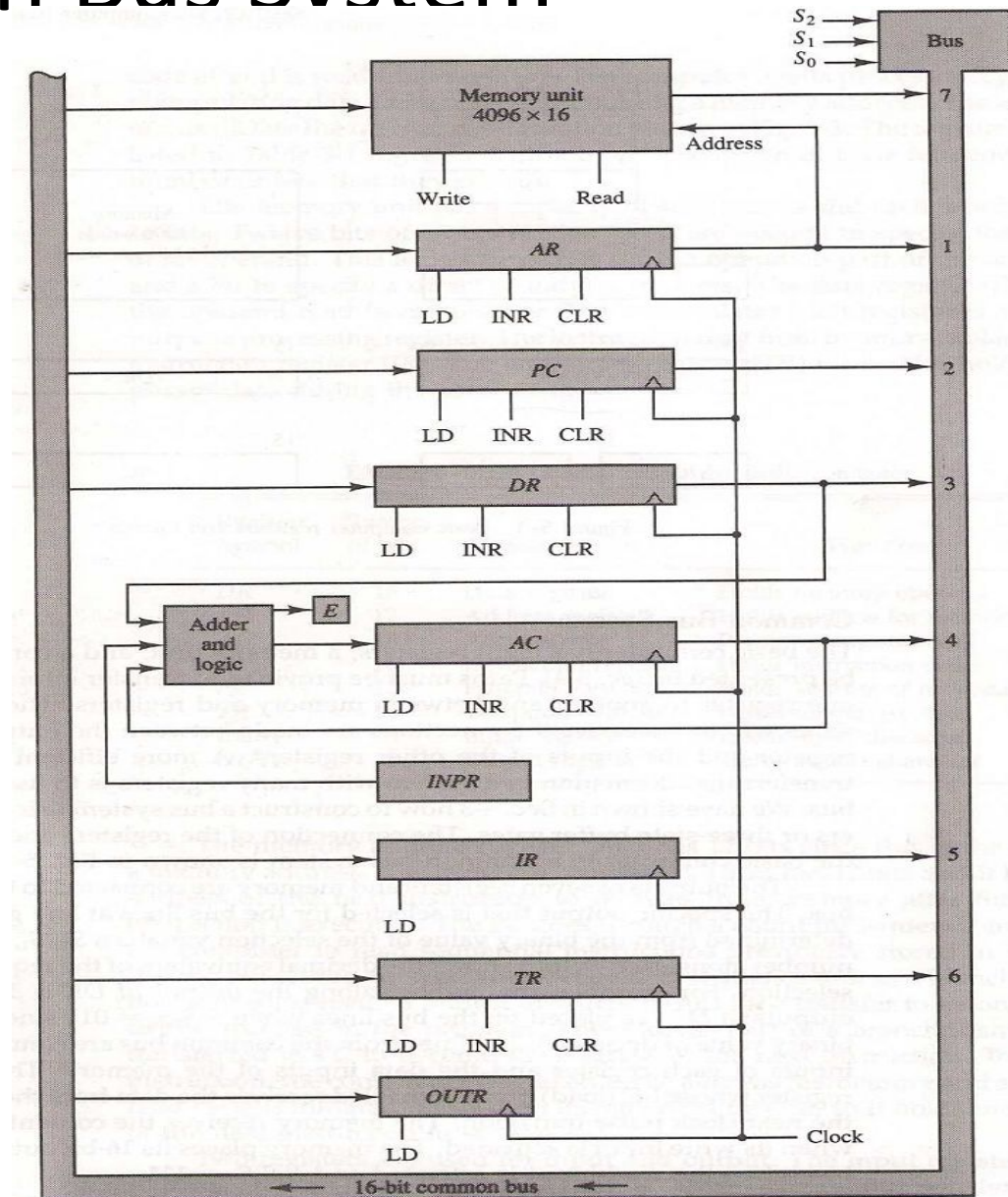
Common Bus System

- The input data and output data of the memory are connected to the common bus.
- But the memory address is connected to AR.



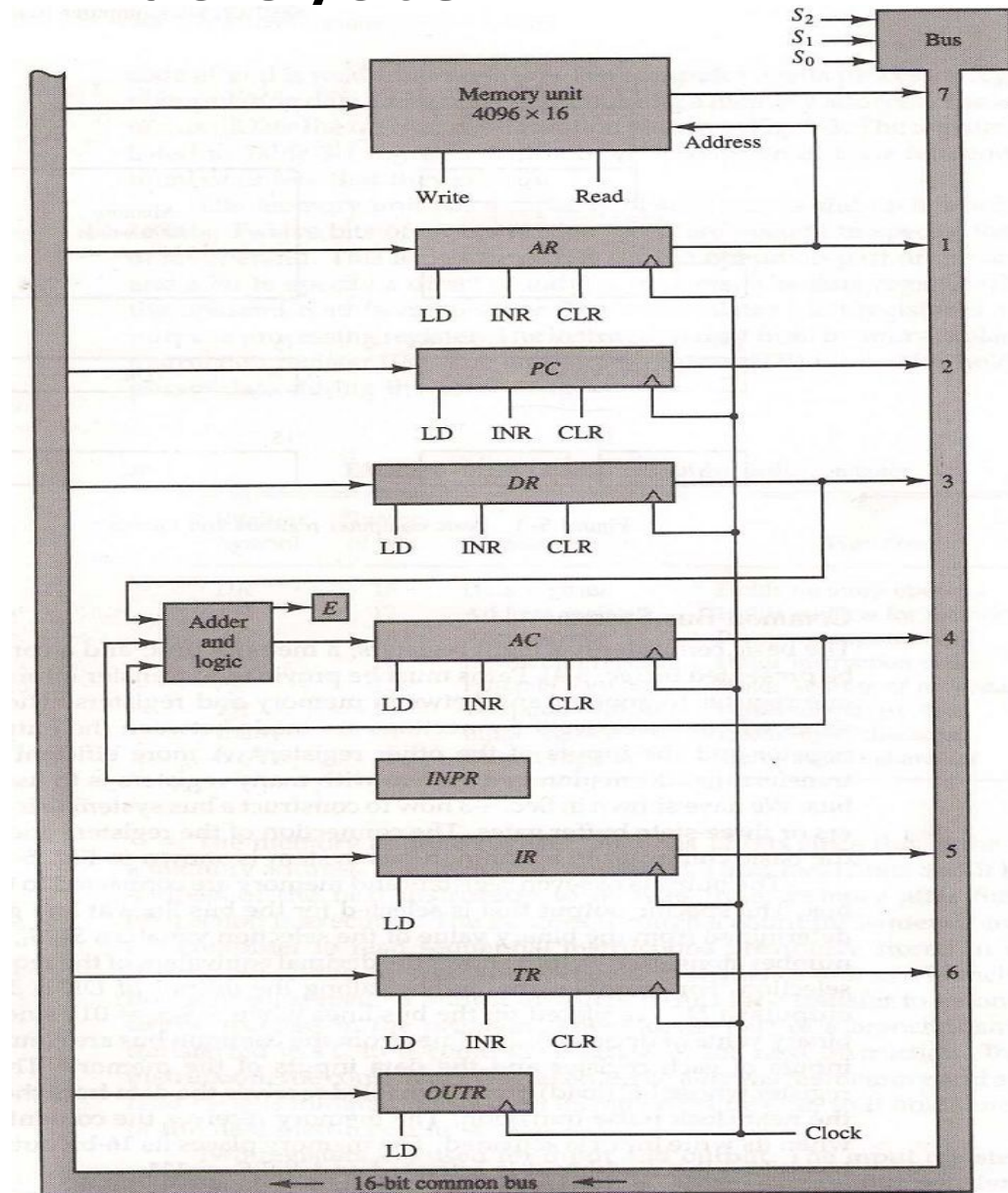
Common Bus System

- Therefore, AR must always be used to specify a memory address.
- By using a single register for the address, we eliminate the need for an address bus that would have been needed otherwise.



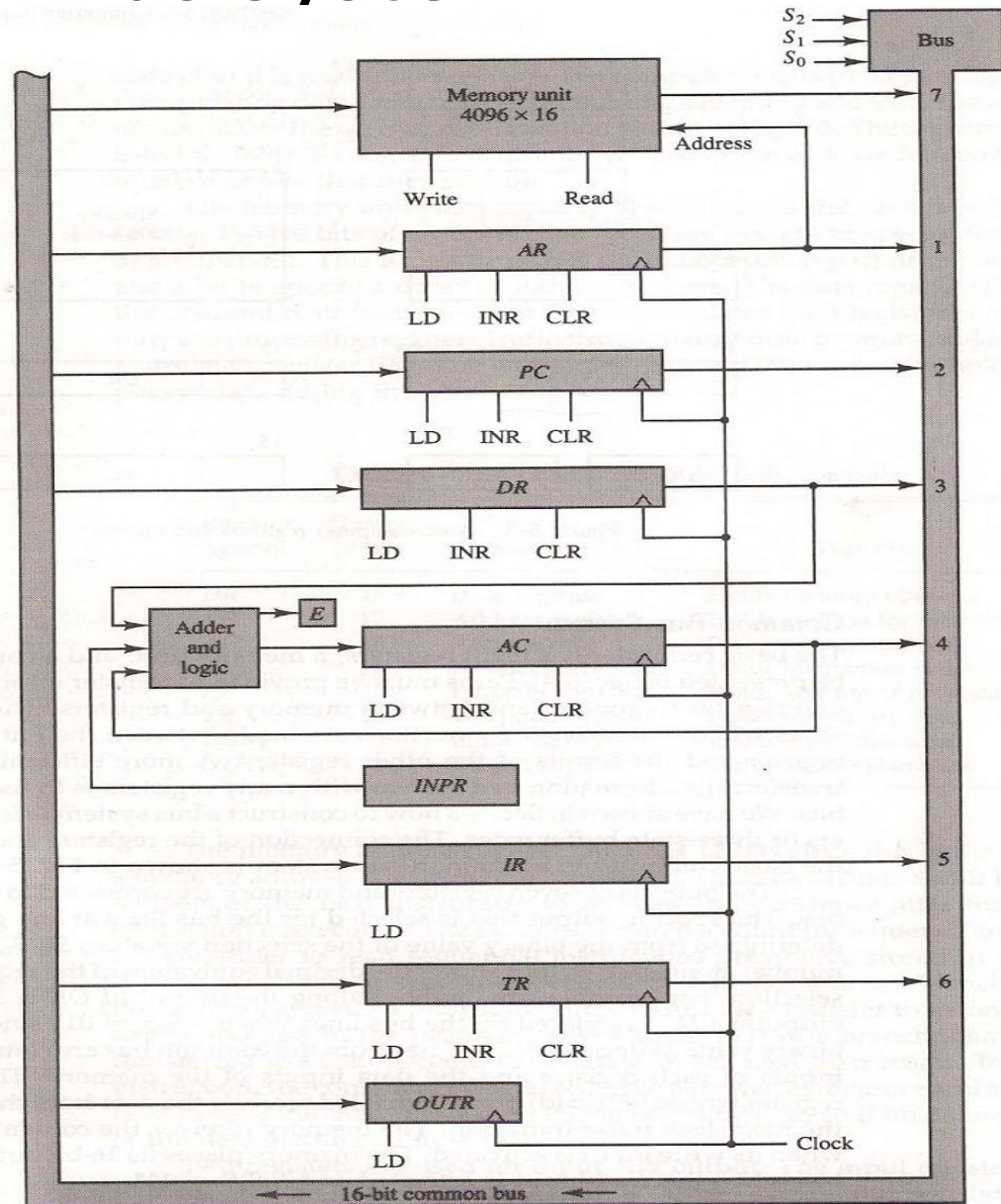
Common Bus System

- Register ☐ Memory:
Write operation
- Memory ☐ Register:
Read operation (note that AC cannot directly read from memory)
- The 16-bit inputs of AC come from an adder and logic circuit. The circuit has three sets of inputs.



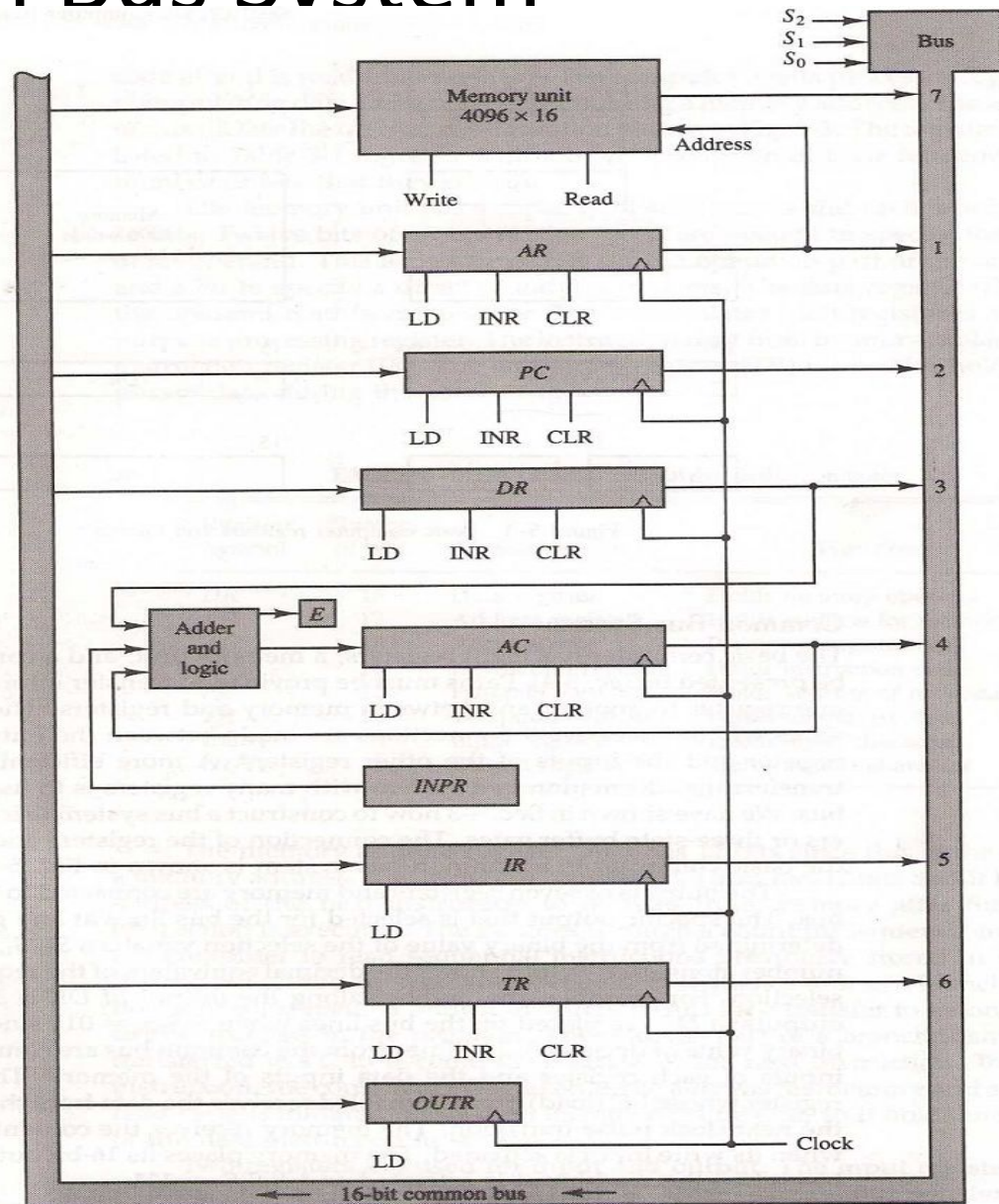
Common Bus System

- One set of 16-bit inputs come from the outputs of AC.
- They are used to implement register micro-operations such as complement AC and shift AC.
- The inputs from DR and AC are used for arithmetic and logic micro-operations, such as add DR to AC, etc.



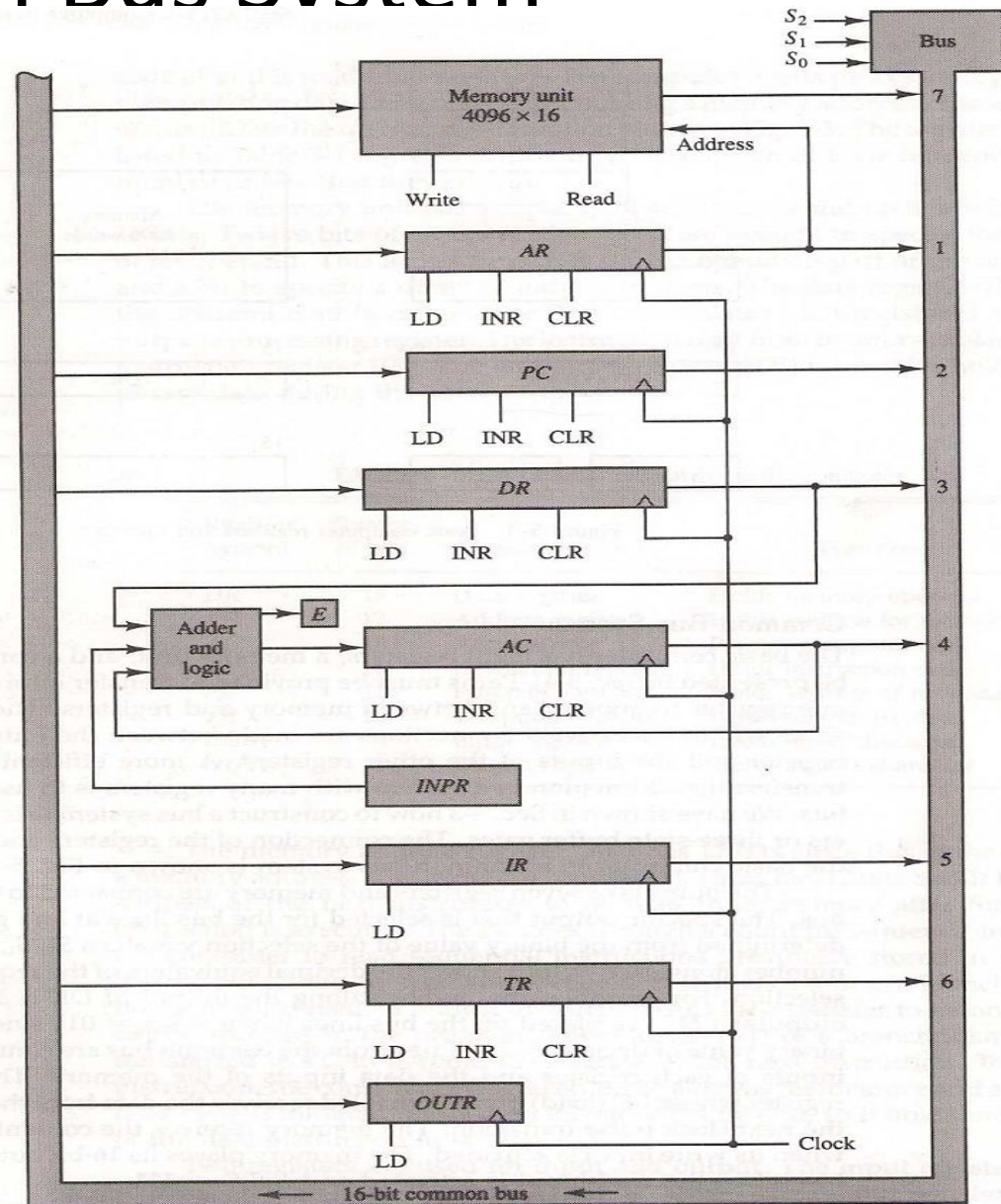
Common Bus System

- The result of an addition is transferred to AC and the end carry-out of the addition is transferred to flip-flop E (extended AC bit).
- The third set of 8-bit inputs come from the input register INPR.



Common Bus System

- The content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle



Computer Instructions

- An **Instruction** is a group of bits that instructs the computer to perform a specific operation.
- The most basic part of an instruction is its **operation code** part.
- The operation code of an instruction is a group of bits that defines **certain operations** such as add, subtract, shift, and complement.

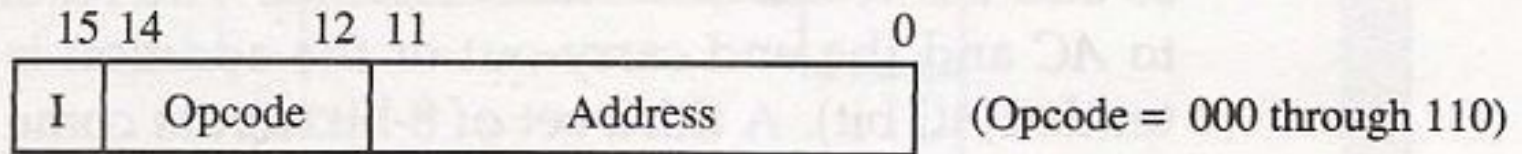
Computer Instructions

- The number of bits required for the operation code depends on the total number of operations available in the computer.
- 2^n (or little less) distinct operations
n bit operation code

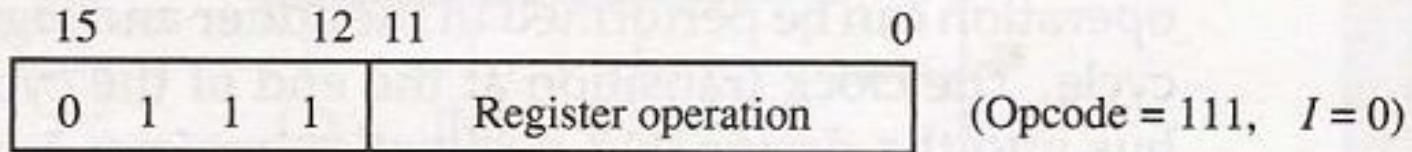
Computer Instructions

The basic computer has three instruction code formats

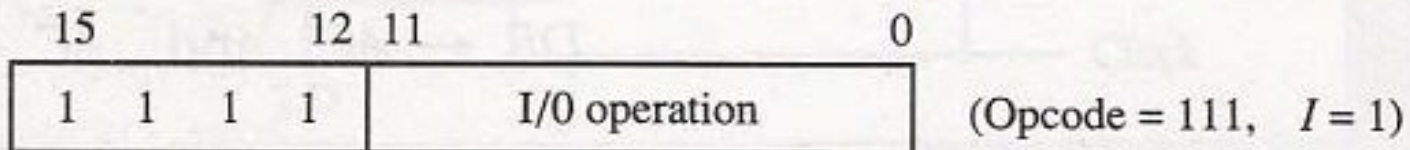
- Each format has 16 bits.



(a) Memory – reference instruction



(b) Register – reference instruction

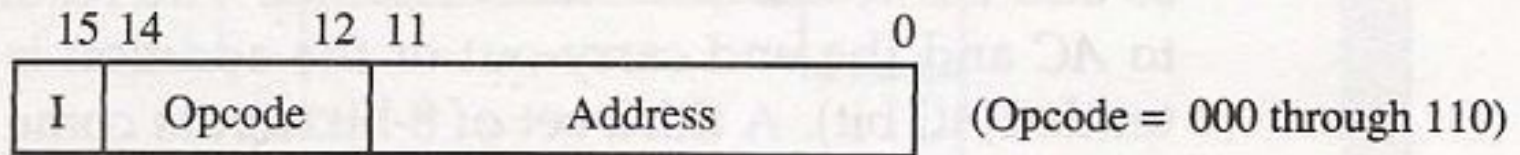


(c) Input – output instruction

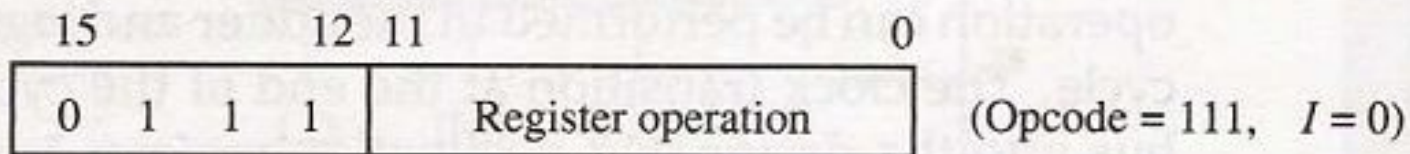
Fig: Basic computer instruction formats

Computer Instructions

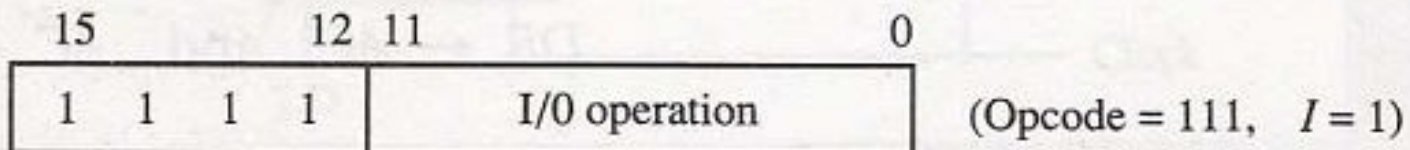
A **memory-reference instruction** uses one bit to specify the addressing mode I . $I = 0$ for direct, $1 = \text{Indirect}$



(a) Memory – reference instruction



(b) Register – reference instruction

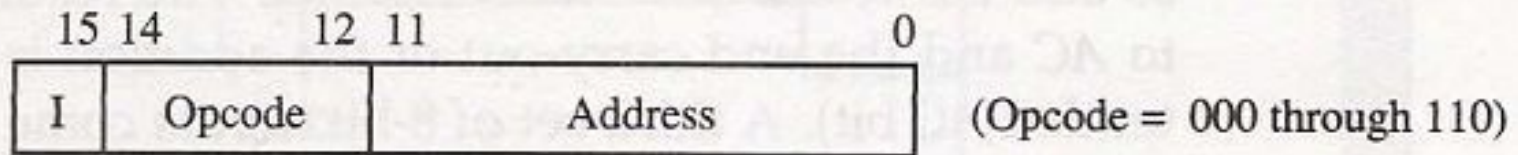


(c) Input – output instruction

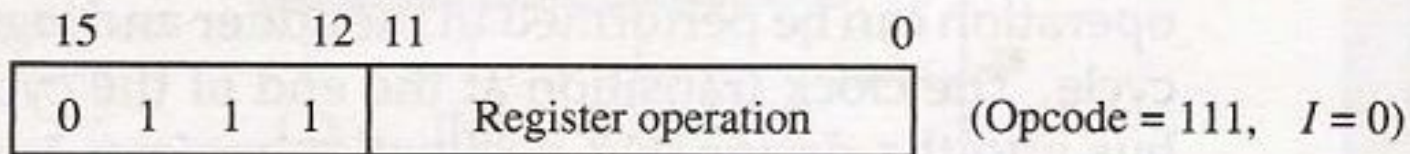
Fig: Basic computer instruction formats

Computer Instructions

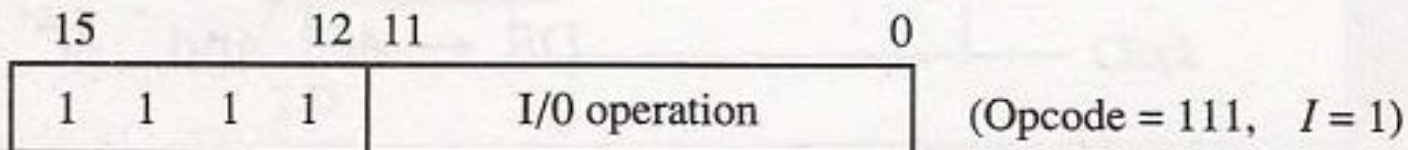
A **register-reference instruction** specifies an operation on AC, other 12 bits are used to specify the operation



(a) Memory – reference instruction



(b) Register – reference instruction

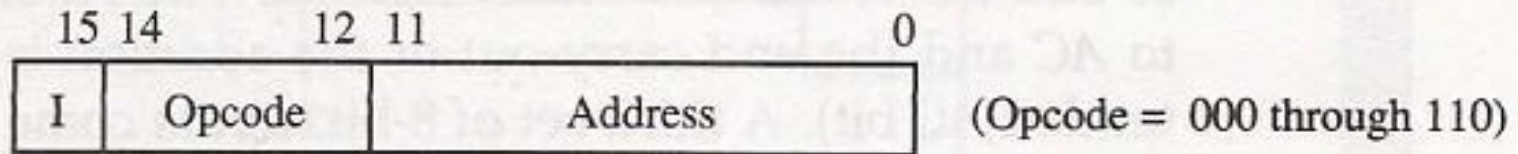


(c) Input – output instruction

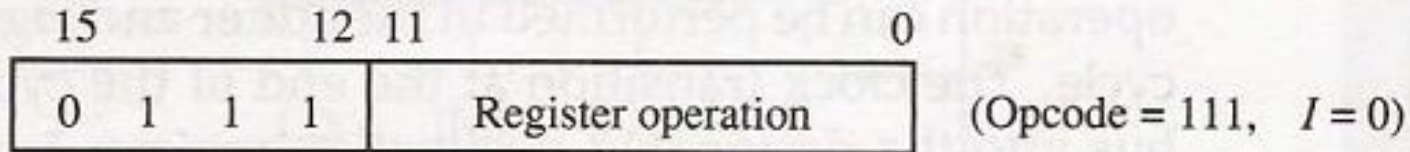
Fig: Basic computer instruction formats

Computer Instructions

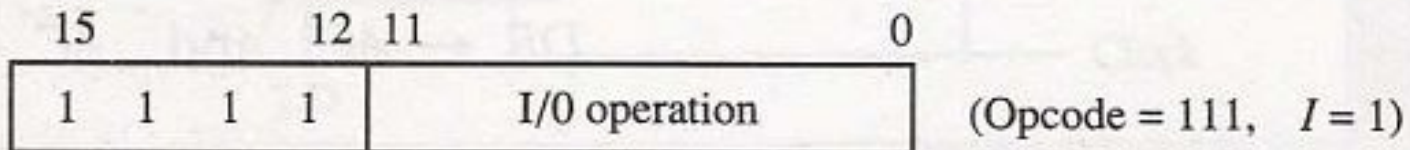
An **IO instruction** does not need a reference to memory, remaining 12 bits are used to specify the type of IO operation



(a) Memory – reference instruction



(b) Register – reference instruction



(c) Input – output instruction

Fig: Basic computer instruction formats

Basic computer instructions

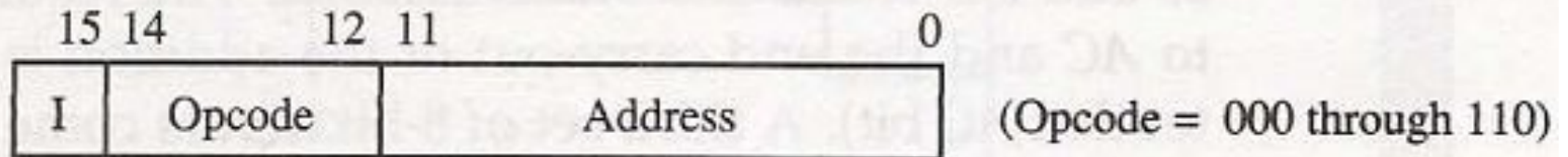
Symbol	Hexadecimal code		Description
	$I = 0$	$I = 1$	
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Tab: Basic computer instructions

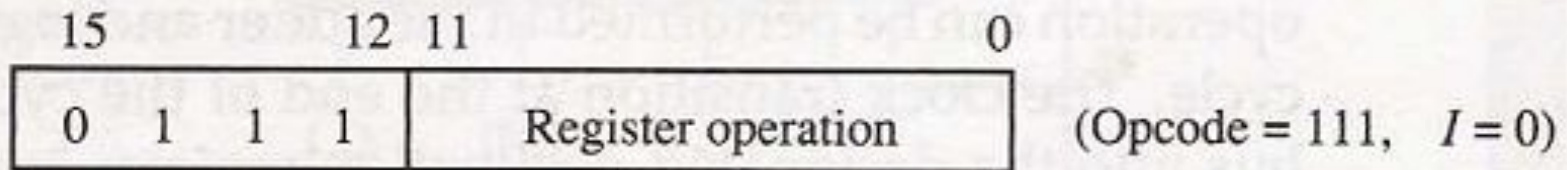
Computer Instructions

Control Unit with Timing,
Instruction Cycle and
Determine the Type of Instruction

Computer Instructions



(a) Memory – reference instruction



(b) Register – reference instruction

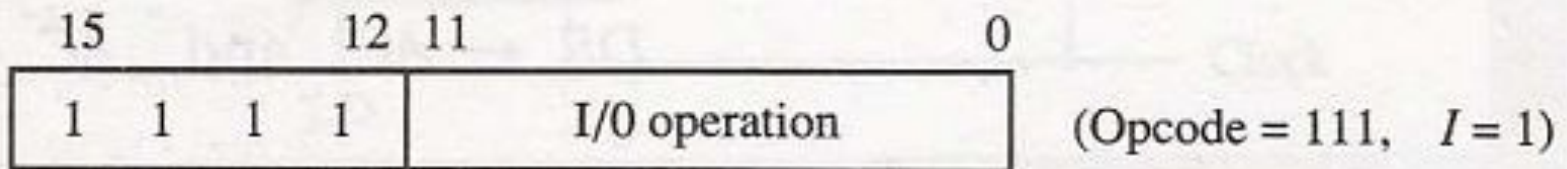


Fig: Basic Computer Instruction code format

Computer Instructions

- 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 direct address and to 1 for indirect address.

Computer Instructions

- The **register-reference** instructions are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction. A register-reference instruction specifies an operation on or a test of the AC register. An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test 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 or test performed.

Basic Computer Instructions

The type of instruction is recognized by the computer control from the four bits in positions 12 through 15 of the instruction.

- **Memory-reference:** If the three opcode bits in position 12 through 14 are not equal to 111.
- Bit 15 is taken as the addressing mode *I*.

If the 3 bits opcode is equal to 111, control then inspects the bit in position 15.

- **Register-reference:** If 15 bit is 0.
- **Input-output:** If 15 bit is 1.

Basic Computer Instructions

Symbol	Hexadecimal code		Description	
	<i>I</i> = 0	<i>I</i> = 1		
AND	0xxx	8xxx	AND memory word to <i>AC</i>	Memory-reference type
ADD	1xxx	9xxx	Add memory word to <i>AC</i>	
LDA	2xxx	Axxx	Load memory word to <i>AC</i>	
STA	3xxx	Bxxx	Store content of <i>AC</i> in memory	
BUN	4xxx	Cxxx	Branch unconditionally	
BSA	5xxx	Dxxx	Branch and save return address	
ISZ	6xxx	Exxx	Increment and skip if zero	
CLA		7800	Clear <i>AC</i>	Register-reference type
CLE		7400	Clear <i>E</i>	
CMA		7200	Complement <i>AC</i>	
CME		7100	Complement <i>E</i>	
CIR		7080	Circulate right <i>AC</i> and <i>E</i>	
CIL		7040	Circulate left <i>AC</i> and <i>E</i>	
INC		7020	Increment <i>AC</i>	
SPA		7010	Skip next instruction if <i>AC</i> positive	
SNA		7008	Skip next instruction if <i>AC</i> negative	
SZA		7004	Skip next instruction if <i>AC</i> zero	
SZE		7002	Skip next instruction if <i>E</i> is 0	
HLT		7001	Halt computer	
INP		F800	Input character to <i>AC</i>	Input-output type
OUT		F400	Output character from <i>AC</i>	
SKI		F200	Skip on input flag	
SKO		F100	Skip on output flag	
ION		F080	Interrupt on	
IOF		F040	Interrupt off	

Basic Computer Instructions

The 16 bits of an instruction code is reduced to equivalent four digits hexadecimal digits.

- **Memory-reference:** the address part is denoted by three x's and stand for the three hexadecimal digits.
- **Register-reference:** The leftmost four bits are always 0111, which is equivalent to hexadecimal 7.
- **Input-output:** The last four bits are always 1111, equivalent to hexadecimal F.

Control Unit with Timing

- 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** (i.e., Load)

Control Unit with Timing

- 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 micro-operations for the accumulator

There are two major types of control organization:

- Hardwired control
- Micro-programmed control

Control Unit with Timing

- **Hardwired Organization:** the control logic is implemented with gates, flip-flops, decoders, and other digital circuits.
- **Micro-programmed Organization:** the control information is stored in a control memory (if the design is modified, the micro-program in control memory has to be updated)

Control Unit of basic computer

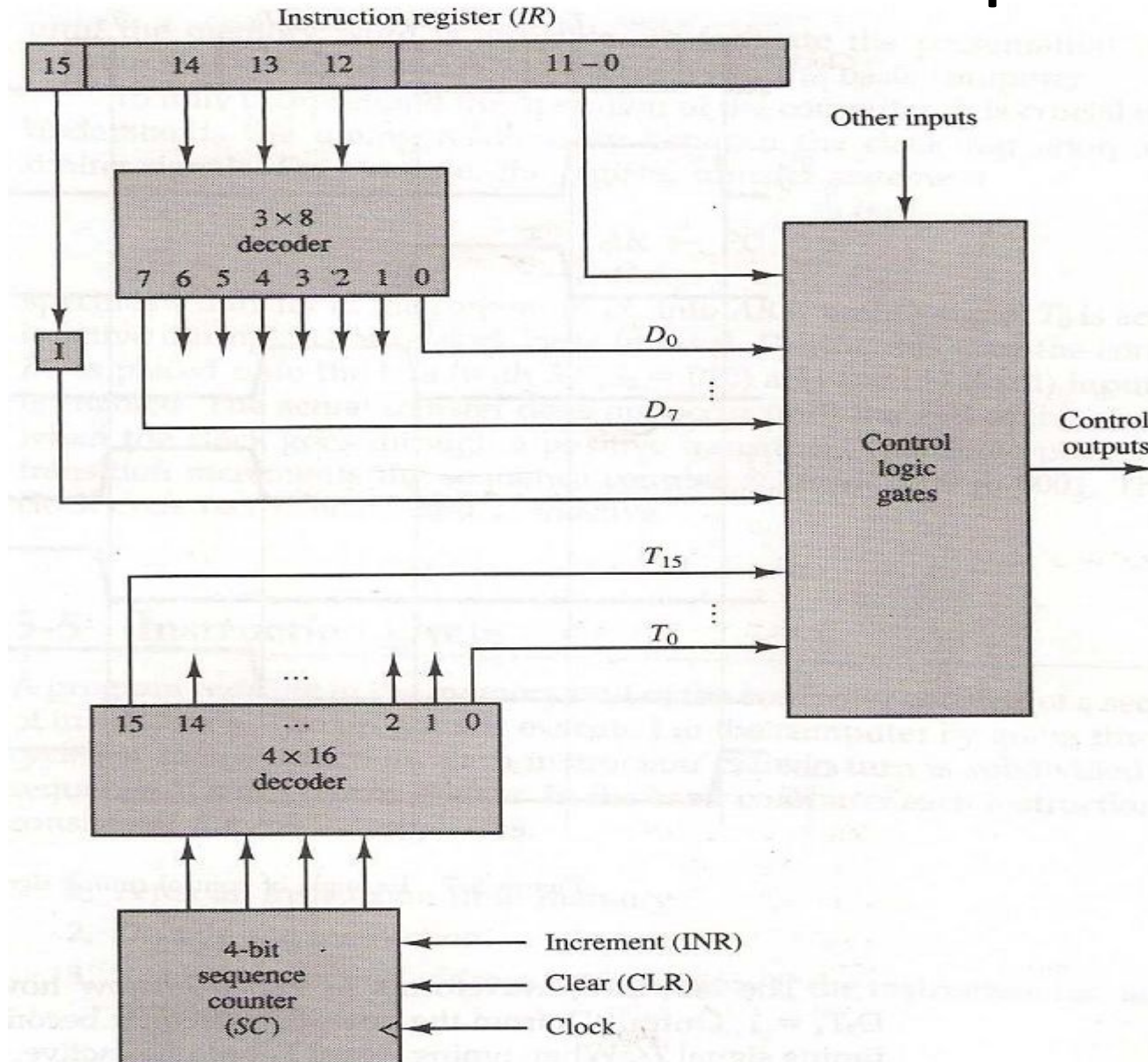


Fig: Block diagram of the hardwired control unit

Control Unit of basic computer

- 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)

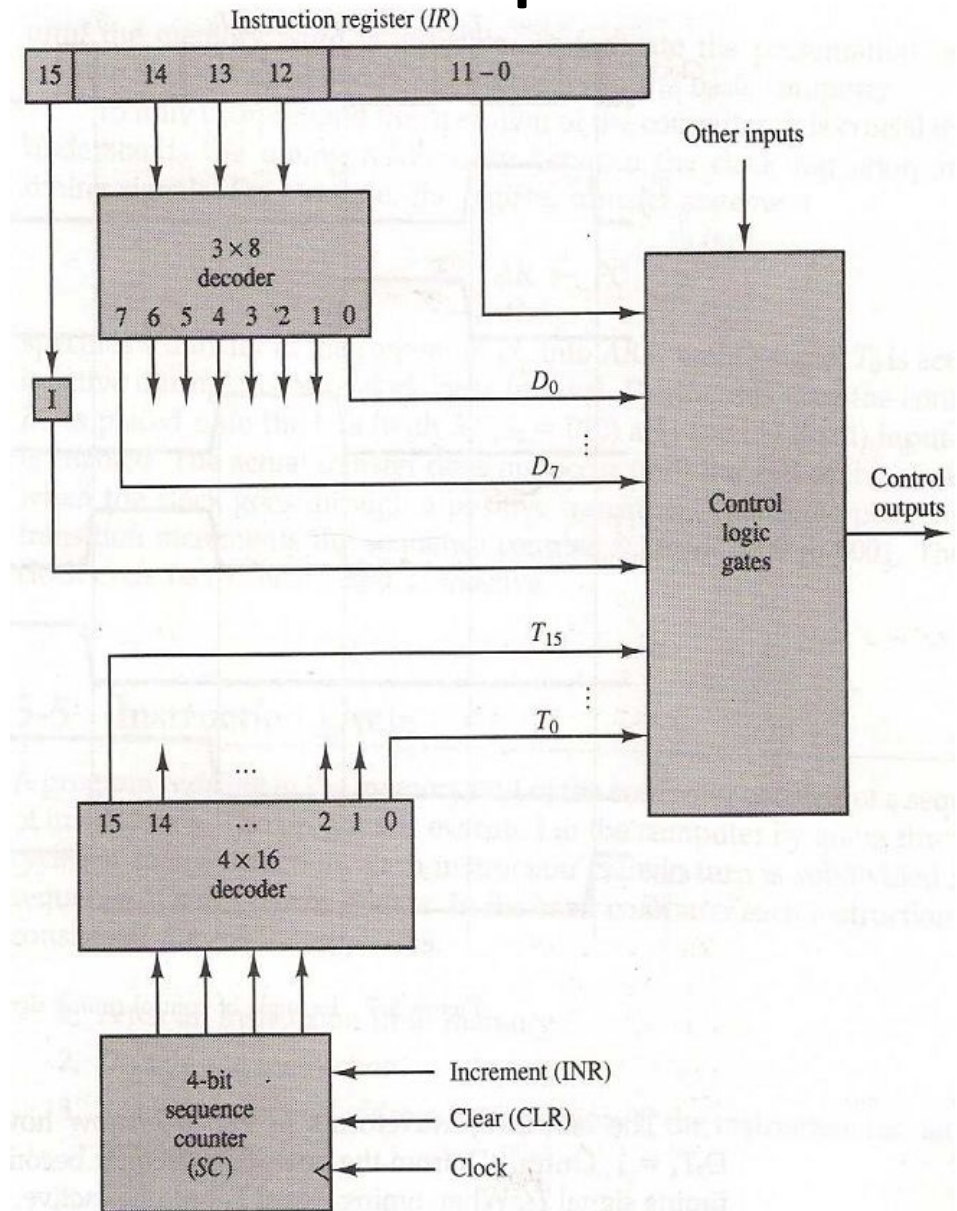


Fig: Block diagram of the hardwired control unit

Control Unit of basic computer

- The opcode in bits are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D_0 through D_7 .
- Bit 15 of the instruction is transferred to a flip-flop I.
- Bits 0 through 11 are applied to the control logic gates.

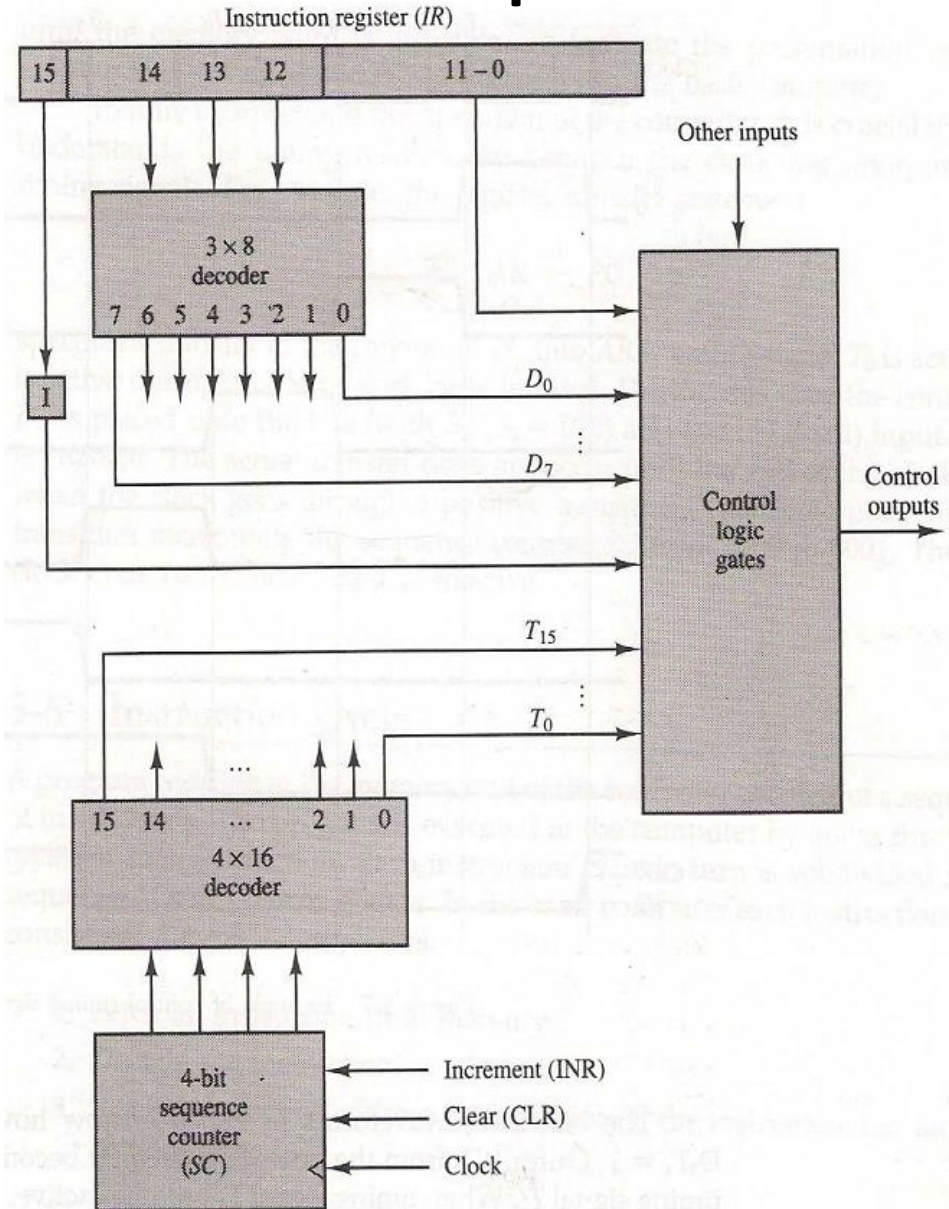


Fig: Block diagram of the hardwired control unit

Control Unit of basic computer

- 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 sequence counter SC can be incremented or cleared synchronously.

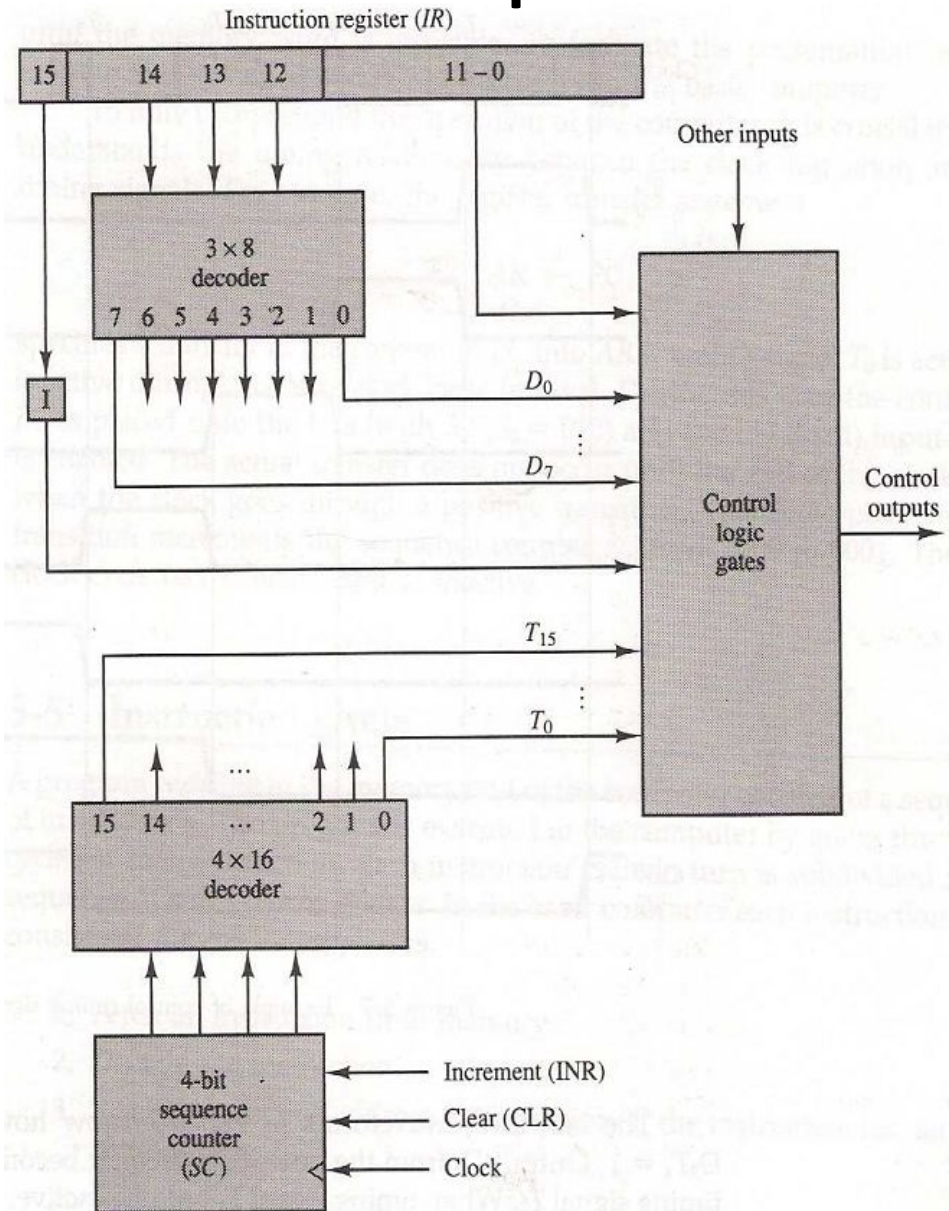


Fig: Block diagram of the hardwired control unit

Control Unit of basic computer

- The counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder.

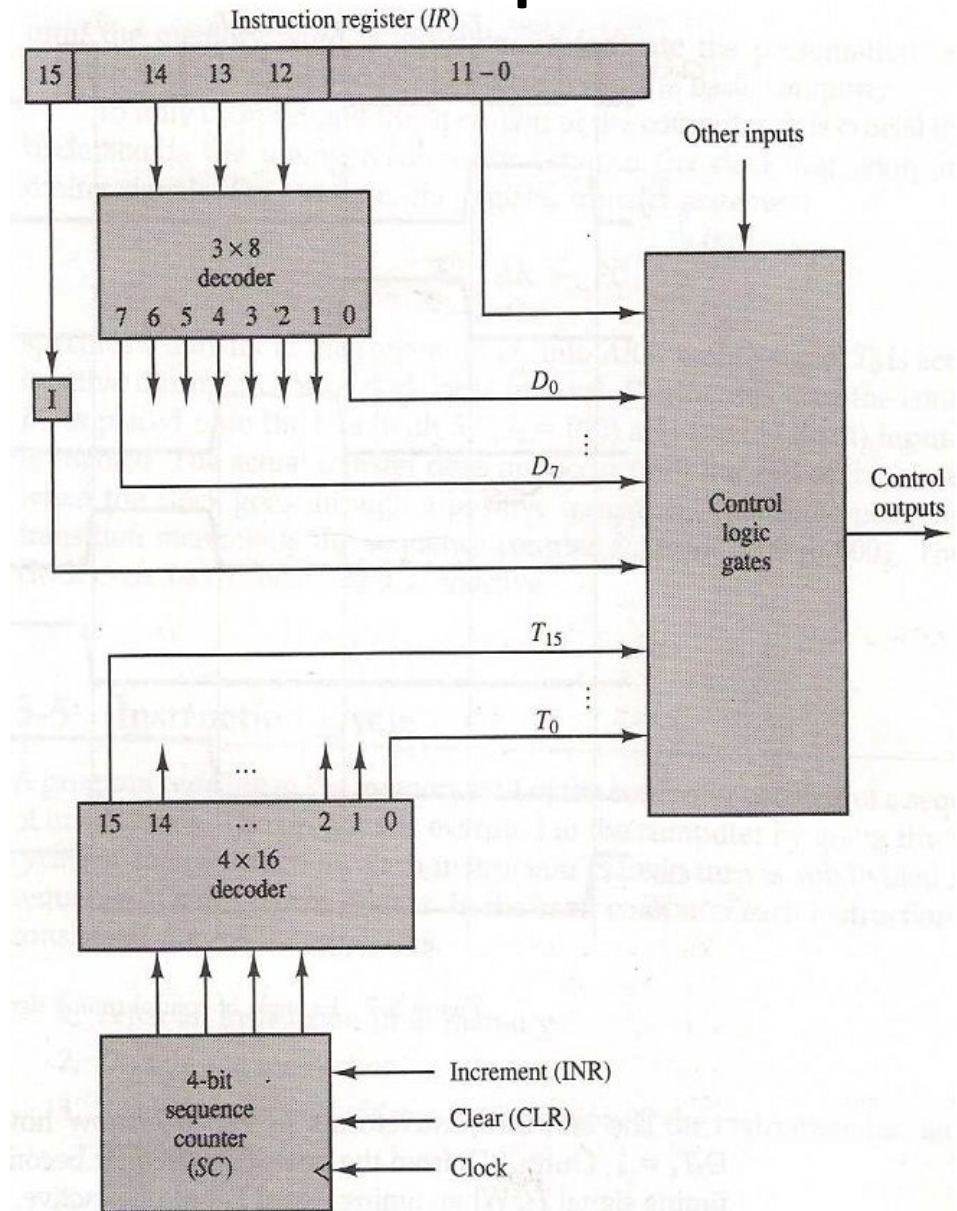
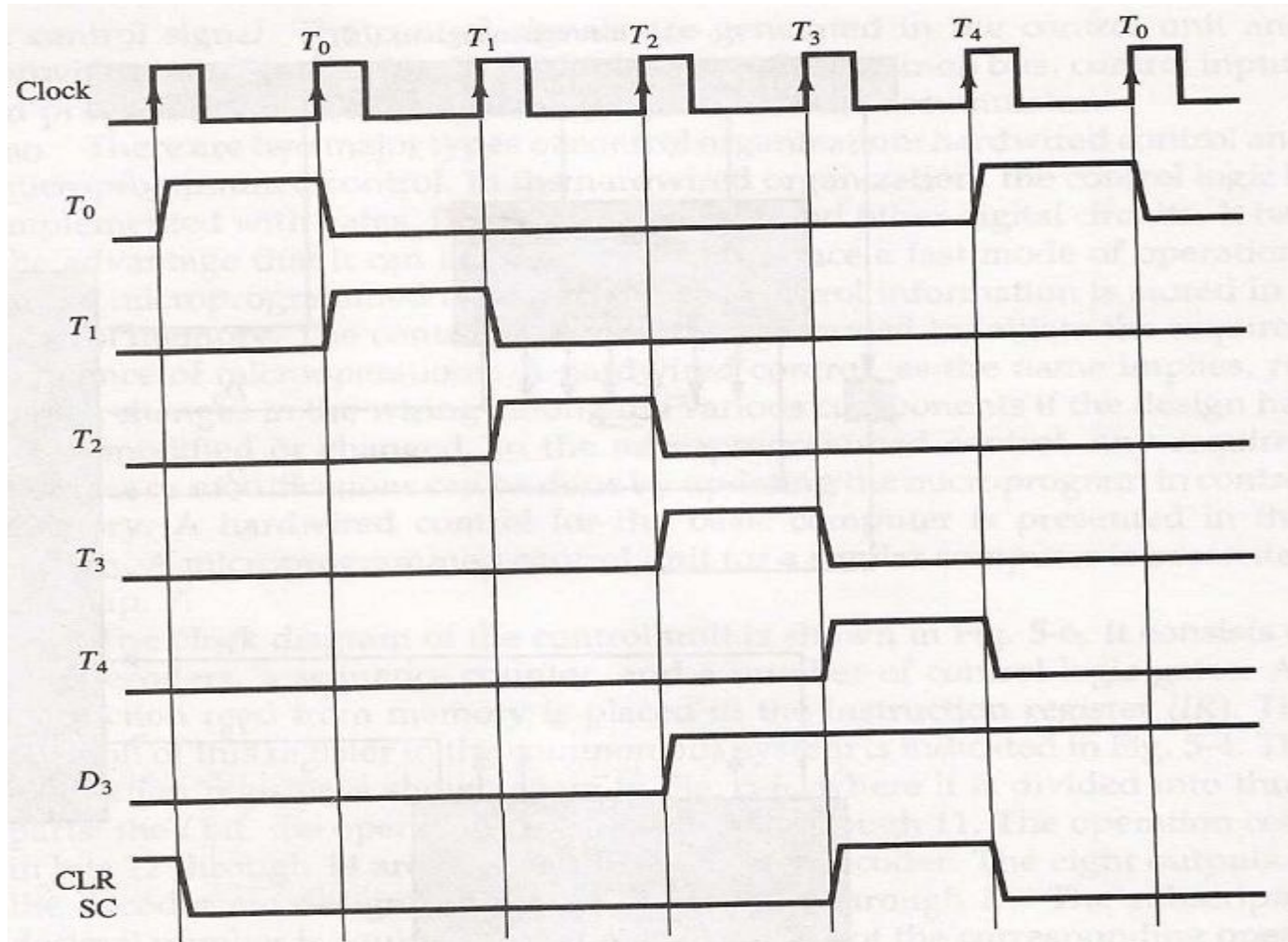


Fig: Block diagram of the hardwired control unit

Control Timing Signals



Control Timing Signals

- The timing diagram shows the time relationship of the control signals.
- The sequence counter SC responds to the positive transition of the clock.

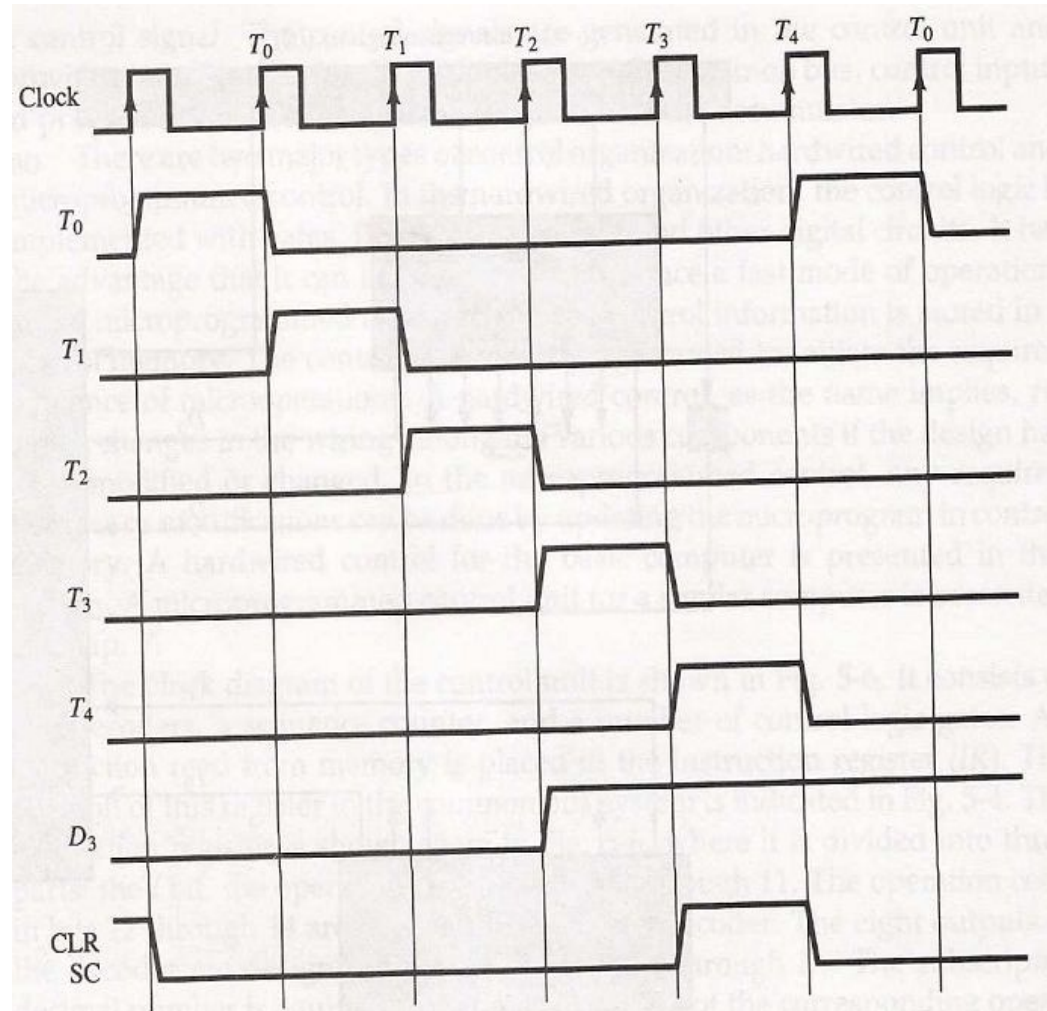


Fig: Diagram of Control Timing Signals

Control Timing Signals

- Initially, the CLR input of SC is active. The first positive transition of the clock 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.

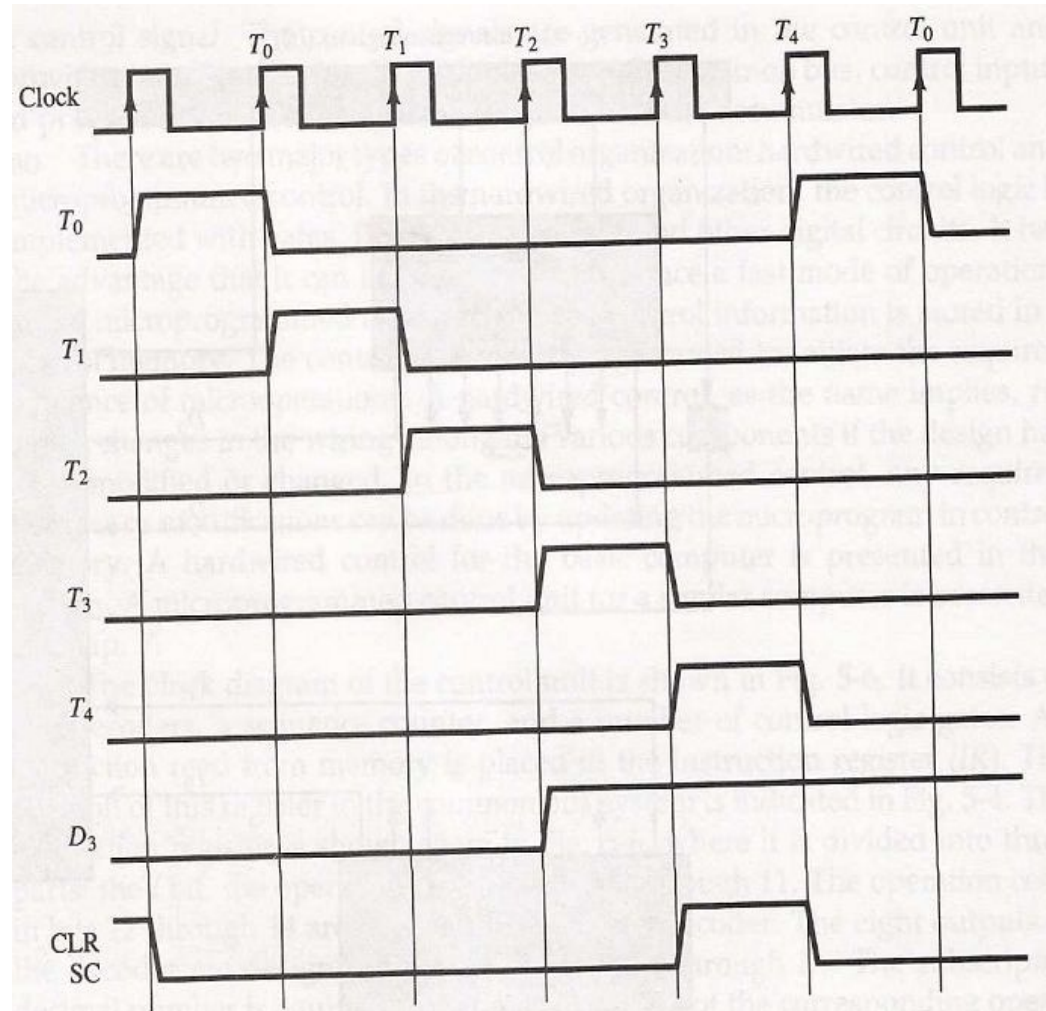


Fig: Diagram of Control Timing Signals

Control Timing Signals

- 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.

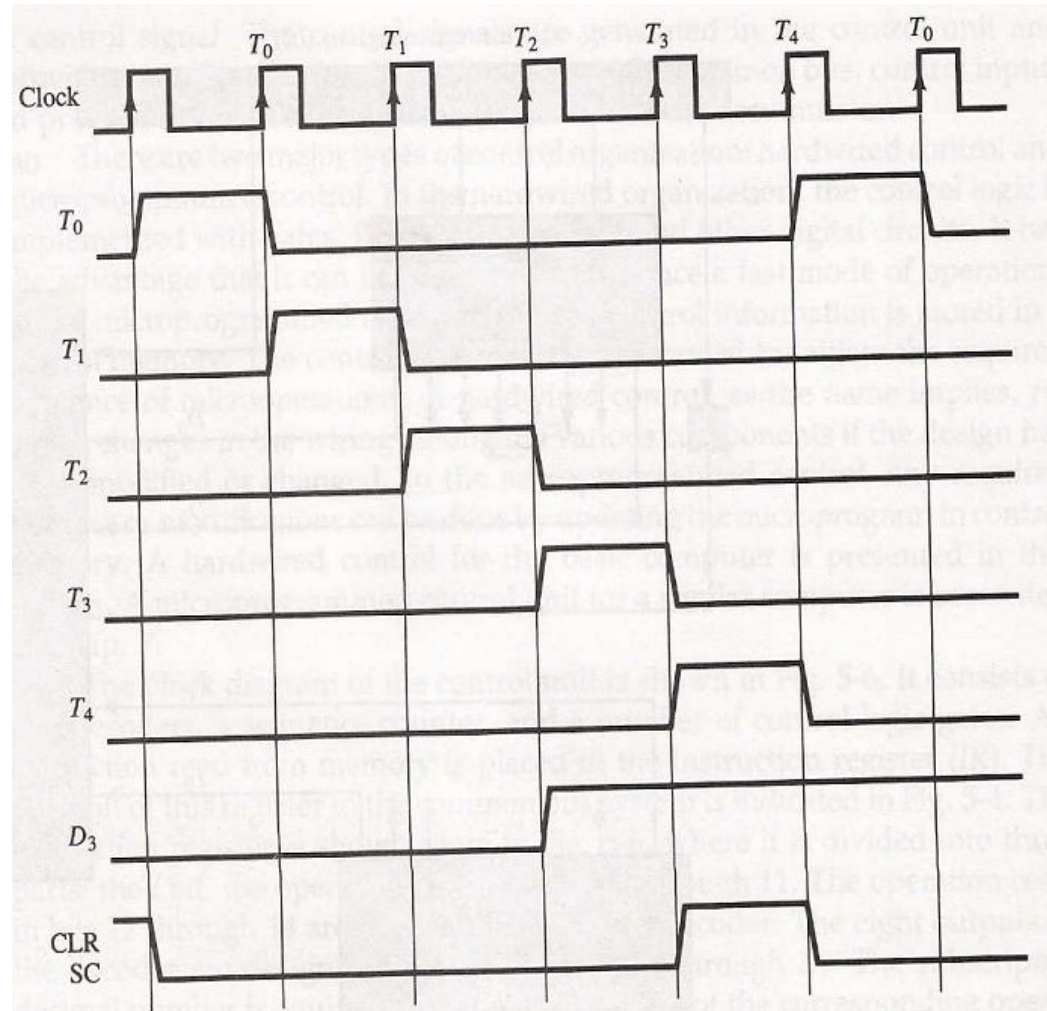


Fig: Diagram of Control Timing Signals

Control Timing Signals

- The last three waveforms in diagram show how SC is cleared when $D_3 T_4 = 1$.
- Output D_3 from the operation decoder becomes active at the end of timing signal T_2 .

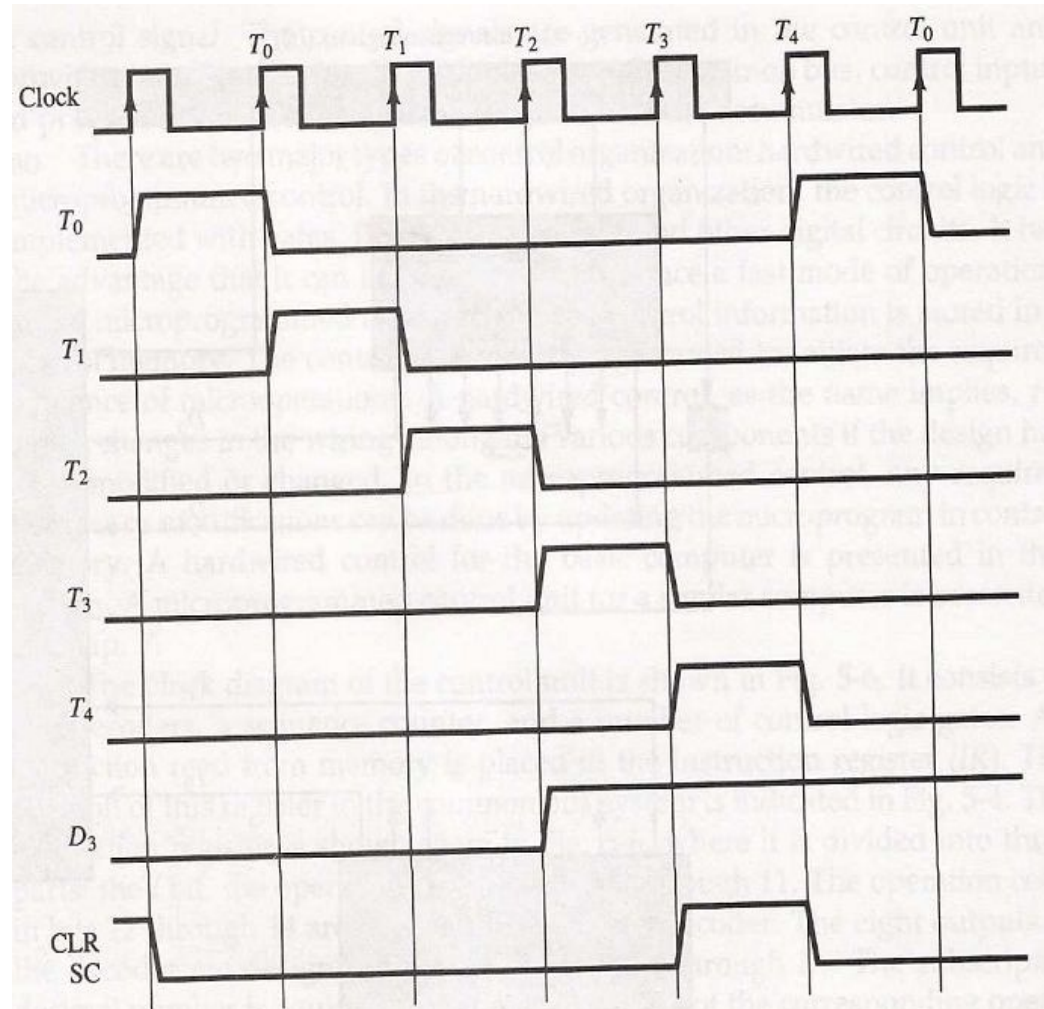


Fig: Diagram of Control Timing Signals

Control Timing Signals

- When timing signal T_4 becomes active, the output of the AND gate that implements the control function $D_3 T_4$ 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.

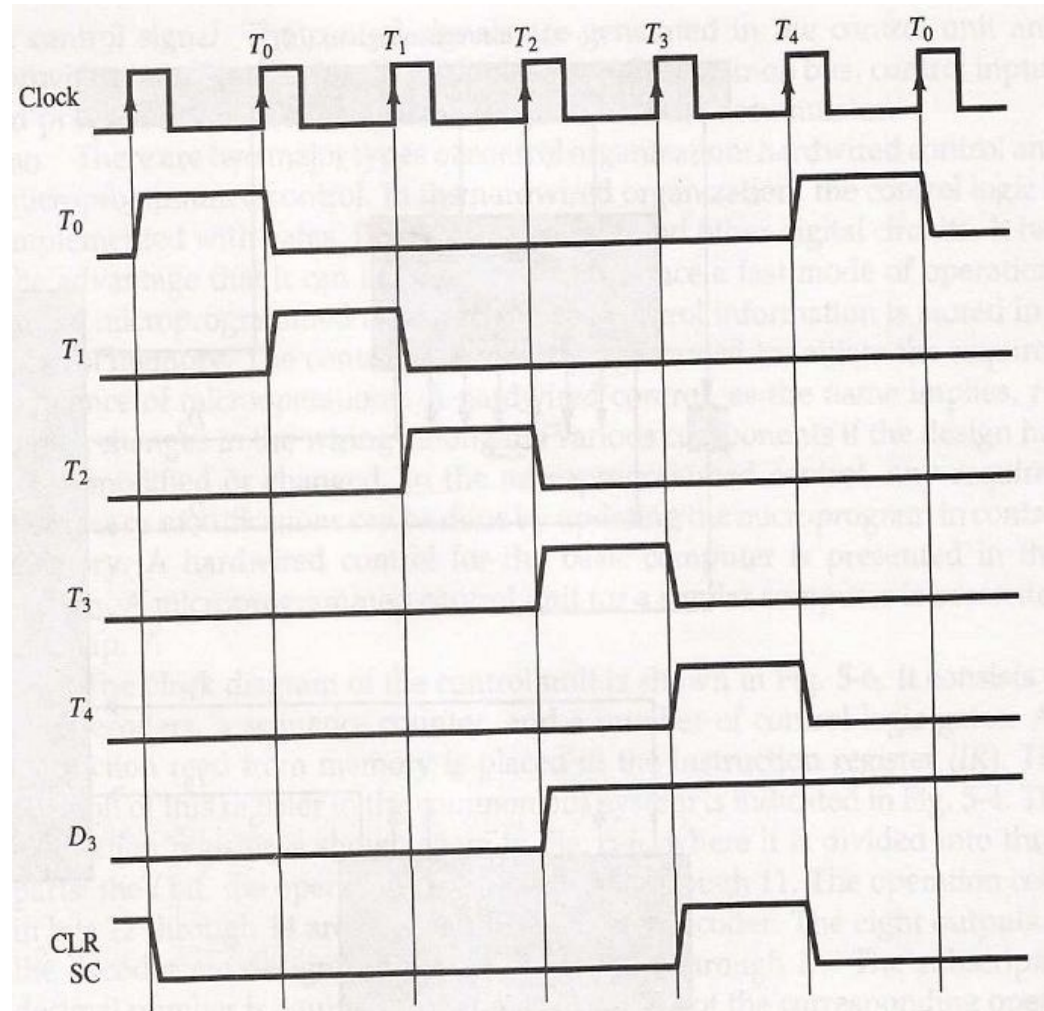


Fig: Diagram of Control Timing Signals

Control Timing Signals

- 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.

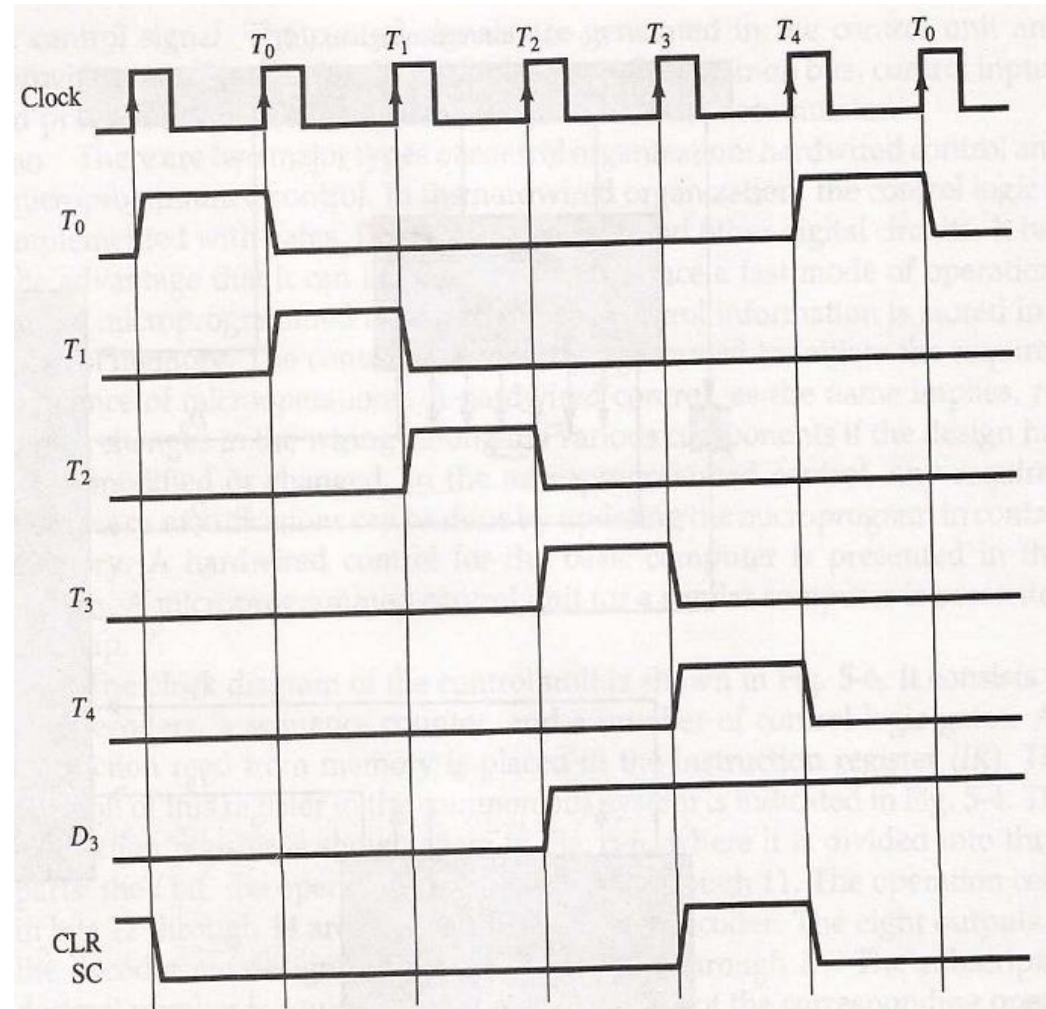


Fig: Diagram of Control Timing Signals

Instruction Cycle

- A program is a sequence of instructions stored in memory.
- The program is executed in the computer by going through a cycle for each instruction (in most cases).
- Each instruction in turn is subdivided into a sequence of sub-cycles or phases.

Instruction Cycle 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
- This cycle repeats 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.

Fetch and Decode

□ $T_0: AR \leftarrow PC$ (this is essential!!)

The address of the instruction is moved to AR.

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

The instruction is fetched from the memory to IR, and the PC is incremented.

□ $T_2: D_0, \dots, D_7 \leftarrow \text{Decode } IR(12-14), AR \leftarrow IR(0-11), I \leftarrow IR(15)$

Register transfer for the fetch phase

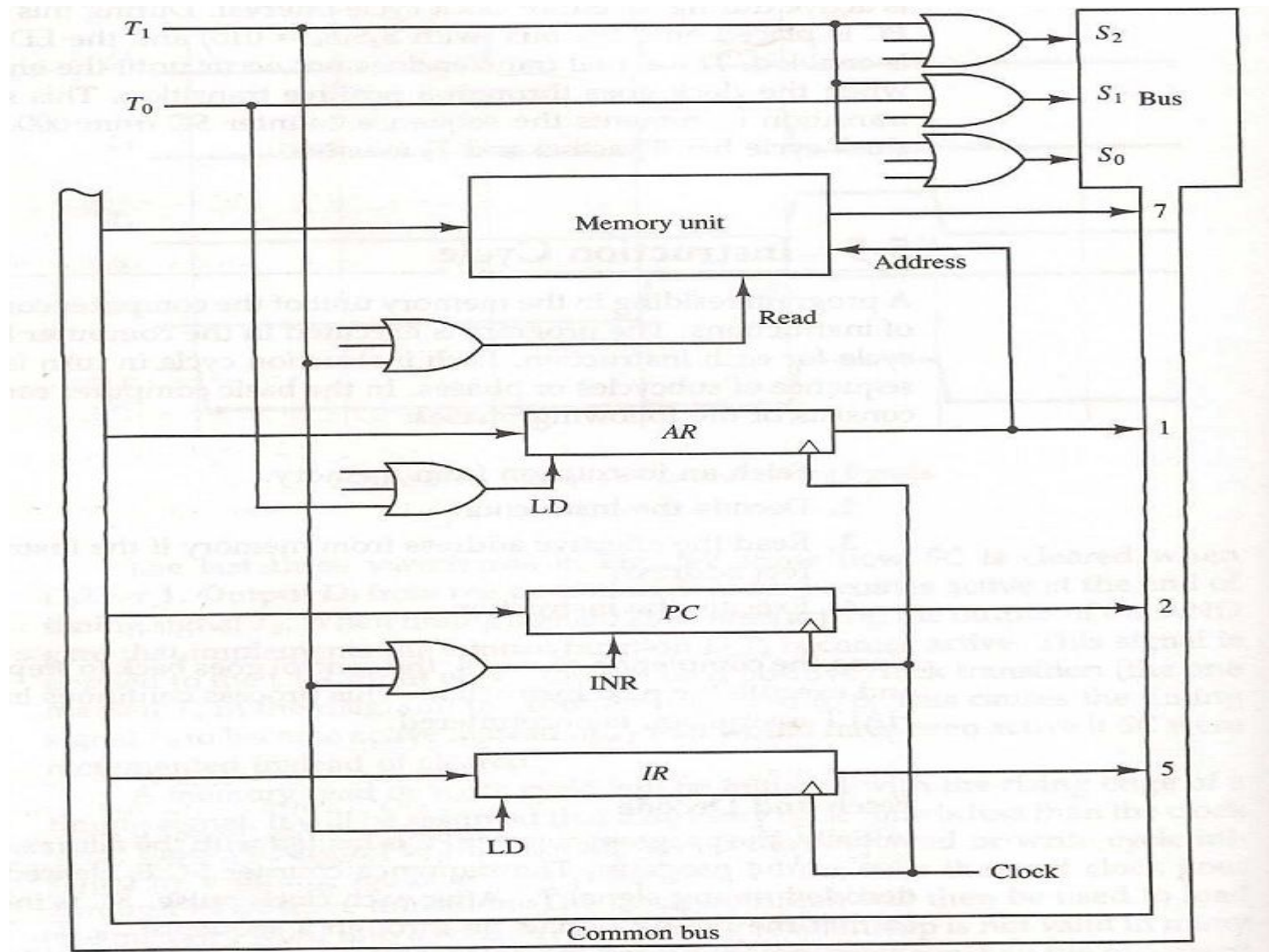


Fig: Register transfer for the fetch phase.

Register transfer for the fetch phase

- Figure shows how the first two register transfer statements are implemented in the bus system.

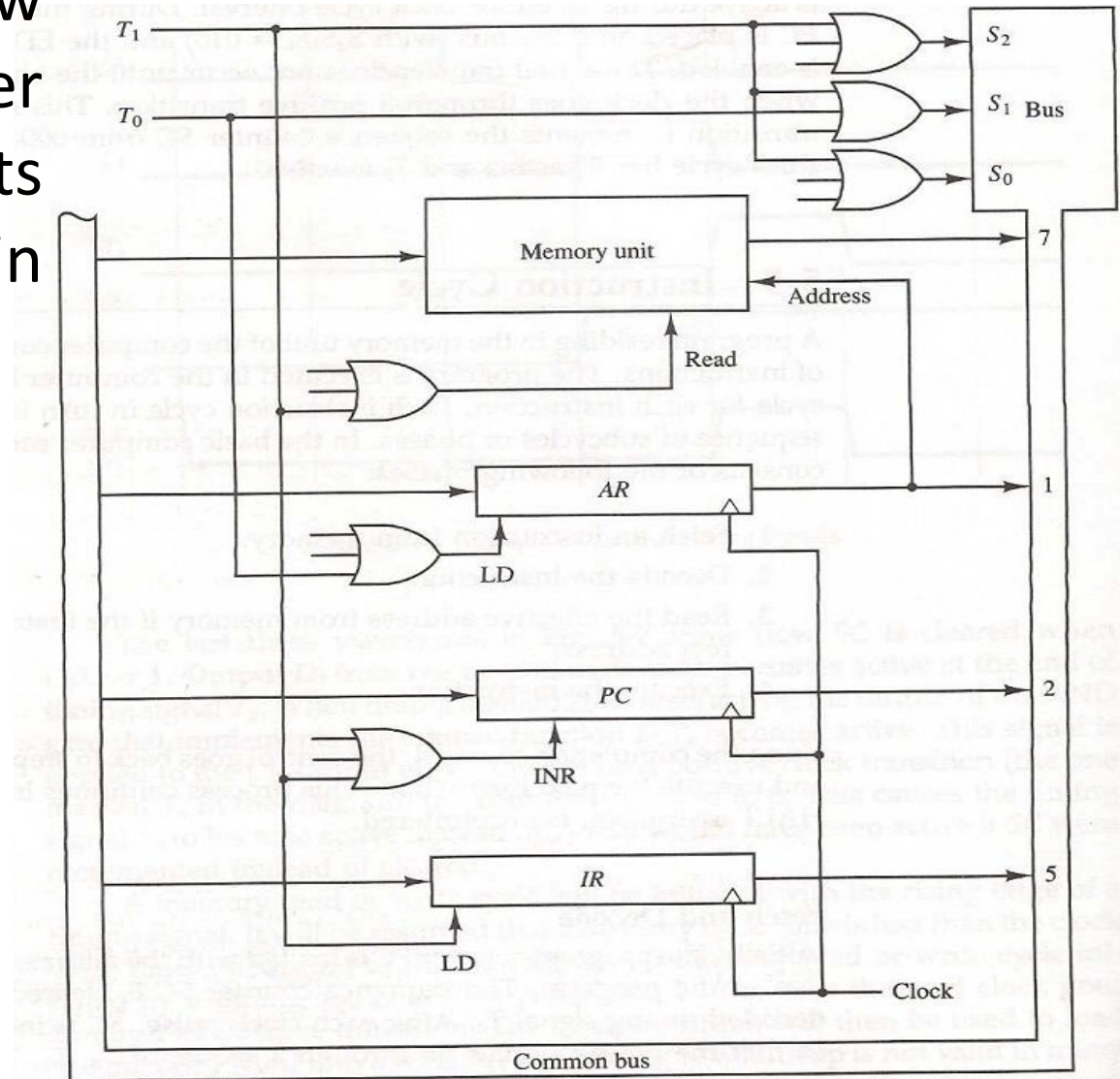


Fig: Register transfer for the fetch phase.

Register transfer for the fetch phase

T_0 . ($T_0: AR \leftarrow PC$)

- To transfer the from PC to AR we must apply timing signal
- Place the content of PC onto the bus by making the bus selection inputs S_2 , S_1 , S_0 equal to 010.
- ❖ Transfer the content of the bus to AR by enabling the LD input of AR.

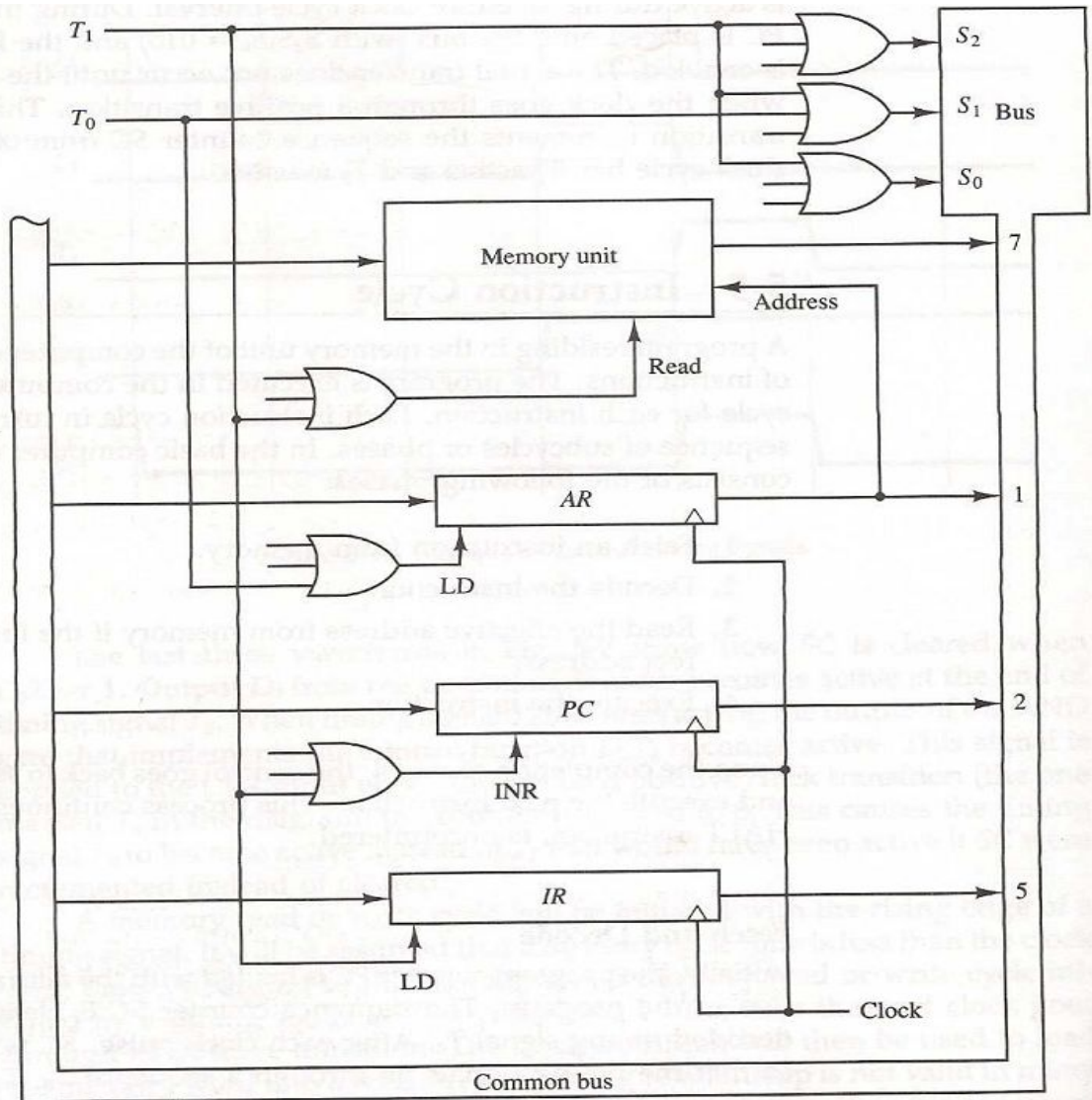


Fig: Register transfer for the fetch phase.

Register transfer for the fetch phase

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

- To implement the second statement, we need timing signal T_1 .

Enable the read input of memory.

- ❖ Place the content of memory onto the bus by making $S_2 S_1 S_0 = 111$.

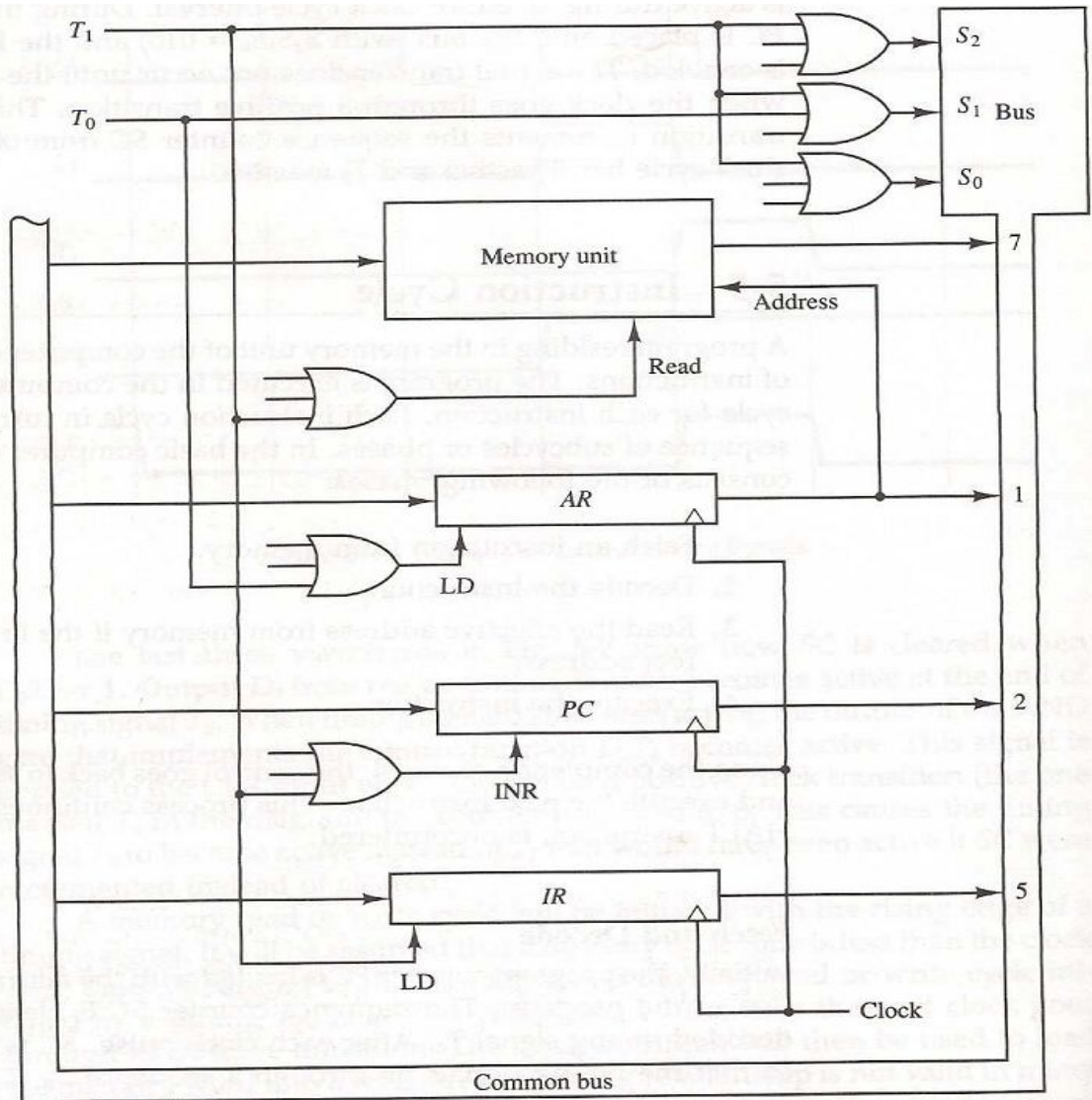


Fig: Register transfer for the fetch phase.

Register transfer for the fetch phase

- ❖ Transfer the content of the bus to IR by enabling the LD input of IR.
- ❖ Increment PC by enabling the INR input of PC.

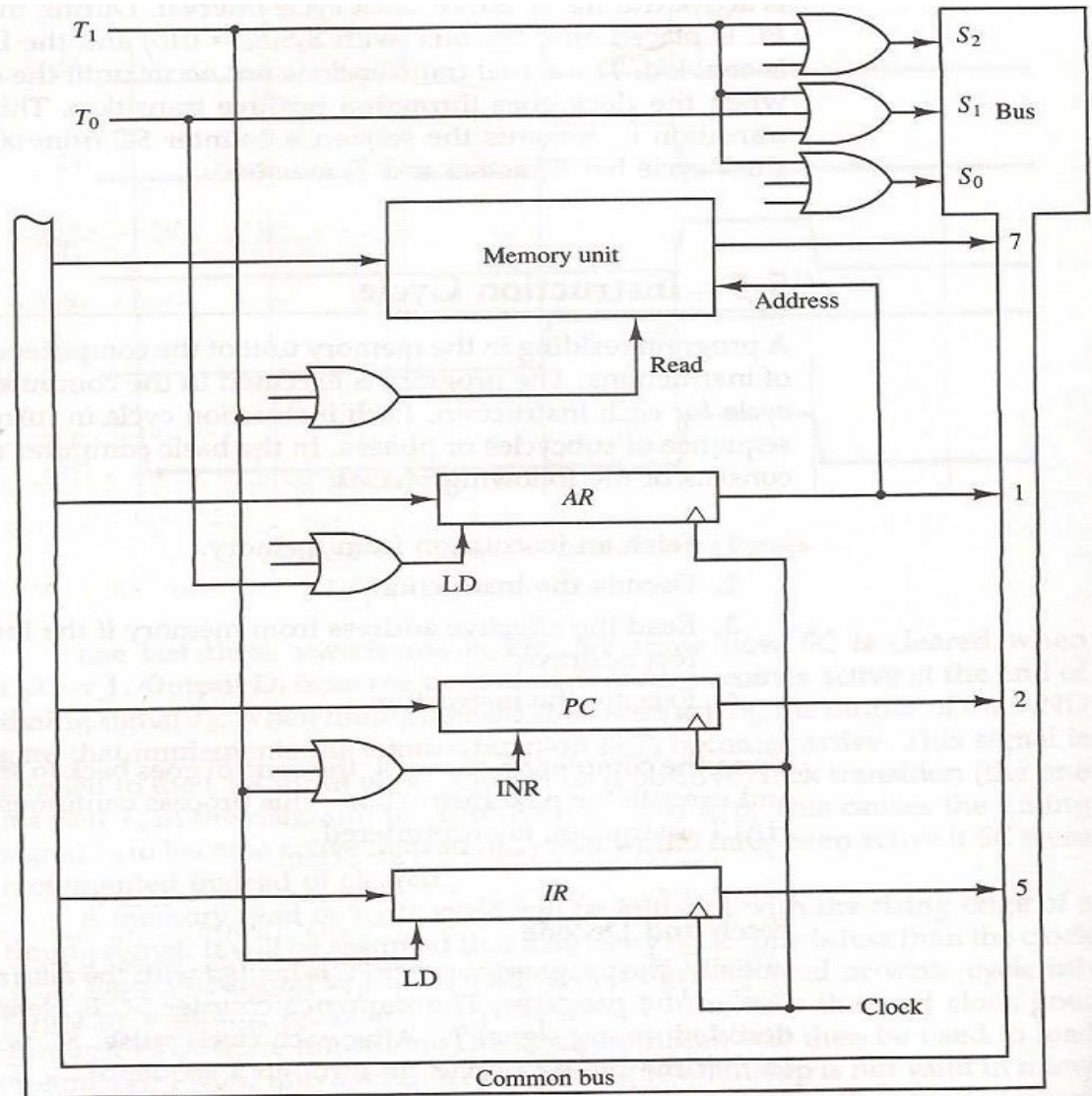


Fig: Register transfer for the fetch phase.

Register transfer for the fetch phase

- The bus system 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 because there are other control functions that will initiate similar operations.

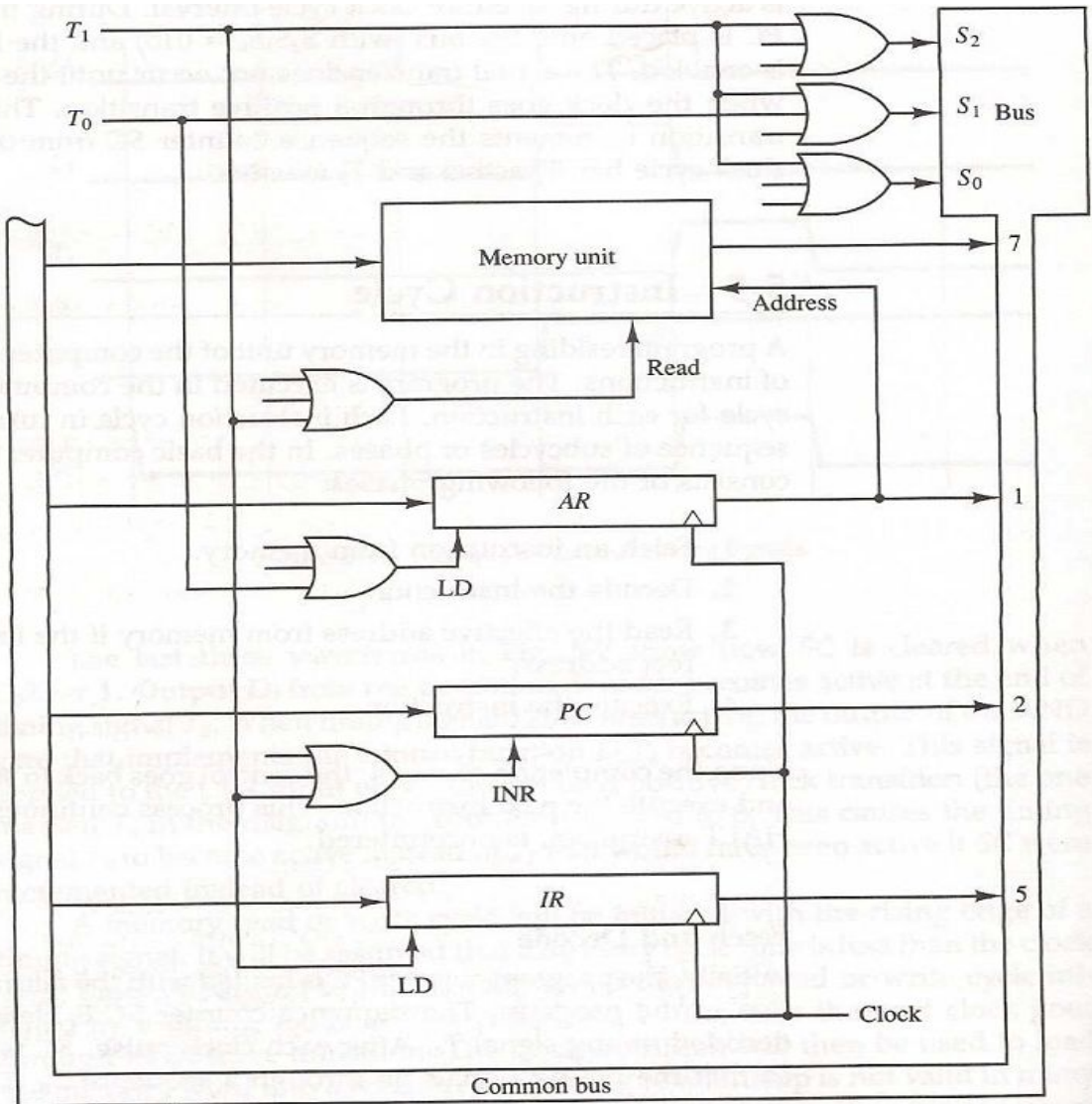


Fig: Register transfer for the fetch phase.

Determine the Type of Instruction

How the control determines the instruction cycle type after the decoding.

Determine the Type of Instruction

- The flowchart shows the initial configurations for the instruction cycle and also how the control determines the instruction cycle type after the decoding.

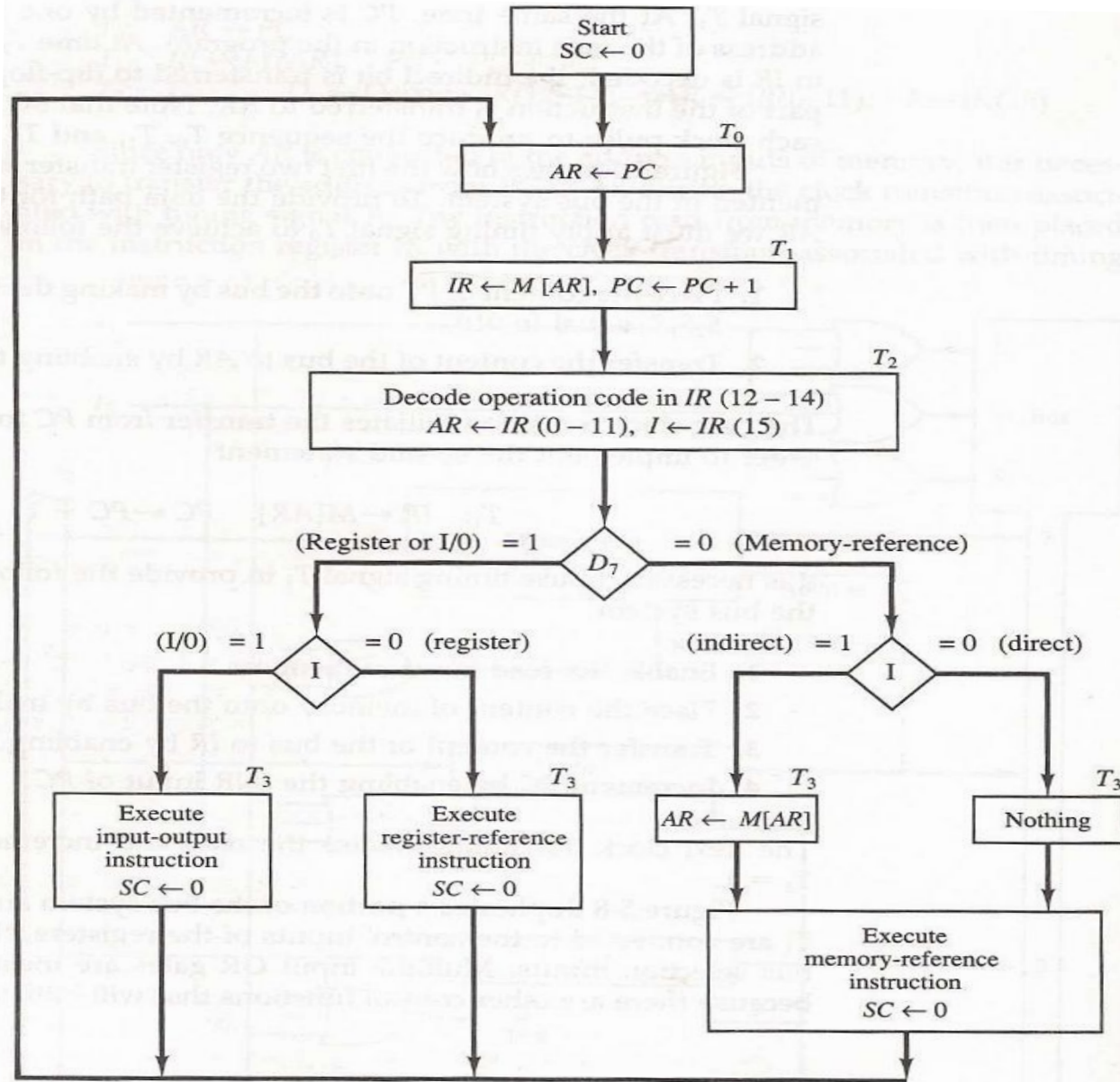


Fig: Flowchart for instruction cycle

Determine the Type of Instruction

- The timing signal that is active after the decoding is T_3 .
- During time T_3 , the control unit determine the type of instruction that was read from the memory.

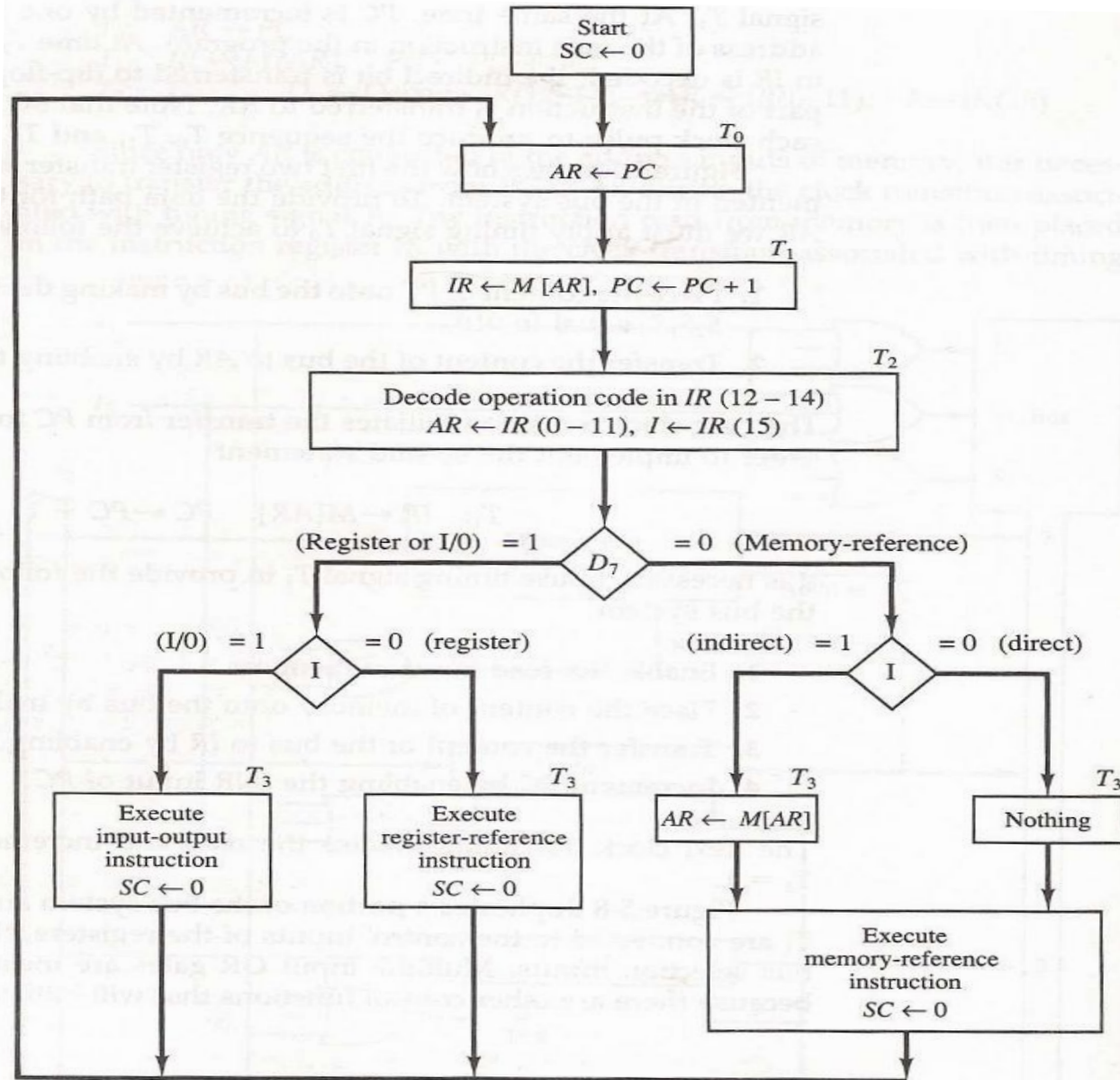


Fig: Flowchart for instruction cycle

Determine the Type of Instruction

- If $D_7=1$, the instruction must be a register-reference or input-output type.
- If $D_7 = 0$, the instruction must be a memory-reference instruction.

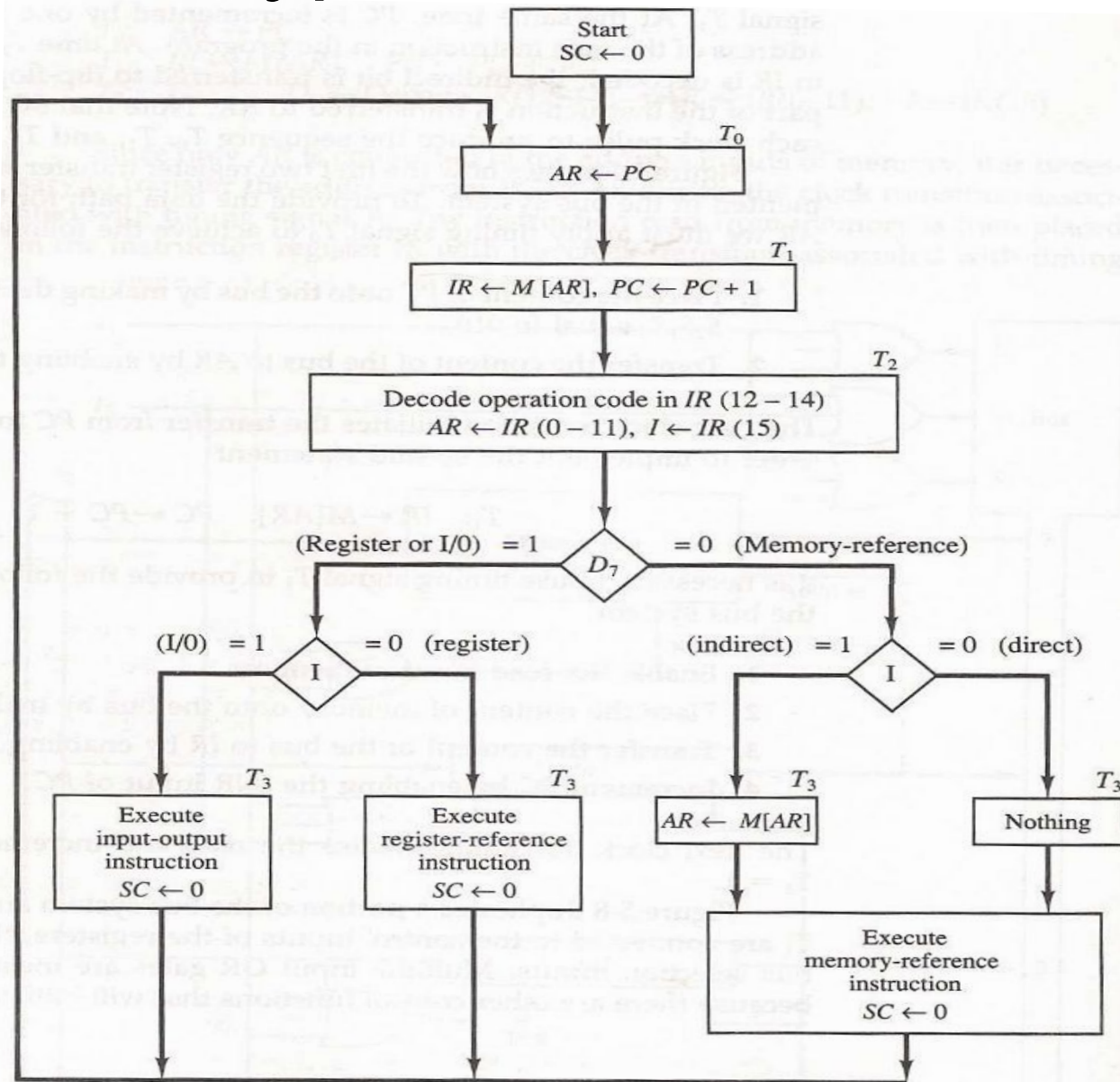


Fig: Flowchart for instruction cycle

Determine the Type of Instruction

- when $D_7 = 0$, the operation code must be one of the other seven values 000 through 110.

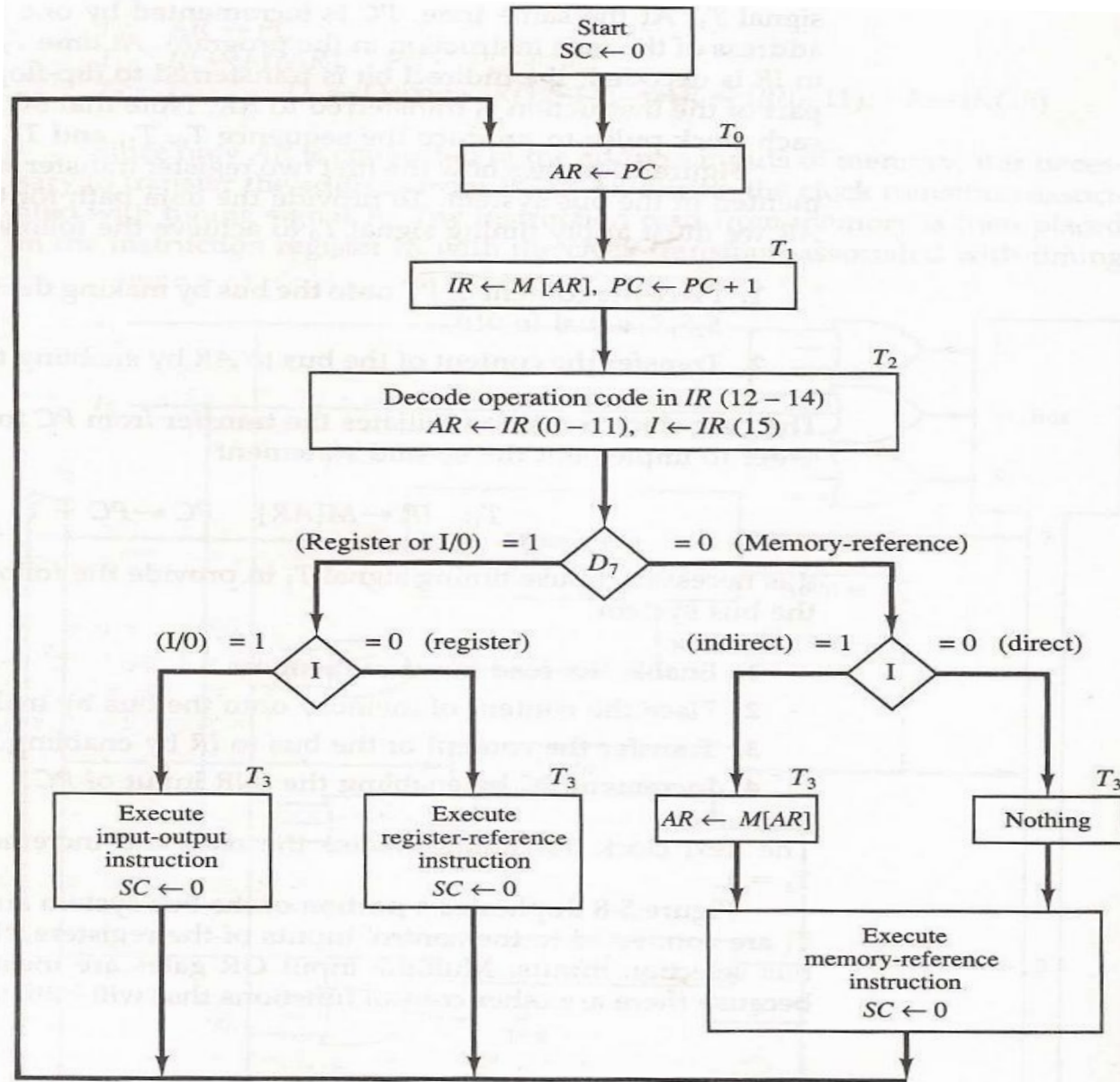


Fig: Flowchart for instruction cycle

Determine the Type of Instruction

- Control then inspects the value of the first bit of the instruction, which is now available in flip-flop I.

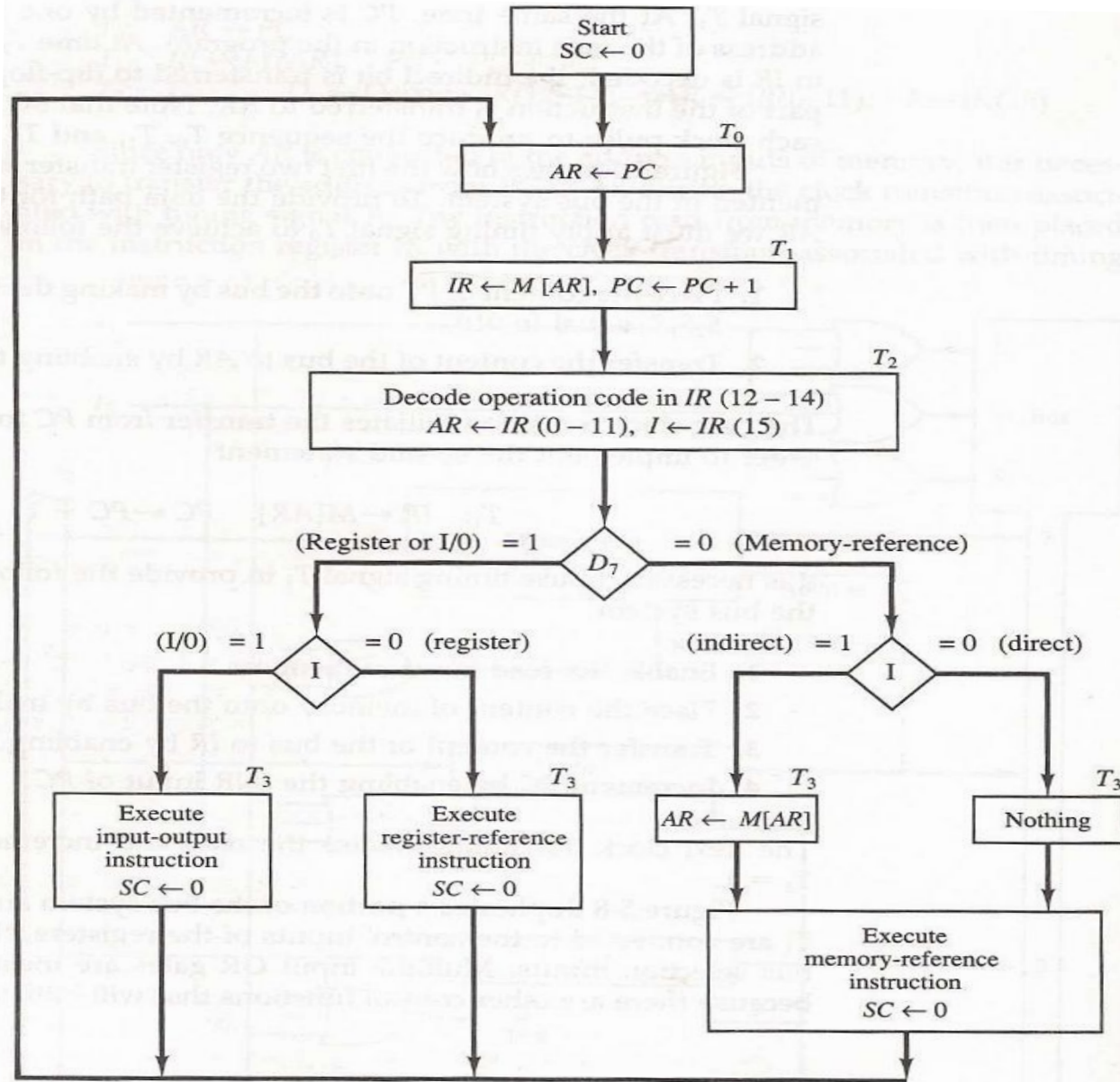


Fig: Flowchart for instruction cycle

Determine the Type of Instruction

- If $D_7 = 0$ and $I = 1$, indicates a memory-reference instruction with an indirect address. So it is then necessary to read the effective address from memory.

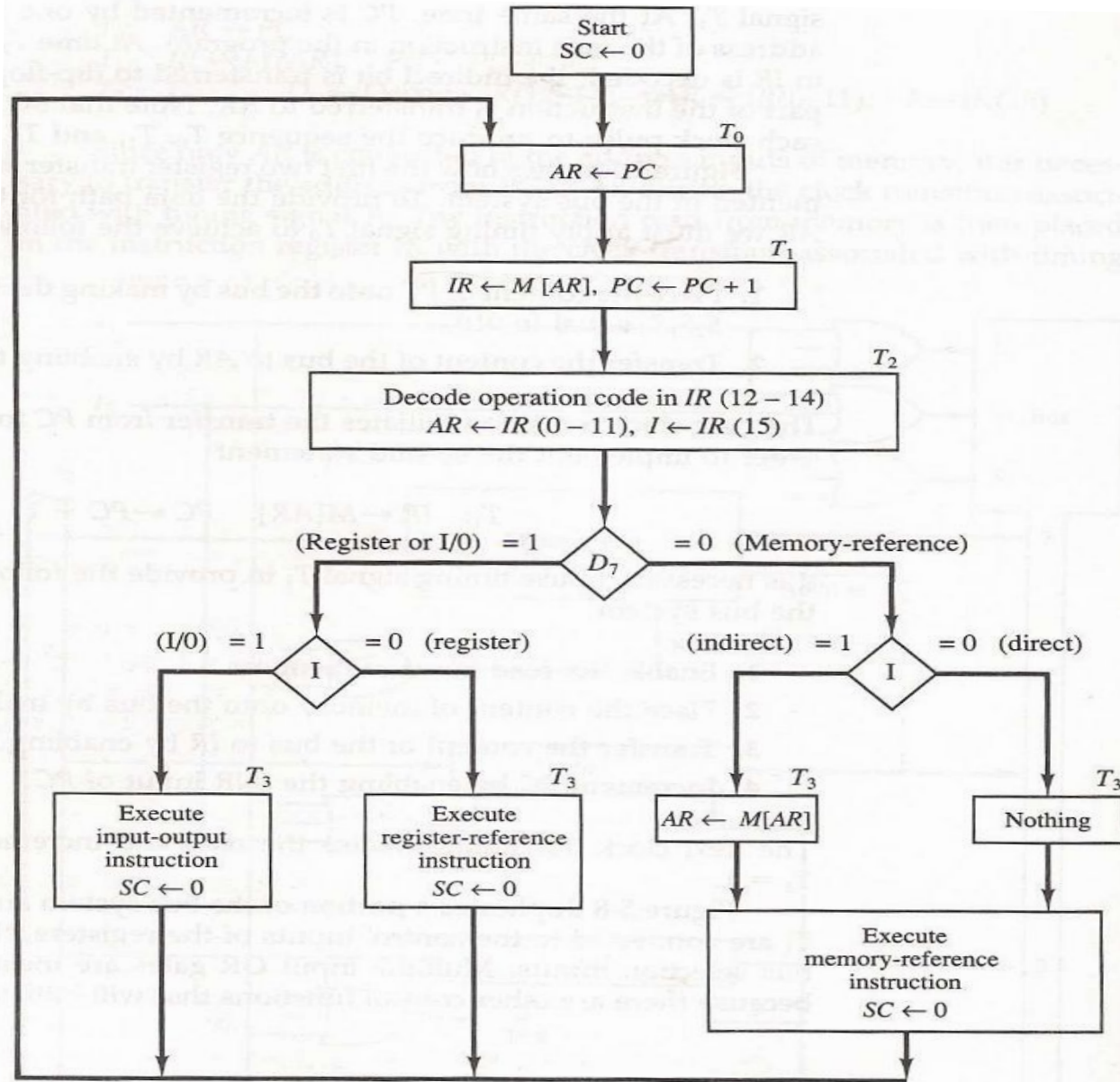


Fig: Flowchart for instruction cycle

Determine the Type of Instruction

- If $D_7 = 0$ and $I = 0$, indicates a memory-reference instruction with a direct address.

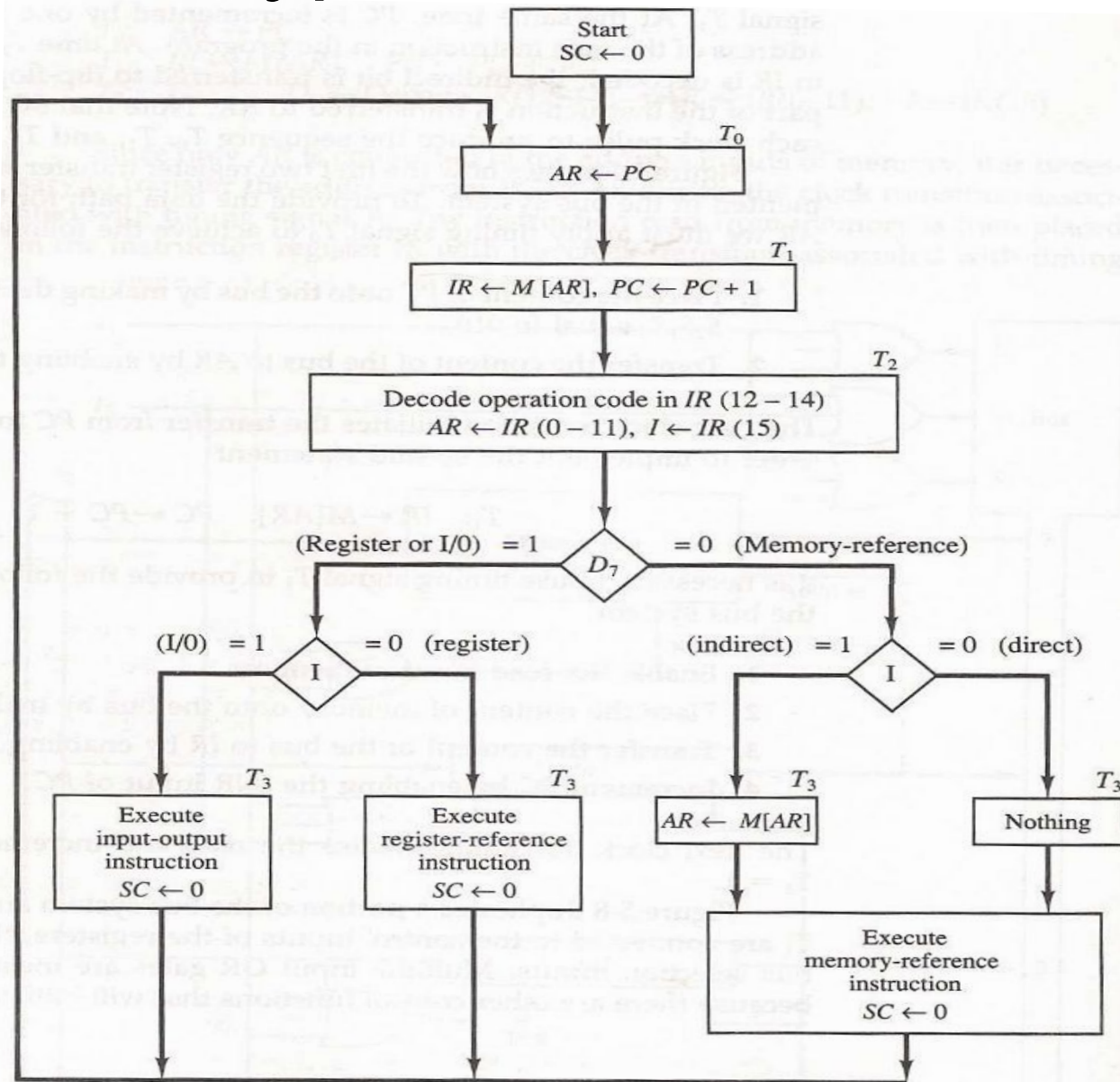


Fig: Flowchart for instruction cycle

Determine the Type of Instruction

- If $D_7 = 1$ and $I = 0$, indicates a register-reference instruction.
- If $D_7 = 1$ and $I = 1$, indicates an input-output instruction.

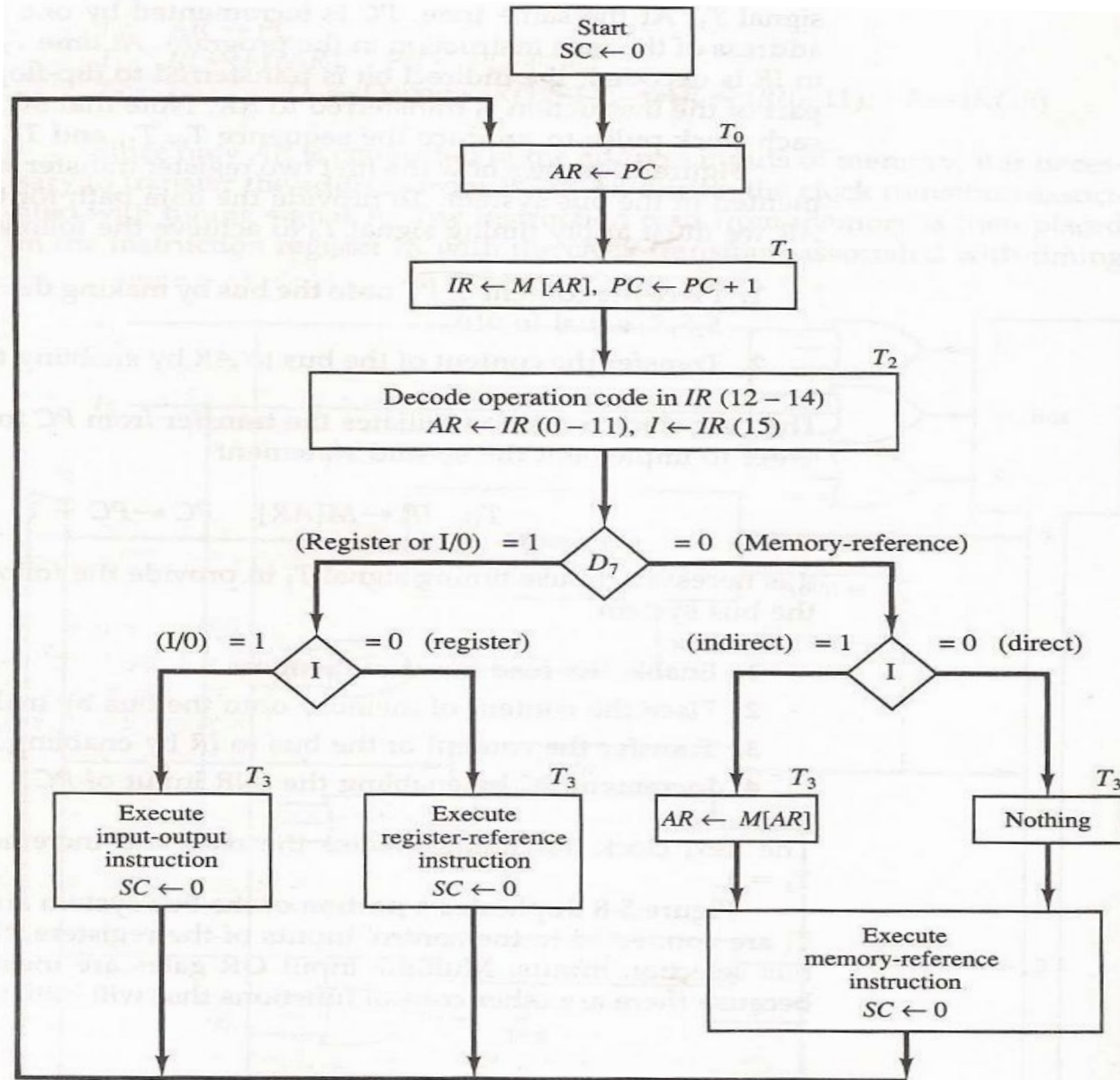


Fig: Flowchart for instruction cycle

Determine the Type of Instruction

- 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 .

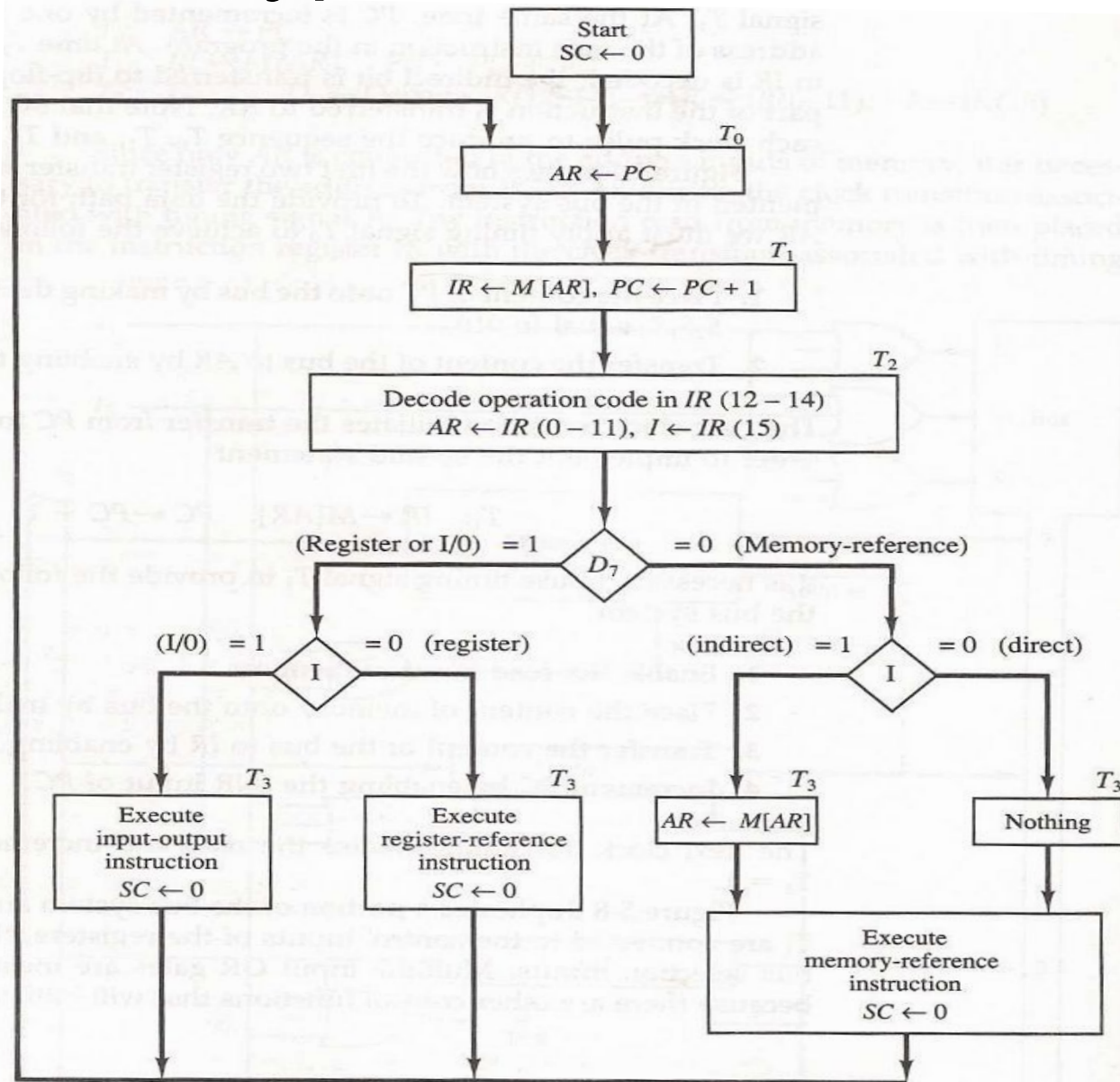


Fig: Flowchart for instruction cycle

Determine the Type of Instruction

Symbolized:-

- D_7I_3 : $AR \leftarrow M[AR]$
- D_7I_3 : Nothing
- D_7I_3 : Execute a register-reference instr.
- D_7I_3 : Execute an input-output instr.

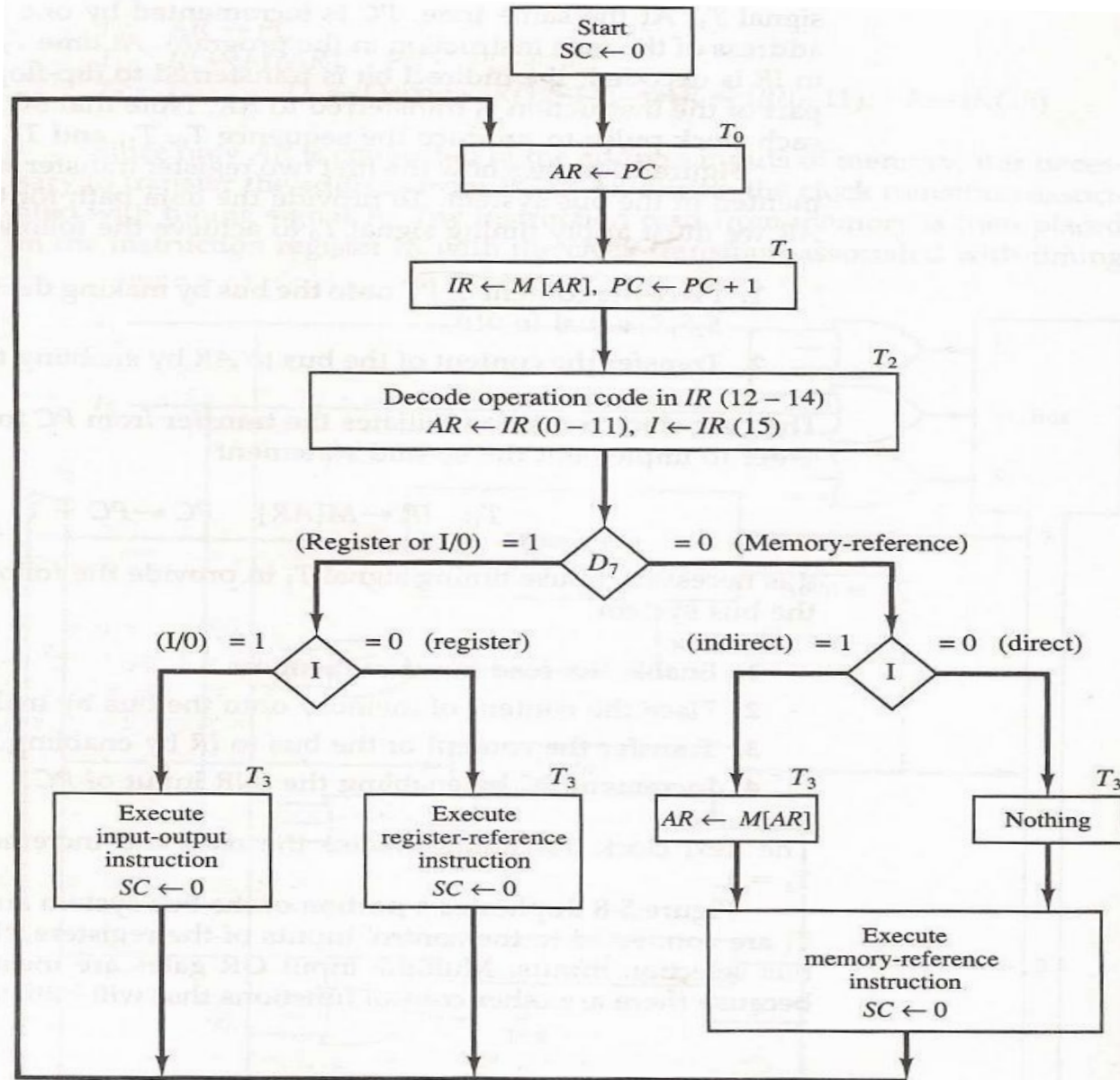


Fig: Flowchart for instruction cycle

Determine the Type of Instruction

- A register-reference or input-output instruction can be executed with the clock associated with timing signal T_3 .

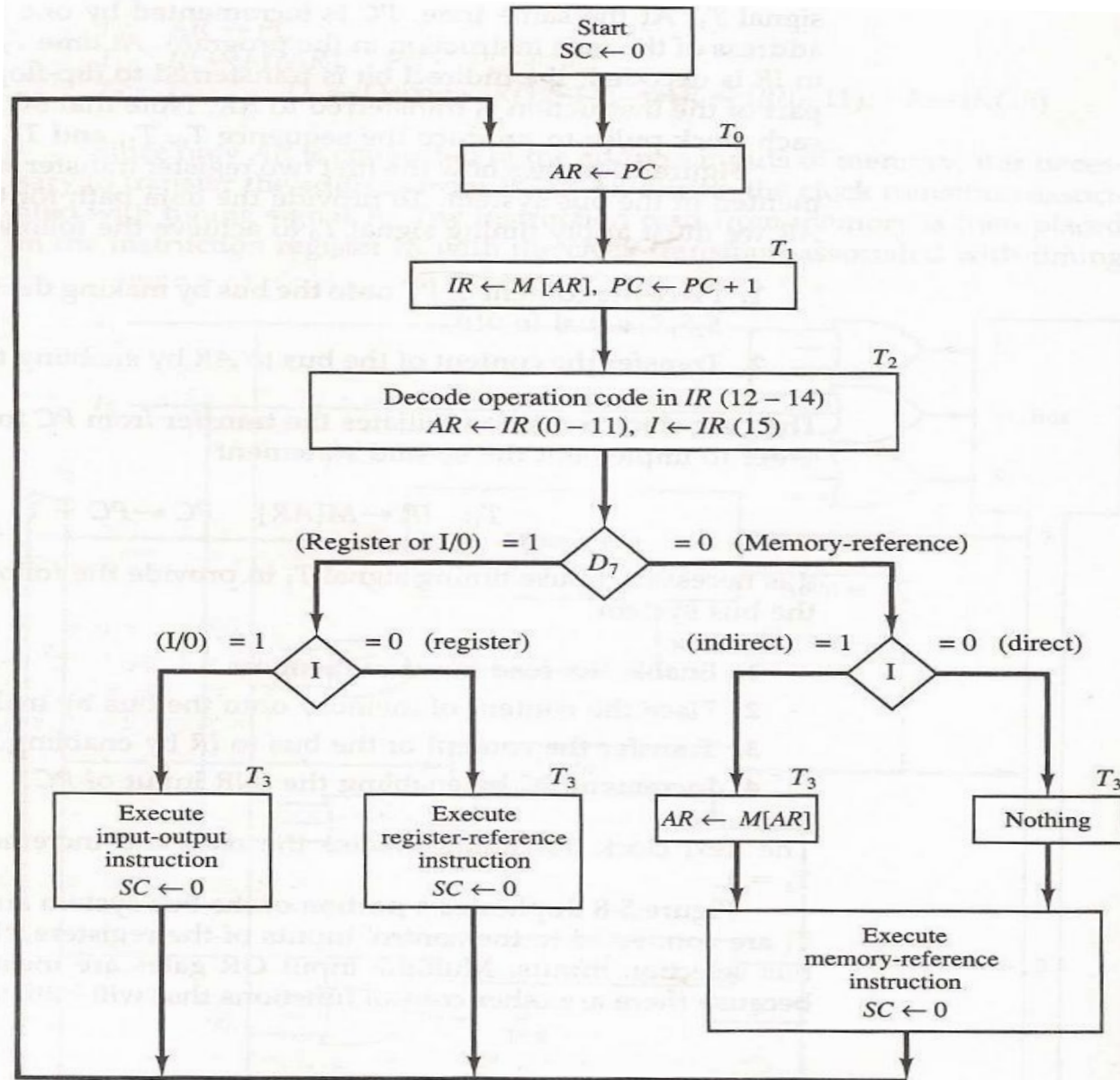


Fig: Flowchart for instruction cycle

Determine the Type of Instruction

- The execution of the memory-reference instruction can be continued with timing variable T_4 , when instruction with $I=0$.

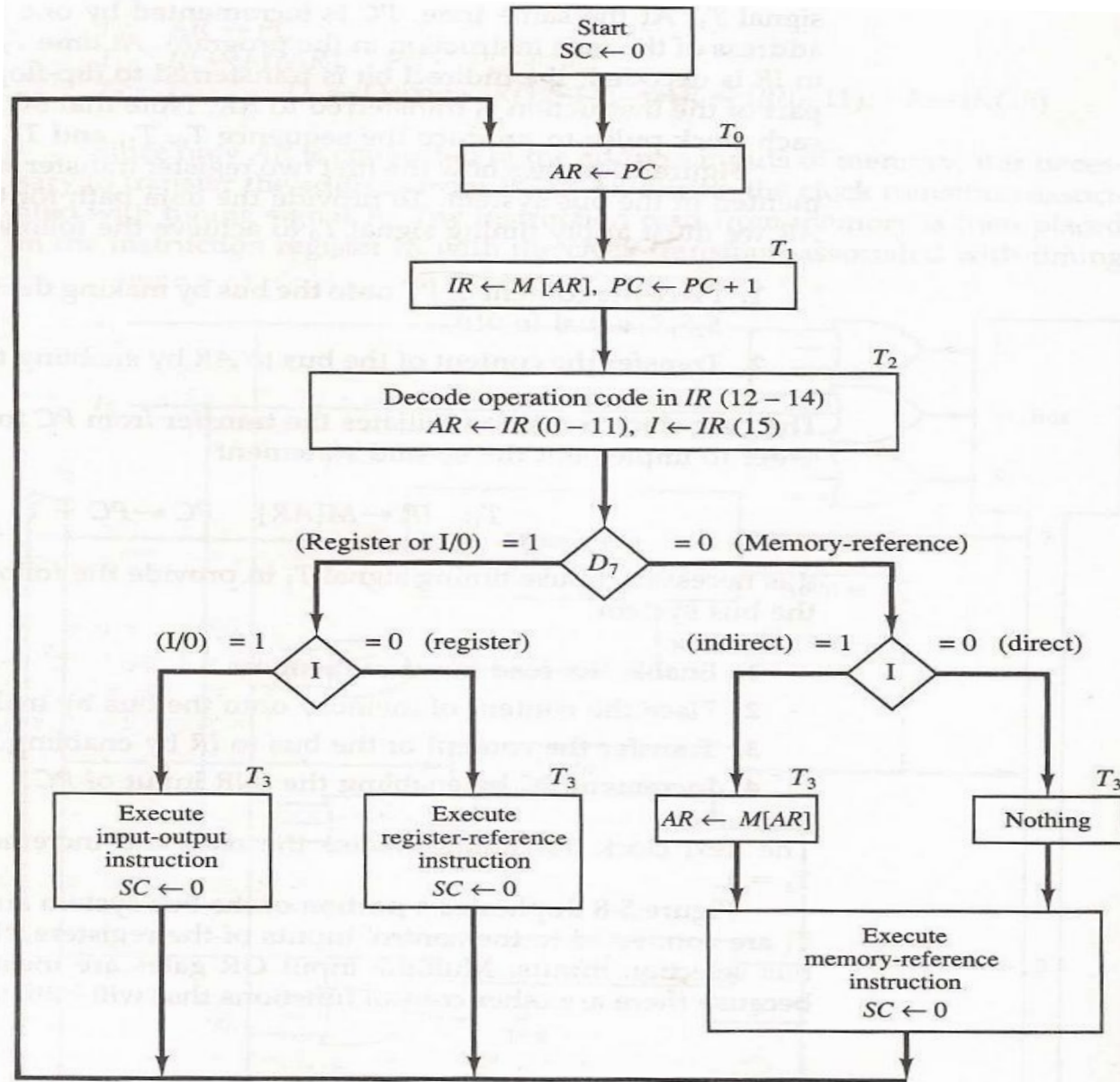


Fig: Flowchart for instruction cycle

Determine the Type of Instruction

After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase with $T_0=1$.

Memory-reference instruction

Memory-Reference Instructions

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

Register-reference instruction

Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)

$IR(i) = B_i$ [bit in $IR(0-11)$ that specifies the operation]

	$r:$	$SC \leftarrow 0$	Clear SC
CLA	$rB_{11}:$	$AC \leftarrow 0$	Clear AC
CLE	$rB_{10}:$	$E \leftarrow 0$	Clear E
CMA	$rB_9:$	$AC \leftarrow \overline{AC}$	Complement AC
CME	$rB_8:$	$E \leftarrow \overline{E}$	Complement E
CIR	$rB_7:$	$AC \leftarrow \text{shr } AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	$rB_6:$	$AC \leftarrow \text{shl } AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	$rB_5:$	$AC \leftarrow AC + 1$	Increment AC
SPA	$rB_4:$	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	$rB_3:$	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	$rB_2:$	If $(AC = 0)$ then $PC \leftarrow PC + 1$	Skip if AC zero
SZE	$rB_1:$	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	$rB_0:$	$S \leftarrow 0$ (S is a start-stop flip-flop)	Halt computer

Input-output instruction



TABLE 5-5 Input-Output Instructions

$D_7IT_3 = p$ (common to all input-output instructions)

$IR(i) = B_i$ [bit in $IR(6-11)$ that specifies the instruction]

	$p:$	$SC \leftarrow 0$	Clear SC
INP	$pB_{11}:$	$AC(0-7) \leftarrow INPR, FGI \leftarrow 0$	Input character
OUT	$pB_{10}:$	$OUTR \leftarrow AC(0-7), FGO \leftarrow 0$	Output character
SKI	$pB_9:$	If ($FGI = 1$) then ($PC \leftarrow PC + 1$)	Skip on input flag
SKO	$pB_8:$	If ($FGO = 1$) then ($PC \leftarrow PC + 1$)	Skip on output flag
ION	$pB_7:$	$IEN \leftarrow 1$	Interrupt enable on
IOF	$pB_6:$	$IEN \leftarrow 0$	Interrupt enable off

Input-Output and Interrupt

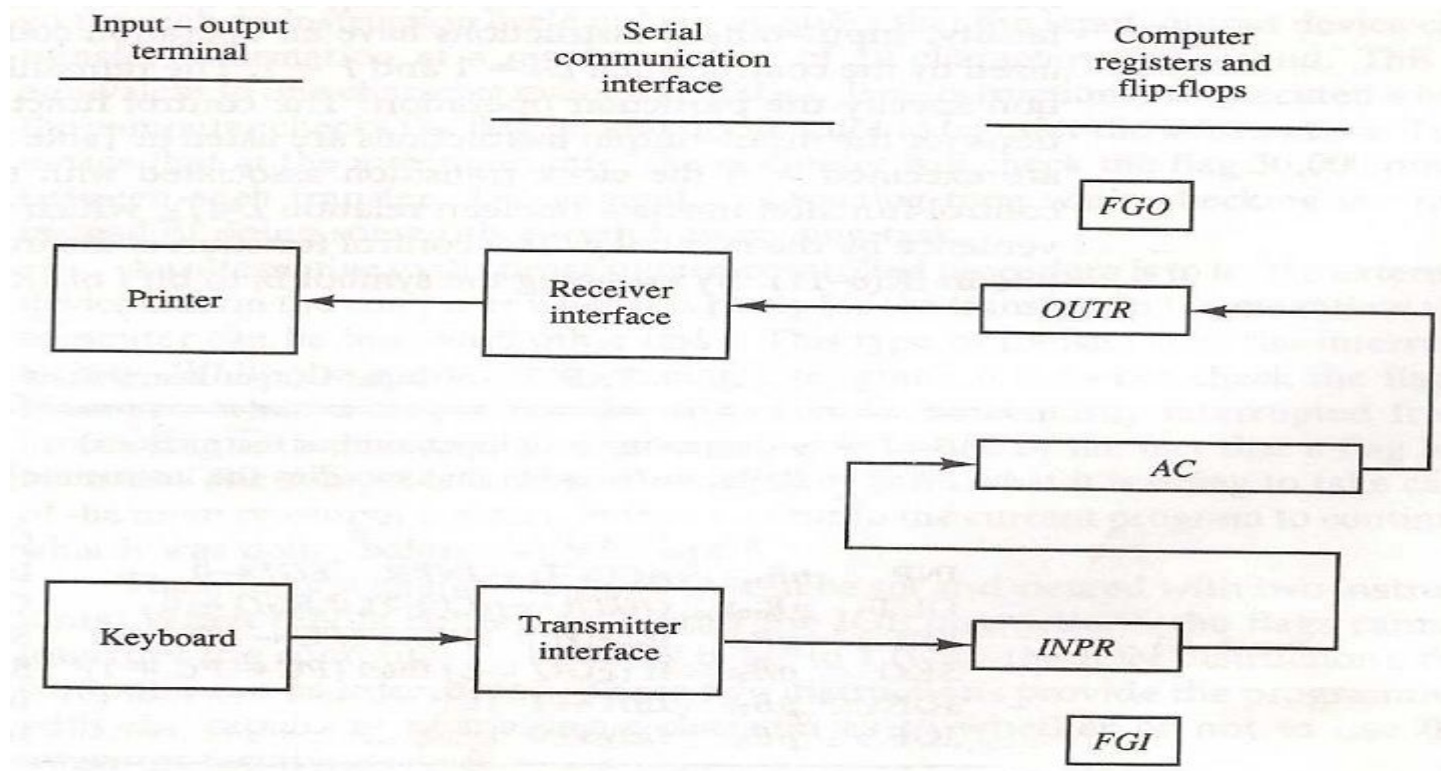
Input-Output Configuration,
Program Interrupt, and
Interrupt cycle

Input-Output and Interrupt

- Instructions and data stored in memory come from some input device.
- Computational results must be transmitted to the user through some output device.

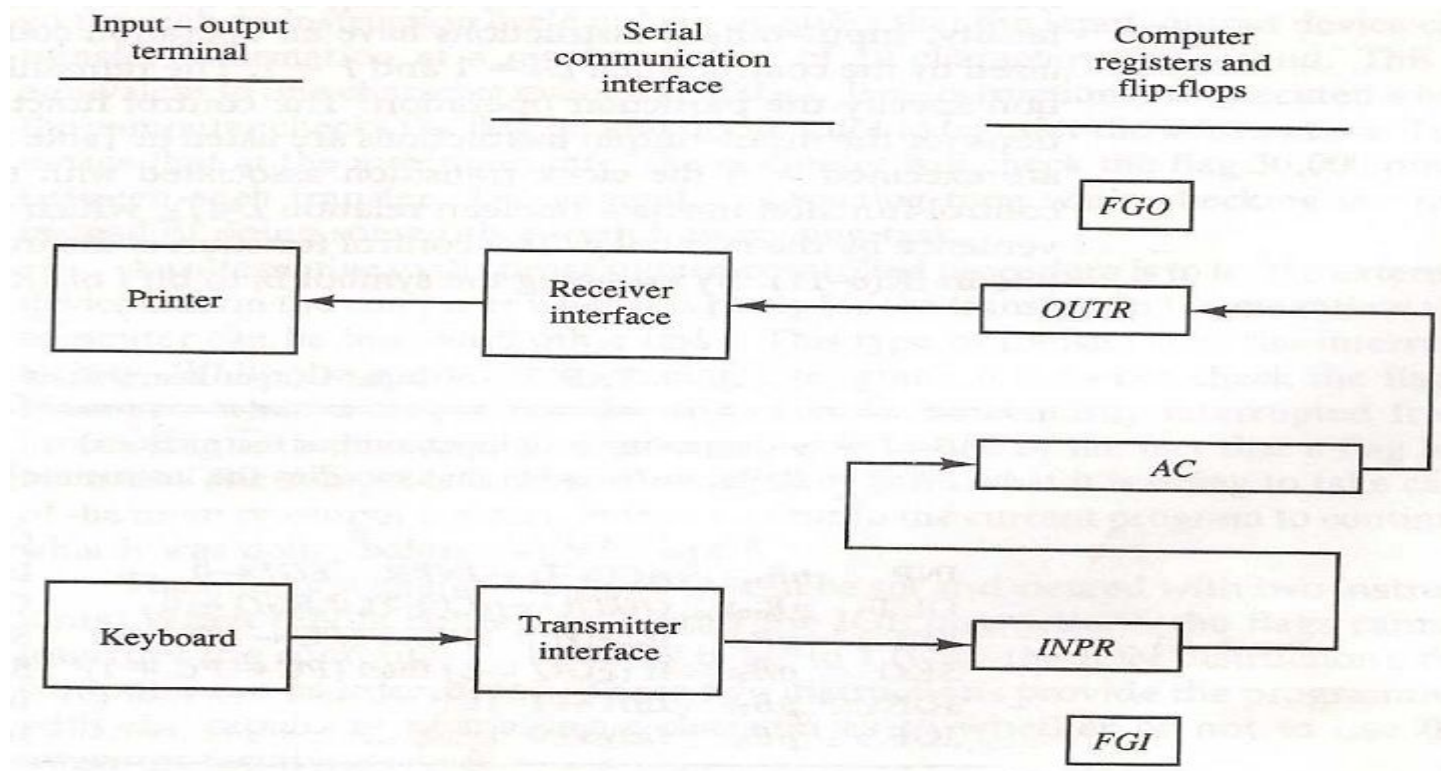
Input-Output Configuration

- The terminal sends and receives serial information.
- Each quantity of information has eight bits of an alphanumeric code.
- The serial information from the keyboard is shifted into the input register INPR.



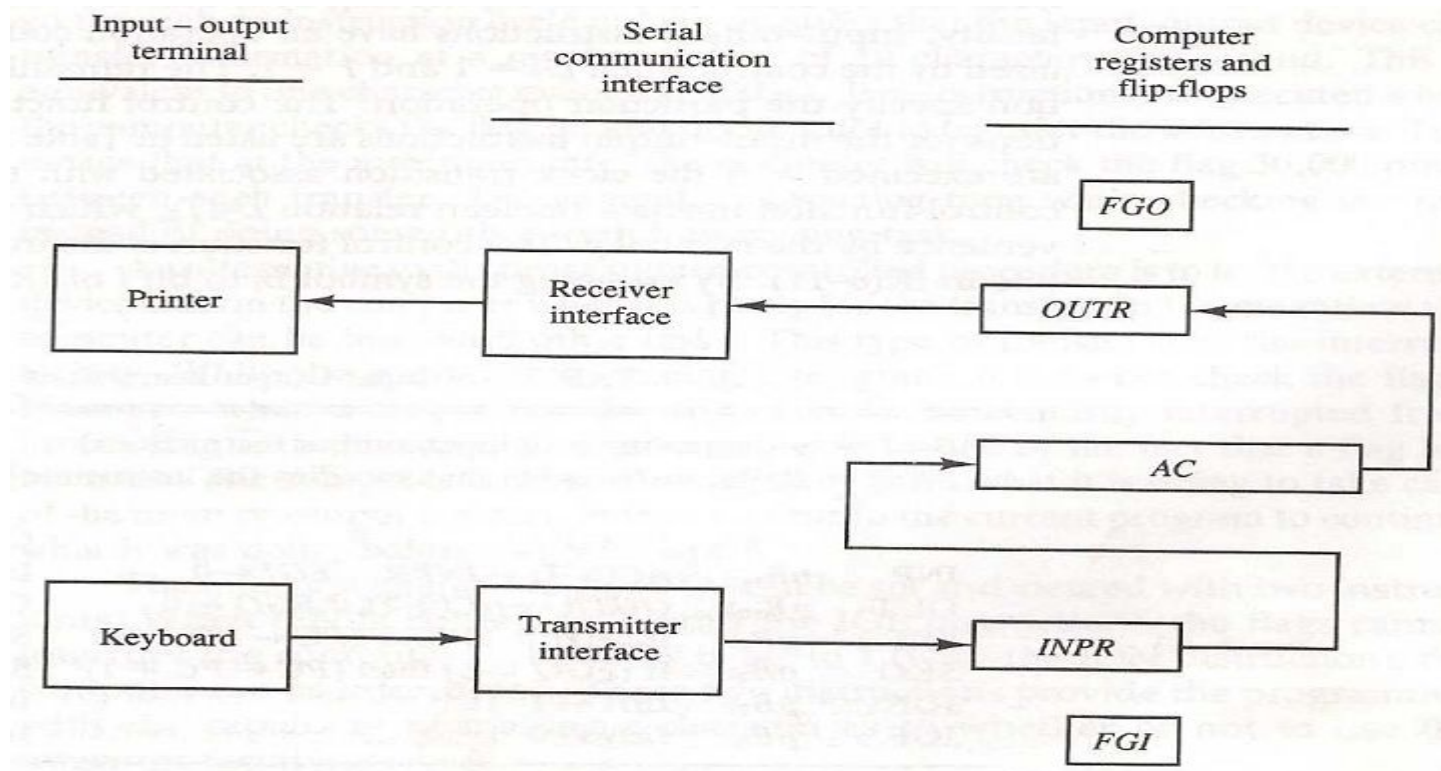
Input-Output Configuration

- The serial information for the printer is stored in the output register OUTR.
- These two registers communicate with a communication interface serially and with the AC in parallel.



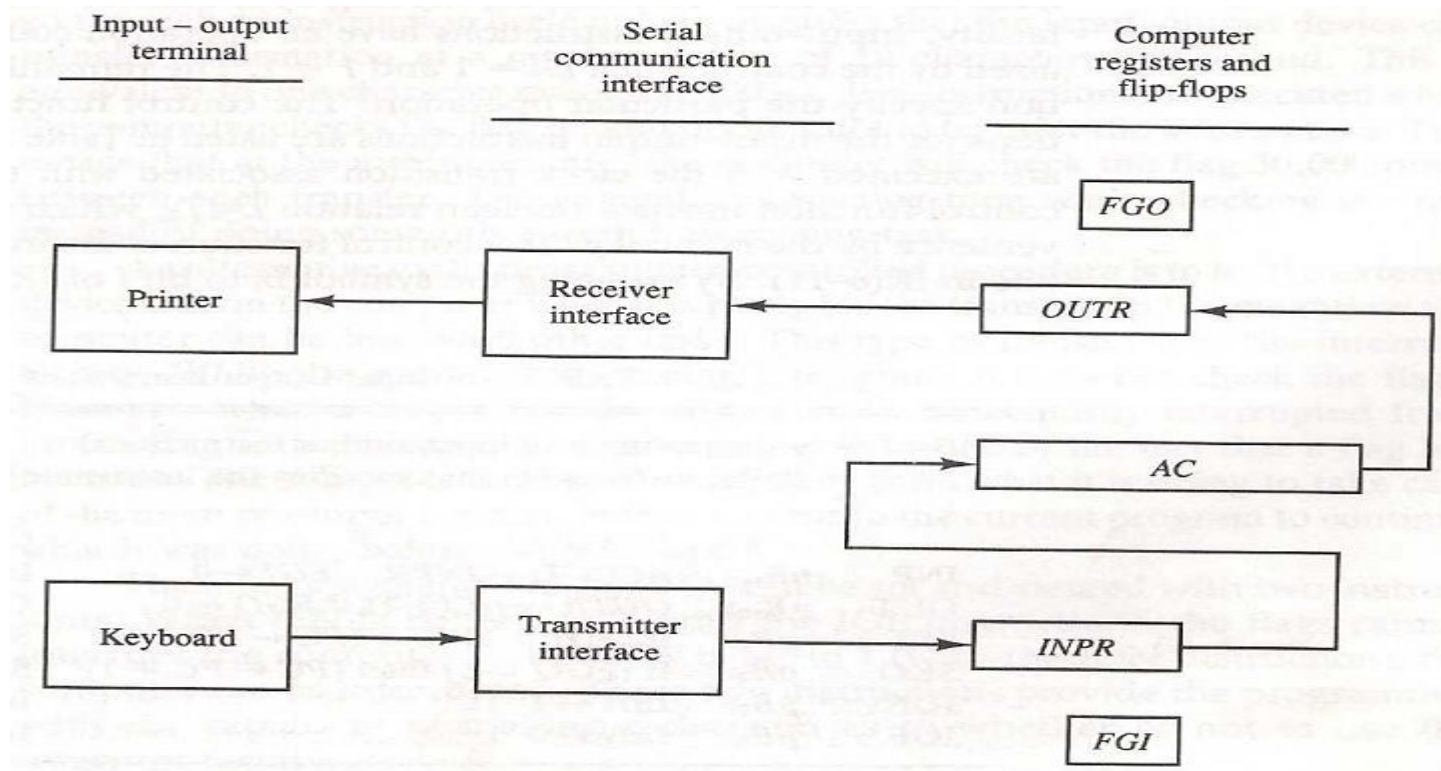
Input-Output Configuration

- The input register INPR consists of eight bits and holds alphanumeric input information.
- The 1-bit input flag FGI is a control flip-flop.



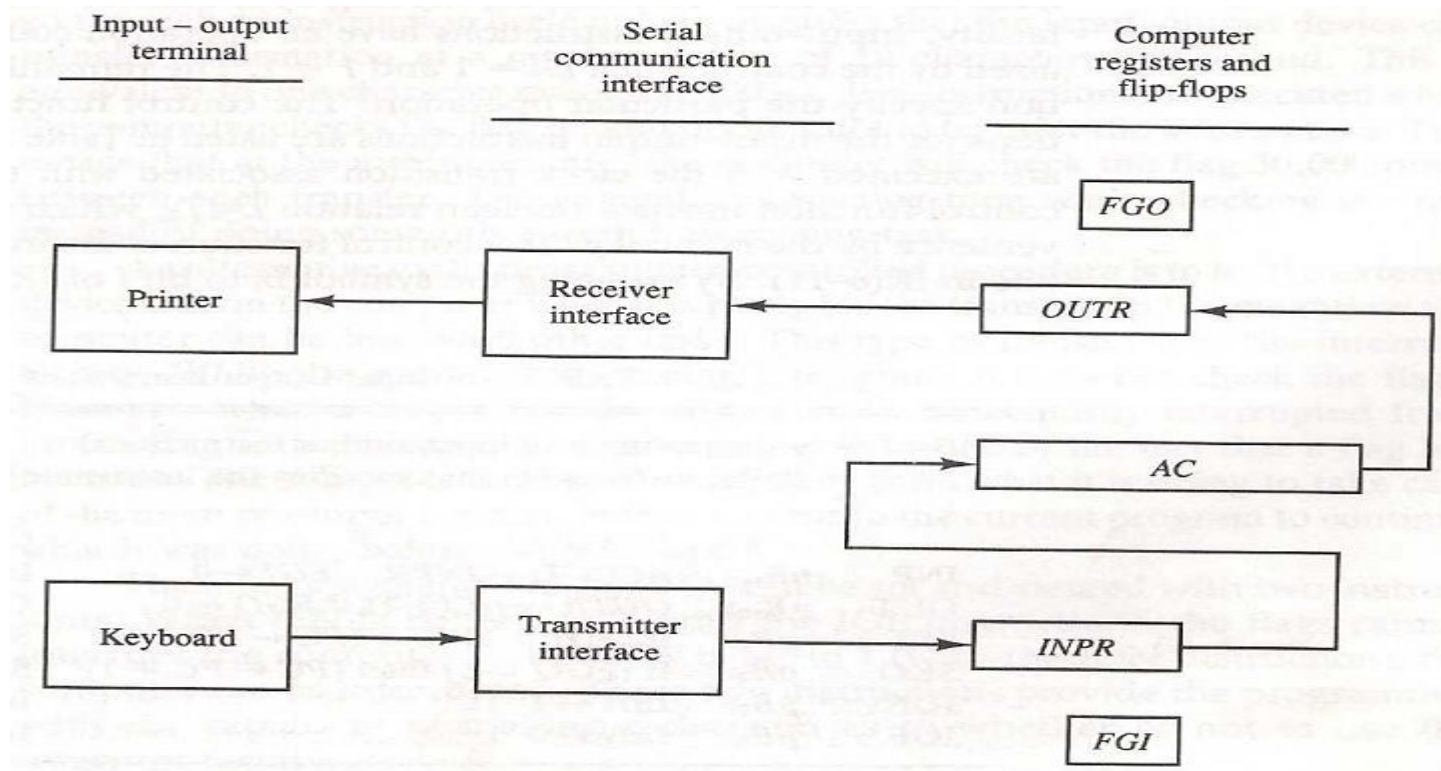
Input-Output Configuration

- The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer.
- The output register OUTR works similarly but the direction of information flow is reversed.



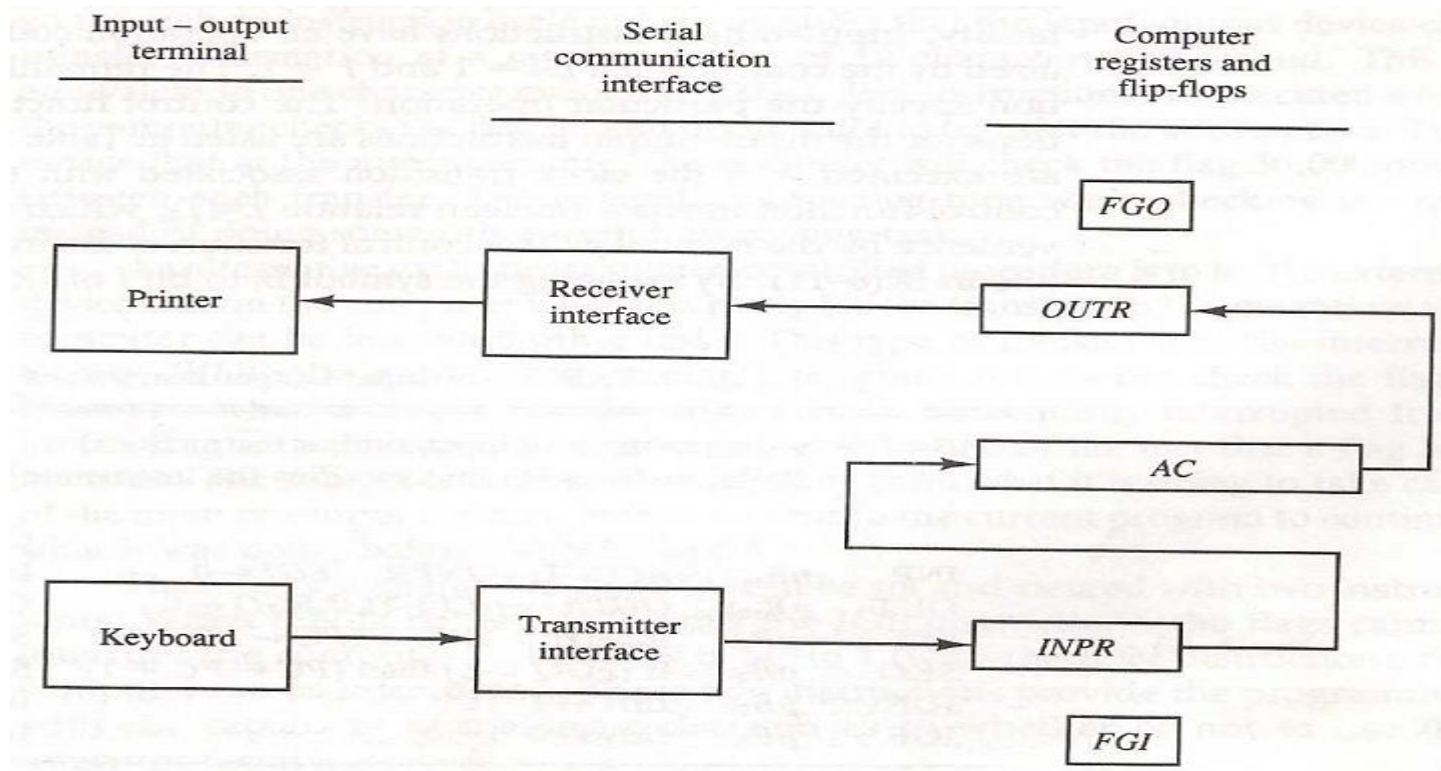
Input-Output Configuration

- Initially, the input flag FGI is cleared to 0.
- When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and input flag FGI is set to 1.



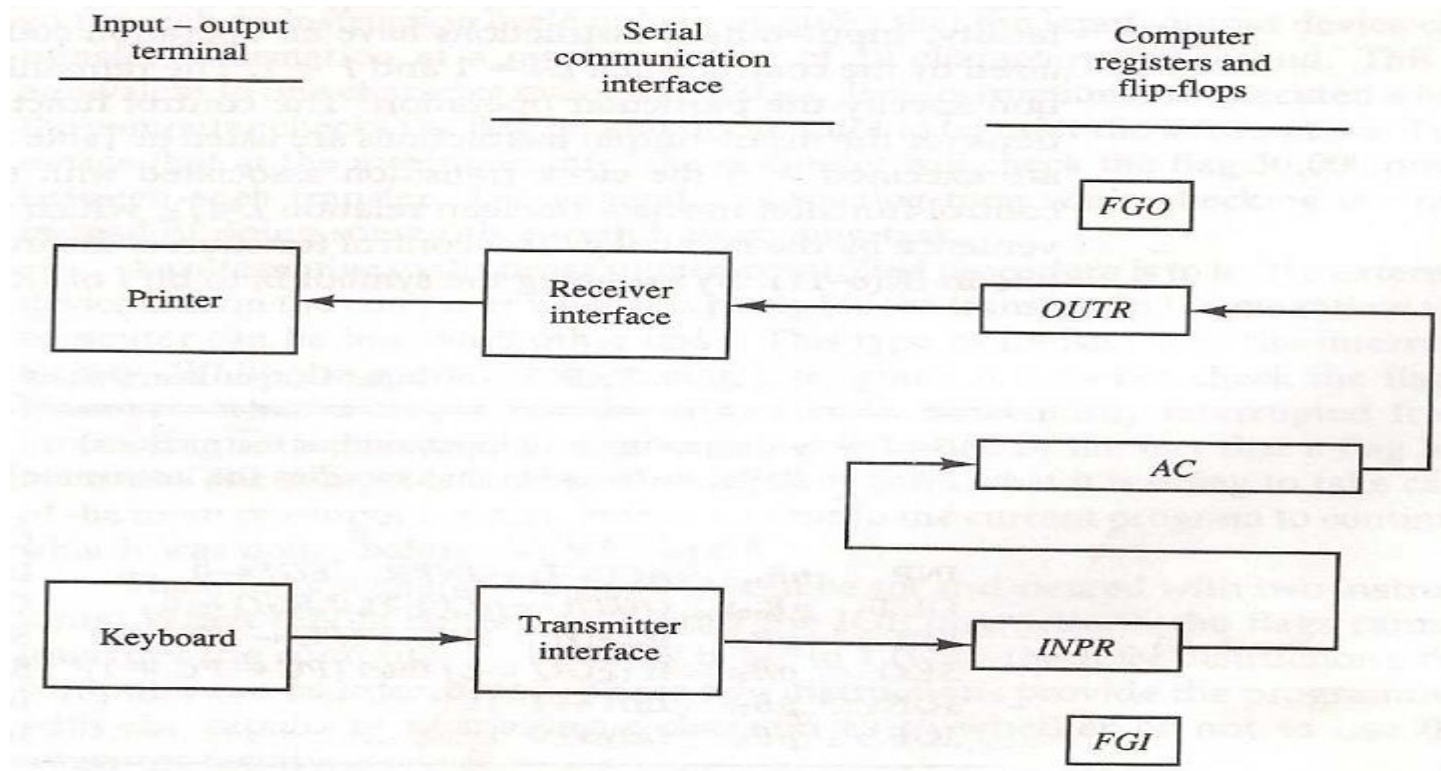
Input-Output Configuration

- As long as the flag is set, the information in INPR cannot be changed by striking another key.



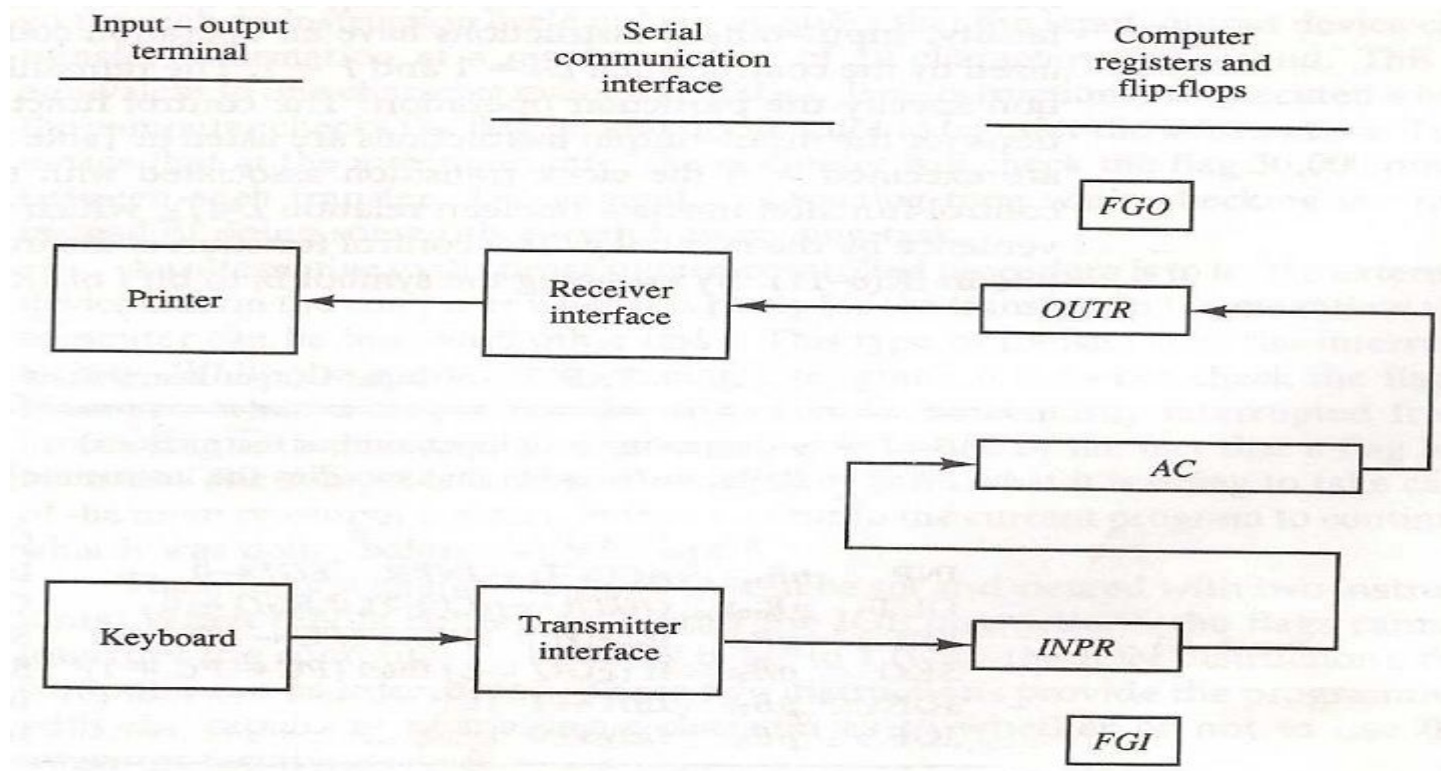
Input-Output Configuration

- Initially, the output flag FGO is set to 1.
- The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0.



Input-Output Configuration

- The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1.
- The computer does not load a new character into OUTR when FGO is 0.



Input-Output Instructions

- Input and output instructions are needed for transferring information to and from AC register, for checking the flag bits, and for controlling the interrupt facility.
- Input-output instructions have an operation code 1111 and are recognized by the control when $D_7 = 1$ and $I = 1$.
- The remaining bits of the instruction specify the particular operation.

Input-Output Instructions

- These instructions are executed with the clock transition associated with timing signal T_3 .
- Each control function needs a Boolean relation D_7IT_3 , which we designate for convenience by the symbol p .

Input-Output Instructions			
$D_7IT_3 = p$ (common to all input-output instructions)			
$IR(i) = B_i$ [bit in $IR(6-11)$ that specifies the instruction]			
	p :	$SC \leftarrow 0$	Clear SC
INP	pB_{11} :	$AC(0-7) \leftarrow INPR, FGI \leftarrow 0$	Input character
OUT	pB_{10} :	$OUTR \leftarrow AC(0-7), FGO \leftarrow 0$	Output character
SKI	pB_9 :	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8 :	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7 :	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6 :	$IEN \leftarrow 0$	Interrupt enable off

Input-Output Instructions

- The control function is distinguished by one of the bits in IR (6-11).
- By assigning the symbol B_i to bit i of IR, all control functions can be denoted by ${}_pB_i$ for $i = 6$ through 11.

Input-Output Instructions			
$D_7IT_3 = p$ (common to all input-output instructions)			
$IR(i) = B_i$ [bit in $IR(6-11)$ that specifies the instruction]			
	p :	$SC \leftarrow 0$	Clear SC
INP	${}_pB_{11}$:	$AC(0-7) \leftarrow INPR, FGI \leftarrow 0$	Input character
OUT	${}_pB_{10}$:	$OUTR \leftarrow AC(0-7), FGO \leftarrow 0$	Output character
SKI	${}_pB_9$:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	${}_pB_8$:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	${}_pB_7$:	$IEN \leftarrow 1$	Interrupt enable on
IOF	${}_pB_6$:	$IEN \leftarrow 0$	Interrupt enable off

Input-Output Instructions

- The sequence counter SC is cleared to 0 when $p=D_7IT_3=1$.
- The last two instructions set and clear an interrupt enable flip-flop IEN.

Input-Output Instructions			
$D_7IT_3 = p$ (common to all input-output instructions)			
$IR(i) = B_i$ [bit in $IR(6-11)$ that specifies the instruction]			
	$p:$	$SC \leftarrow 0$	Clear SC
INP	$pB_{11}:$	$AC(0-7) \leftarrow INPR, FGI \leftarrow 0$	Input character
OUT	$pB_{10}:$	$OUTR \leftarrow AC(0-7), FGO \leftarrow 0$	Output character
SKI	$pB_9:$	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	$pB_8:$	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	$pB_7:$	$IEN \leftarrow 1$	Interrupt enable on
IOF	$pB_6:$	$IEN \leftarrow 0$	Interrupt enable off

Program Interrupt

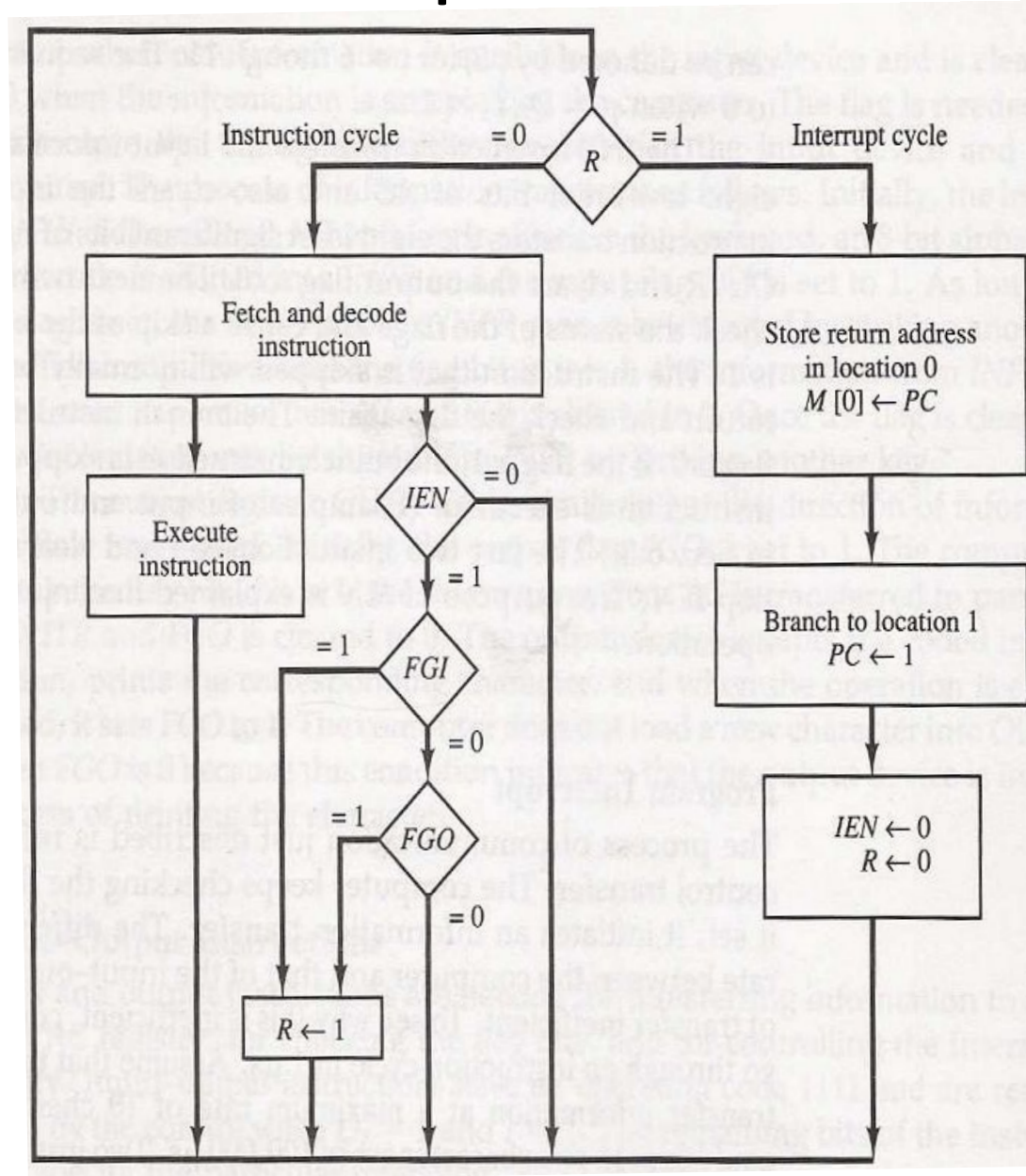
- The computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer.
- The difference of information flow rate between the computer and that of the input—output device makes this type of transfer inefficient.
- An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer.
- In the meantime the computer can be busy with other tasks. This type of transfer uses the interrupt facility.
- While the computer is running a program, it does not check the flags.

Program Interrupt

- When a flag is set, the computer is momentarily interrupted from the current program.
- The computer deviates momentarily from what it is doing to perform of the input or output transfer.
- It then returns to the current program to continue what it was doing before the interrupt.
- The interrupt enable flip-flop IEN can be set and cleared with two instructions.
 - When IEN is cleared to 0 (with the IOF instruction), the flags cannot interrupt the computer.
 - When IEN is set to (with the ION instruction), the computer can be interrupted.

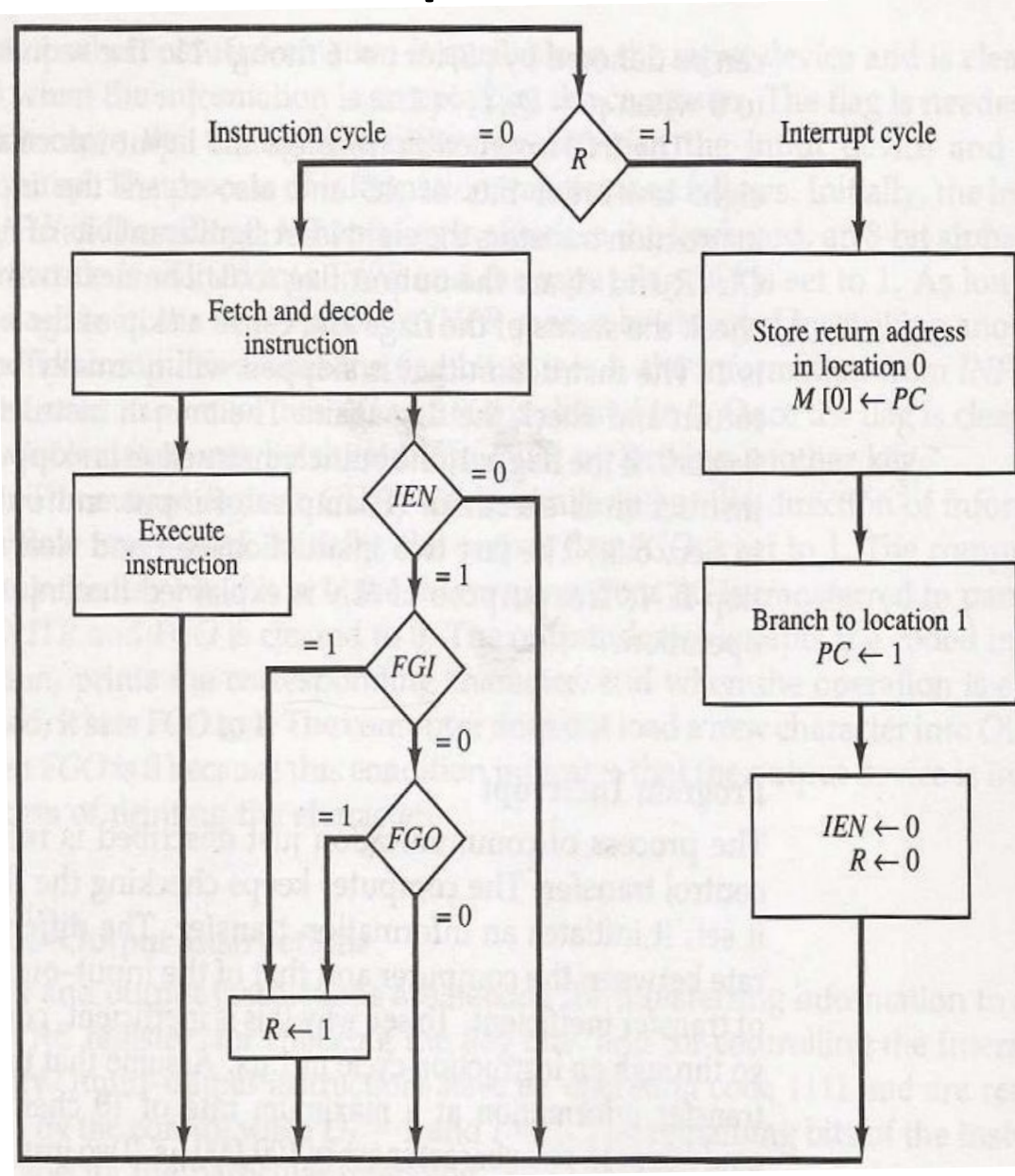
Program Interrupt

- An interrupt flip-flop R is included in the computer. When $R = 0$, the computer goes through an instruction cycle.
- During the execute phase of the instruction cycle IEN is checked by the control.



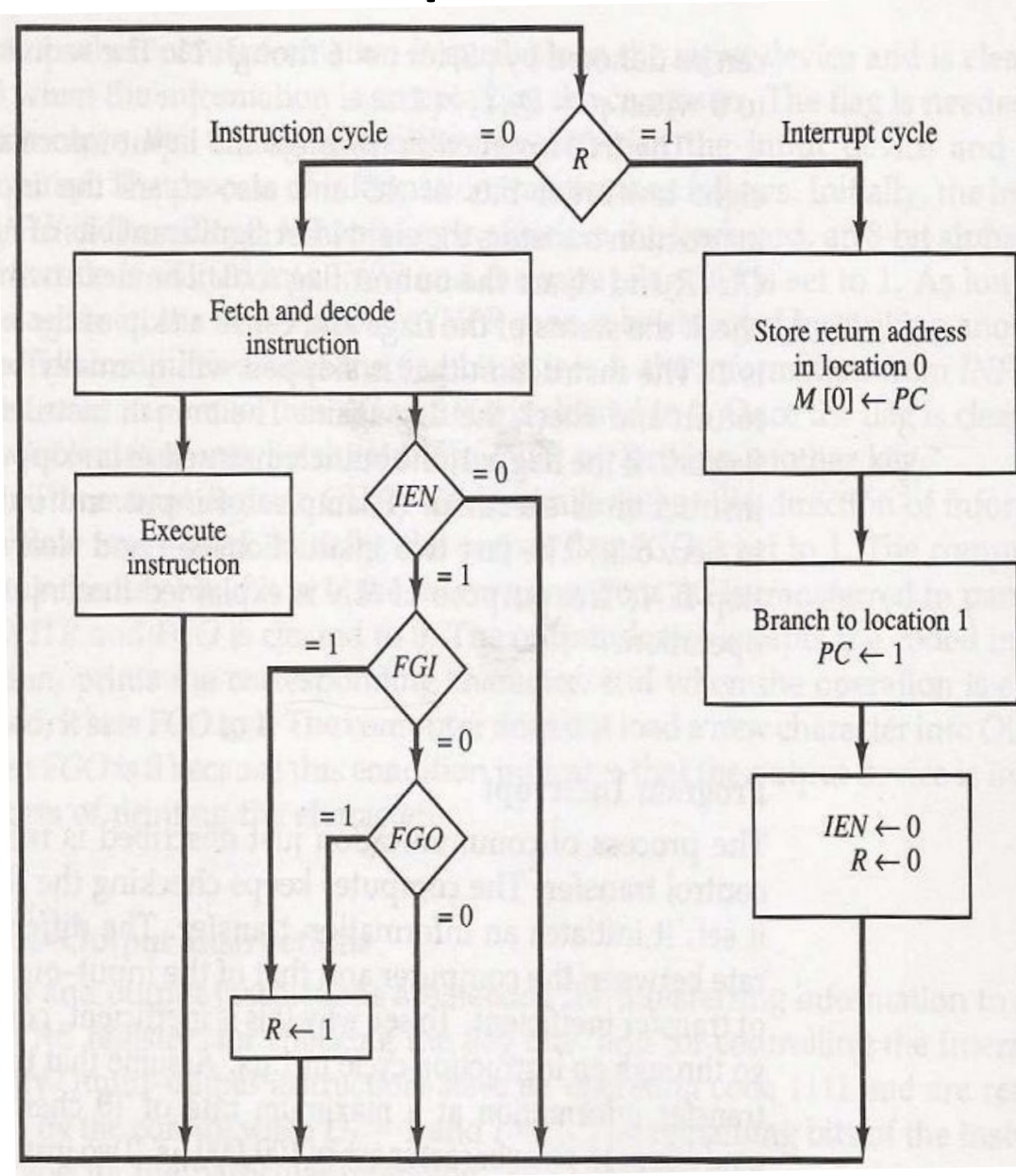
Program Interrupt

- If IEN is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle.
- If IEN is 1, control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information. In this case, control continues with the next instruction cycle.



Program Interrupt

- If either flag is set to 1 while $IEN = 1$, flip-flop R is set to 1. At the end of the execute phase, control checks the value of R , and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.



Interrupt Cycle

- The interrupt cycle is a hardware implementation of a branch and save return address operation.
- The return address available in PC is stored in a specific location.
- This location may be a processor register, a memory stack, or a specific memory location.

Interrupt Cycle

- When an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255.
- At this time, the returns address 256 is in PC.

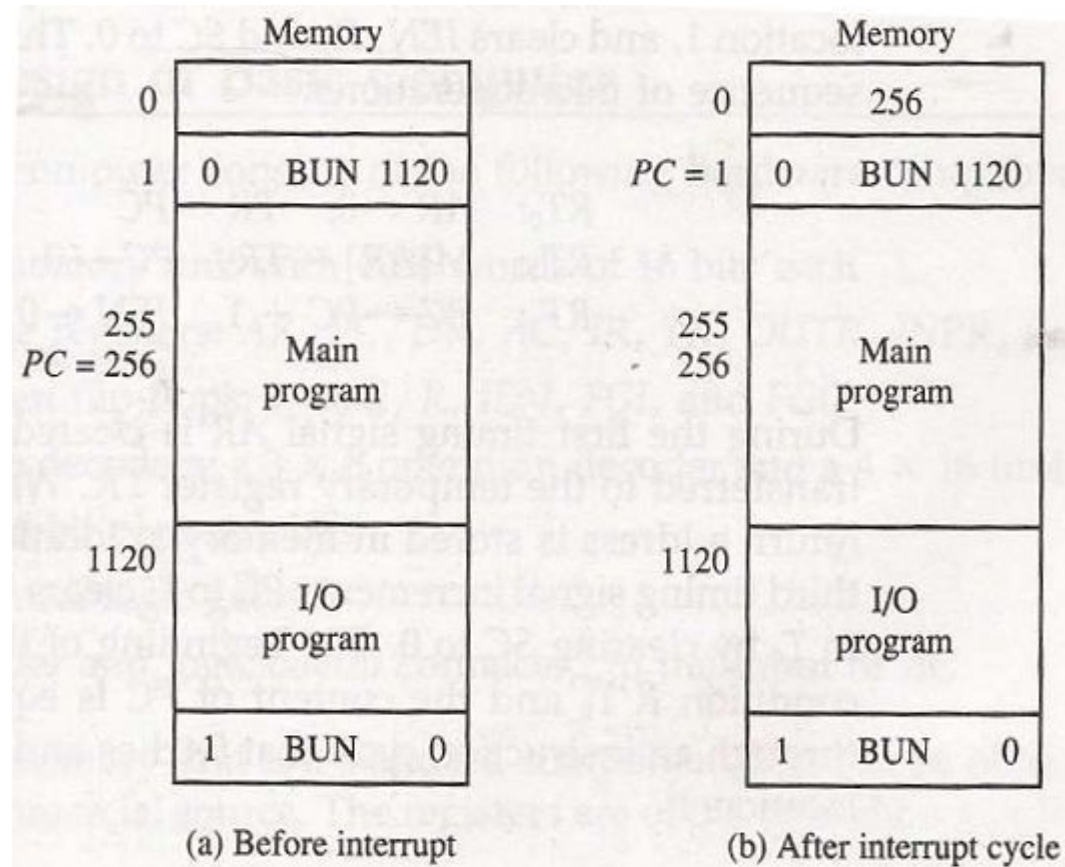


Fig: Demonstration of the interrupt cycle

- The programmer has previously placed an input—output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1.

Interrupt Cycle

- When control reaches timing signal T_0 and finds that $R = 1$, it proceeds with the interrupt cycle.
- The content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0.

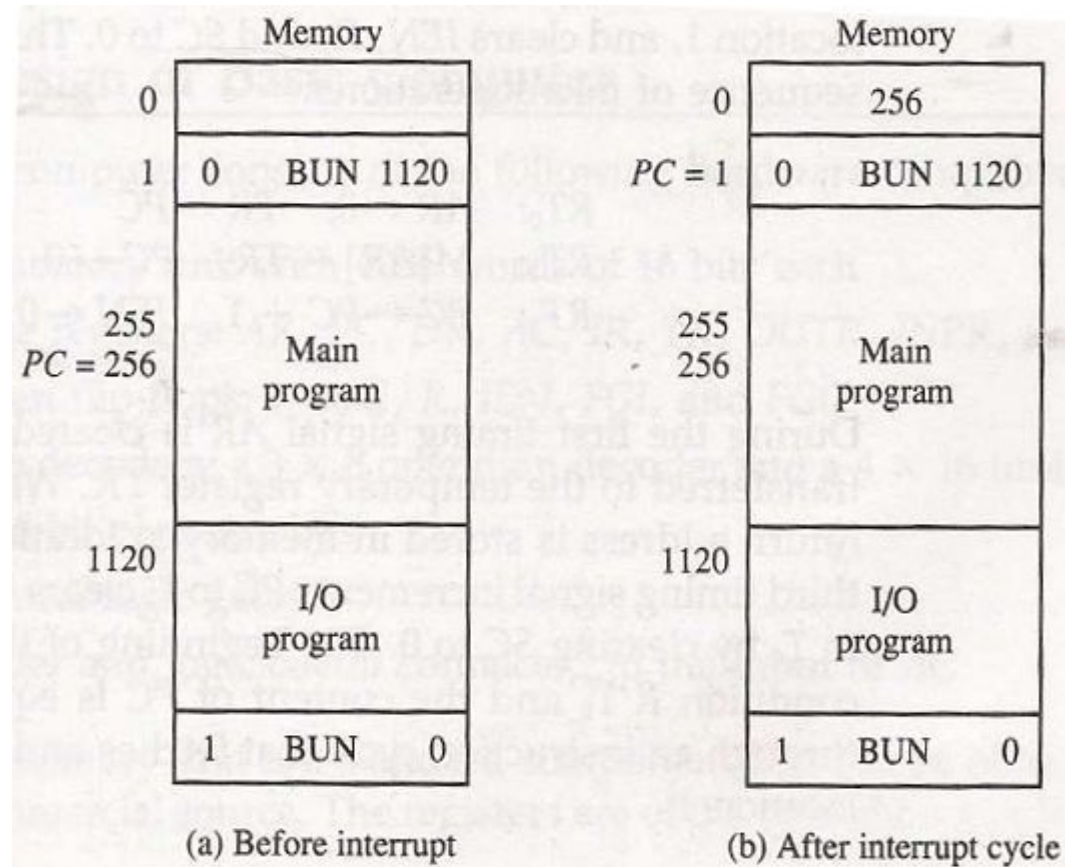


Fig: Demonstration of the interrupt cycle

- The branch instruction at address 1 causes the program to transfer to the input—output service program at address 1120.

Interrupt Cycle

- This program checks the flags, determines which flag is set, and then transfers the required input or output information.
- Once this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location where it was interrupted.

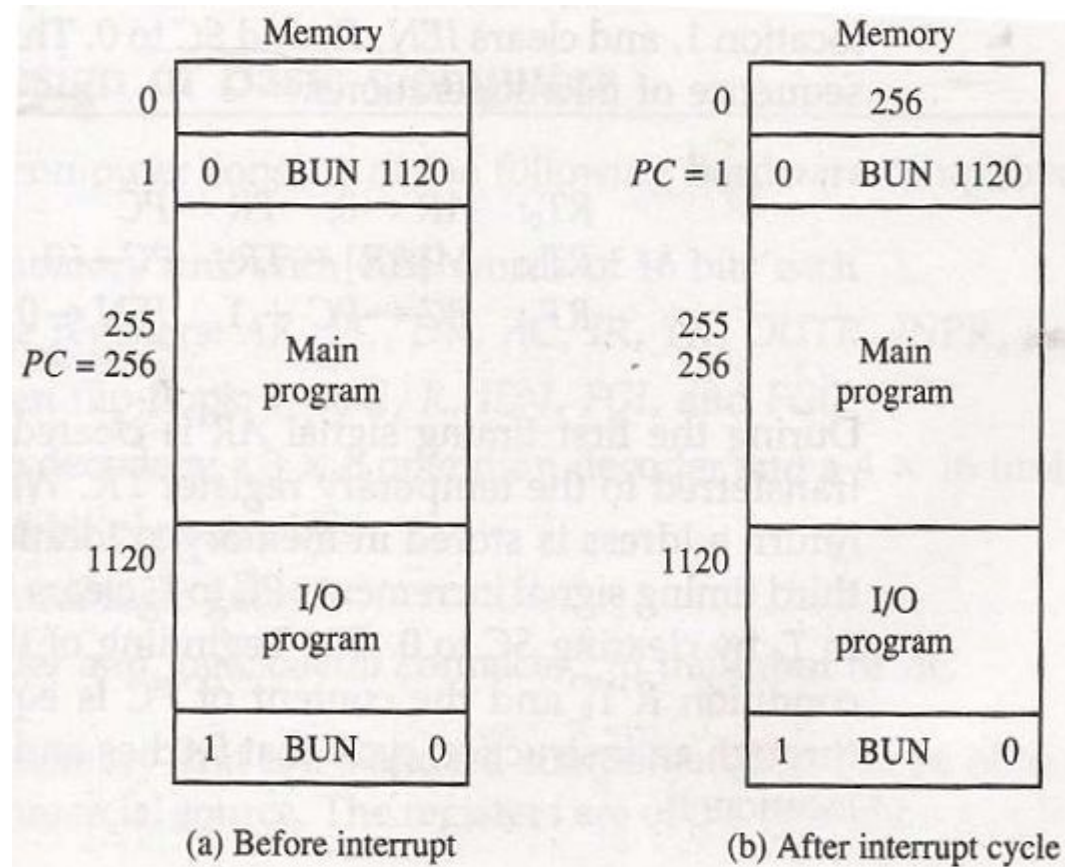


Fig: Demonstration of the interrupt cycle

BUN: Branch Unconditionally

- BUN: Branch Unconditionally is an Memory-Reference Instructions.
- The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally.
- The instruction is executed with one microoperation:

$$D_4T_4: PC \leftarrow AR, SC \leftarrow 0$$

- This instruction transfers the program to the instruction specified by the effective address.