Computer System Architecture

Ms. Yuzana Hlaing
M.E(IT), Ph.D. Candidate(thesis)
Lecturer
Computer Engineering and Information Technology Dept.
Yangon Technological University

Course Schedule

Periods	Lectures
Week-1	Introduction to Computer System Architecture
Week-2	Data Presentations
Week-3	Register Transfer and Microoperations
Week-4	Basic Computer Organization and Design
Week-5	Programming the Basic Computer
Week-6	Microprogrammed Control
Week-7	Central Processing Unit
Week-8	Pipeline and Vector Processing
Week-9	Computer Arithmetic
Week-10	Input-Output Organization
Week-11	Memory Organization
Week-12	Multiprocessors

Lecture-4

Basic Computer Organization and Design

Lecture-4
Basic Computer Organization and Design

- Instruction Codes
- Computer Registers
- Computer Instructions
- Timing and Control
- Instruction Cycle
- Memory -Reference Instructions
- Input -Output and Interrupt
- Compete Computer Description
- Design of Basic Computer
- Design of Accumulator Logic

Instruction Codes

Program:

A **program** is a set of instructions that specify the operations operands, and the sequence by which processing has to occur.

Computer Instruction:

A **computer instruction** is a binary code that specifies a sequence of microoperations for the computer.

Instruction Code:

An **instruction code** is a group of bits that instruct the computer to perform a specific operation.

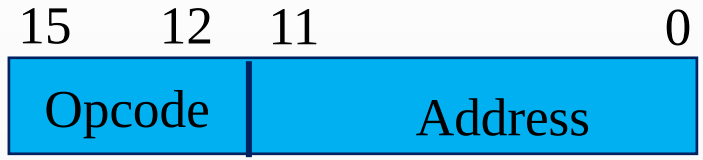
Operation Code:

The **operation code** of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement.

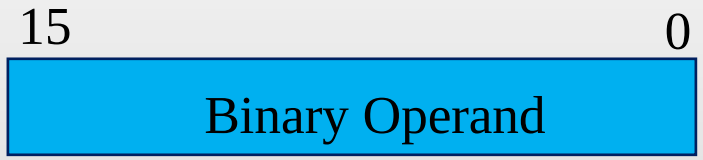
Instruction Codes

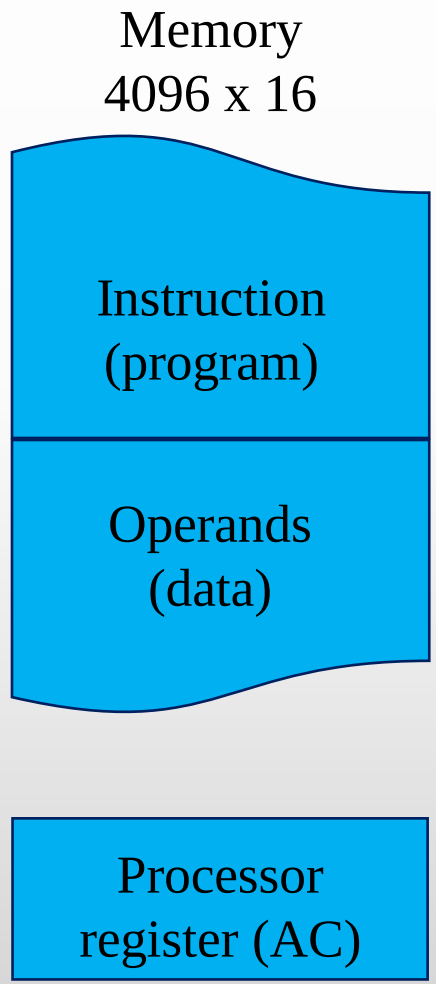
- An instruction code is a group of bits that instruct the computer to perform a specific operation.
- It is usually divided into two parts: operation code and address , each having its own particular interpretation.
- The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement.
- For every operation code, the control issues a sequence of microoperations needed for the hardware implementation of the specified operation.
- For this reason, an operation code is sometimes called a macrooperation because it specifies a set of microoperations.
- Address defines the registers or the memory words where the operands are to be found, as well as the register or memory word where the result is to be stored.
- Memory words can be specified in instruction codes by their address.

Stored program organization



Instruction format

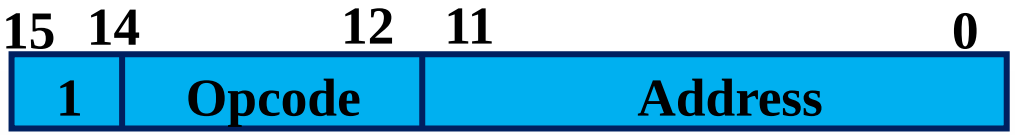


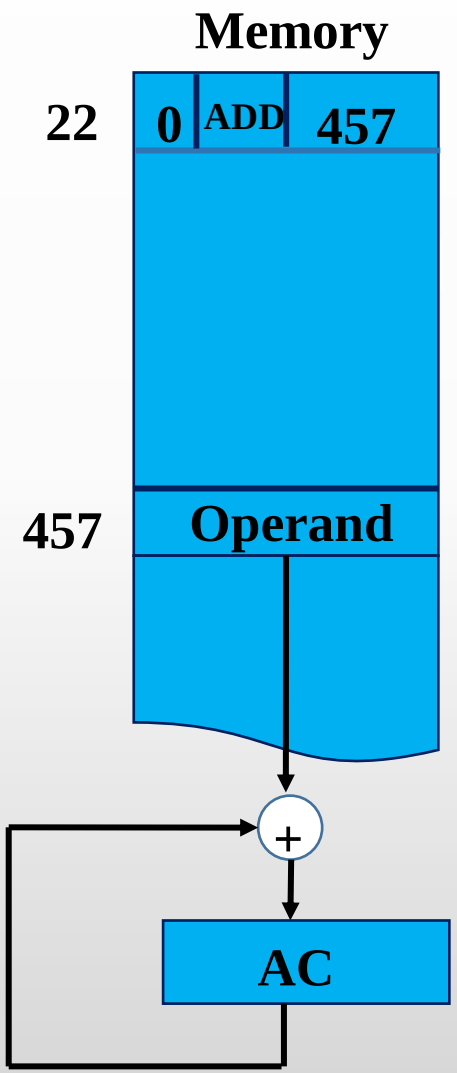


Indirect Address

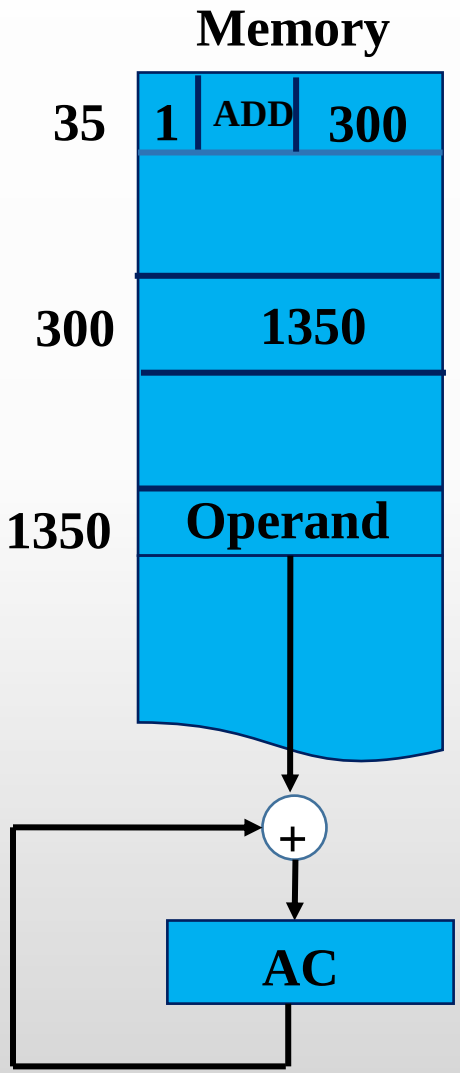
- When the second part of an instruction code specifies an operand, the instruction is said to have an immediate operand.
- When the second part specifies the address of an operand, the instruction is said to have a **direct address**.
- The second part of the instruction designate an address of a memory word in which the address of the operand is found is said to have an **indirect address**.
- The indirect address instruction needs two references to memory to fetch an operand.
  - The first reference is needed to read the address of the operand;
  - The second reference is for the operand itself.
- One bit of the instruction code can be used to distinguish between a direct and an indirect address.

(a) Instruction Format





(b) Direct Address



(c) Indirect Address

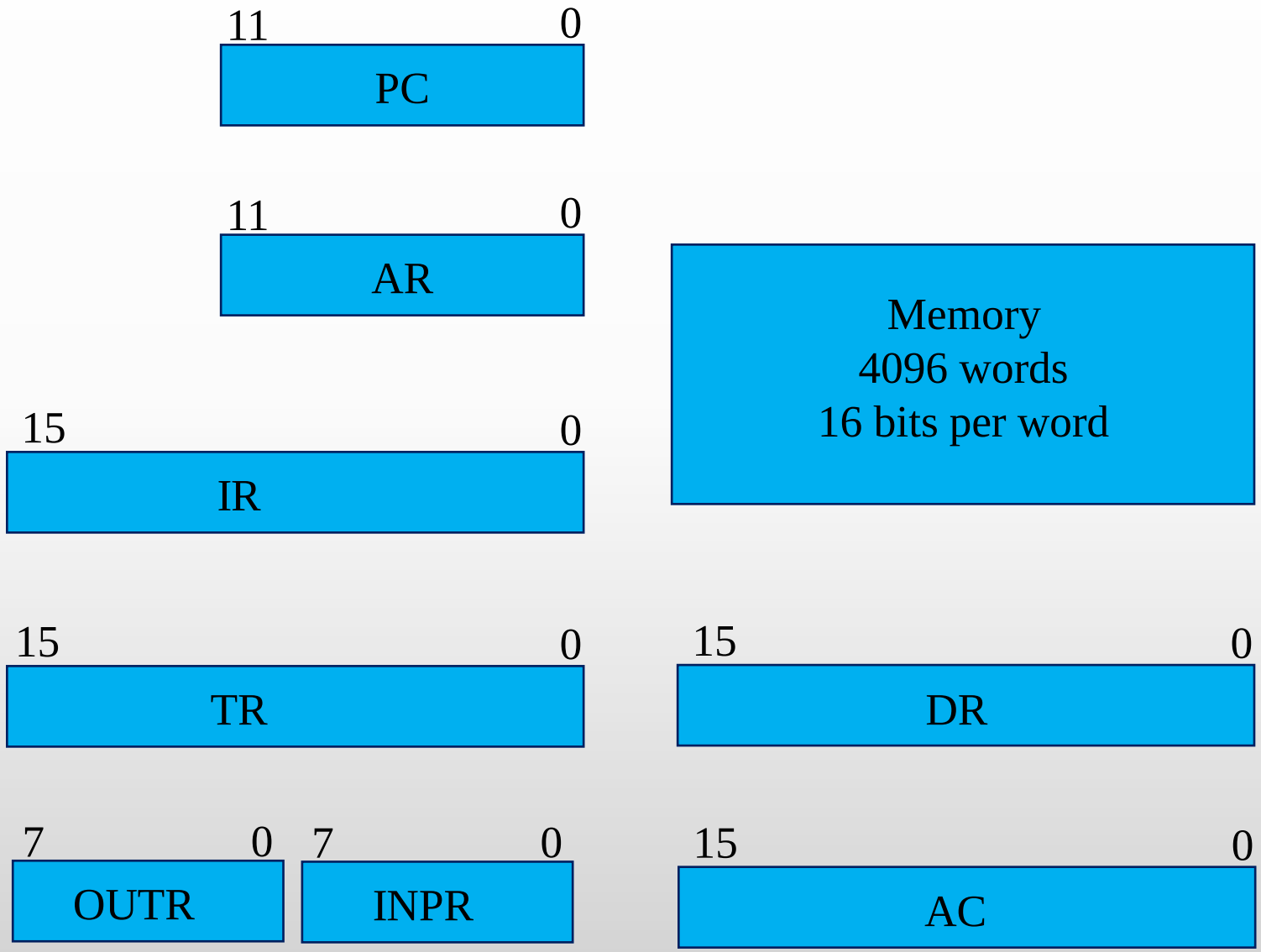
Computer Registers

Computer Registers

List of registers for the basic computer

Register Symbol	Number of Bits	Register Name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

Basic computer registers and memory



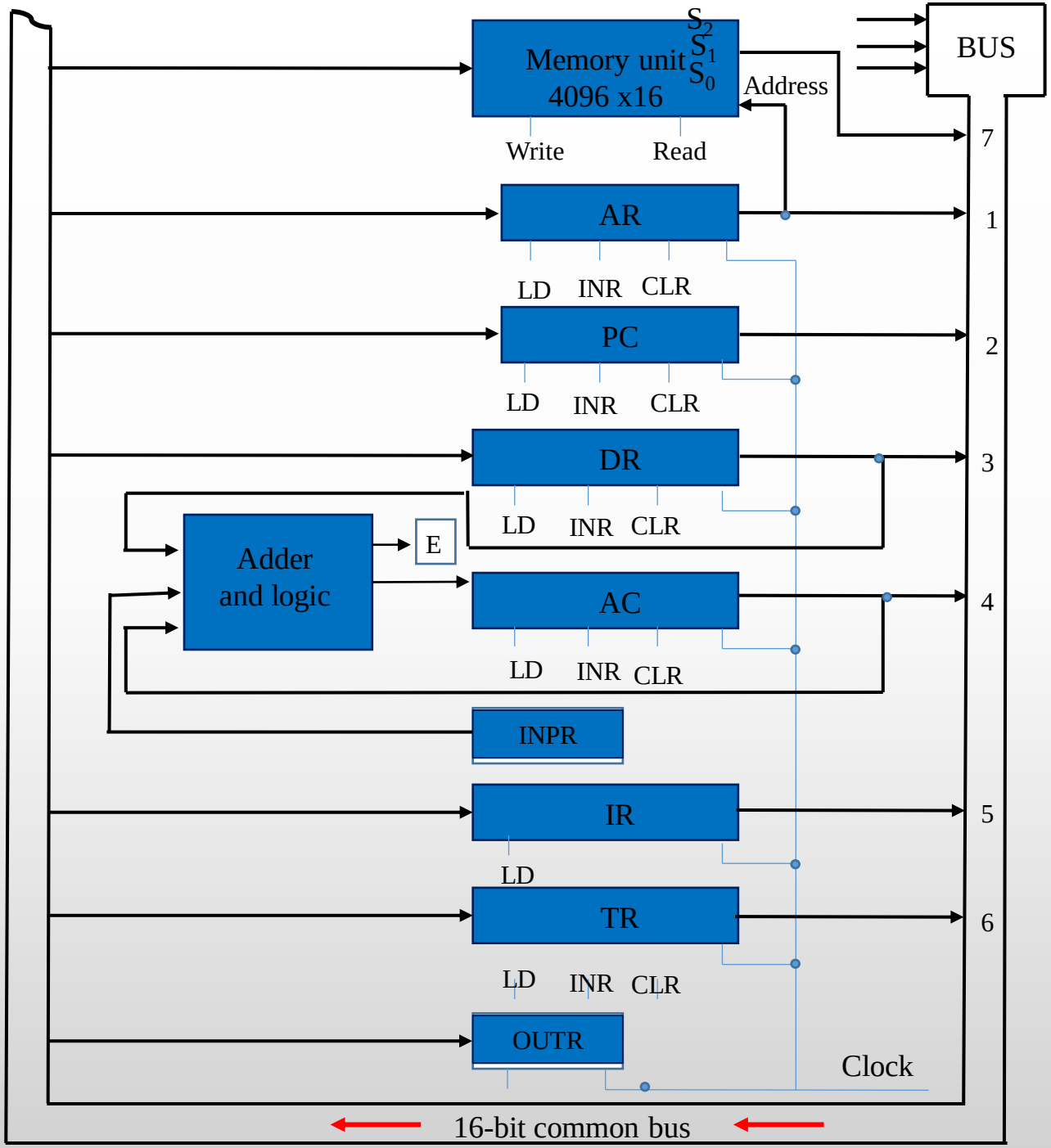
Common Bus System

- Instruction and data can be transferred among the registers and memory unit by using a common bus system.
- A common bus system for basic computer is designed using multiplexers.
- For example, a common bus system for the following register transfer statements can be considered by performing its process.

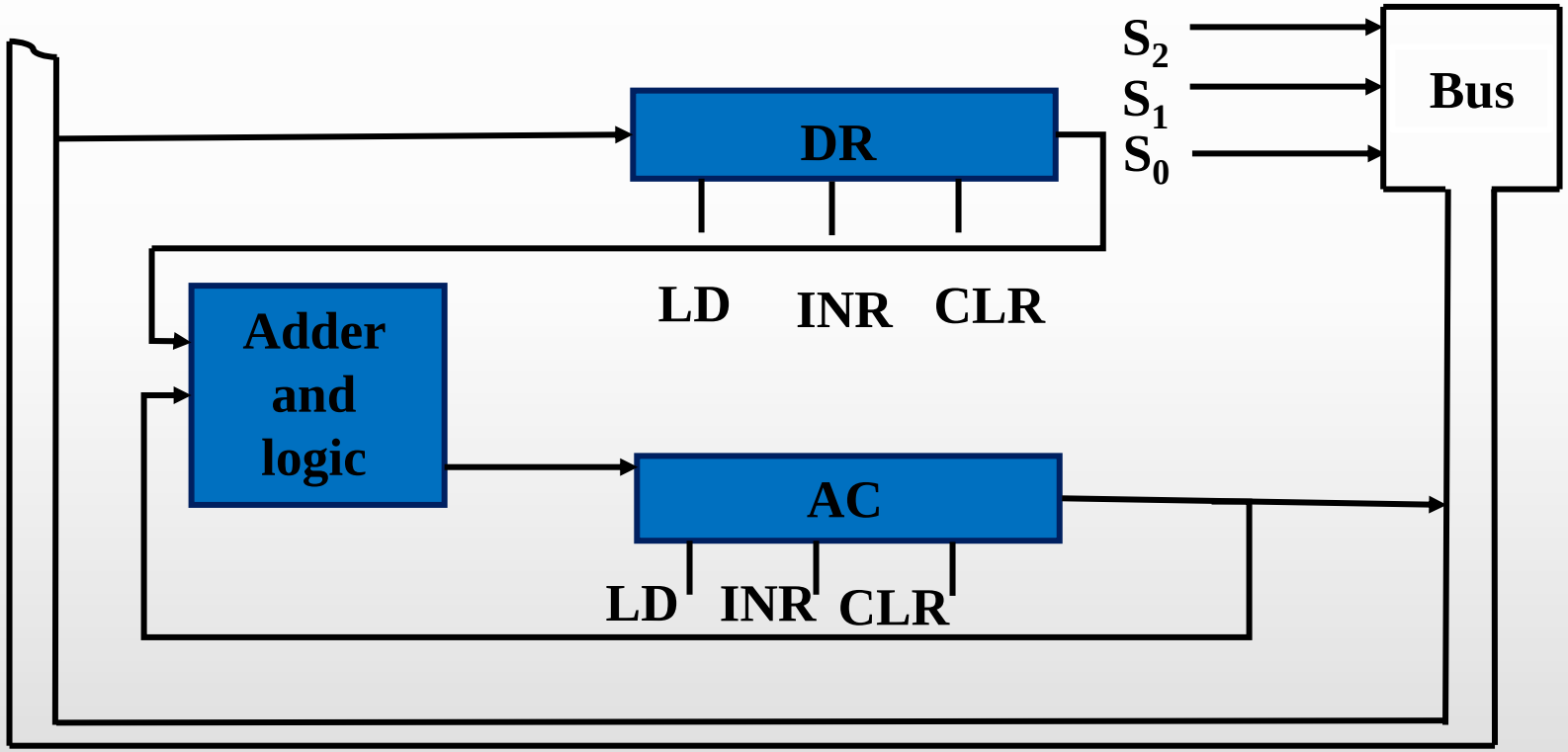
DR ← AC and **AC ← DR**

- These two microoperations can be executed at the same time.
  1. Place the content of AC onto the bus by making the bus selection inputs $S_2S_1S_0$ equal to **100**.
  2. Transfer the content of the bus to **DR** by enabling **LD** input of **DR**, transferring the content of **DR** through the adder and logic circuit into AC, and enabling the **LD** input of **AC**, all during the same clock cycle.

Common Bus System for Basic Computer System



Common Bus System for Specified Register Transfer Statements



Computer Instructions

Computer Instructions

- The basic computer has three instruction code formats.
- 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.
- The register- reference instructions are recognized by the operation code **111** with a **0** in the leftmost bit (bit **15**) of the instruction.
- An input-output instruction is recognized by the operation code **111** with a **1** in the leftmost bit of the instruction.

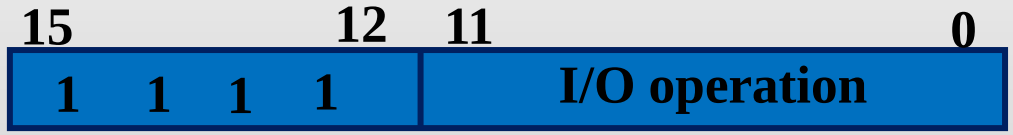
Basic Computer Instruction Formats

(Opcode = 000 through 110)

(a) Memory-reference instruction

(Opcode = 111, I = 0)

(b) Register-reference instruction



(Opcode = 111, I = 1)

(c) Input-output instruction

For Memory-reference Instruction

Symbol	Hexadecimal code I=0	Hexadecimal code I=1	Description
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

For Register-reference Instruction

Symbol	Hexadecimal code	Description
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 ins: if AC positive
SNA	7008	Skip next ins: if AC negative
SZA	7004	Skip next ins: if AC zero
SZE	7002	Skip next ins: if E zero
HLT	7001	Halt computer

For Input-Output Instruction

Symbol	Hexadecimal code	Description
INP	F800	Input character to AC
OUT	F400	Output character from AC
SKI	F200	Skip on input flag
SKO	F100	Skip on output flg
ION	F080	Interrupt on
IOF	F040	Interrupt off

Instruction Set Completeness

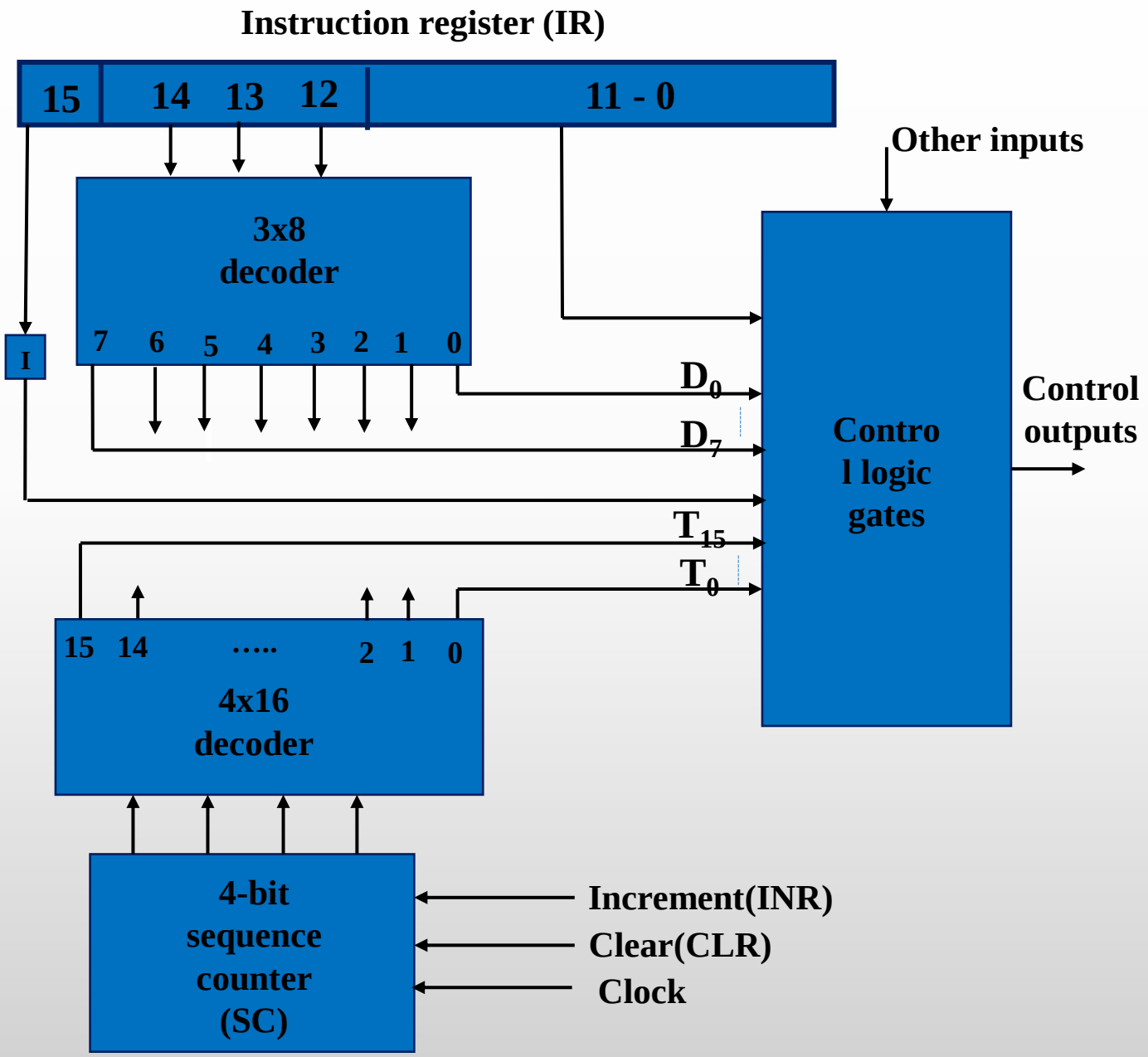
- A computer has a set of instructions so that the user can construct machine language programs to evaluate any function to be computed.
- The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:
  1. Arithmetic, logical, and shift instructions
  2. Instructions for moving information to and from memory and processor registers
  3. Program control instructions together with instructions that check status conditions
  4. Input and output instructions
- Arithmetic, logic, and shift instructions provide computational capabilities for the processing the type of data.

Timing and Control

Timing and Control

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

Control Unit of Basic Computer



Instruction Cycle

Instruction Cycle

- The program is executed in the computer by going through a cycle for each instruction.
- Each instruction cycle is subdivided into a sequence of subcycles or phases.
- In the basic computer each instruction cycle consists of the following phases:
  1. Fetch an instruction from memory.
  2. Decode the instruction.
  3. Read the effective address from memory if the instruction has an indirect address.
  4. Execute the instruction.
- This process continues unless a HALT instruction is encountered.

Fetch and Decode

- The register transfer statements for the fetch and decode phases .

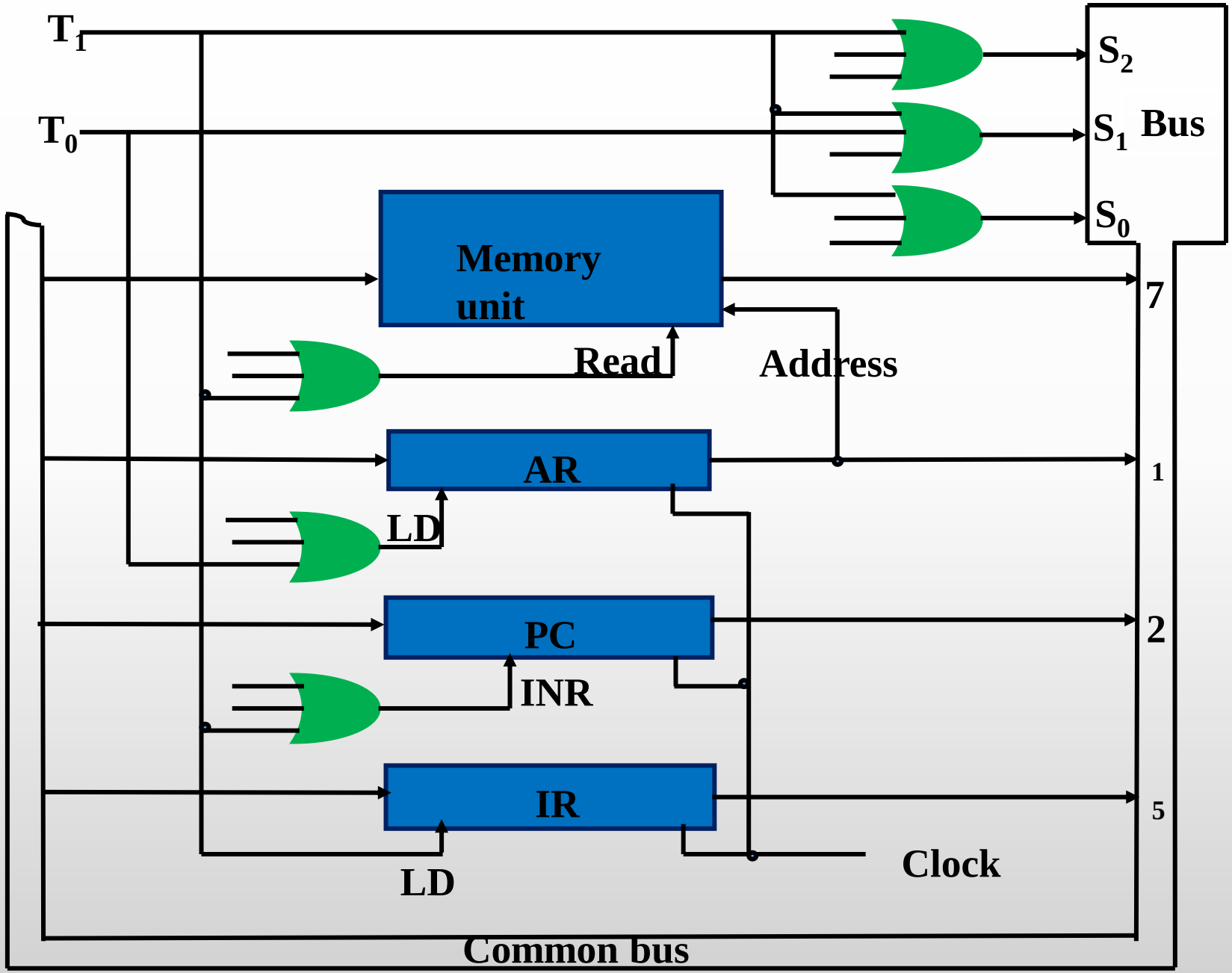
  T_0 : AR ← PC

  T_1 : IR ← M[AR], PC← PC + 1

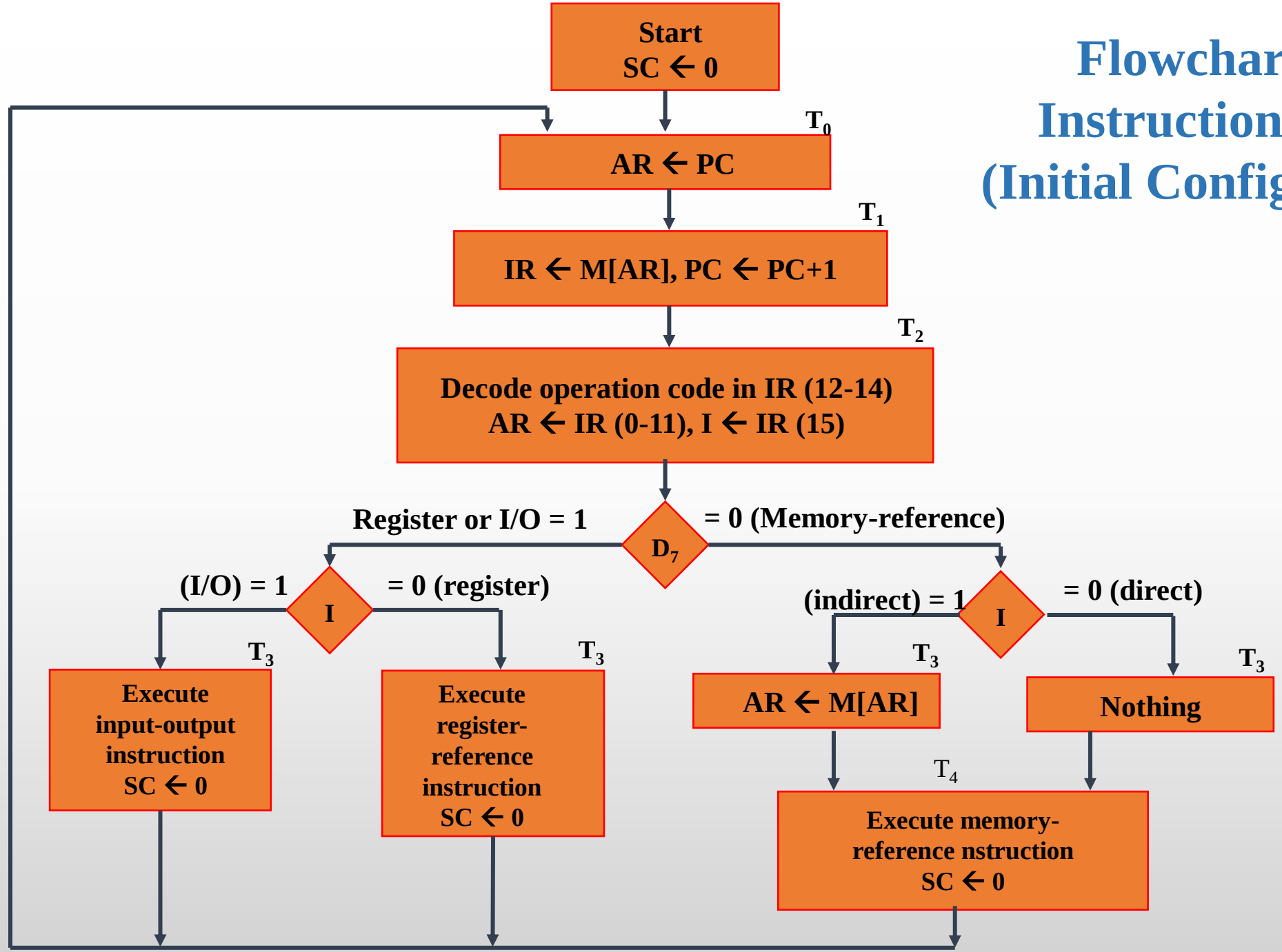
  T_2 : D_0 , . . . , D_7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)

- To provide the data path for the transfer of **PC** to **AR** need to apply timing signal **T_0** to achieve the following connection:
  1. Place the content of **PC** onto the bus by making the bus selection inputs **$S_2S_1S_0$** equal to **010**.
  2. Transfer the content of the bus to **AR** by enabling **LD** input of **AR**.
- It is necessary to use timing signal **T_1** to provide the following connections in the bus system.
  1. Enable the read input of memory.
  2. Place the content of memory onto the bus by making **$S_2S_1S_0$ =111**.
  3. Transfer the content of the bus to **IR** by enabling **LD** input of **IR**.
  4. Increment **PC** by enabling the **INR** input of **PC**.

Register Transfers for Fetch Phase



Flowchart for Instruction Cycle (Initial Configuration)



- In instruction cycle, the processes of each type of instructions are expressed as register transfer statements.
- The three instruction types are subdivided into four separate paths. This can be symbolized as follows:
- For memory-reference instruction, indirect address mode;

  D'_7IT_3: AR ← M[AR]
- For memory-reference instruction, direct address mode;

  $D'_7I'T_3$: Nothing
- For register-reference instruction;

  $D_7I'T_3$: Execute a register-reference instruction
- For input-out instruction

  D_7IT_3: Execute an input-output 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 ← 0	Clear SC
CLA	rB_{11}:	AC ← 0	Clear AC
CLE	rB_{10}:	E ← 0	Clear E
CMA	rB_9:	AC ← $\overline{AC}$	Complement AC
CME	rB_8:	E ← $\overline{E}$	Complement E
CIR	rB_7:	AC←shr AC, AC(15)←E, E←AC(0)	Circular right
CIL	rB_6:	AC←shl AC, AC(0)←E, E←AC(15)	Circular left
INC	rB_5:	AC←AC+1	Increment AC
SPA	rB_4:	If (AC(15) = 0)then (PC←PC+1)	Skip if positive
SNA	rB_3:	If (AC(15) = 1)then (PC←PC+1)	Skip if negative
SZA	rB_2:	If (AC = 0)then (PC←PC+1)	Skip if AC zero
SZE	rB_1:	If E = 0)then (PC←PC+1)	Skip if E zero
HLT	rB_0:	S ← 0 (S is a start-stop flip-flop)	Halt computer

Memory Reference Instruction

Memory-reference Instructions

- The different memory-reference instructions are described in the following table.

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

- For each memory reference instructions, the microoperations required to execute each of these instructions are expressed in the followings:

AND to AC (operation decoder-D_0)

D_0T_4 : DR ← M[AR]
D_0T_5 : AC ← AC ^ DR, SC ← 0

ADD to AC (operation decoder-D_1)

D_1T_4 : DR ← M[AR]
D_1T_5: AC ← AC + DR, E ← C_{out}, SC ← 0

LDA: Load to AC (operation decoder-D_2)

D_2T_4 : DR ← M[AR]
D_2T_5 : AC ← DR, SC ← 0

STA: Store AC (operation decoder-D_3)

D_3T_4 : M[AR] ← AC, SC ← 0

BUN: Branch Unconditionally (operation decoder-D_4)

D_4T_4 : PC ← AR, SC ← 0

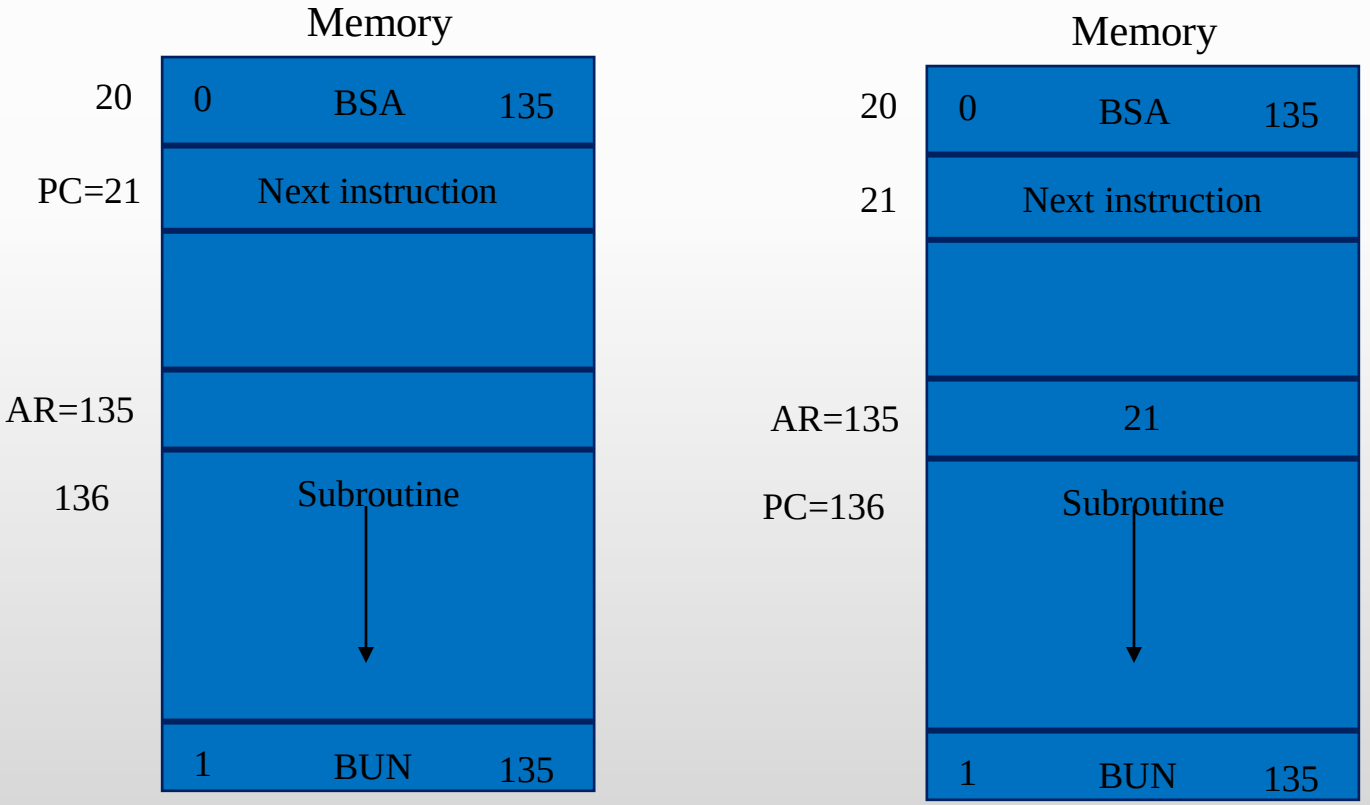
ISZ: Increment and Skip if Zero (operation decoder-D_6)

D_6T_4 : DR ← M[AR],
D_6T_5 : DR ← DR +1
D_6T_6 : M[AR] ← DR, if (DR = 0) then (PC← PC+1), SC ← 0

BSA: Branch and Save Return Address (operation decoder-D_5)

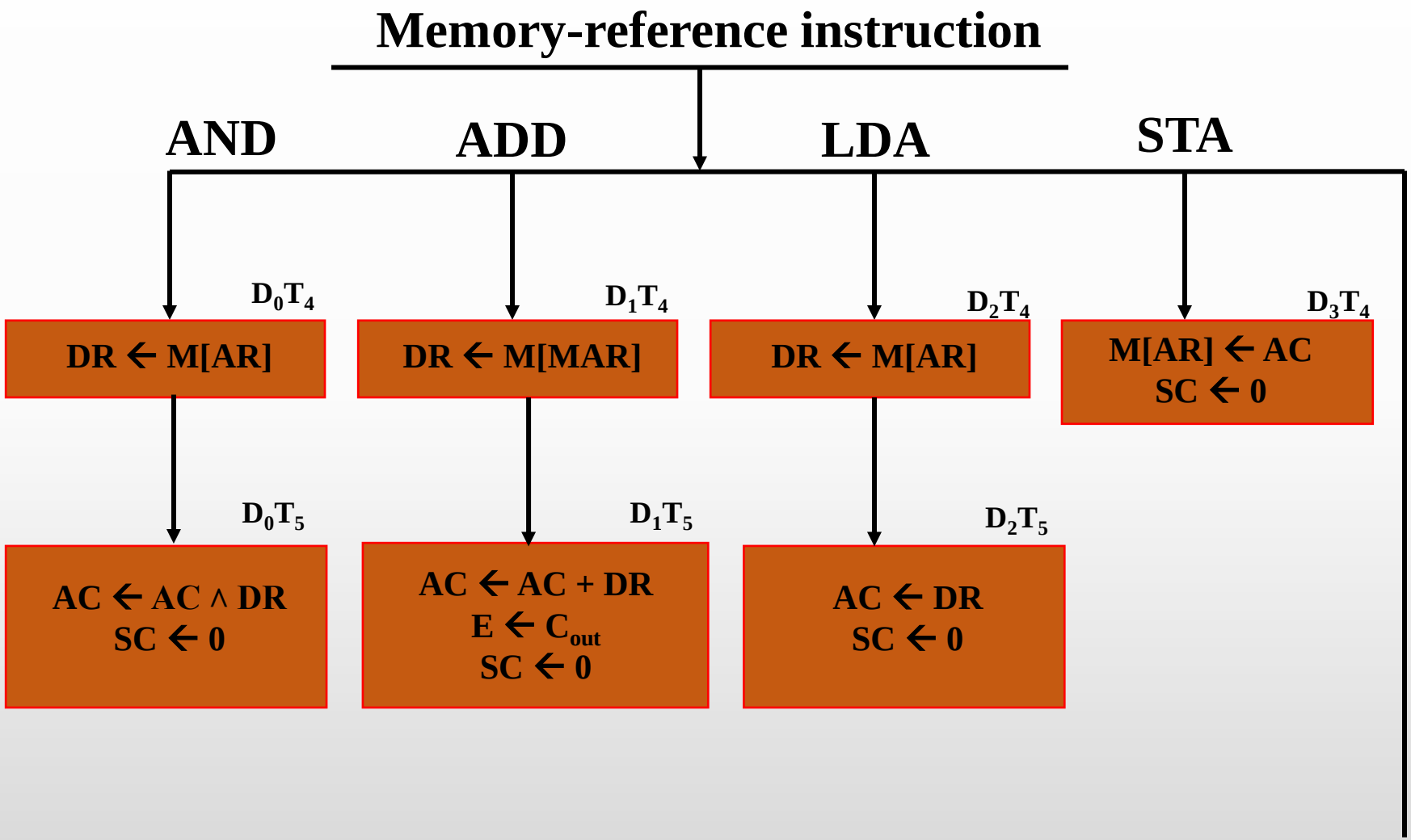
- For BSA, it is needed to save the return address before branch the instruction of the effective address.

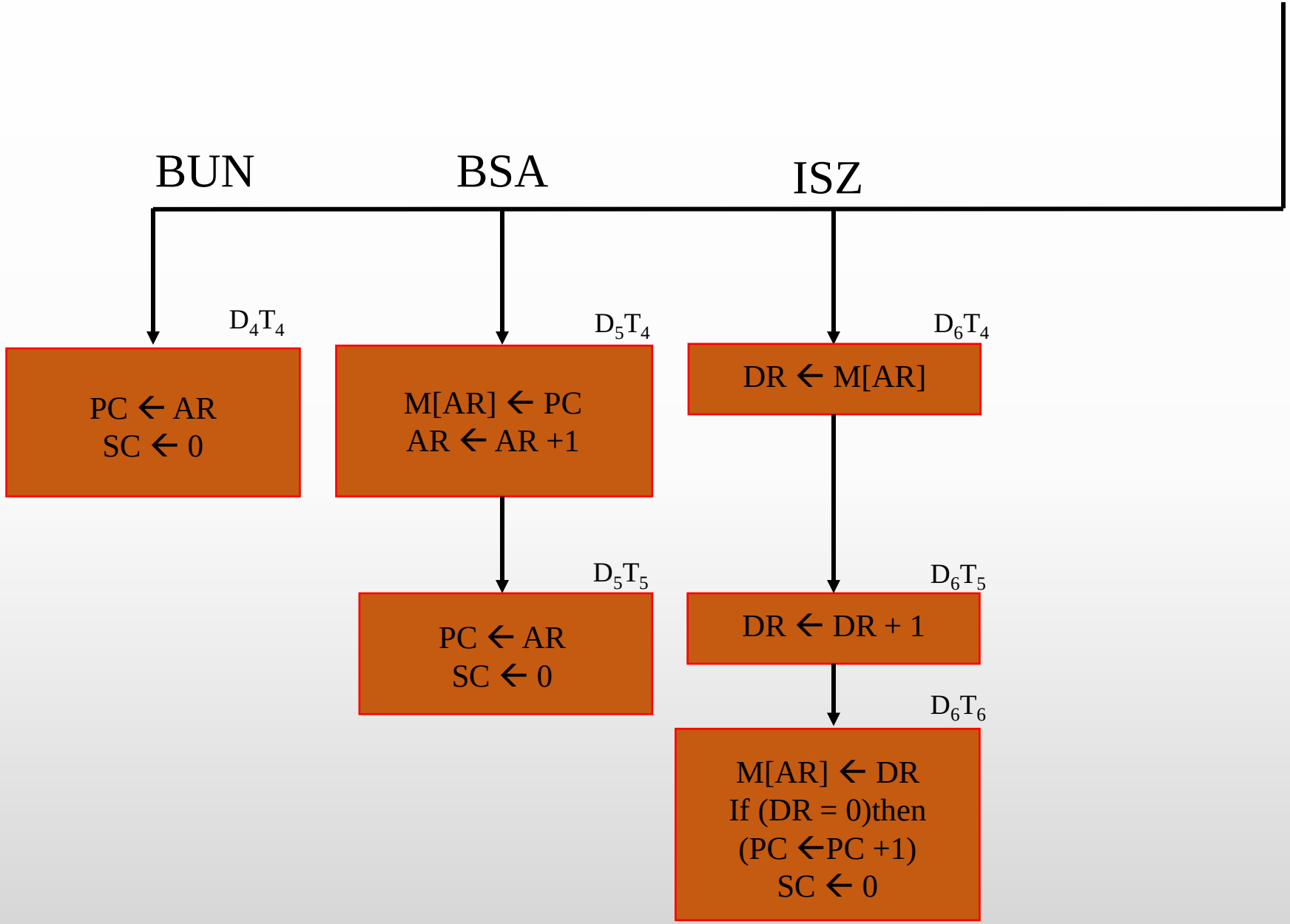
$$D_5T_4 : M[AR] \leftarrow PC, AR \leftarrow AR + 1$$
$$D_5T_5 : PC \leftarrow AR, SC \leftarrow 0$$



(a) Memory, PC, and AR at time T_4 (b) Memory and PC after execution

Flowchart for memory-reference instructions.




BUN
BSA
ISZ
D4T4
PC ← AR
SC ← 0
D5T4
M[AR] ← PC
AR ← AR +1
D6T4
DR ← M[AR]
D5T5
PC ← AR
SC ← 0
D6T5
DR ← DR + 1
D6T6
M[AR] ← DR
If (DR = 0)then
(PC ←PC +1)
SC ← 0

Input-Output and Interrupt

Input-Output and Interrupt

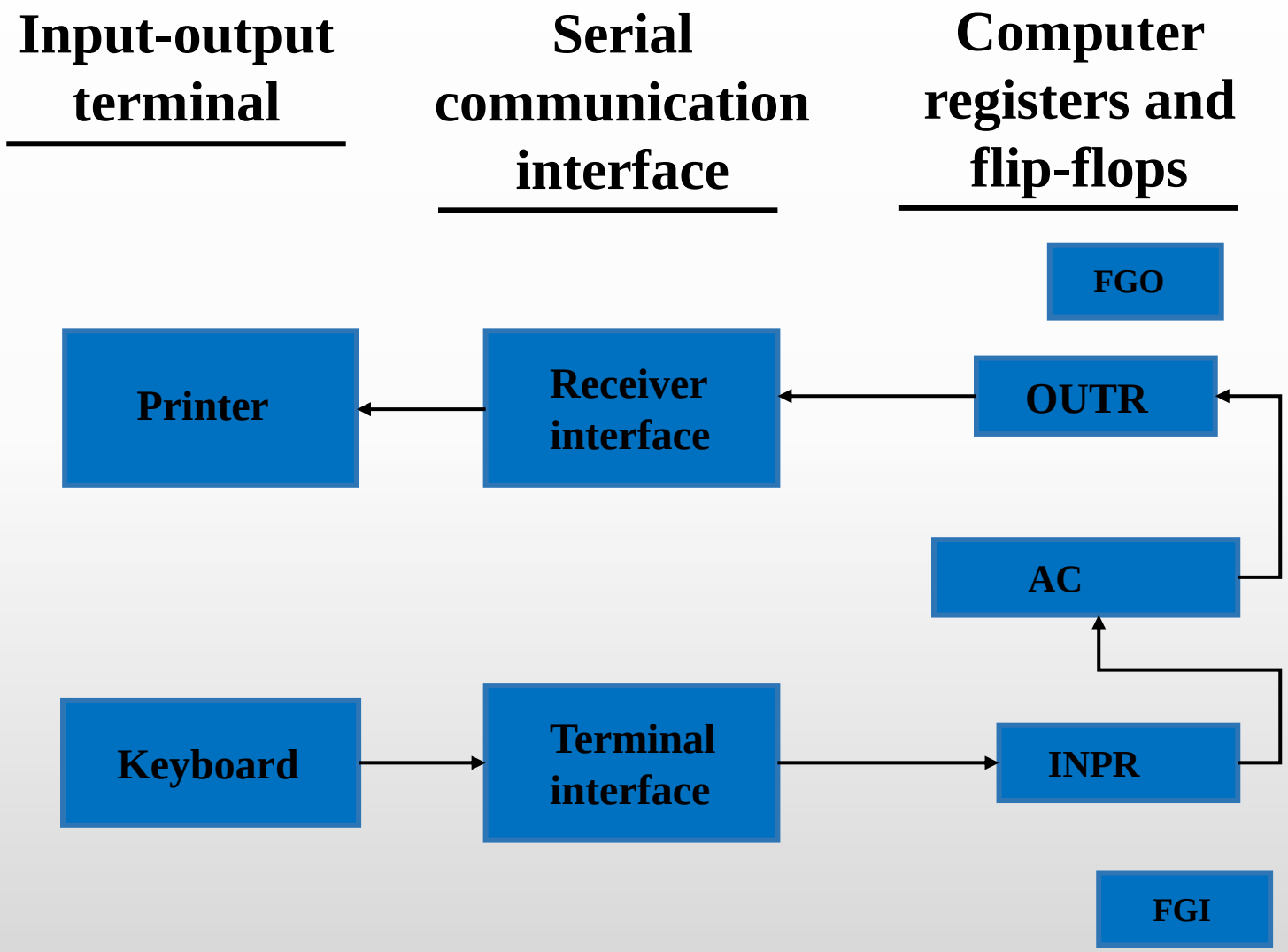


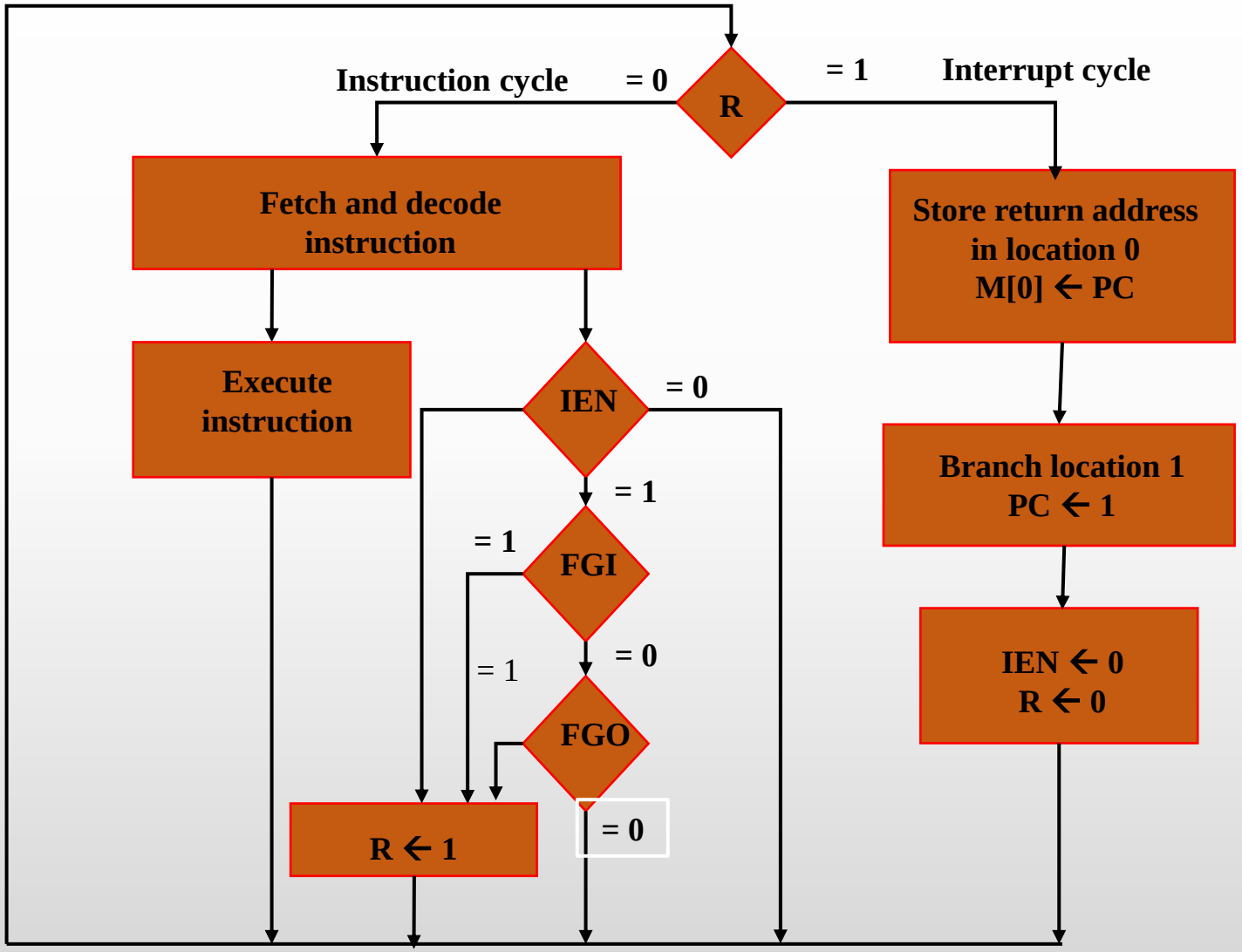
Figure: Input-output configuration.

Input-Output Instructions

D_7IT_3 = p (common to all input-output instructions)
IR (i) = Bi [bit in IR (6-11) that specifies the instruction]

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

Flowchart of Interrupt Cycle



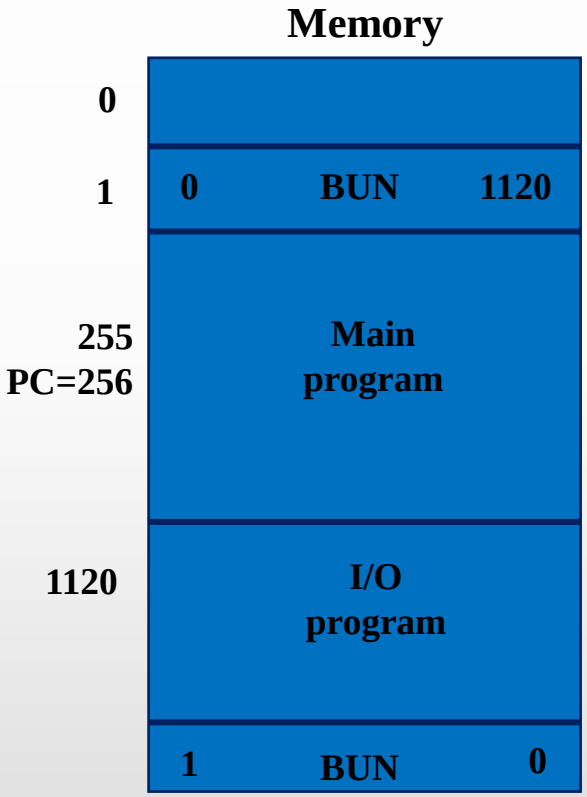
- The register transfer statements and the sequence of microoperations for the interrupt cycle.

$$T_0'T_1'T_2'(IEN)(FGI + FGO): R \leftarrow 1$$

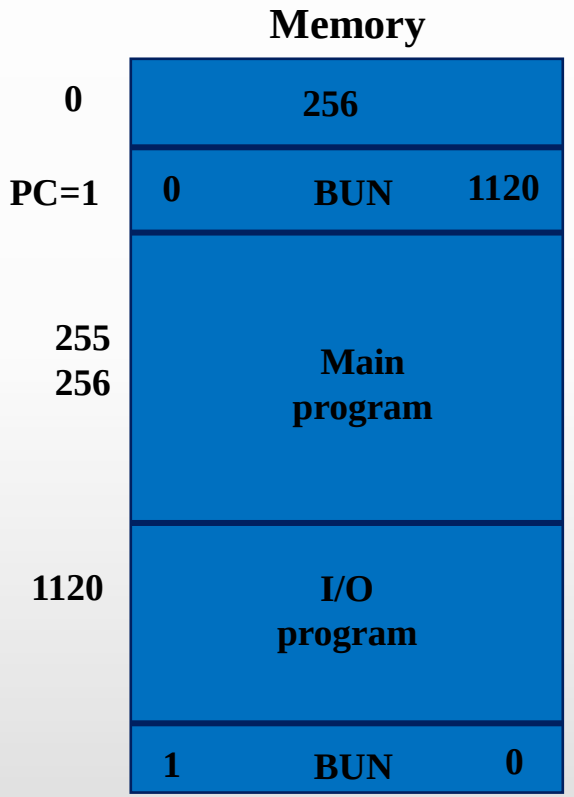
- In the interrupt cycle,
  - stores the return address (available in PC) into memory location 0,
  - branches to memory location 1, and
  - clears IEN, R, and SC to 0.
- This can be done with the following sequence of microoperations:

$$RT_0 : AR \leftarrow 0, TR \leftarrow PC$$
$$RT_1 : M[AR] \leftarrow TR, PC \leftarrow 0$$
$$RT_2 : PC \leftarrow PC + 1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$$

Memory Location Demonstration of Interrupt Cycle



(a) Before interrupt

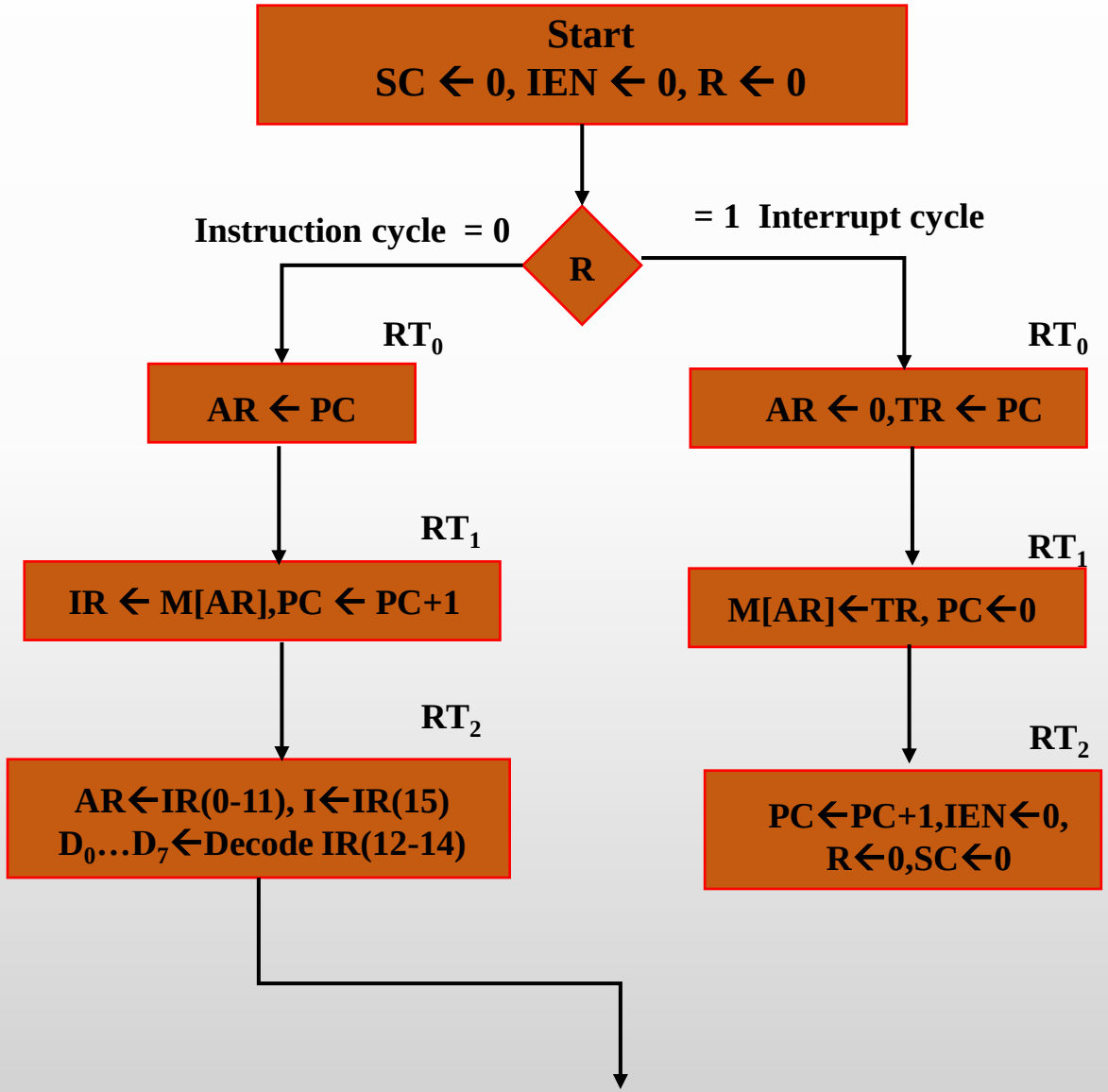


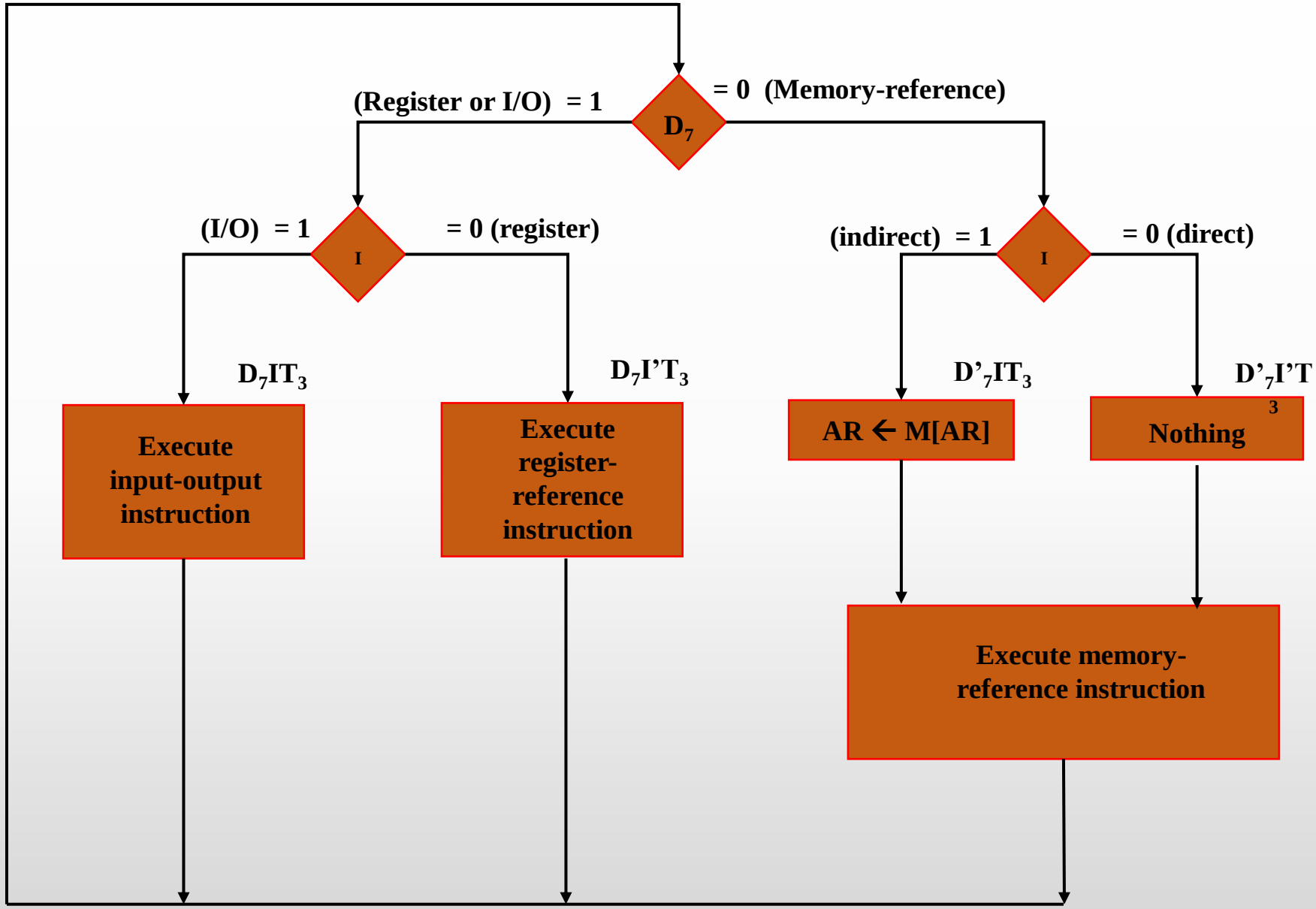
(a) After interrupt

Complete Computer Description

Complete Computer Description

Flowchart for computer operation




(Register or I/O) = 1
D7
= 0 (Memory-reference)
(I/O) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
D7IT3
D7I'T3
D'7IT3
D'7I'T3
Execute input-output instruction
Execute register-reference instruction
AR ← M[AR]
Nothing
Execute memory-reference instruction

Design of Basic Computer

Design of Basic Computer

The basic computer consists of the following hardware components:

1. A memory unit with **4096** words of **16** bits each
2. Nine registers: **AR, PC, DR, AC, IR, TR, OUTR, INPR**, and **SC**
3. Seven flip-flops**: I, S, E, R, IEN, FGI**, and **FGO**
4. Two decoders: a **3** x **8** operation decoder and a **4** x **16** timing decoder
5. A **16**-bit common bus
6. Control logic gates
7. Adder and logic circuit connected to the input of **AC**

The outputs of the control logic circuit are:

1. Signals to control the inputs of the nine registers
2. Signals to control the read and write inputs of memory
3. Signals to set, clear, or complement the flip-flops
4. Signals for $\mathbf{S_2}$, $\mathbf{S_1}$, and $\mathbf{S_0}$ to select a register for the bus
5. Signals to control the **AC** adder and logic circuit

Control of Registers and Memory

- By using the register transfer statements of control inputs for AR, and the logic circuit for control inputs associated with AR is designed as follows.

For **LD**,

$$R'T_0 : AR \leftarrow PC$$
$$R'T_2 : AR \leftarrow IR\ (0\text{-}11)$$
$$D_7'IT_3 : AR \leftarrow M\ [AR]$$

For **CLR**,

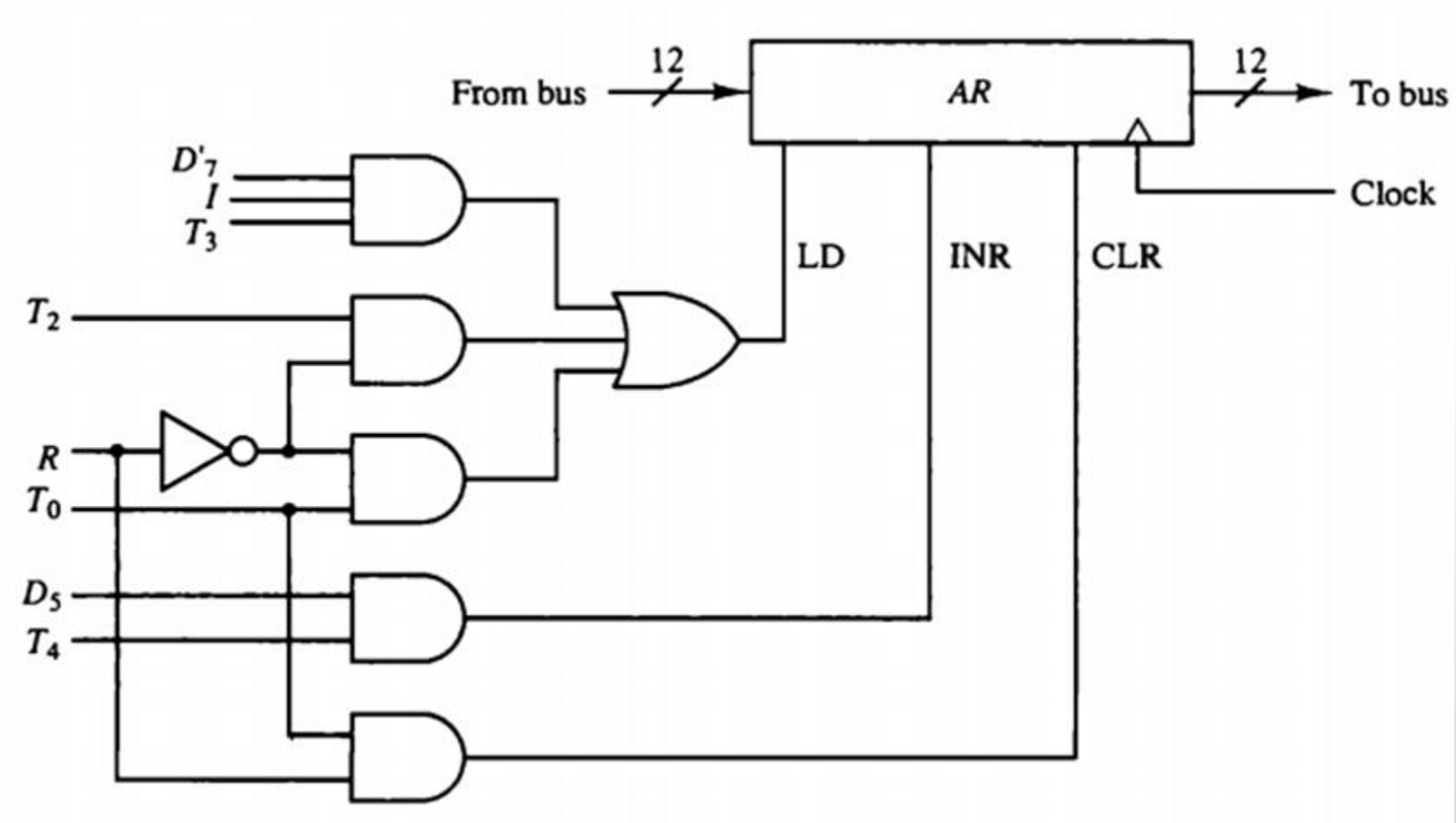
$$RT_0 : AR \leftarrow 0$$

For **INR**,

$$D_5T_4 : AR \leftarrow AR+1$$

Required control input for **LD** of **AR** $= R'T_0 + R'T_2 + D_7'IT_3$
Required control input for **CLR** of **AR** $= RT_0$
Required control input for **INR** of **AR** $= D_5T_4$

Logic Diagram of Control gates associated with AR



- For memory, there are two control input lines to manage the information transfer to and from the memory and registers.
- The required control inputs for **READ** line are described in the following as the register transfer statement.

For **READ** ,

$$R'T_1 : IR \leftarrow M[AR]$$
$$D_7'IT_3 : AR \leftarrow M[AR]$$
$$D_0T_4 : DR \leftarrow M[AR]$$
$$D_1T_4 : DR \leftarrow M[AR]$$
$$D_2T_4 : DR \leftarrow M[AR]$$
$$D_6T_4 : DR \leftarrow M[AR]$$

$$READ = R'T_1 + D_7'IT_3 + (D_0 + D_1 + D_2 + D_6)\,T_4$$

Control of Common Bus

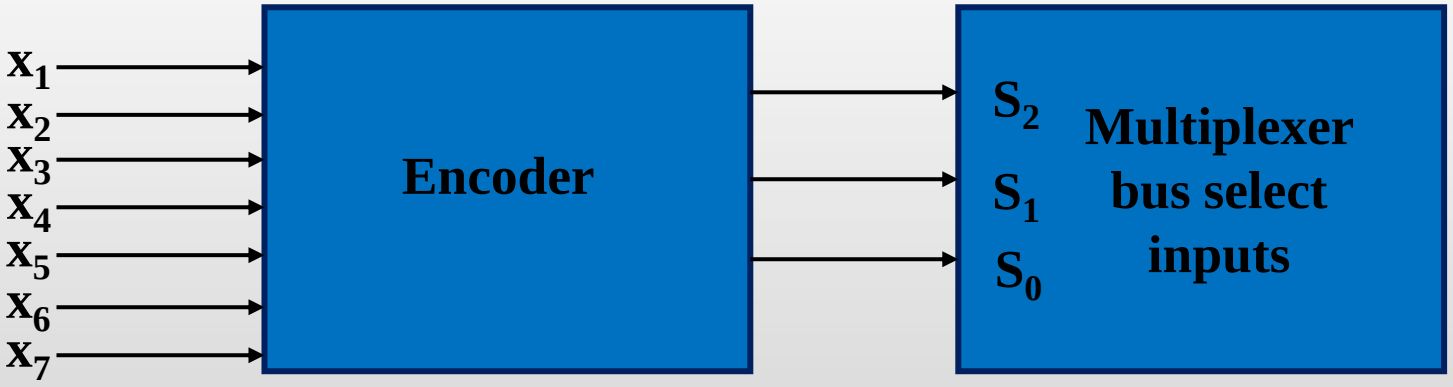
- The 16-bit common bus is controlled by the selection lines S_2, S_1, and S_0.
- The Boolean function for encoder is described as follows.

$S_0 = x_1 + x_3 + x_5 + x_7$

$S_1 = x_2 + x_3 + x_6 + x_7$

$S_2 = x_4 + x_5 + x_6 + x_7$

Encoder for Bus Selection Inputs



Process of Encoder to Control the Common Bus

Inputs							Outputs			Register selected for bus
x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

Design of Accumulator Logic

Design of Accumulator Logic

- The register transfer statements for **AC** are described in the following.
- The block symbol as well as the logic gate structure for control inputs of **AC** can be designed using these register transfer statements.

For **LD** control input,

D_0T_5	: AC ← AC ^ DR	AND with DR
D_1T_5	: AC ← AC + DR	ADD with DR
D_2T_5	:AC ← DR	Transfer from DR
pB_{11}	:AC(0-7) ← INPR	Transfer from INPR
rB_9	:AC ← AC	Complement
rB_7	:AC ← shr AC, AC(15)← E	Shift right
rB_6	:AC ← shl AC, AC(0) ← E	Shift left

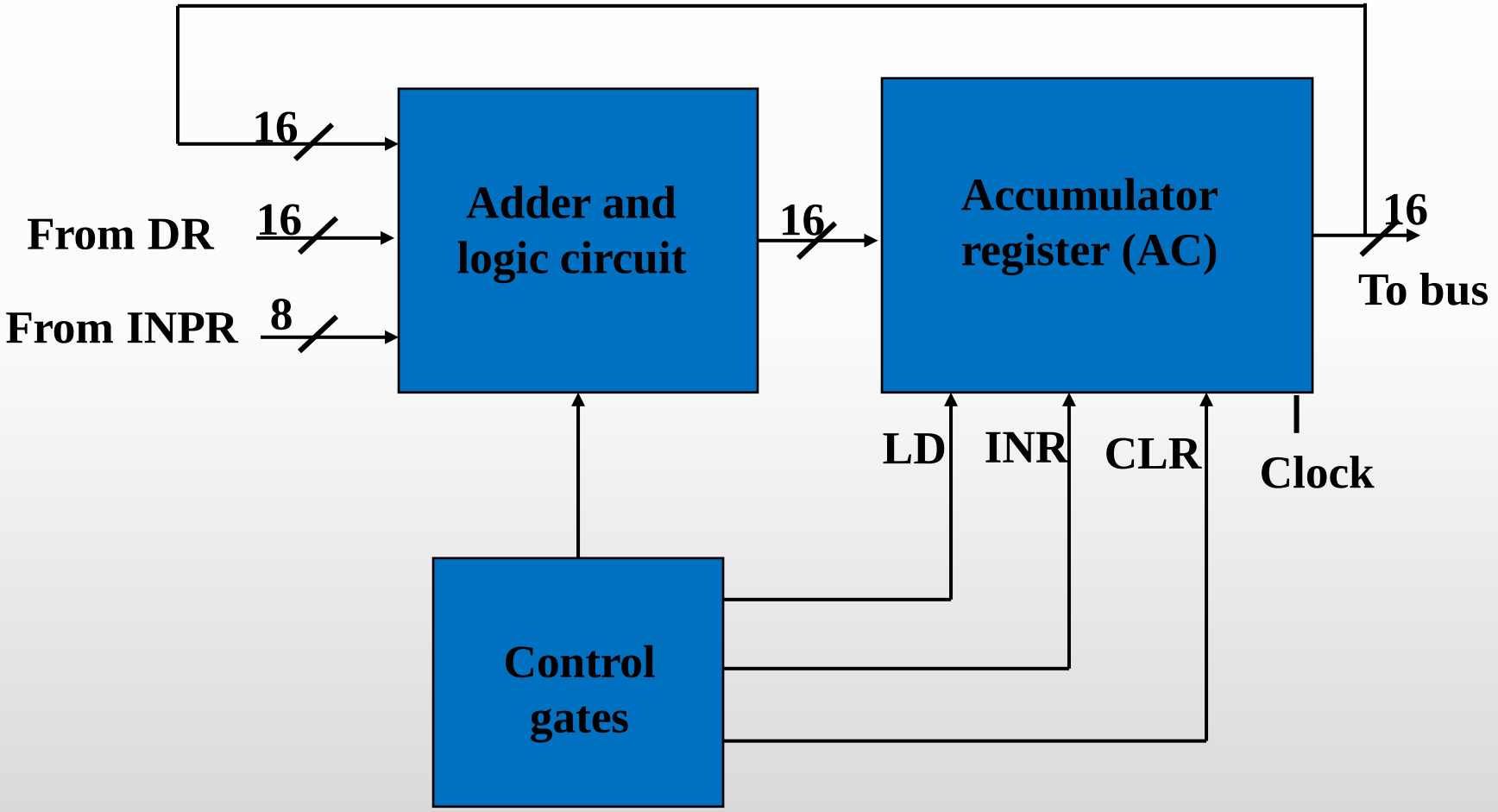
For **CLR** control input,

rB_{11}	:AC ← 0	Clear

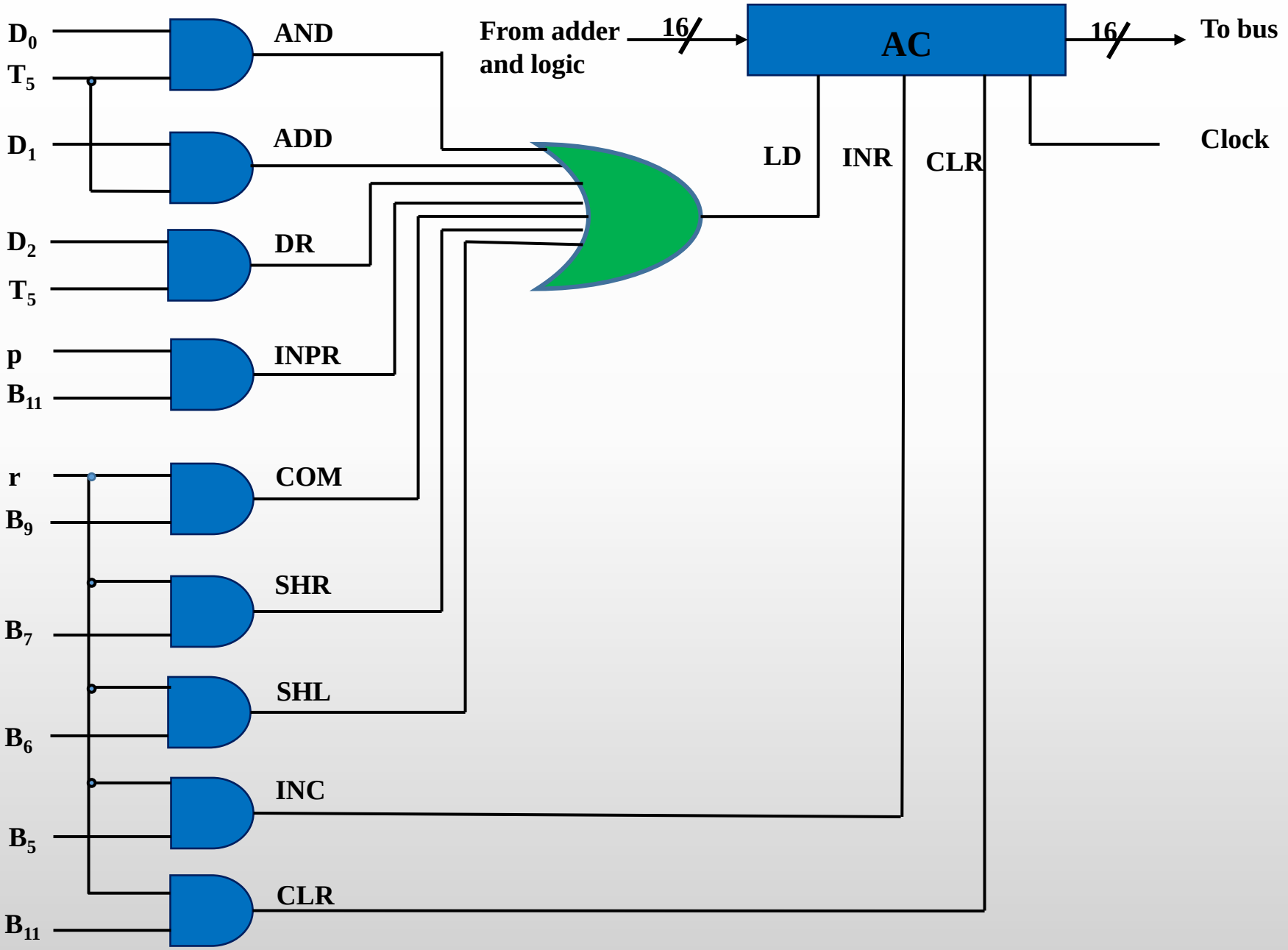
For **INR** control input,

rB_9	:AC ← AC + 1	Increment

Logic Diagram of Circuits associated with AC



Gate structure for controlling the LD, INR, and CLR of AC



Thank You

Lecture for Next Week

- **Programming the Basic Computer**
- **What is Machine Language, Assembly Language?**
- **The Assembler**
- **Program Loops**
- **Programming Arithmetic and Logic Operations**
- **Input-Output Programming**