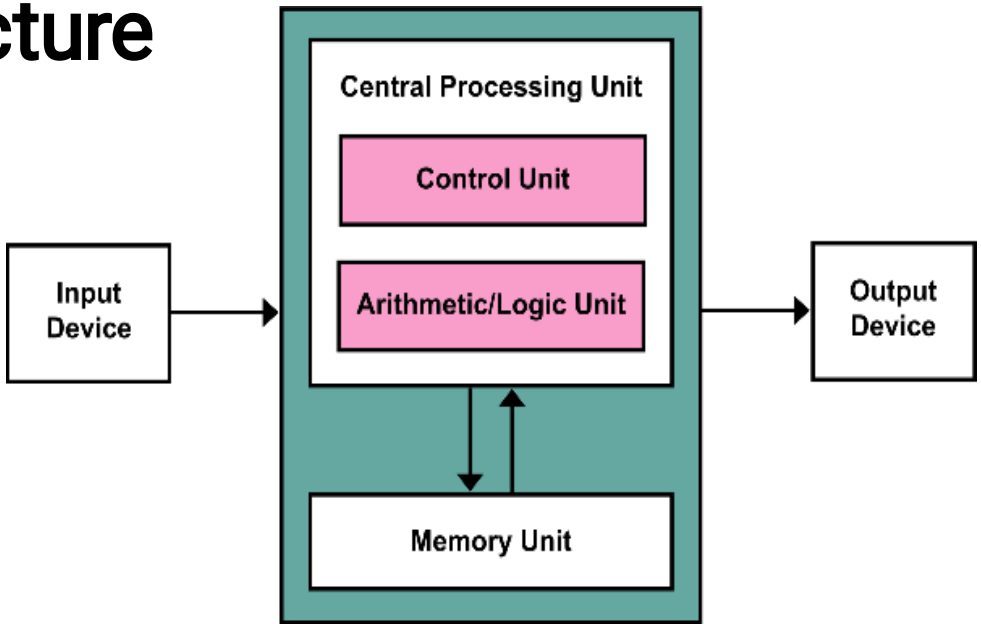
Computer Organisation and Architecture
Unit – I
Structure of Digital Computer

- **Digital Components**
  1. Gates :- AND , OR, NOT, XOR
  2. Flip- Flops :- J-K flip flop , R-S Flip Flops
  3. Decoder
  4. Encoder
  5. Multiplexer
  6. Demultiplexer

- **Von –Neumann Architecture**
  1. CPU
  2. Memory Unit
  3. Input unit
  4. Output unit



Computer Organisation and Architecture
Unit – I
Structure of Digital Computer

- Computer Architecture vs. Organization

1. Architecture

– Abstract model

– Programmer's view of Instruction sets, addressing modes, registers

– WHAT ?

2. Organization

– Realization and implementation of architecture

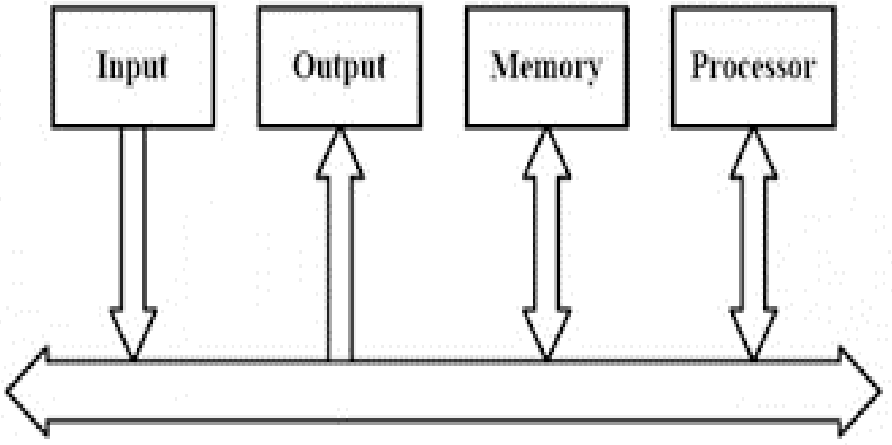
– Hardware

– HOW ?

Computer Organisation and Architecture
Unit – I
Bus Structure

Bus Architecture

- There are many ways to connect different parts inside a computer together.
- A group of lines that serves as a connecting path for several devices is called a bus.
- Three different types of lines :- Address/data/ control

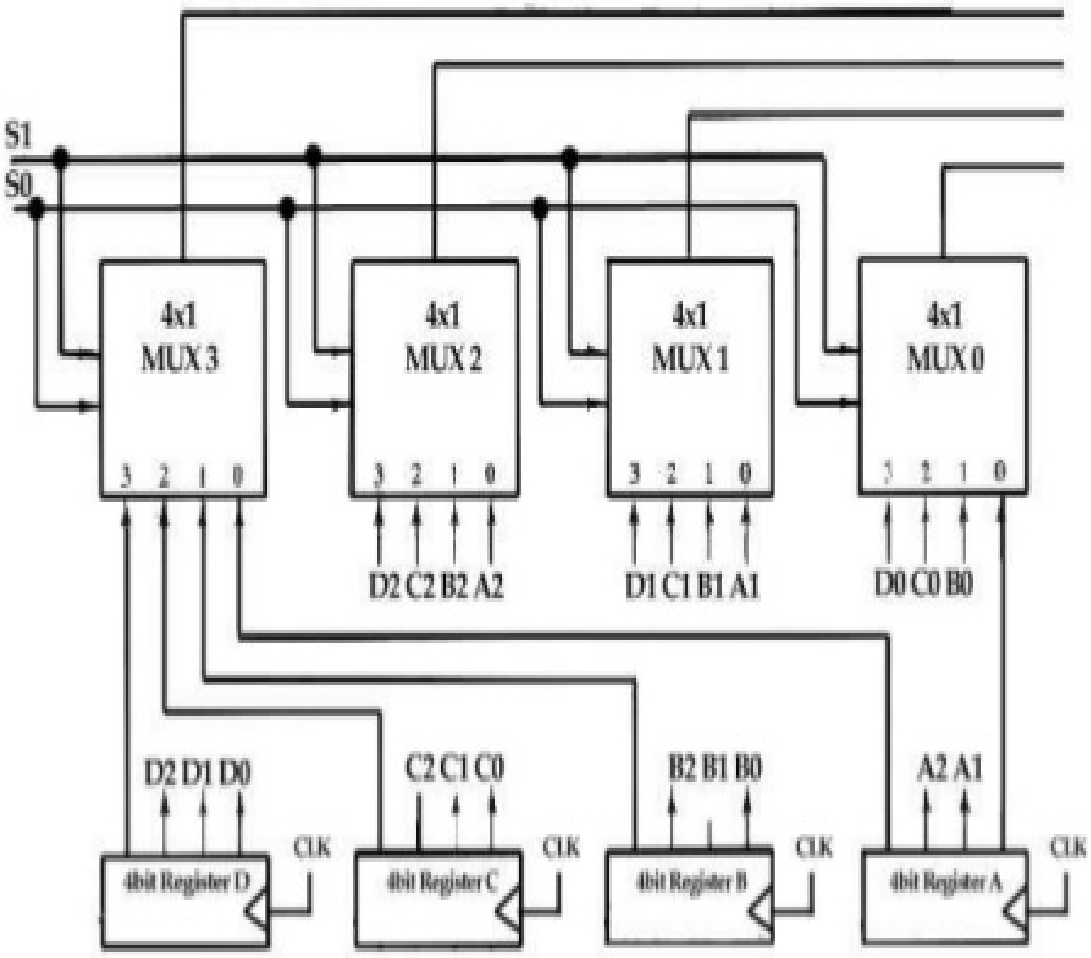


Computer Organisation and Architecture
Unit – I
Bus Structure

Bus Structure usig Multiplexers

Number of Multiplexers =
Number of bits in Register

Size of each Multiplexer =
Number of Registers



Computer Organisation and Architecture
Unit – I
Bus Structure

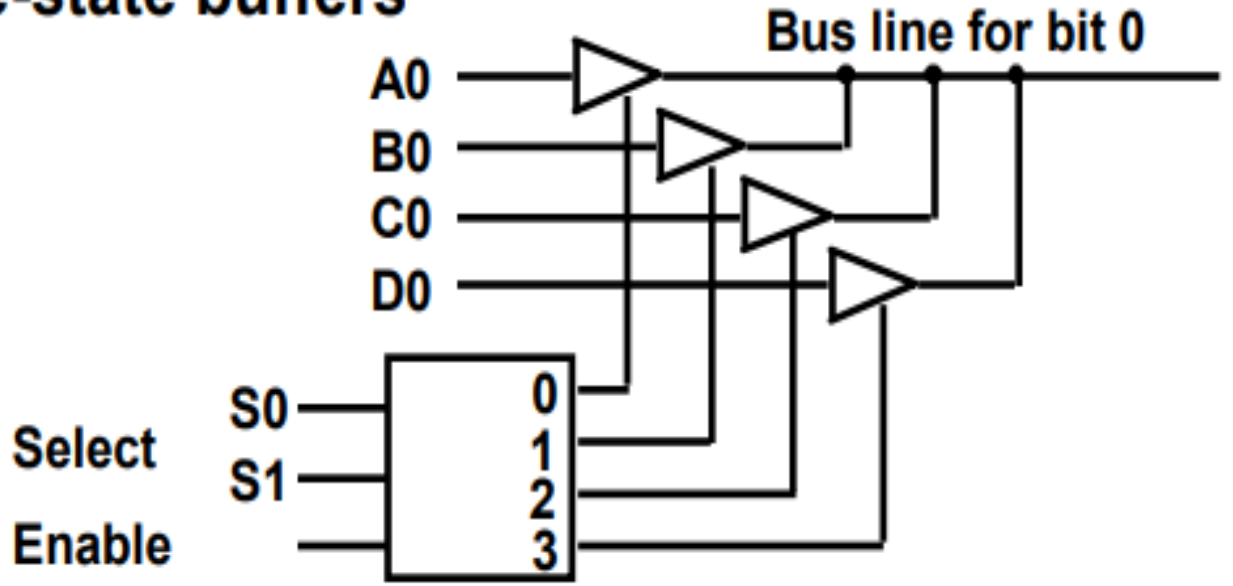
Three-State Bus Buffers

Normal input A

Control input C

Output Y=A if C=1
High-impedence if C=0

Bus line with three-state buffers

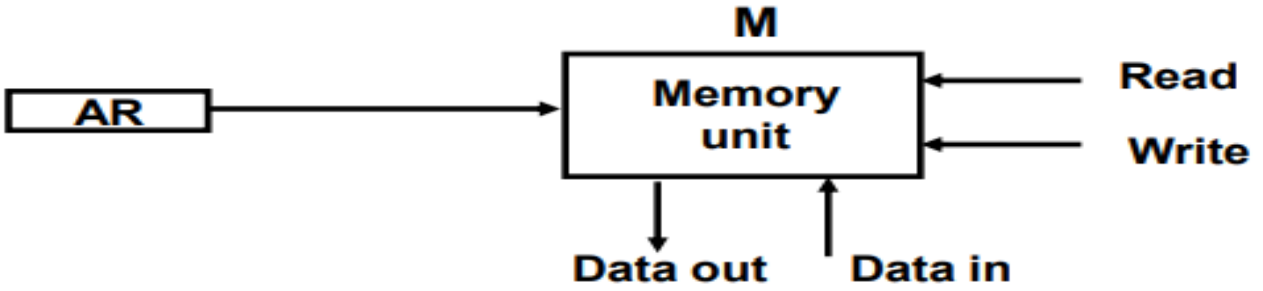


Computer Organisation and Architecture
Unit – I
Memory Transfer

Memory Transfers

- **Collectively, the memory is viewed at the register level as a device, M.**
- **Since it contains multiple locations, we must specify which address in memory we will be using**
- **This is done by indexing memory references**

- **Memory is usually accessed in computer systems by putting the desired address in a special register, the *Memory Address Register* (*MAR*, or *AR*)**
- **When memory is accessed, the contents of the MAR get sent to the memory unit's address lines**



Computer Organisation and Architecture
Unit – I
Register Transfer Language

- **Rather than specifying a digital system in words, a specific notation is used, *register transfer language***

- **For any function of the computer, the register transfer language can be used to describe the (sequence of) microoperations**

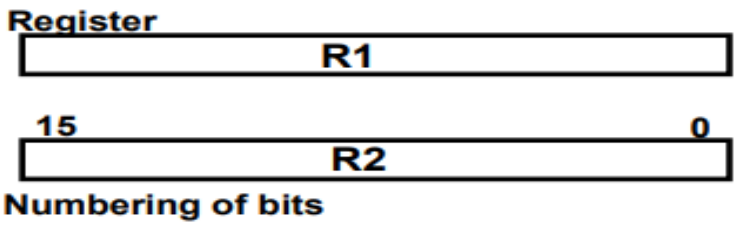
- **Register transfer language**
  - **A symbolic language**
  - **A convenient tool for describing the internal organization of digital computers**
  - **Can also be used to facilitate the design process of digital systems.**

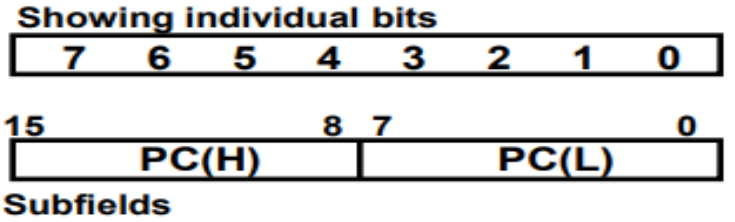
Computer Organisation and Architecture
Unit – I
Register Transfer Language

- **Registers are designated by capital letters, sometimes followed by numbers (e.g., A, R13, IR)**
- **Often the names indicate function:**
  - **MAR - memory address register**
  - **PC - program counter**
  - **IR - instruction register**

- **Registers and their contents can be viewed and represented in *various ways***
  - **A register can be viewed as a single entity:**

  - **Registers may also be represented showing the bits of data they contain**
- **Common ways of drawing the block diagram of a register**





Computer Organisation and Architecture
Unit – I
Register Transfer Language

Register Transfer

- **Copying the contents of one register to another is a register transfer**

- **A register transfer is indicated as**

  R2 ← R1

  - **In this case the contents of register R2 are copied (loaded) into register R1**
  - **A simultaneous transfer of all bits from the source R1 to the destination register R2, during one clock pulse**
  - **Note that this is a non-destructive; i.e. the contents of R1 are not altered by copying (loading) them to R2**

Computer Organisation and Architecture
Unit – I
Register Transfer Language

Control Function

- **Often actions need to only occur if a certain condition is true**
- **This is similar to an "if" statement in a programming language**
- **In digital systems, this is often done via a *control signal*, called a *control function***
  - **If the signal is 1, the action takes place**
- **This is represented as:**

P: R2 ← R1

Which means "if P = 1, then load the contents of register R1 into register R2", i.e., if (P = 1) then (R2 ← R1)

Computer Organisation and Architecture
Unit – I
Microoperations

Arithematic Microoperations

- **The basic arithmetic microoperations are**
  - **Addition**
  - **Subtraction**
  - **Increment**
  - **Decrement**

- **The additional arithmetic microoperations are**
  - **Add with carry**
  - **Subtract with borrow**
  - **Transfer/Load**
  - **etc. …**

Summary of Typical Arithmetic Micro-Operations

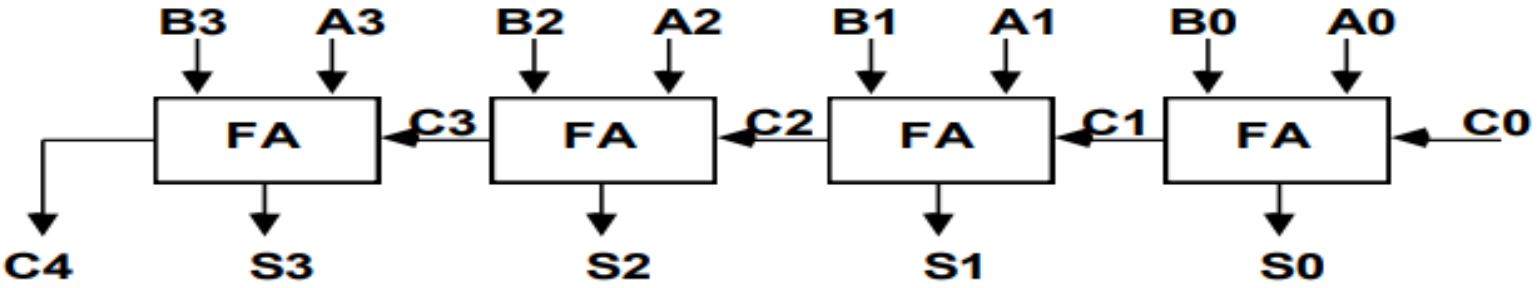
R3 ← R1 + R2	Contents of R1 plus R2 transferred to R3
R3 ← R1 - R2	Contents of R1 minus R2 transferred to R3
R2 ← R2’	Complement the contents of R2
R2 ← R2’+ 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2’+ 1	subtraction
R1 ← R1 + 1	Increment
R1 ← R1 - 1	Decrement

Computer Organisation and Architecture

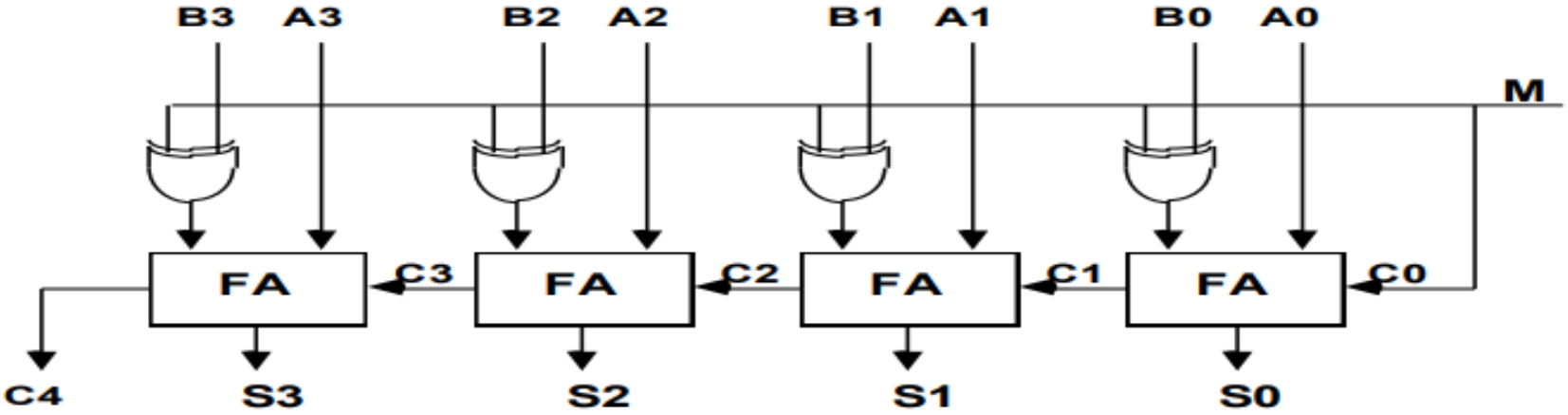
Unit – I

Arithematic Microoperations

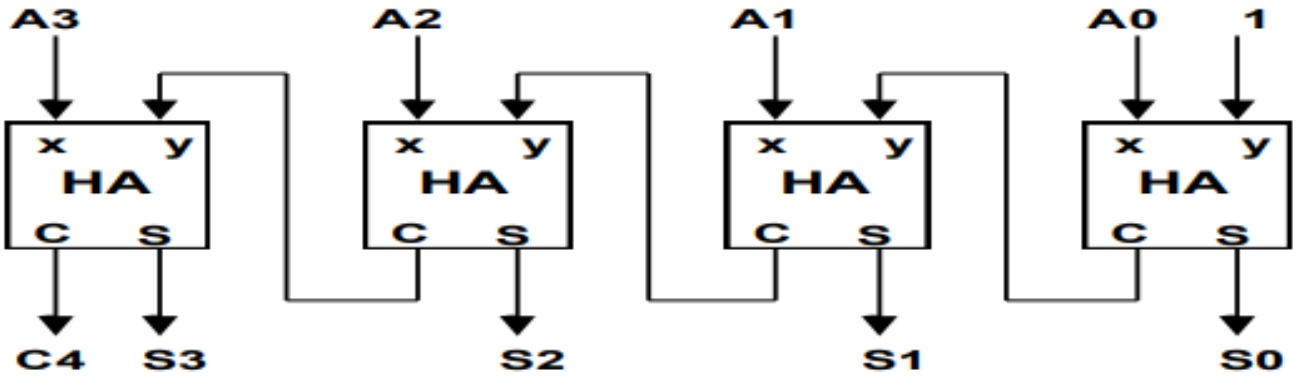
Binary Adder



Binary Adder-Subtractor



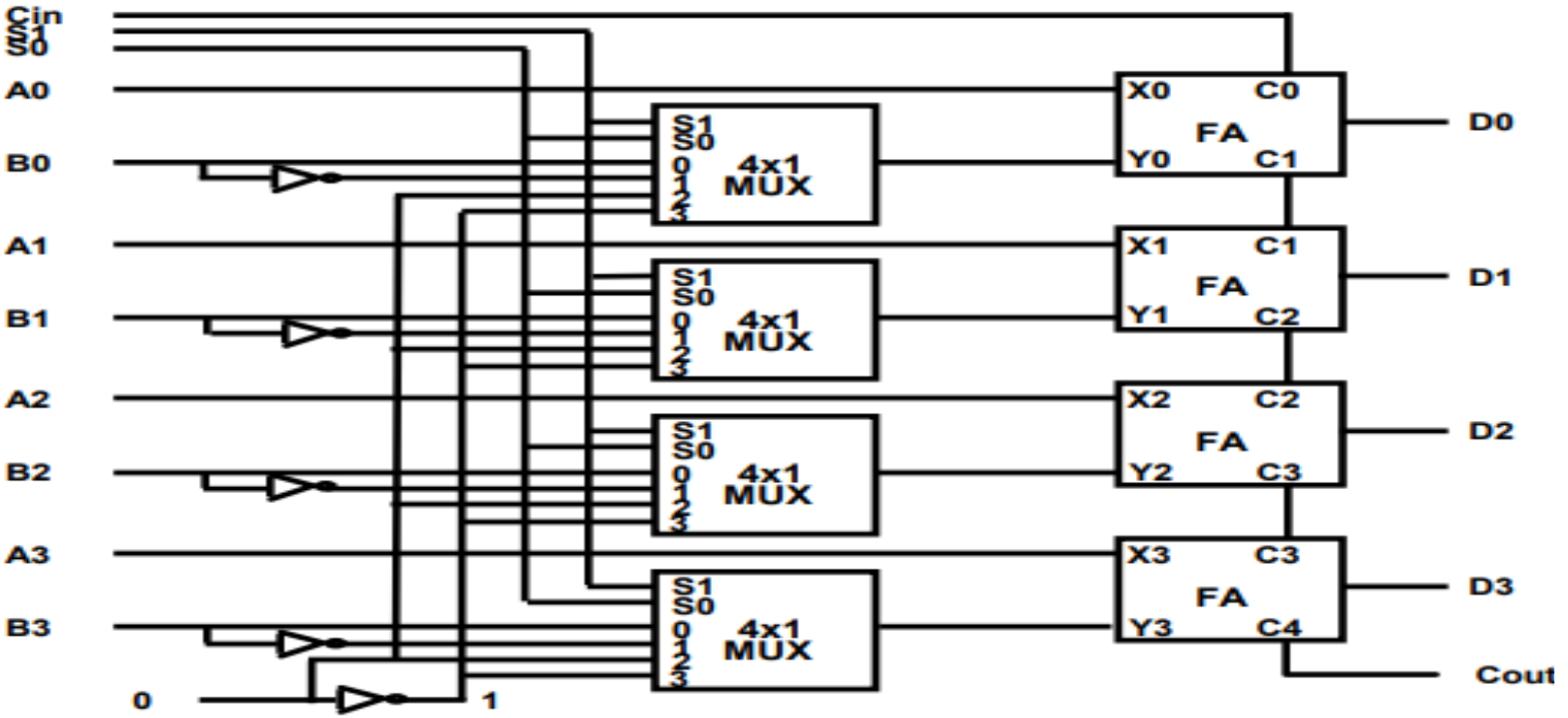
Binary Incrementer



Computer Organisation and Architecture

Unit – I

Arithematic Circuit



S1	S0	Cin	Y	Output	Microoperation
0	0	0	B	D = A + B	Add
0	0	1	B	D = A + B + 1	Add with carry
0	1	0	B'	D = A + B'	Subtract with borrow
0	1	1	B'	D = A + 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

Computer Organisation and Architecture
Unit – I
Logic Microoperations

- **Specify binary operations on the strings of bits in registers**
  - **Logic microoperations are bit-wise operations, i.e., they work on the individual bits of data**
  - **useful for bit manipulations on binary data**
  - **useful for making logical decisions based on the bit value**
- **There are, in principle, 16 different logic functions that can be defined over two binary input variables**

A	B	F_0	F_1	F_2	…	F_{13}	F_{14}	F_{15}
0	0	0	0	0	…	1	1	1
0	1	0	0	0	…	1	1	1
1	0	0	0	1	…	0	1	1
1	1	0	1	0	…	1	0	1

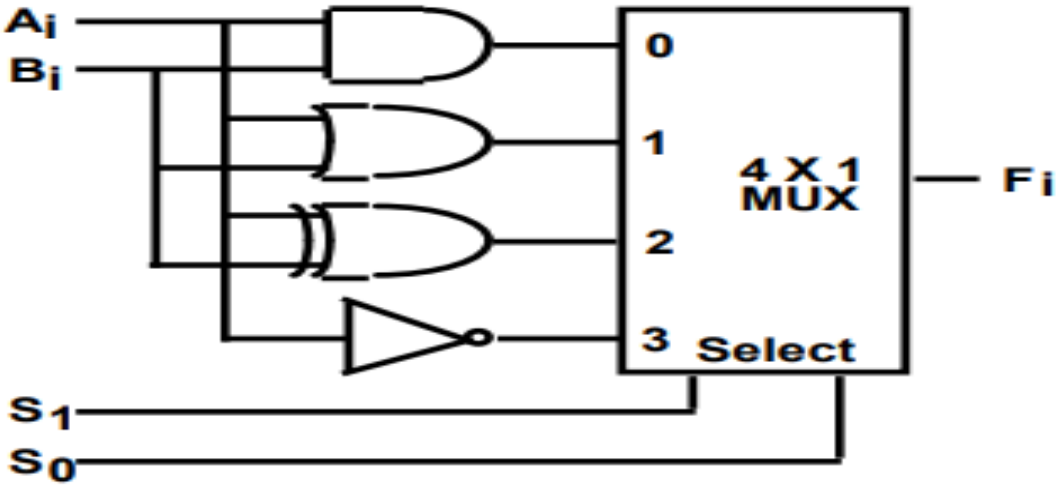
- **However, most systems only implement four of these**
  - **AND (∧), OR (∨), XOR (⊕), Complement/NOT**
- **The others can be created from combination of these**

Computer Organisation and Architecture

Unit – I

Logic Microoperations

Hardware implementation of Logic Microoperations



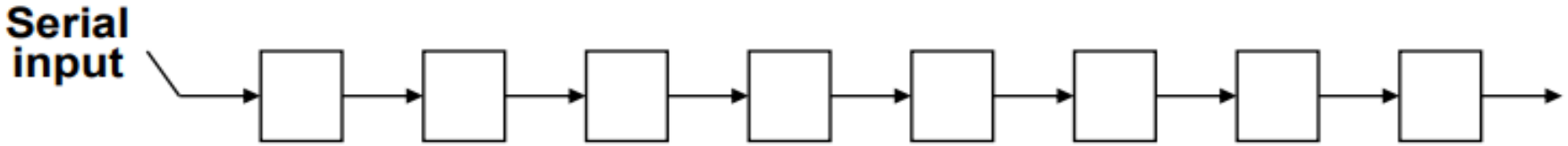
Function table

S_1	S_0	Output	μ-operation
0	0	$F = A \wedge B$	AND
0	1	$F = A \vee B$	OR
1	0	$F = A \oplus B$	XOR
1	1	$F = A'$	Complement

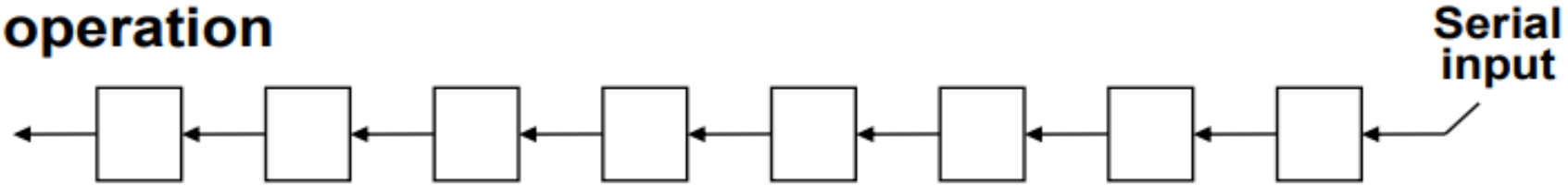
Computer Organisation and Architecture
Unit – I
Shift Microoperations

- **There are three types of shifts**
  - ***Logical shift***
  - ***Circular shift***
  - ***Arithmetic shift***
- **What differentiates them is the information that goes into the serial input**

- **A right shift operation**



- **A left shift operation**

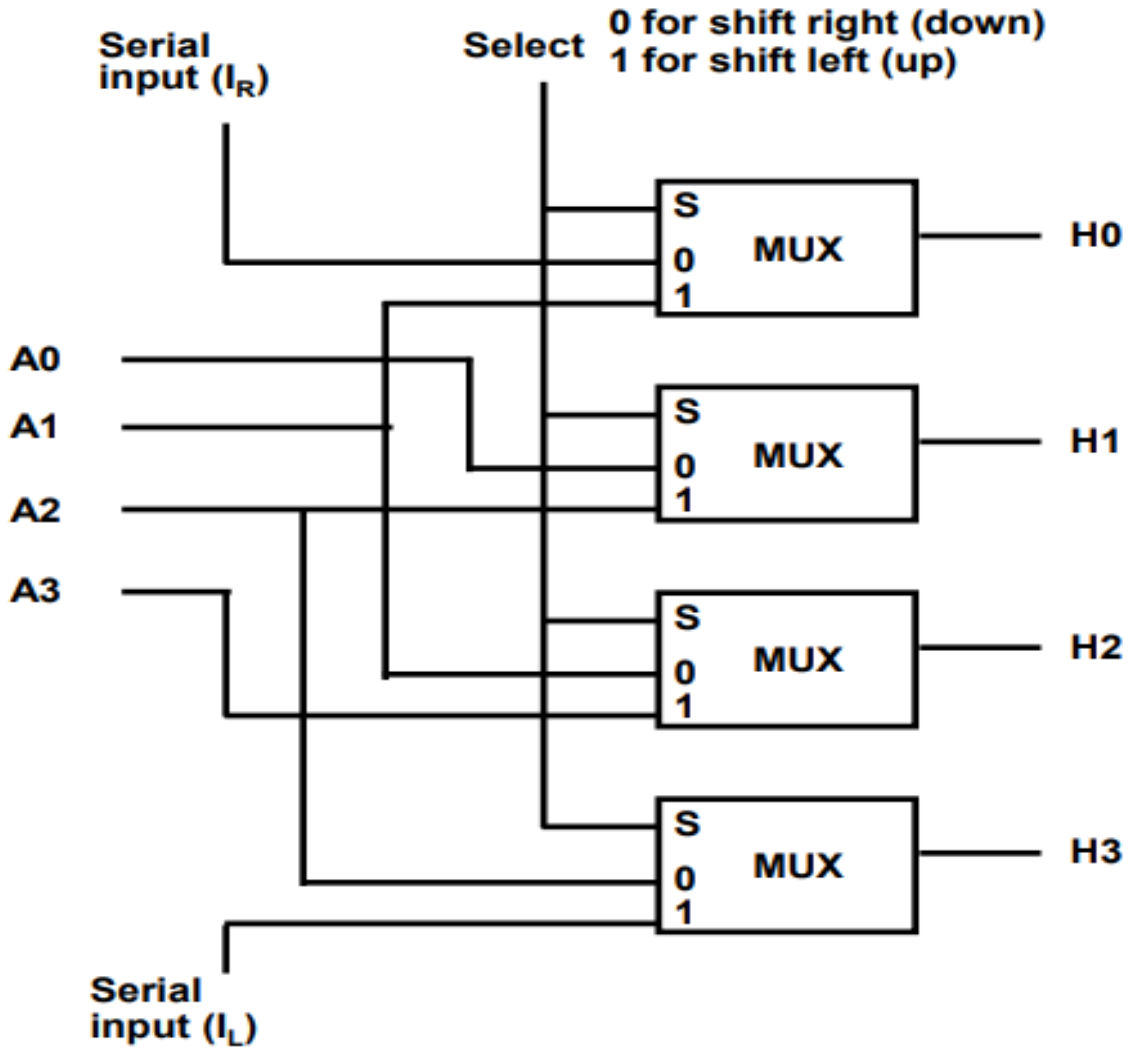


Computer Organisation and Architecture

Unit – I

Shift Microoperations

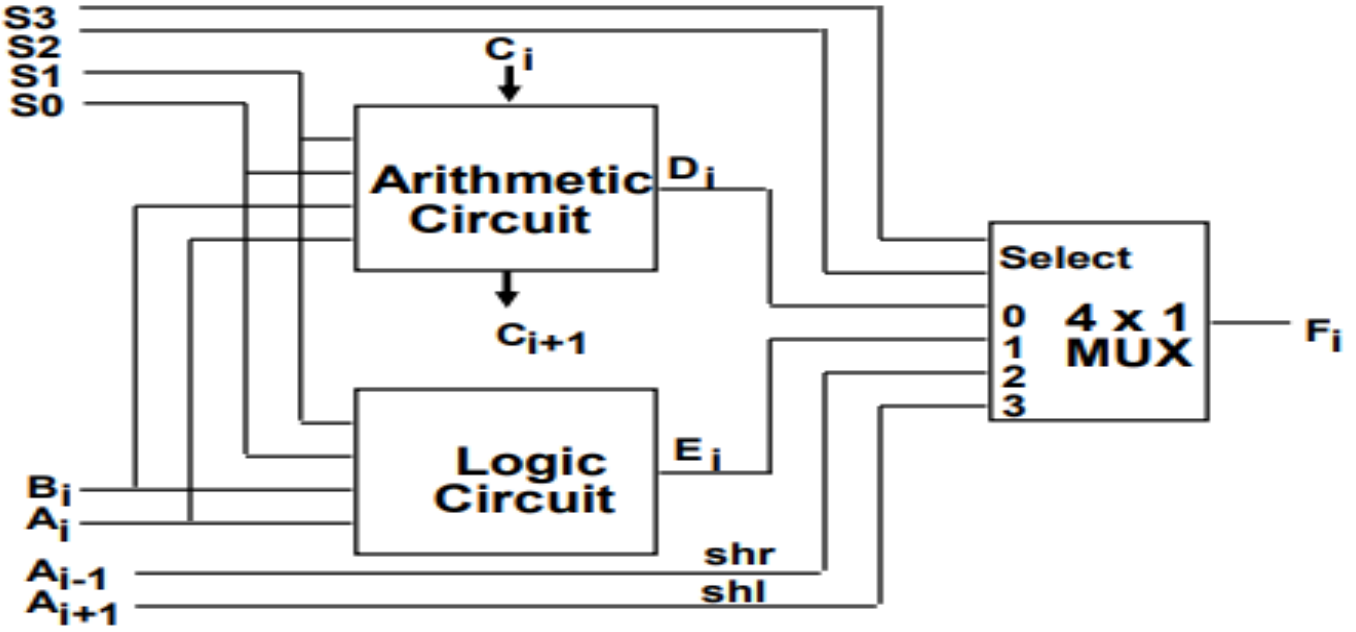
Hardware implementation of Shift Microoperations



Computer Organisation and Architecture

Unit – I

ALU Unit



S3	S2	S1	S0	Cin	Operation	Function
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Increment A
0	0	0	1	0	F = A + B	Addition
0	0	0	1	1	F = A + B + 1	Add with carry
0	0	1	0	0	F = A + B'	Subtract with borrow
0	0	1	0	1	F = A + B'+ 1	Subtraction
0	0	1	1	0	F = A - 1	Decrement A
0	0	1	1	1	F = A	TransferA
0	1	0	0	X	F = A ∧ B	AND
0	1	0	1	X	F = A ∨ B	OR
0	1	1	0	X	F = A ⊕ B	XOR
0	1	1	1	X	F = A'	Complement A
1	0	X	X	X	F = shr A	Shift right A into F
1	1	X	X	X	F = shl A	Shift left A into F

Computer Organisation and Architecture

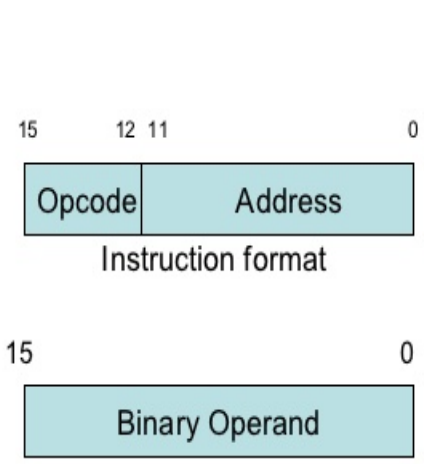
Unit – II

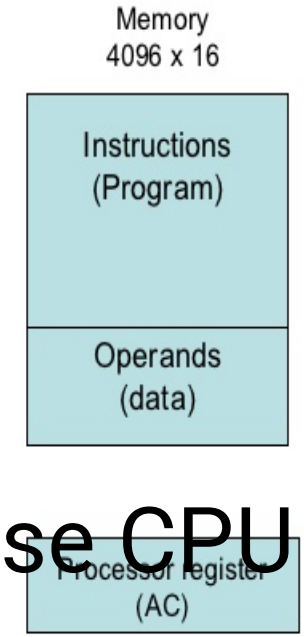
Instruction Codes

- Organisation of computer is defined by:
  - Internal Register
  - Set of Instructions
  - Timing and Control
- Program is a set of instructions that specify:
  - Operations
  - Operands
  - Sequence of processing
- Computer instruction is a binary code that specifies sequence of microoperations.

Computer Organisation and Architecture
Unit – II
Instruction Codes

- Instruction codes together with data are stored in memory.
- An instruction code is divided into two parts:-
  - Address
  - Opcode

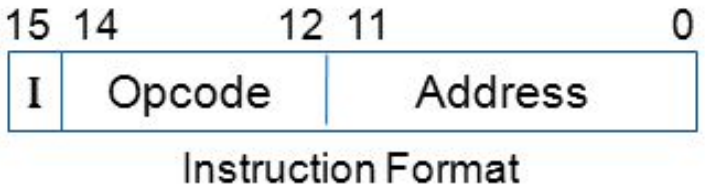




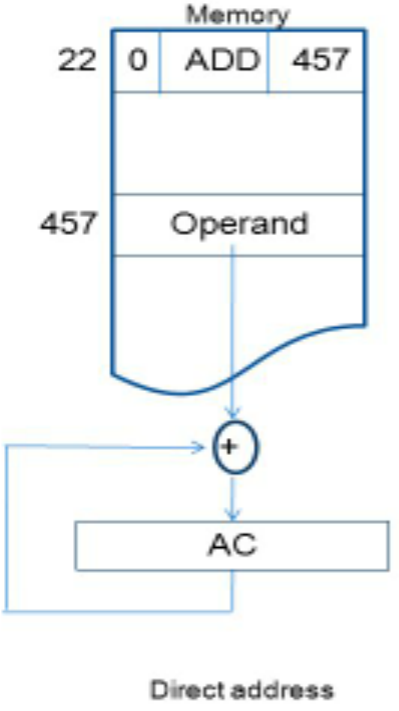
- Accumulator (AC) is a general purpose CPU register.

Computer Organisation and Architecture
Unit – II
Instruction Codes

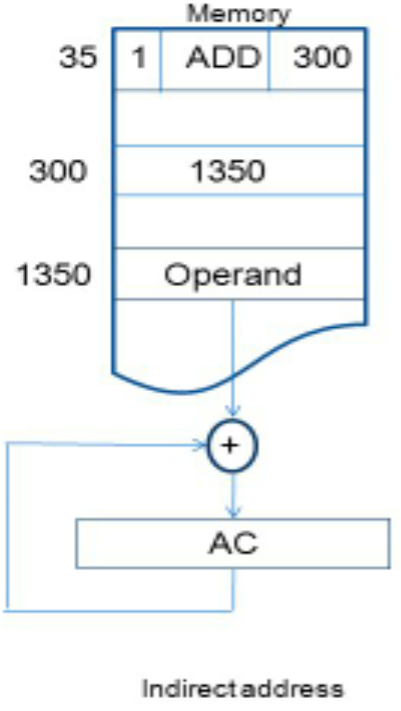
- Immediate operand uses the address bits of the instruction to specify the actual operand.
- Direct & Indirect Address:-



Instruction Format



Direct address



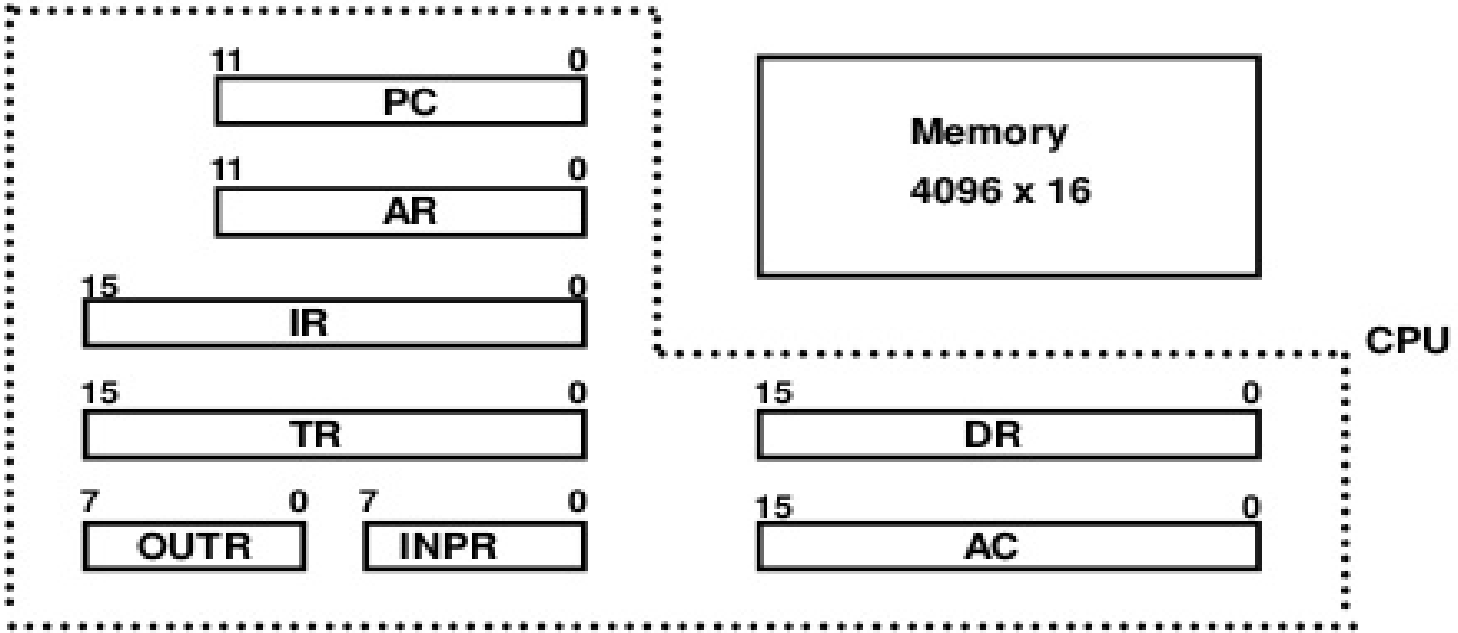
Indirect address

Computer Organisation and Architecture

Unit – II

Computer Registers

Registers in the Basic Computer



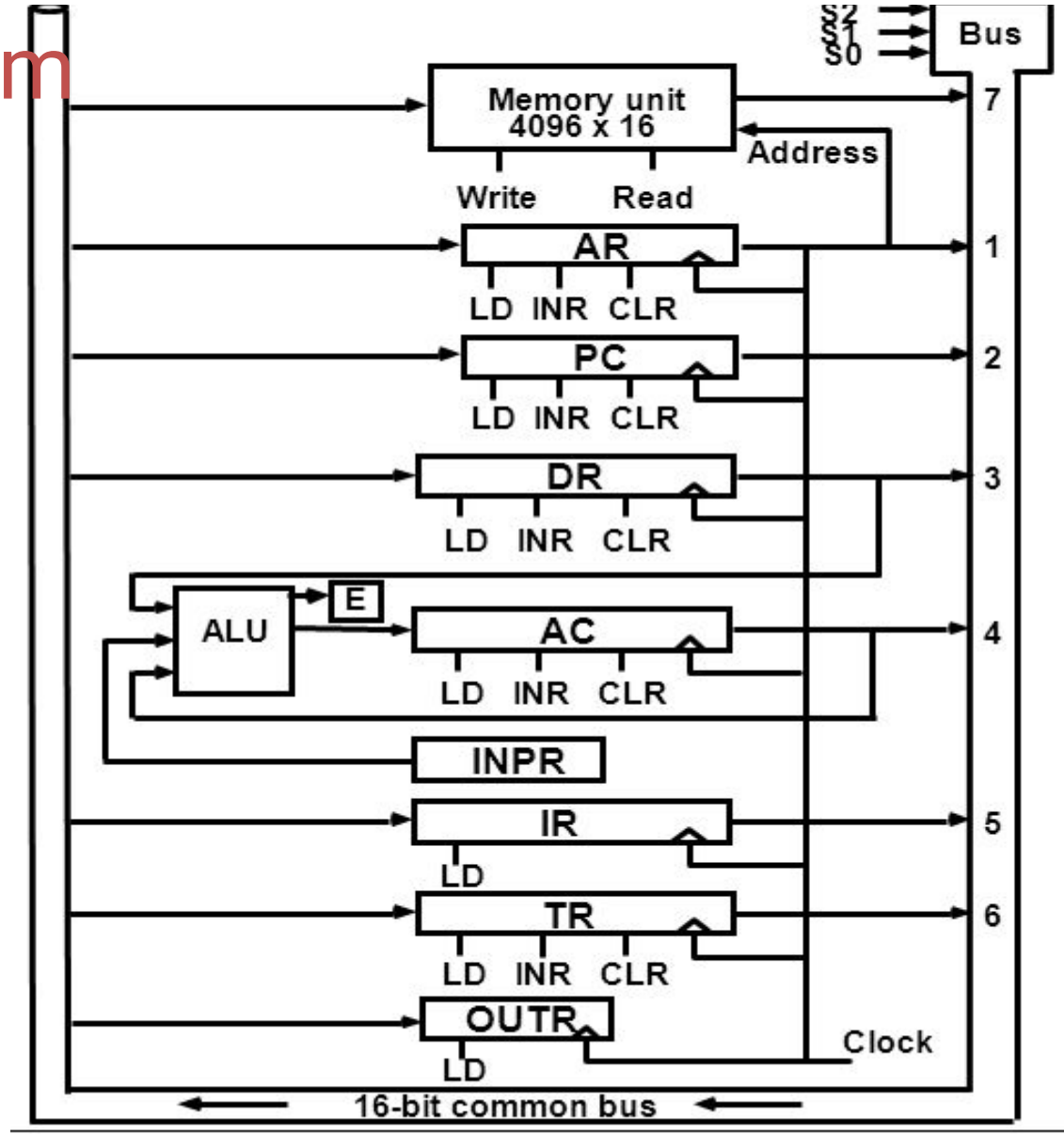
List of BC Registers

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

Computer Organisation and Architecture
Unit – II
Computer Registers

- Common Bus system

- There are 8 registers, a memory unit, and a control unit.
- Paths must be provided to transfer information from one register to another and between memory and registers.



Computer Organisation and Architecture
Unit – II
Computer Registers

- **Three control lines, S_2, S_1, and S_0 control which register the bus selects as its input**

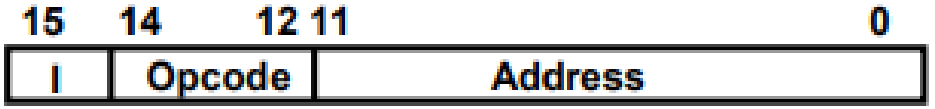
S_2	S_1	S_0	Register
0	0	0	x
0	0	1	AR
0	1	0	PC
0	1	1	DR
1	0	0	AC
1	0	1	IR
1	1	0	TR
1	1	1	Memory

- **Either one of the registers will have its load signal activated, or the memory will have its read signal activated**
  - **Will determine where the data from the bus gets loaded**
- **The 12-bit registers, AR and PC, have 0's loaded onto the bus in the high order 4 bit positions**
- **When the 8-bit register OUTR is loaded from the bus, the data comes from the low order 8 bits on the bus**

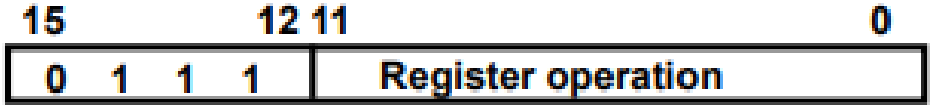
Computer Organisation and Architecture
Unit – II
Computer Instructions

- **Basic Computer Instruction Format**

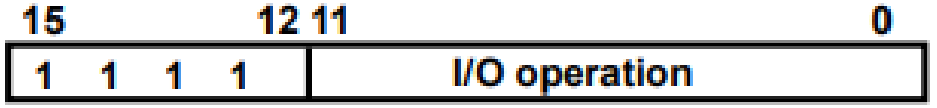
Memory-Reference Instructions (OP-code = 000 ~ 110)



Register-Reference Instructions (OP-code = 111, I = 0)



Input-Output Instructions (OP-code =111, I = 1)



Computer Organisation and Architecture
Unit – II
Computer Instructions

➢ Basic Computer Instructions

Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into 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 instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
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

Computer Organisation and Architecture
Unit – II
Timing & Control

➢ Control Unit

- **Control unit (CU) of a processor translates from machine instructions to the control signals for the microoperations that implement them**

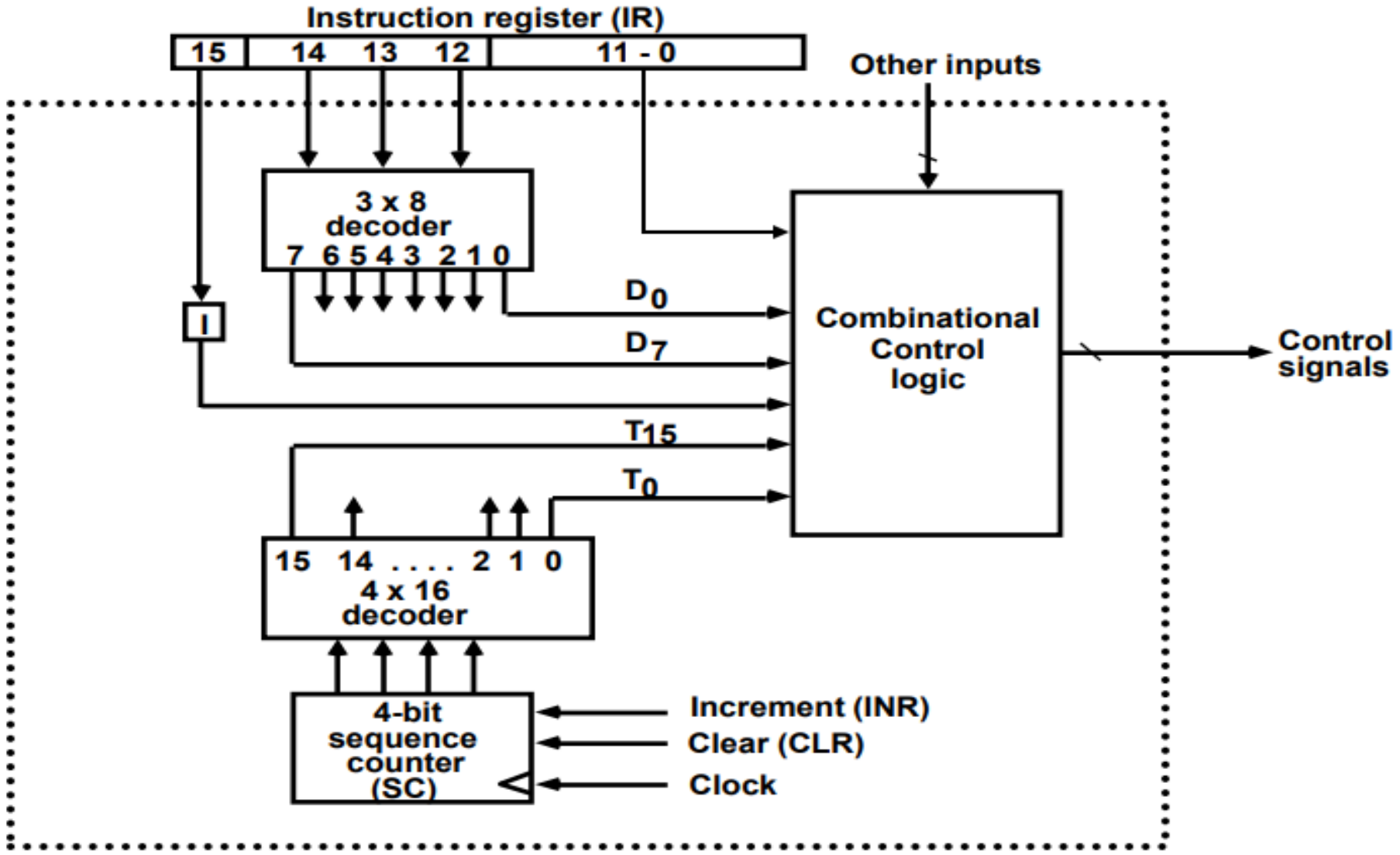
- **Control units are implemented in one of two ways**
- ***Hardwired* Control**
  - **CU is made up of sequential and combinational circuits to generate the control signals**
- ***Microprogrammed* Control**
  - **A control memory on the processor contains microprograms that activate the necessary control signals**

- **We will consider a hardwired implementation of the control unit for the Basic Computer**

Computer Organisation and Architecture

Unit – II

Timing & Control

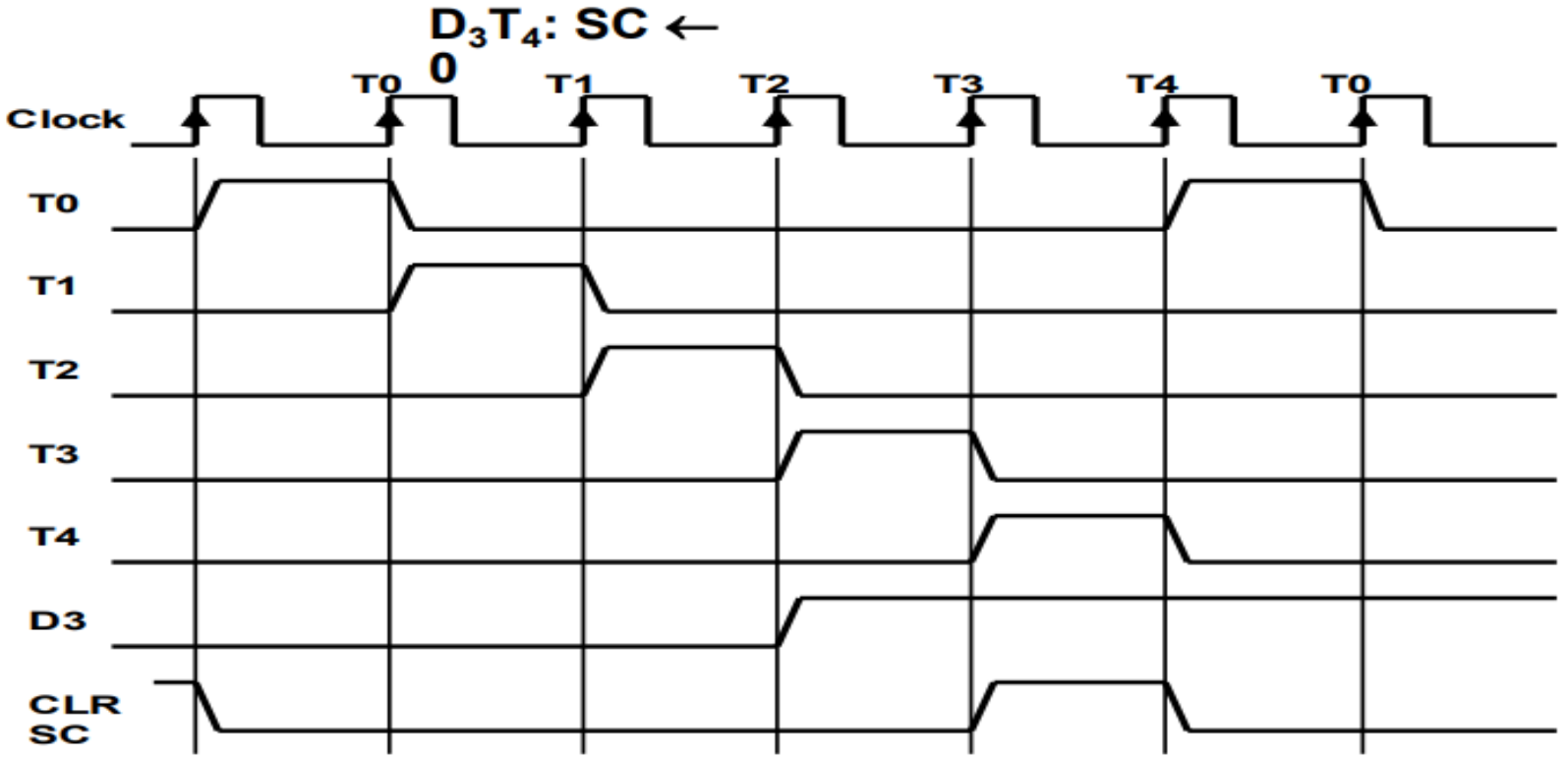


Computer Organisation and Architecture
Unit – II
Timing & Control

- Timing signals

- Generated by 4-bit sequence counter and 4×16 decoder
- The SC can be incremented or cleared.

- Example: $T_0, T_1, T_2, T_3, T_4, T_0, T_1, \ldots$
  Assume: At time T_4, SC is cleared to 0 if decoder output D3 is active.



Computer Organisation and Architecture
Unit – II
Instruction Cycle

- **In Basic Computer, a machine instruction is executed in the following cycle:**
  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**

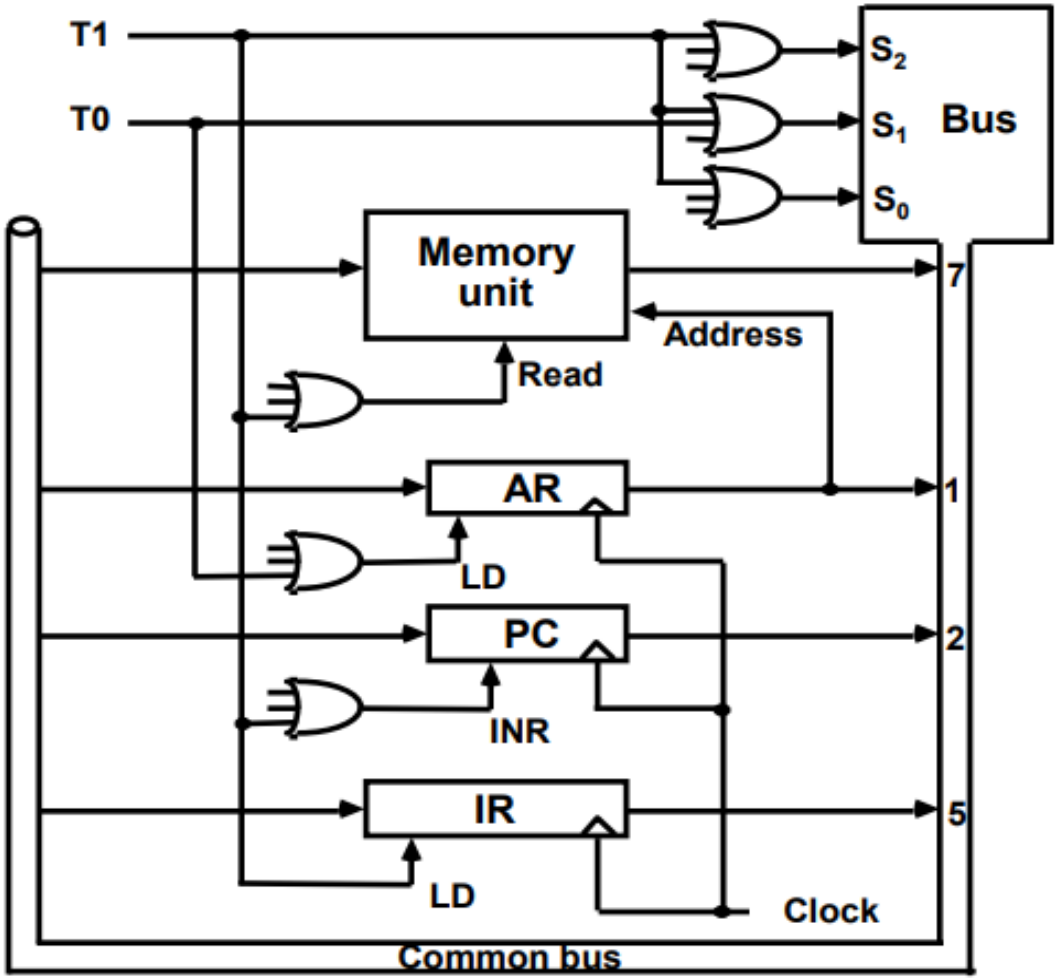
- **After an instruction is executed, the cycle starts again at step 1, for the next instruction**

- ***Note*: Every different processor has its own (different) instruction cycle**

Computer Organisation and Architecture
Unit – II
Instruction Cycle

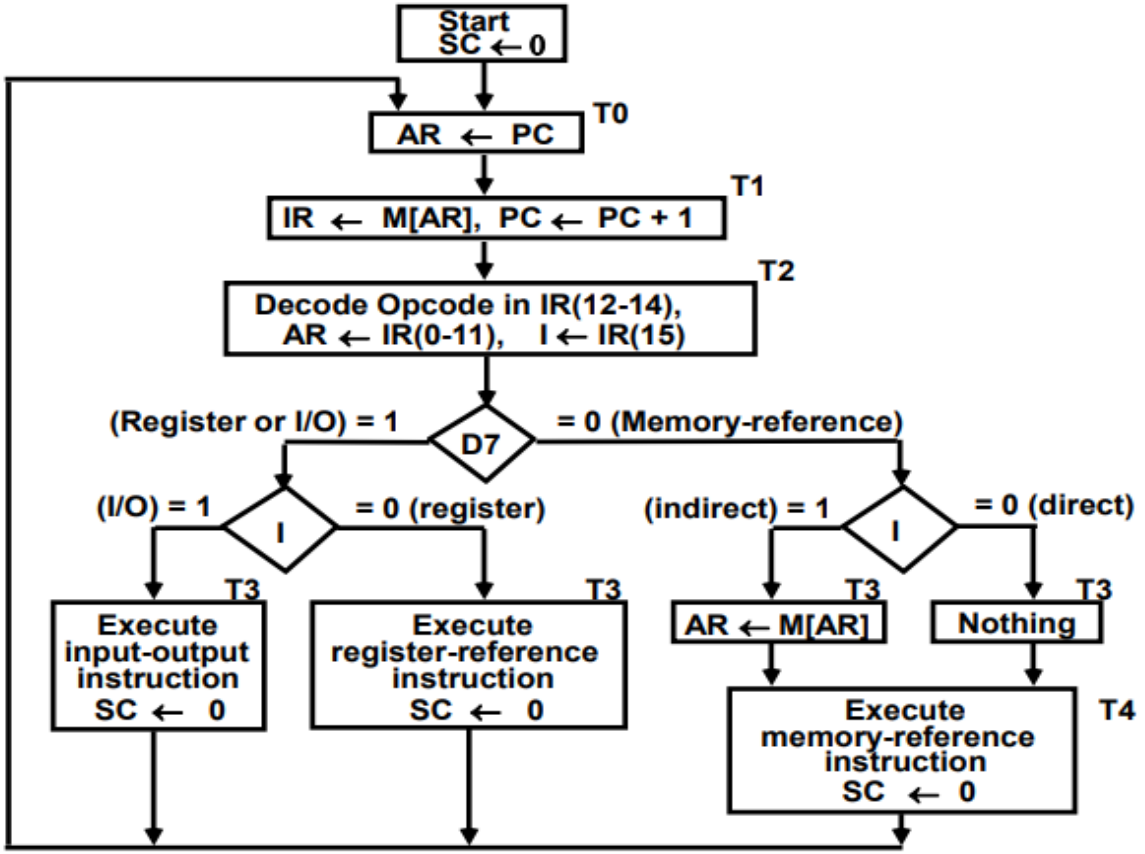
- **Fetch and Decode**

T0: AR ← PC ($S_0S_1S_2$=010, T0=1)
T1: IR ← M [AR], PC ← PC + 1 (S0S1S2=111, T1=1)
T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)



Computer Organisation and Architecture
Unit – II
Instruction Cycle

- Determine Type of Instruction



D'7IT3: AR ← M[AR]
D'7I'T3: Nothing
D7I'T3: Execute a register-reference instr.
D7IT3: Execute an input-output instr.

Computer Organisation and Architecture
Unit – II
Types of Instruction

- Register Reference Instructions:-
  - $D_7 = 1,\ I = 0$
  - Register Ref. Instr. is specified in $b_0 \sim b_{11}$ of IR
  - Execution starts with timing signal T_3

$r = D_7\, I'T_3$ => Register Reference Instruction
$B_i = IR(i)$, i=0,1,2,...,11

	r:	SC ← 0
CLA	rB_{11}:	AC ← 0
CLE	rB_{10}:	E ← 0
CMA	rB_9:	AC ← AC'
CME	rB_8:	E ← E'
CIR	rB_7:	AC ← shr AC, AC(15) ← E, E ← AC(0)
CIL	rB_6:	AC ← shl AC, AC(0) ← E, E ← AC(15)
INC	rB_5:	AC ← AC + 1
SPA	rB_4:	if (AC(15) = 0) then (PC ← PC+1)
SNA	rB_3:	if (AC(15) = 1) then (PC ← PC+1)
SZA	rB_2:	if (AC = 0) then (PC ← PC+1)
SZE	rB_1:	if (E = 0) then (PC ← PC+1)
HLT	rB_0:	S ← 0 (S is a start-stop flip-flop)

Computer Organisation and Architecture
Unit – II
Types of 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 $PC \leftarrow PC+1$

- The effective address of the instruction is in AR and was placed there during timing signal T_2 when I = 0, or during timing signal T_3 when I = 1
- Memory cycle is assumed to be short enough to complete in a CPU cycle
- The execution of MR instruction starts with T_4

AND to AC

D_0T_4: $DR \leftarrow M[AR]$ — Read operand

D_0T_5: $AC \leftarrow AC \wedge DR, SC \leftarrow 0$ — AND with AC

ADD to AC

D_1T_4: $DR \leftarrow M[AR]$ — Read operand

D_1T_5: $AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$ — Add to AC and store carry in E

Computer Organisation and Architecture
Unit – II
Types of Instruction

- Memory Reference Instructions:-

LDA: Load to AC

D_2T_4: DR ← M[AR]

D_2T_5: AC ← DR, SC ← 0

STA: Store AC

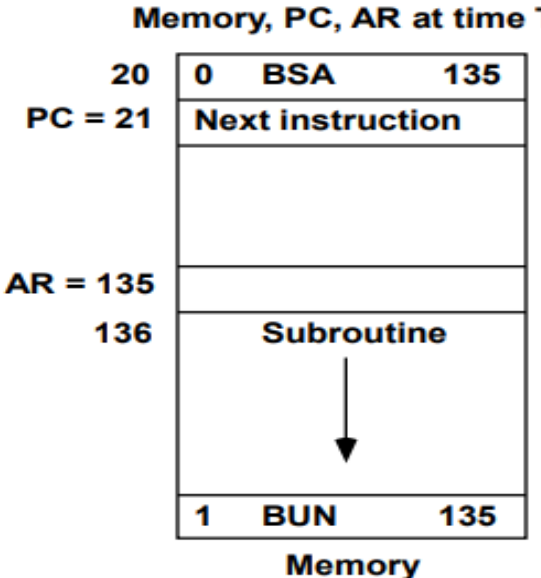
D_3T_4: M[AR] ← AC, SC ← 0

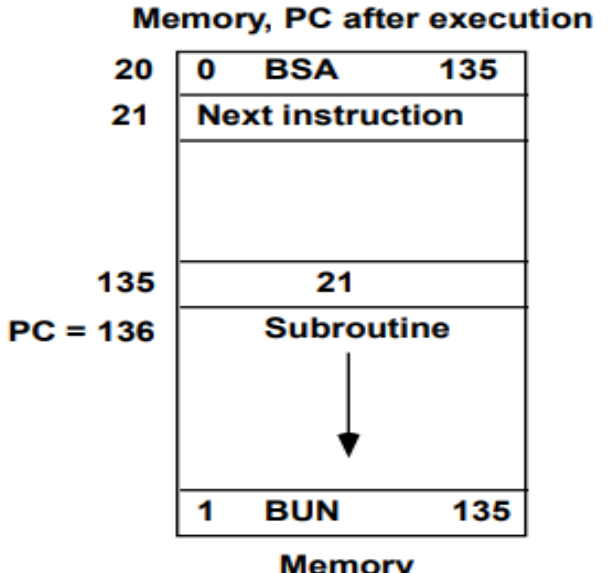
BUN: Branch Unconditionally

D_4T_4: PC ← AR, SC ← 0

BSA: Branch and Save Return Address

M[AR] ← PC, PC ← AR + 1





Computer Organisation and Architecture

Unit – II

Types of Instruction

- Memory Reference Instructions:-

BSA:

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

D_5T_5: $PC \leftarrow AR,\ SC \leftarrow 0$

ISZ: Increment and Skip-if-Zero

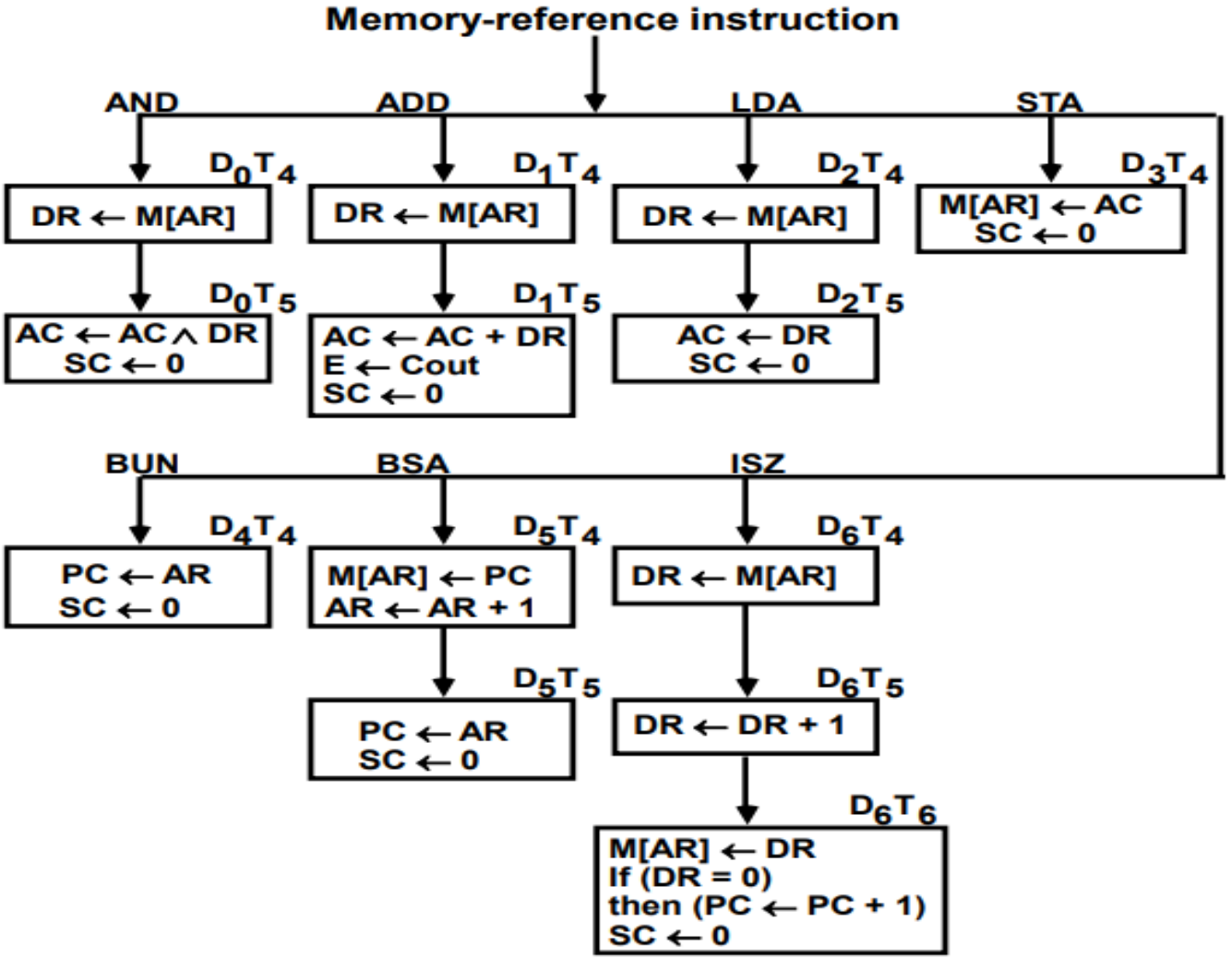
D_6T_4: $DR \leftarrow M[AR]$

D_6T_5: $DR \leftarrow DR + 1$

D_6T_4: $M[AR] \leftarrow DR,\ \text{if } (DR = 0) \text{ then } (PC \leftarrow PC + 1),\ SC \leftarrow 0$

Computer Organisation and Architecture
Unit – II
Types of Instruction

- Memory Reference Instructions:-

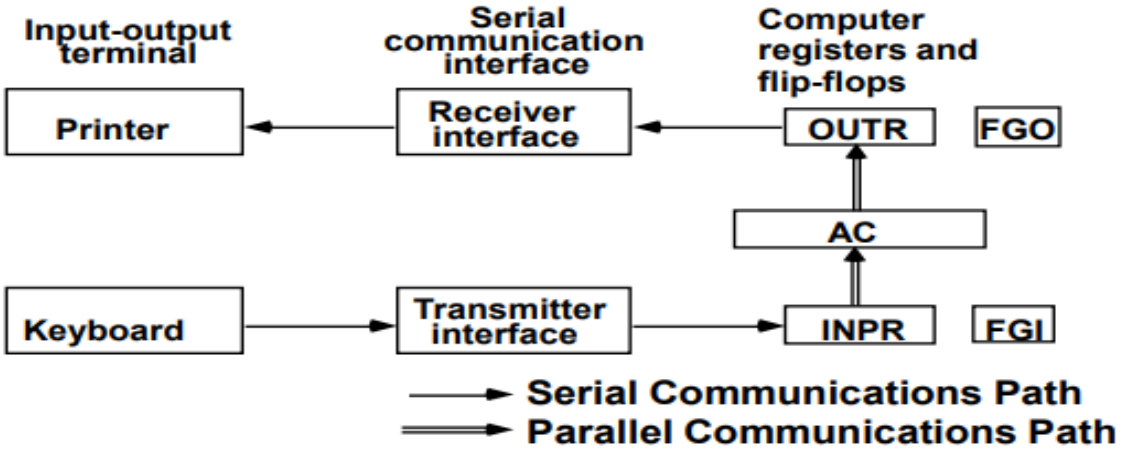


Computer Organisation and Architecture
Unit – II
Types of Instruction

➢ Input/Output & Interrupt

A Terminal with a keyboard and a Printer

- **Input-Output Configuration**



INPR	Input register - 8 bits
OUTR	Output register - 8 bits
FGI	Input flag - 1 bit
FGO	Output flag - 1 bit
IEN	Interrupt enable - 1 bit

- The terminal sends and receives serial information
- The serial info. from the keyboard is shifted into INPR
- The serial info. for the printer is stored in the OUTR
- INPR and OUTR communicate with the terminal serially and with the AC in parallel.
- The flags are needed to *synchronize* the timing difference between I/O device and the computer

Computer Organisation and Architecture
Unit – II
Types of Instruction

- I/O Instructions

$D_7IT_3 = p$
$IR(i) = B_i$, i = 6, ..., 11

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$	Input char. to AC
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$	Output char. from AC
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

Computer Organisation and Architecture
Unit – II
Interrupt Cycle

➢ Program Interrupt

- Open communication only when some data has to be passed --> *interrupt*.

- The I/O interface, instead of the CPU, monitors the I/O device.

- When the interface founds that the I/O device is ready for data transfer, it generates an interrupt request to the CPU

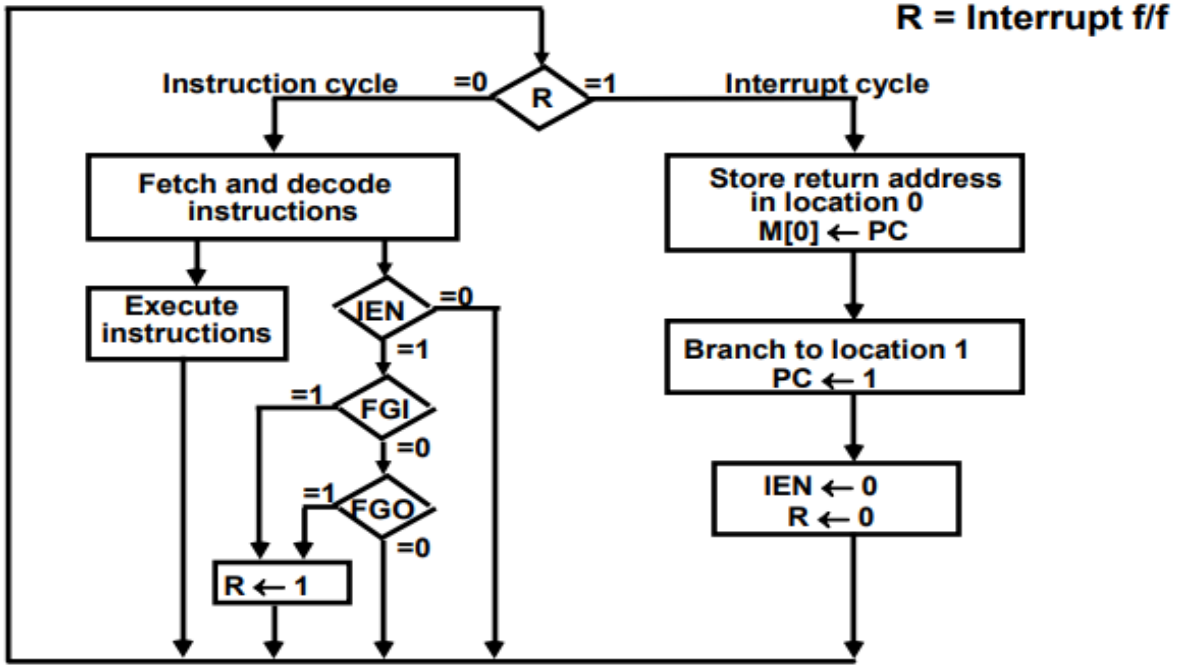
- Upon detecting an interrupt, the CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing.

* IEN (Interrupt-enable flip-flop)

  - can be set and cleared by instructions
  - when cleared, the computer cannot be interrupted

Computer Organisation and Architecture
Unit – II
Interrupt Cycle

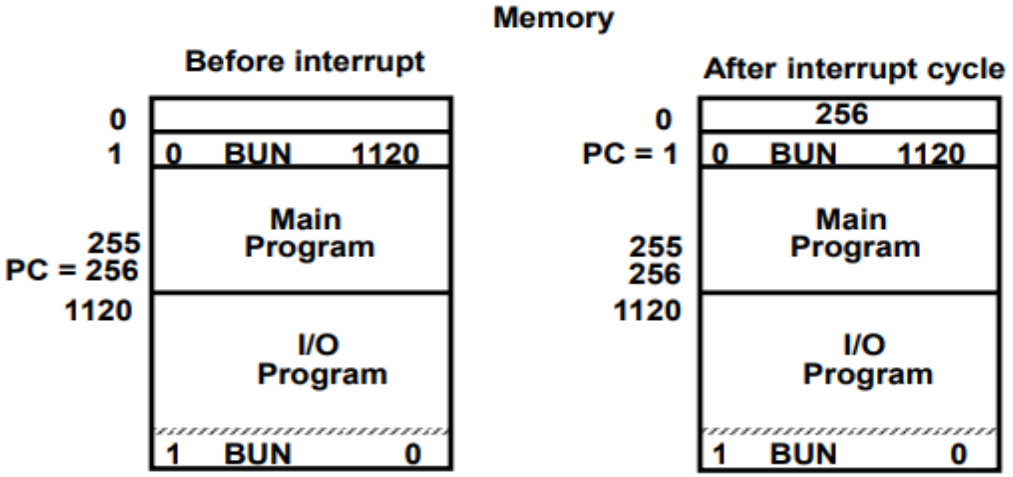
- Interrupt Cycle



- The interrupt cycle is a HW implementation of a branch and save return address operation.
- At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1.
- At memory address 1, the programmer must store a branch instruction that sends the control to an interrupt service routine
- The instruction that returns the control to the original program is "indirect BUN 0"

Computer Organisation and Architecture
Unit – II
Interrupt Cycle

- Register Transfer in interrupt cycle



Register Transfer Statements for Interrupt Cycle

- R F/F ← 1 if IEN (FGI + FGO)$T_0'T_1'T_2'$

  ⇔ $T_0'T_1'T_2'$ (IEN)(FGI + FGO): R ← 1

- The fetch and decode phases of the instruction cycle must be modified ➔Replace T_0, T_1, T_2 with $R'T_0$, $R'T_1$, $R'T_2$
- The interrupt cycle :

RT_0: AR ← 0, TR ← PC

RT_1: M[AR] ← TR, PC ← 0

RT_2: PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0

Computer Organisation and Architecture
Unit – II
Addressing Modes

- The addressing mode specifies a rule for interpreting or modifying the **address field** of the instruction before the operand is actually executed.
- Computers use addressing mode techniques for the purpose of accommodating one of the following provisions:

  1. To give programming versatilities to the user to be more flexible.

  2. To reduce the number of bits in the addressing field of the instruction.
- In some some computers, the addressing mode of the instruction is specified with distinct binary code.

Instruction format with mode field

Computer Organisation and Architecture
Unit – II
Addressing Modes

- Other computers use a single binary for operation & Address mode.
- The mode field is used to locate the operand.
- Address field may designate a memory address or a processor register.
- There are 2 modes that need no address field at all (Implied & immediate modes).

Computer Organisation and Architecture

Unit – II

Addressing Modes

- The most well known addressing mode then are:
  - Implied mode.
  - Immediate mode
  - Register mode
  - Register Indirect mode
  - Auto-increment or Auto-decrement mode
  - Direct Mode
  - Indirect Mode
  - Relative Address Mode
  - Index Addressing Mode

Computer Organisation and Architecture
Unit – II
Addressing Modes

- Different ways in which the address of an operand in specified in an instruction is referred to as addressing modes.
- Register mode
  - Operand is the contents of a processor register.
  - Address of the register is given in the instruction.
  - E.g. *Clear R1*
- Absolute mode
  - Operand is in a memory location.
  - Address of the memory location is given explicitly in the instruction.
  - E.g. *Clear A*
  - Also called as "Direct mode" in some assembly languages
- Register and absolute modes can be used to represent variables

Computer Organisation and Architecture
Unit – II
Addressing Modes

Immediate Mode

- Operand is given explicitly in the instruction.
- E.g. *Move #200, R0*
- Can be used to represent constants.

- Register, Absolute and Immediate modes contained either the address of the operand or the operand itself.
- Some instructions provide information from which the memory address of the operand can be determined
  - That is, they provide the "Effective Address" of the operand.
  - They do not provide the operand or the address of the operand explicitly.
- Different ways in which "Effective Address" of the operand can be generated.

Computer Organisation and Architecture
Unit – II
Addressing Modes

Numerical Example of Addressing Modes

PC=200	R1=400
XR=100	AC

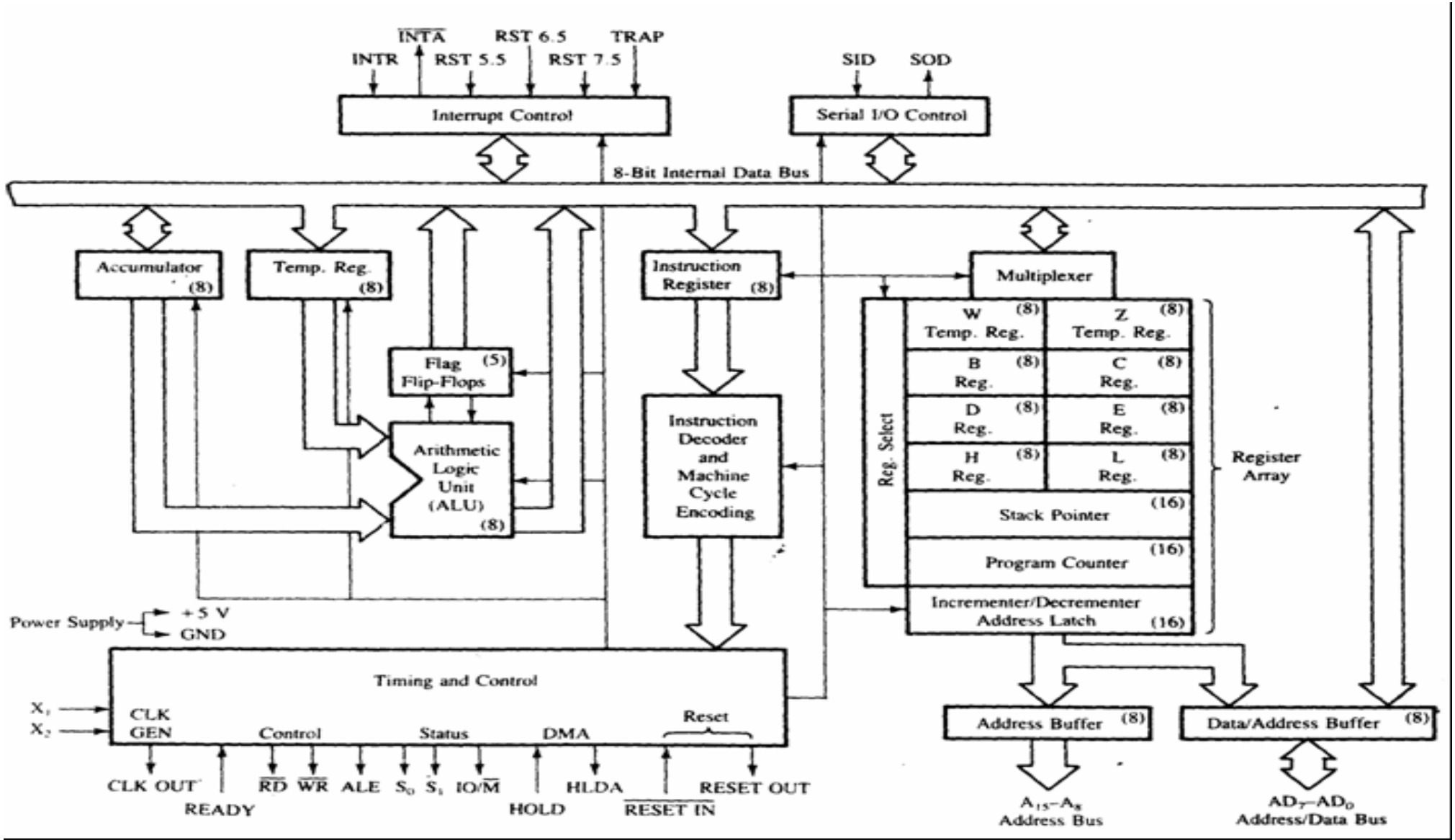
Addressing mode	eff. Add	Content of AC
Direct Address	500	800
Immediate operand	201	500
Indirect Address	800	300
Relative Address	702(PC=PC+2)	325
Indexes Address	600(XR+500)	900
Register	---	400
Register Indirect	400	700
Auto-increment	400	700
Auto-decrement	399	450

Address	Memory	
200	Load to AC	Mode
201	Address=500	
202	Next Instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

Computer Organisation and Architecture

Unit – II

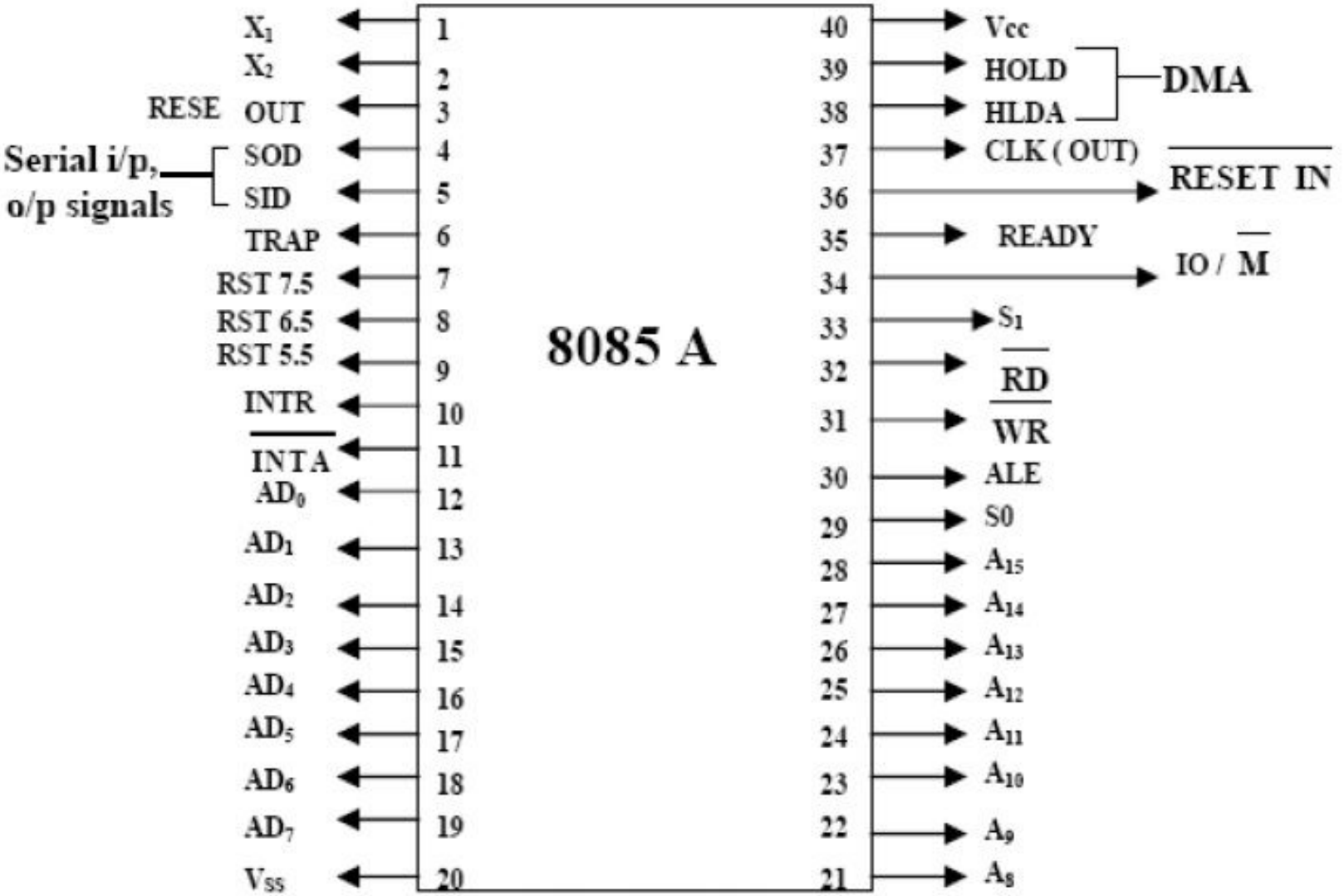
8085 Architecture (Block Diagram)



Computer Organisation and Architecture

Unit – II

8085 Architecture (Pin Diagram)



8085 Pin Diagram