Basic computer organization and design

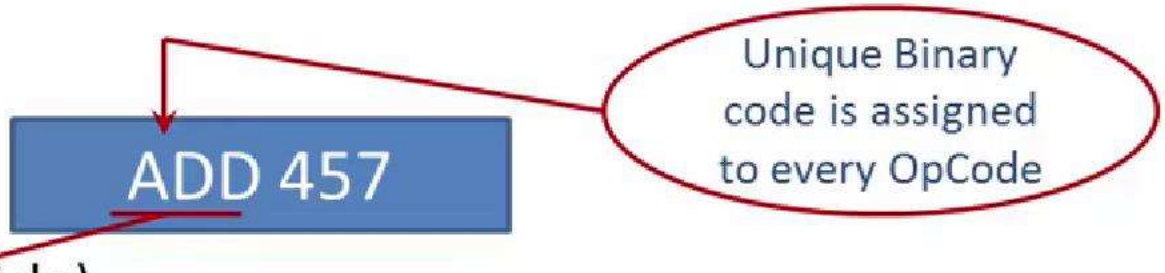
R Anuj, Assistant Professor, Dept. of CSE,
SVECW

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.
  - The computer reads each instruction from memory and places it in a control register.
  - The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of microoperations.

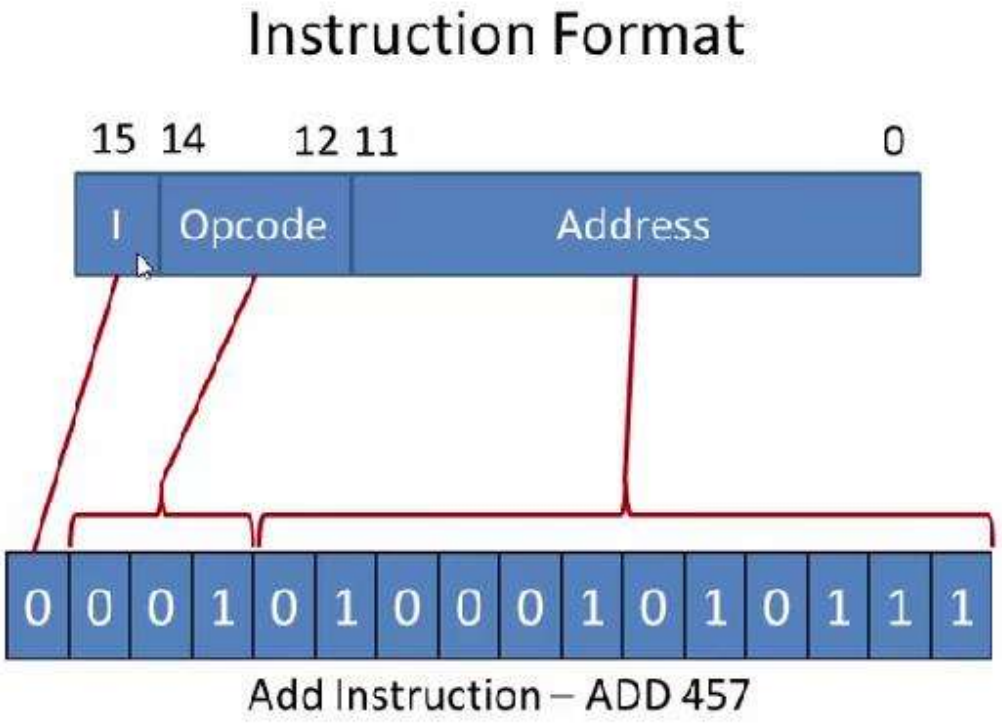
Instruction codes:

- Instruction Code
  - An instruction code is a group of bits that instruct the computer to perform a specific operation.
  - Example

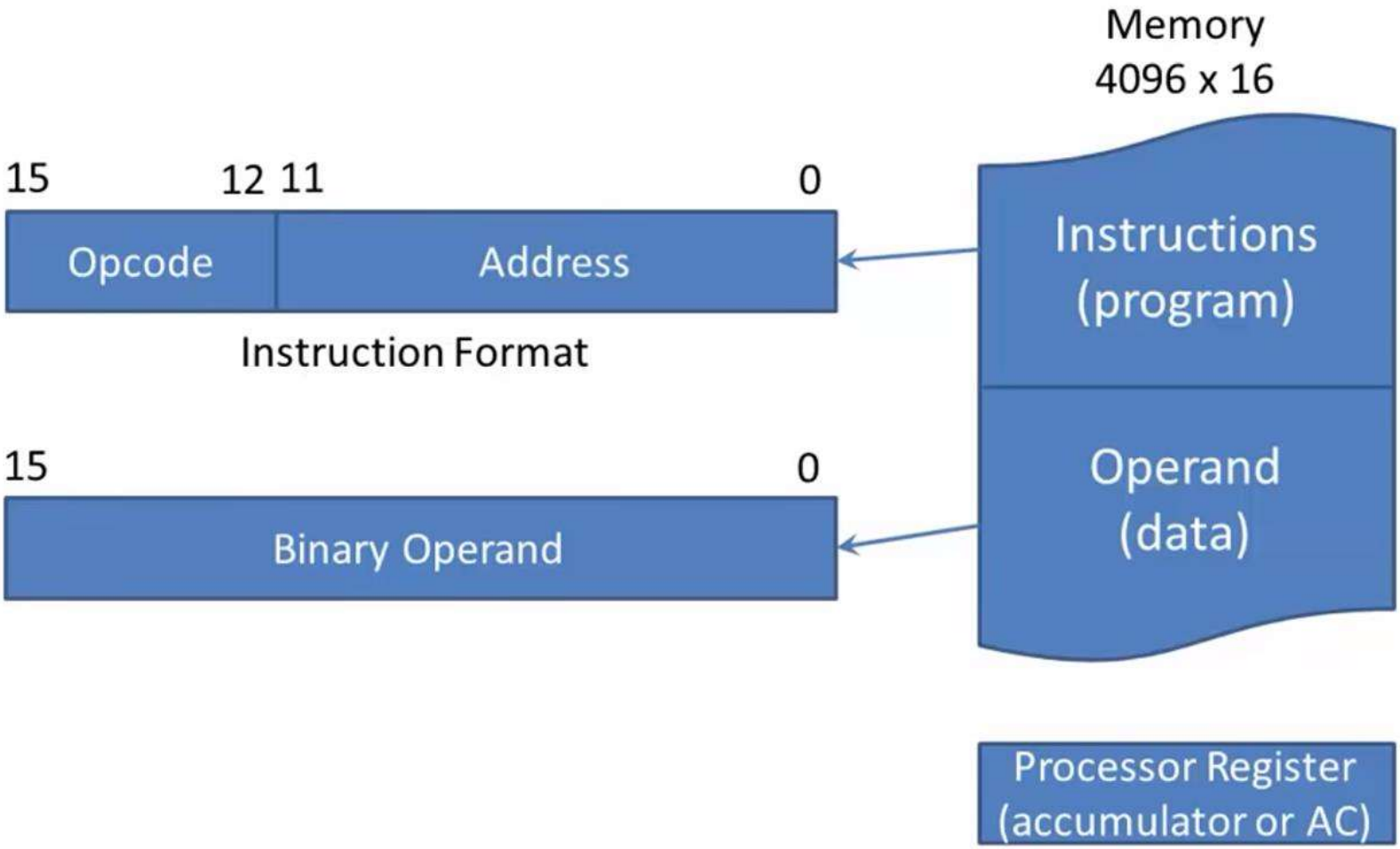


- Operation Code (Opcode)
  - The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement.
  - The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.
  - The operation code must consist of at least n bits for a given 2^n (or less) distinct operations.

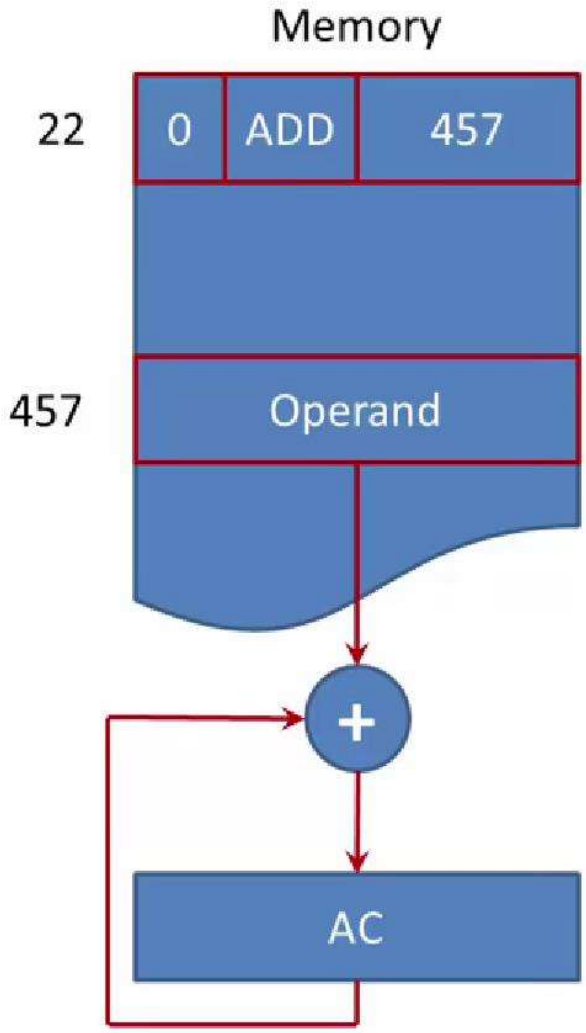
Instruction Format of a Basic Computer:

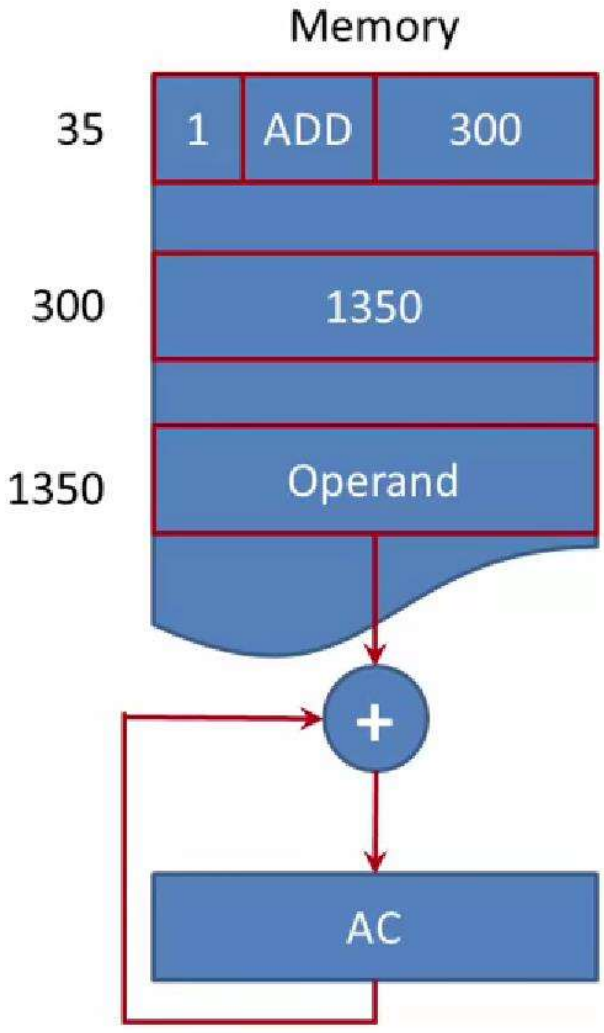


Stored Program Organization:



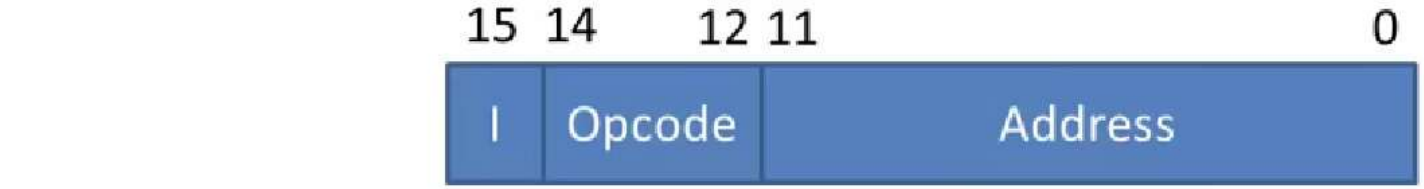
Direct & Indirect Addressing of Memory:

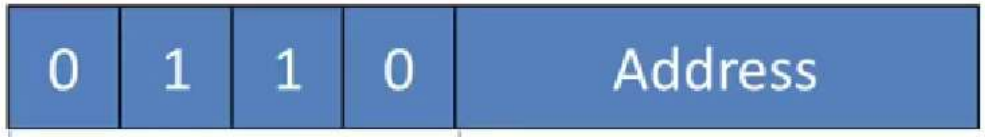




Types of Computer Instructions

1. Memory Reference Instruction

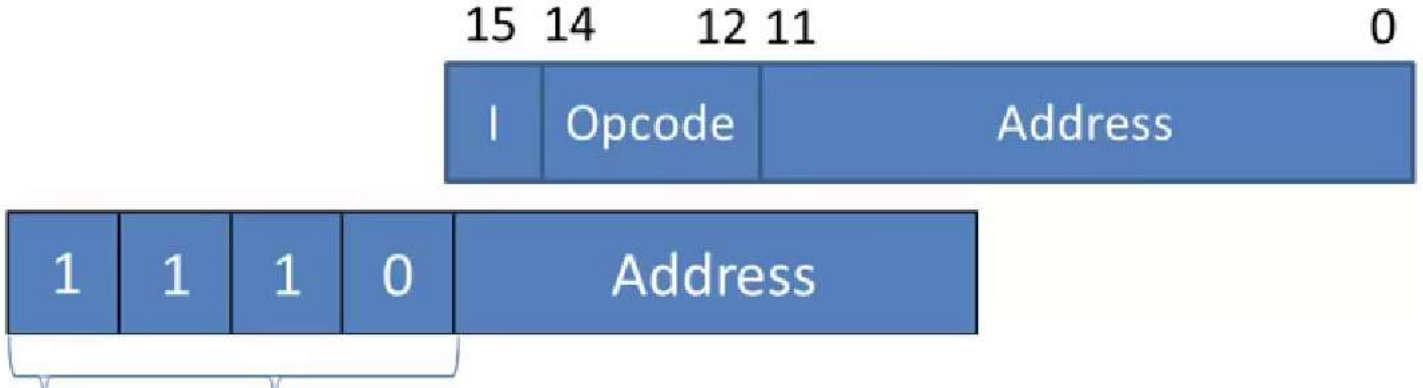




0xxx	AND	AND the content of memory to AC
1xxx	ADD	Add the content of memory to AC
2xxx	LDA	Load memory word to AC
3xxx	STA	Store content of AC in memory
4xxx	BUN	Branch unconditionally
5xxx	BSA	Branch and save return address
6xxx	ISZ	Increment and skip if zero

Types of Computer Instructions

1. Memory Reference Instruction



0xxx	8xxx	AND	AND the content of memory to AC
1xxx	9xxx	ADD	Add the content of memory to AC
2xxx	Axxx	LDA	Load memory word to AC
3xxx	Bxxx	STA	Store content of AC in memory
4xxx	Cxxx	BUN	Branch unconditionally
5xxx	Dxxx	BSA	Branch and save return address
6xxx	Exxx	ISZ	Increment and skip if zero

Types of Computer Instructions

2. Register Reference Instruction

7800	CLA	Clear AC
7400	CLE	Clear E
7200	CMA	Complement AC
7100	CME	Complement E
7080	CIR	Circulate right AC and E
7040	CIL	Circulate left AC and E
7020	INC	Increment AC

Types of Computer Instructions

2. Register Reference Instruction

7010	SPA	Skip next instruction if AC is positive
7008	SNA	Skip next instruction if AC is negative
7004	SZA	Skip next instruction if AC is zero
7002	SZE	Skip next instruction if E is zero
7001	HLT	Halt computer

Types of Computer Instructions

3. Input – Output Instruction

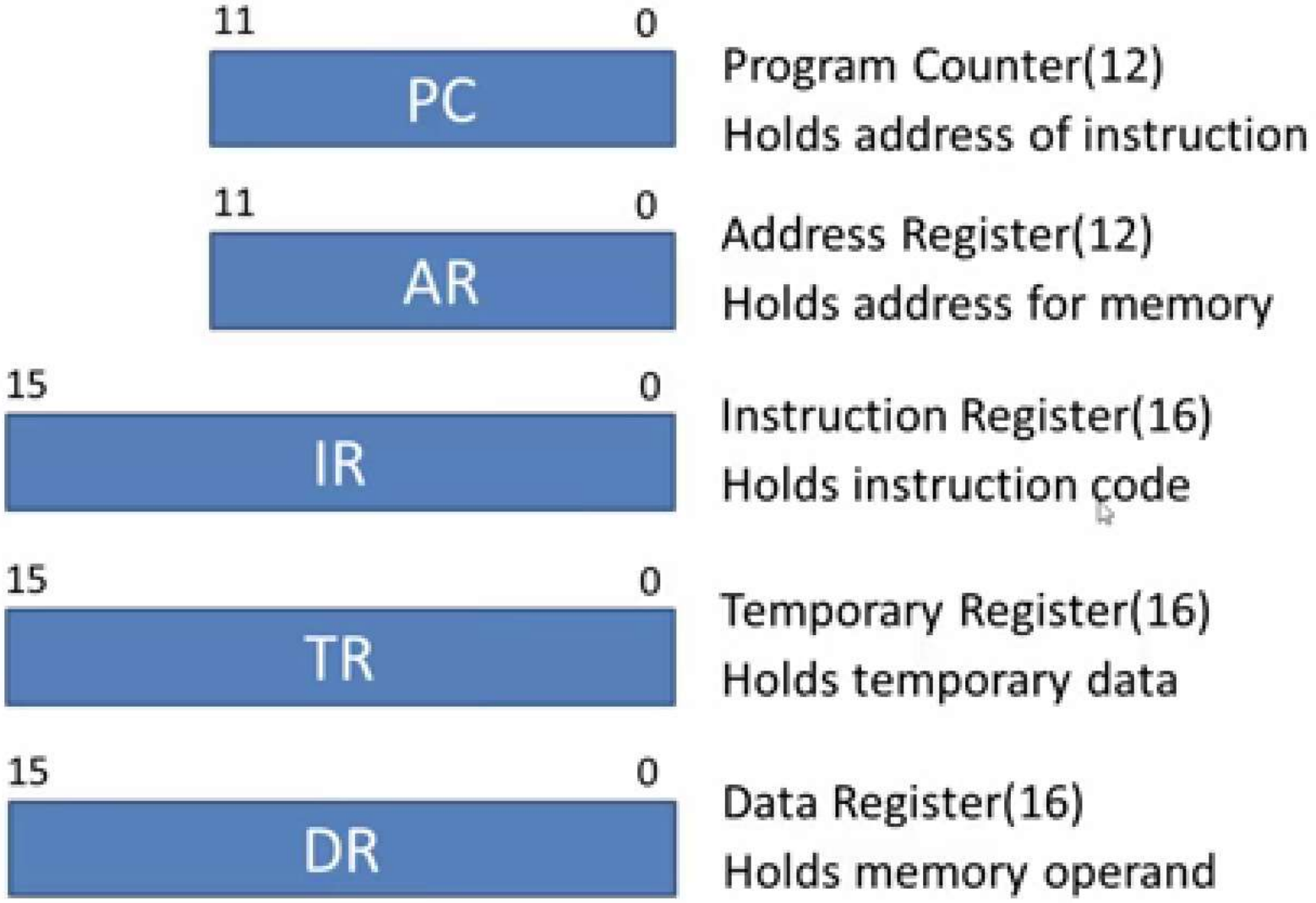
1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0

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

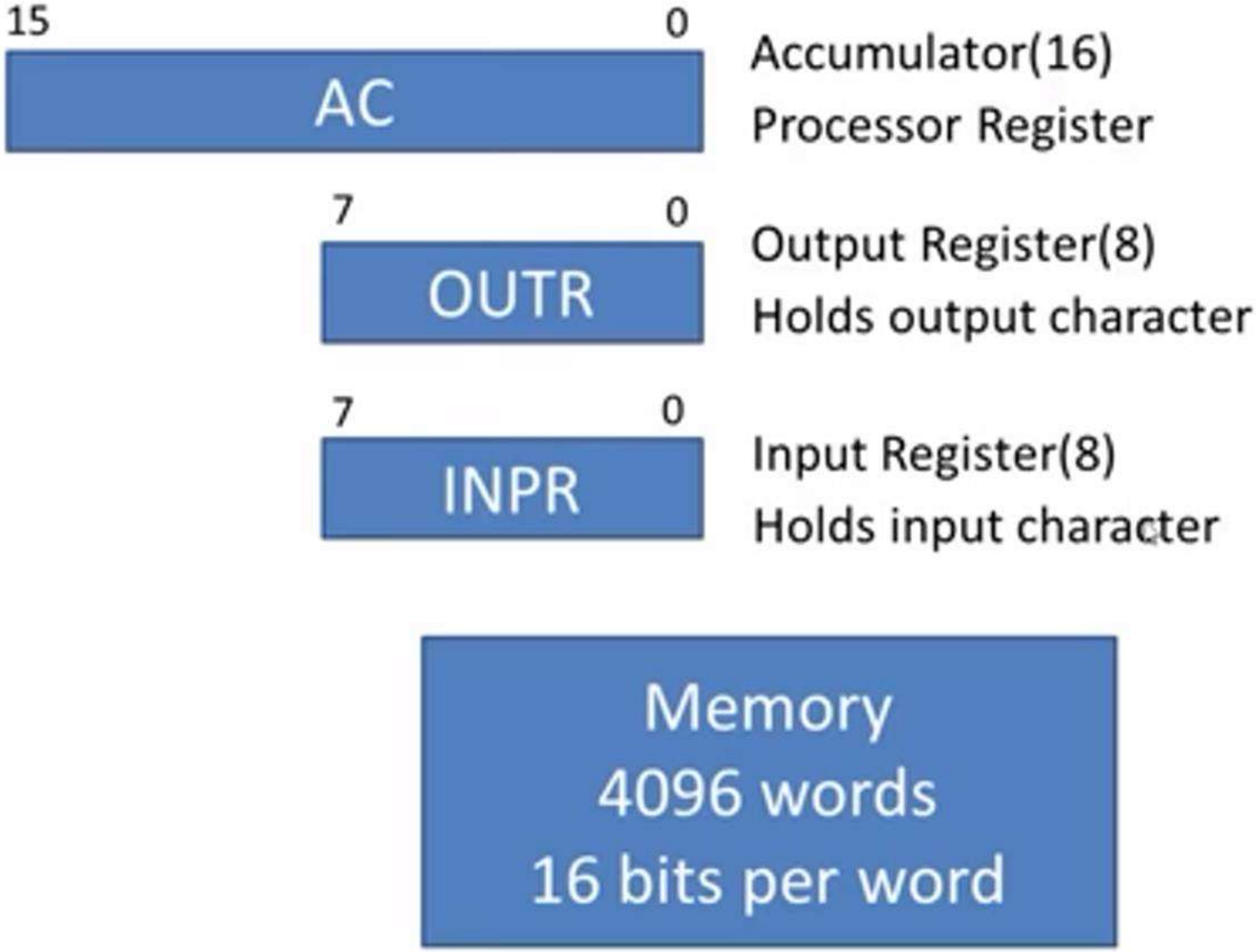
Instruction Set Completeness:

- Instruction set is said to be complete if it includes 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

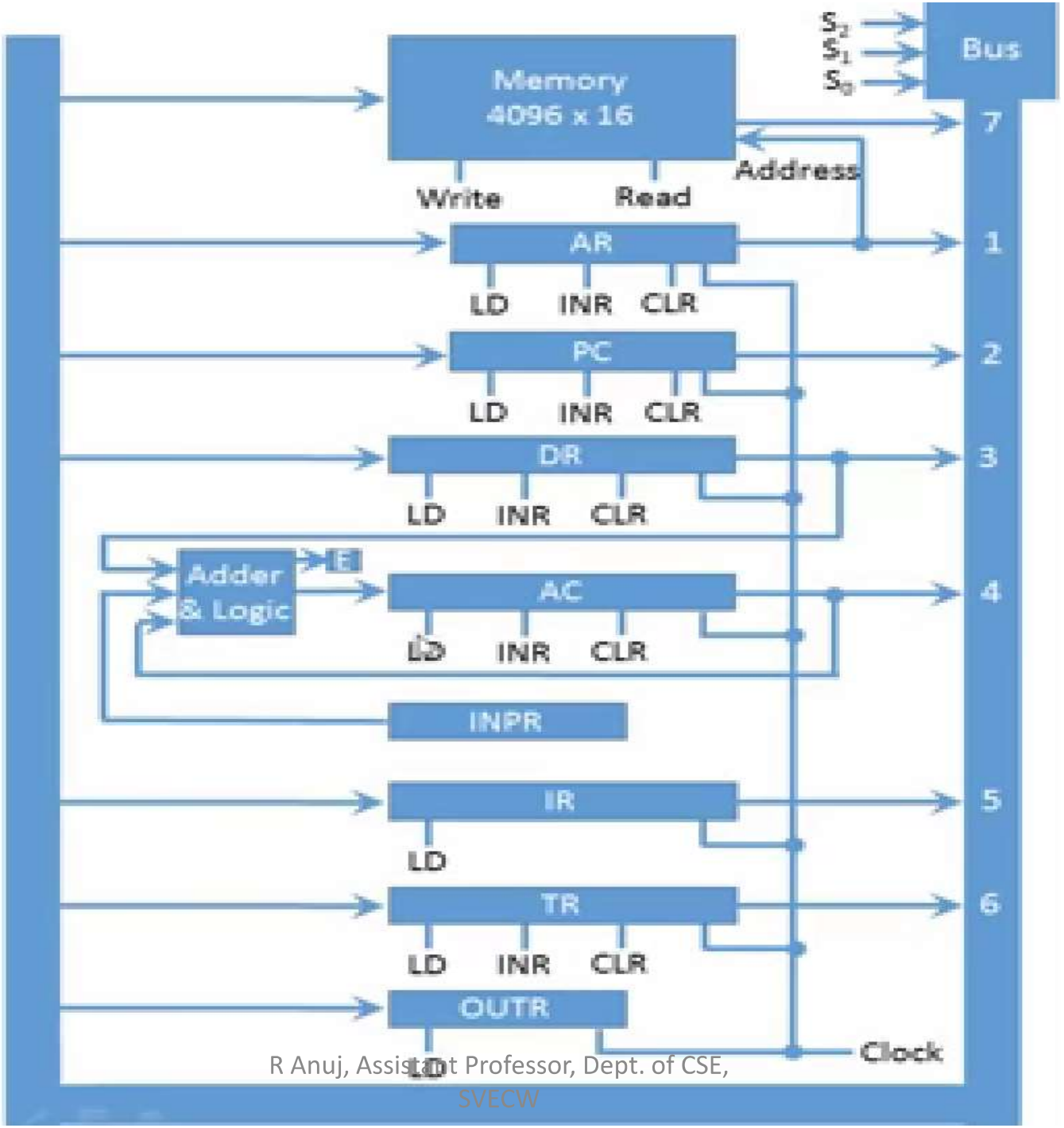
Computer Registers:



Computer Registers:



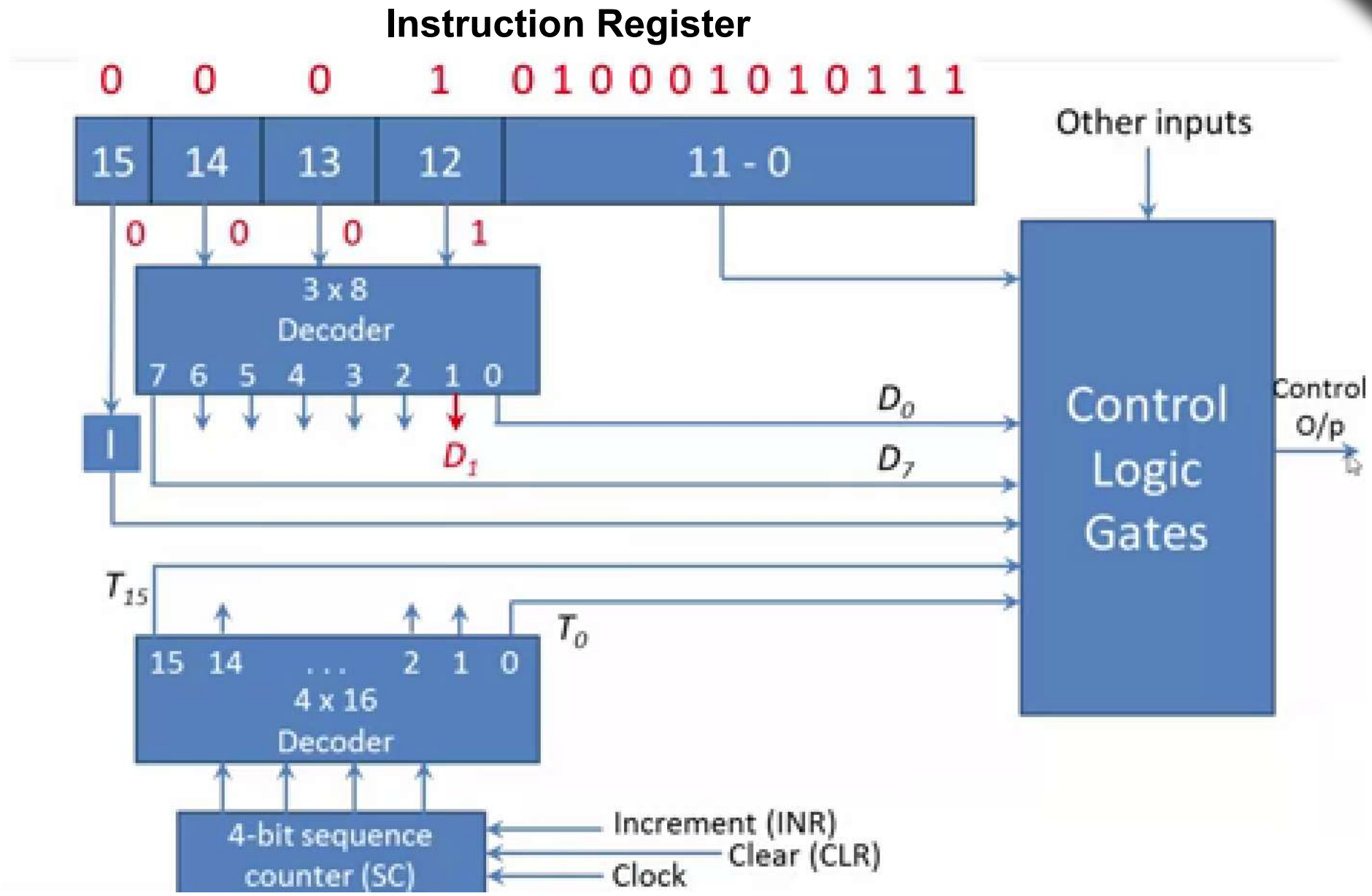
Common BUS System of Basic Computer:



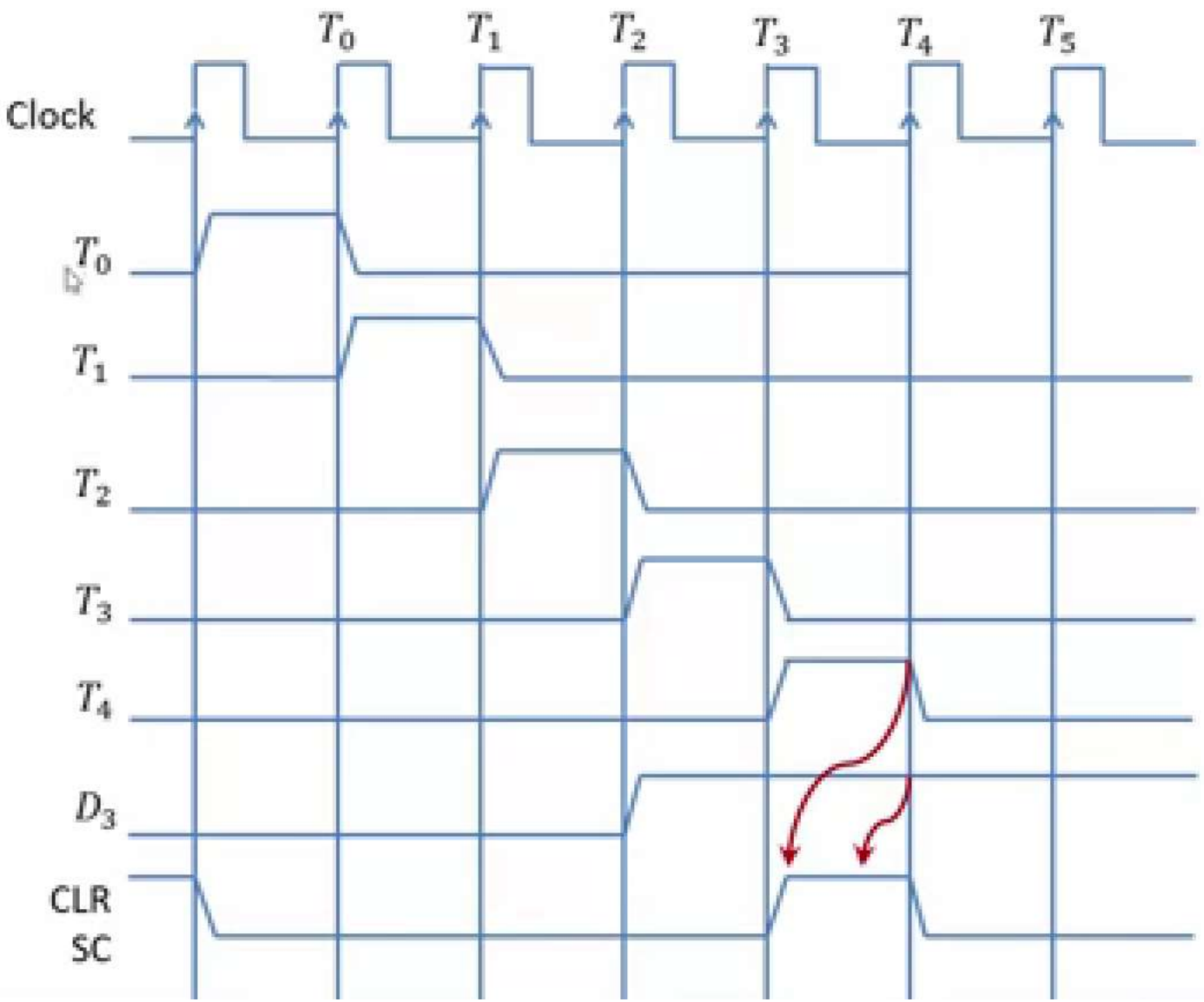
Control Organization:

- Hardwired Control
  - The control logic is implemented with gates, flips-flops, decoders and other digital circuits.
  - It can be optimized to produce a fast mode of operation.
  - It requires changes in the wiring among the various components if the design has to be modified or changed.
- Microprogrammed Control
  - The control information is stored in a control memory.
  - The control memory is programmed to initiate the required sequence of microoperations.
  - Any required changes or modifications can be done by updating the microprogram in control memory.

Control Unit of Basic Computer:



Timing Cycle for D_3T_4 : SC←0



Instruction cycle:

- A program will reside in some memory location of the computer where each program will have a set of instructions.
- In basic computer each instruction will go through following phases(Instruction Cycle):

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

After step 4 the control goes back to step 1 to fetch, decode and execute the next instruction

This process will continue until a HALT instruction is encountered

Instruction cycle:

Fetch & Decode phase:

- PC is loaded with the address of the next instruction in the program
- The microoperations for fetch and decode phases are as follows:

$$T_0 : AR \leftarrow PC$$

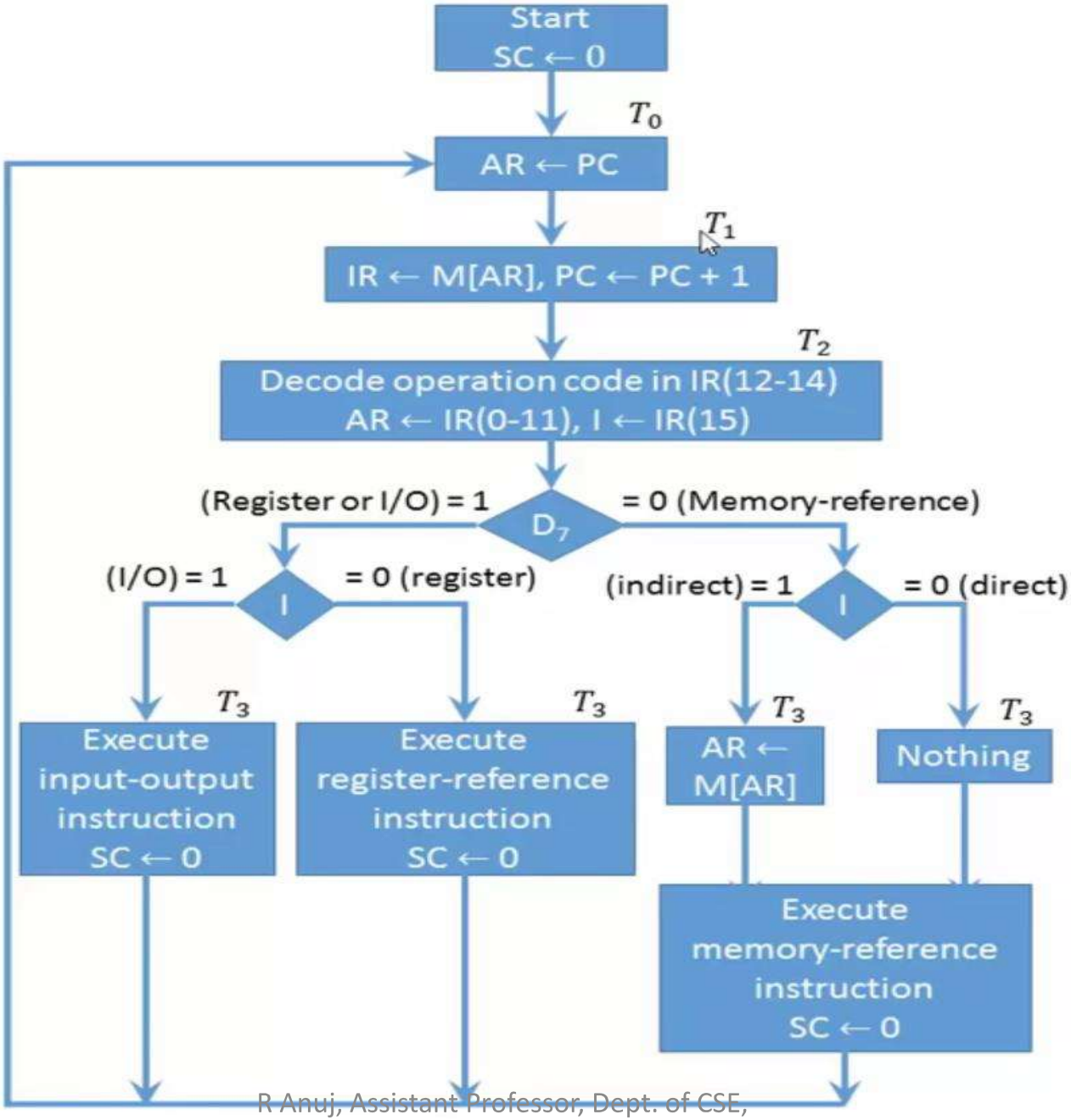
$$T_1 : IR \leftarrow M[AR], PC \leftarrow PC + 1$$

$$T_2 : D_0, \ldots, D_7 \leftarrow Decode\ IR(12-14), AR \leftarrow IR(0-11), I \leftarrow IR(15)$$

Instruction cycle:

- Determine the type of instruction
  - During time T_3, the control unit determines the type of instruction i.e. Memory reference, Register reference or Input-Output instruction.
  - If $D_7 = 1$ then instruction must be register reference or input-output else memory reference instruction.

Instruction cycle flow chart:



1. Register Reference Instructions:

$D_7 I' T_3 = r$ (common to all register reference instructions)

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

CLA	rB_{11}	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}	$E \leftarrow 0$	Clear E
CMA	rB_9	$AC \leftarrow AC'$	Complement AC
CME	rB_8	$E \leftarrow E'$	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 AC is positive
SNA	rB_3	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if AC is negative
SZA	rB_2	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$	Skip if AC is zero
SZE	rB_1	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E is zero
HLT	rB_0	$S \leftarrow 0$ (S is a start-stop flip-flop)	Halt Computer

2.Memory Reference Instruction:

1. AND: AND to AC

   This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC.

$$D_0T_4: DR \leftarrow M[AR]$$

$$D_0T_5: AC \leftarrow AC \wedge DR, SC \leftarrow 0$$

2. ADD: ADD to AC

   This instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and the output carry C_{out} is transferred to the E (extended accumulator) flip-flop.

$$D_1T_4: DR \leftarrow M[AR]$$

$$D_1T_5: AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$$

2.Memory Reference Instruction:

3. LDA: Load to AC

   This instruction transfers the memory word specified by the effective address to AC.

$$D_2T_4: DR \leftarrow M[AR]$$

$$D_2T_5: AC \leftarrow DR, SC \leftarrow 0$$

4. STA: Store AC

   This instruction stores the content of AC into the memory word specified by the effective address.

$$D_3T_4: M[AR] \leftarrow AC, SC \leftarrow 0$$

2.Memory Reference Instruction:

5. BUN: Branch Unconditionally

This instruction transfers the program to instruction specified by the effective address. The BUN instruction allows the programmer to specify an instruction out of sequence and the program branches (or jumps) unconditionally.

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

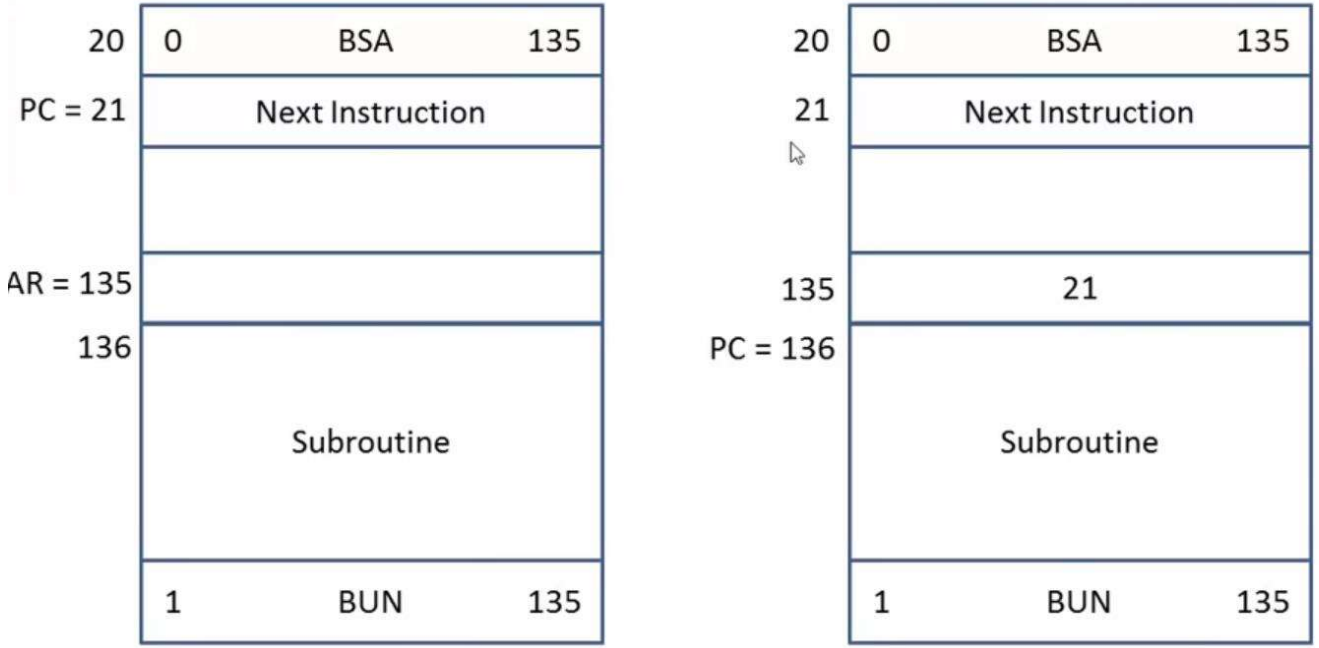
2.Memory Reference Instruction:

6. BSA: Branch and Save Return Address

This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address.

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

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



Memory, PC and AR at Time T_4

2.Memory Reference Instruction:

7. ISZ: Increment and Skip if Zero

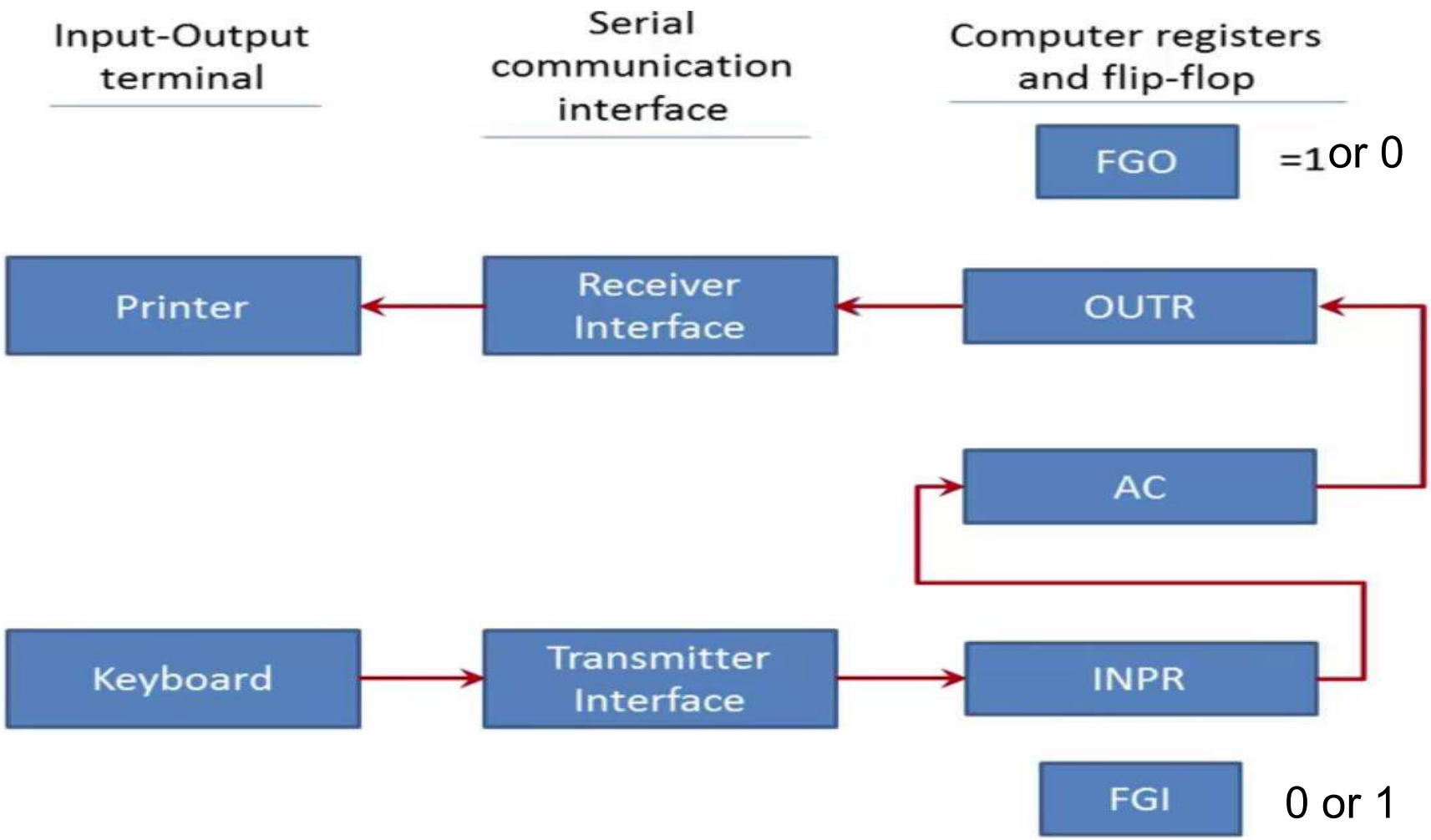
   These instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1. Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory.

D_6T_4: $DR \leftarrow M[AR]$

D_6T_5: $DR \leftarrow DR + 1$

D_6T_6: $M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

3.I/O Reference Instruction:



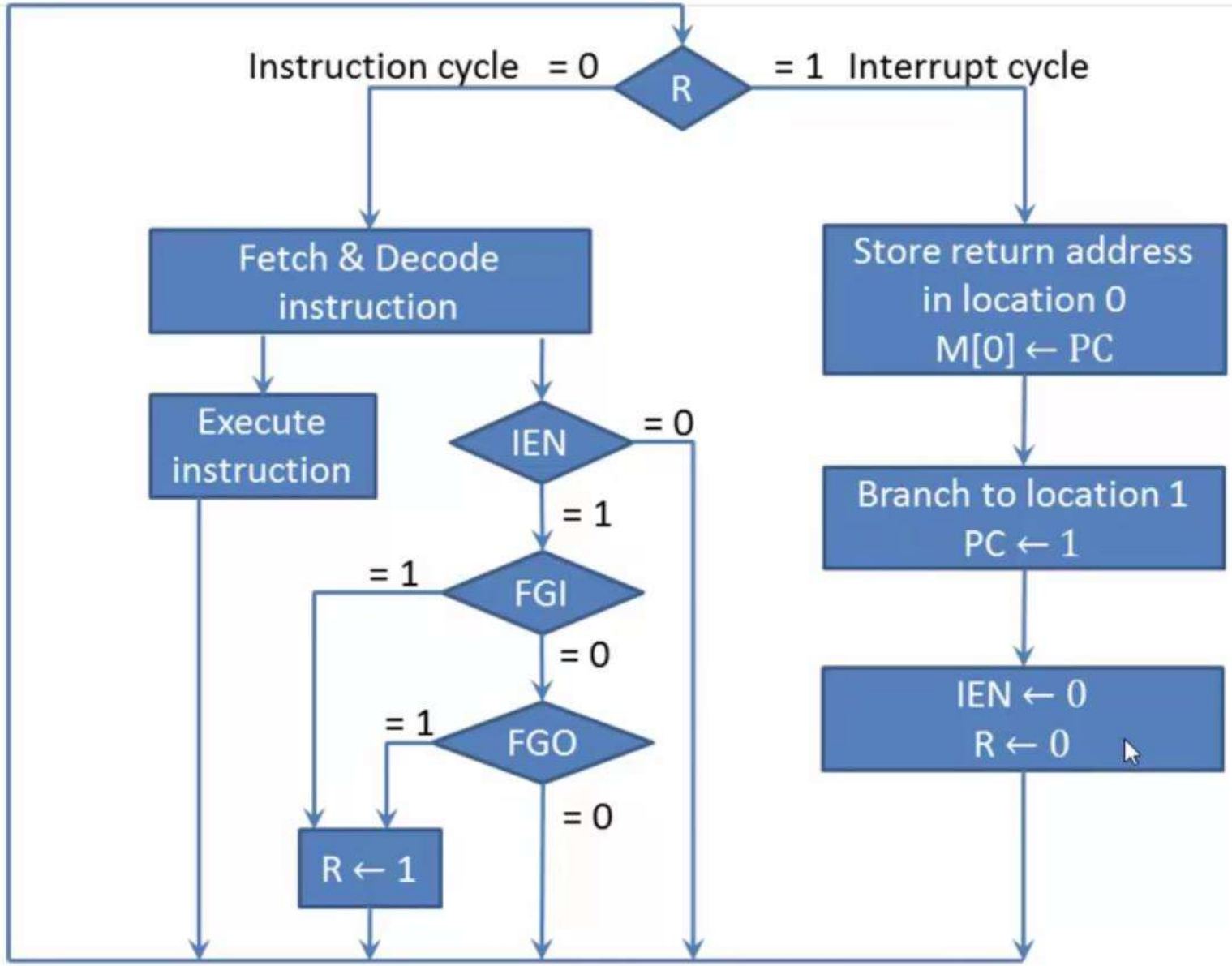
3.I/O Reference Instruction:

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

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

INP	pB_{11}	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$	Input Character
OUT	pB_{10}	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$	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

Interrupt Cycle:



Register Transfer statements for Interrupt cycle:

- The condition for setting flip-flop R= 1 can be expressed with the following register transfer statement:

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

- Therefore the interrupt cycle statements are :

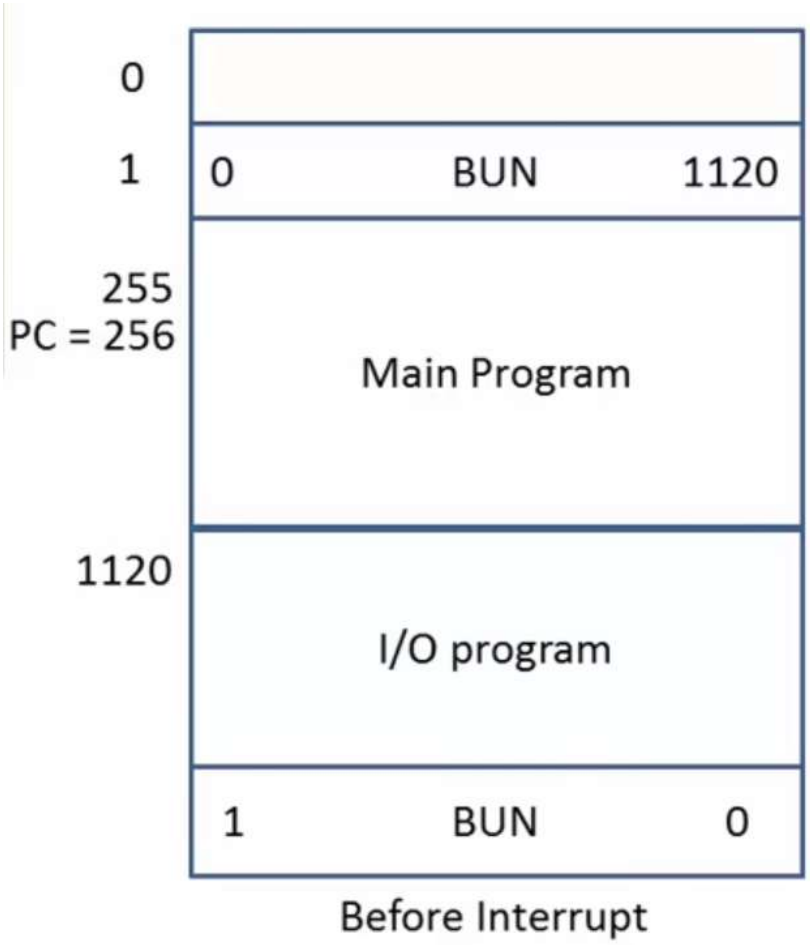
$$RT_0: AR \leftarrow 0,\ TR \leftarrow PC$$

Return address is stored in TR

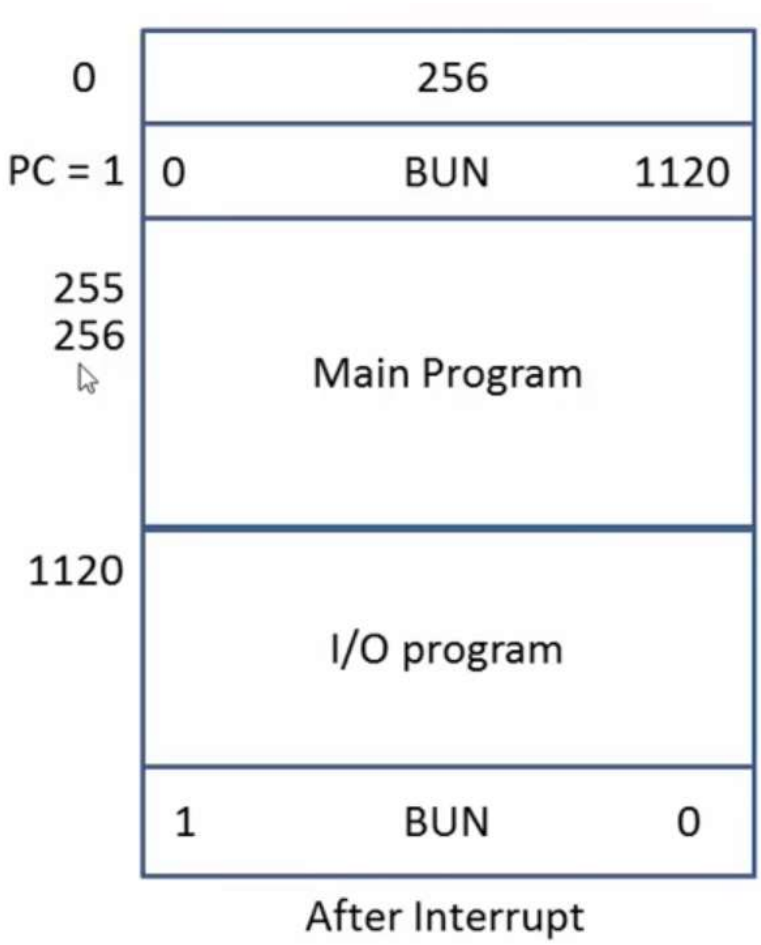
$$RT_1: M[AR] \leftarrow TR,\ PC \leftarrow 0$$

$$RT_2: PC \leftarrow PC + 1,\ IEN \leftarrow 0,\ R \leftarrow 0,\ SC \leftarrow 0$$

Demonstration of Interrupt cycle:

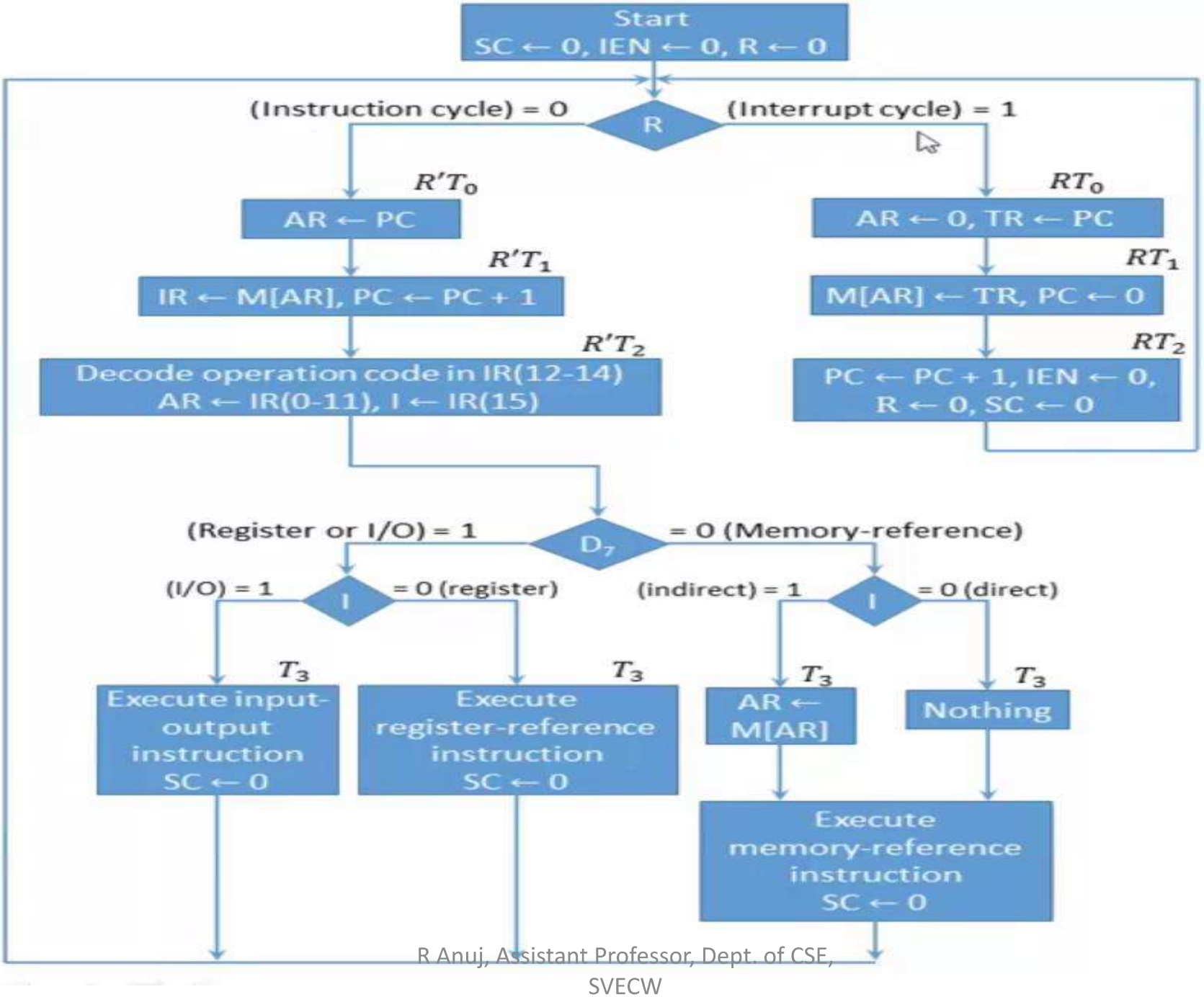


Before Interrupt

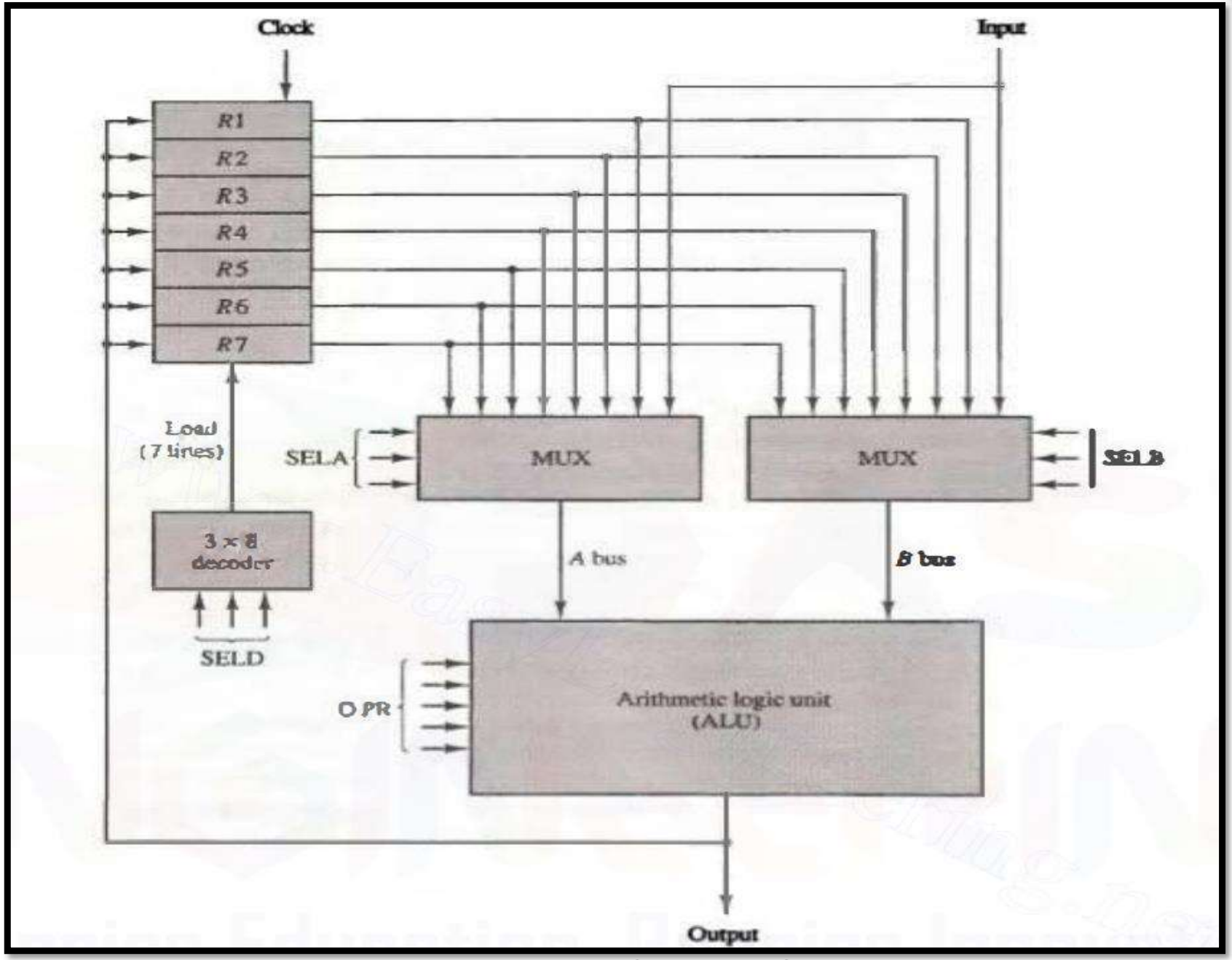


After Interrupt

Design of Basic Computer:



General Register Organization



General Register Organization

Control Word

3	3	3	5
SLE A	**SLE B**	**SLE D**	**OPR**

For example, to perform the operation R1 <--R2 + R3 the control must provide binary selection variables to the following selector inputs:

1. MUX A selector (SELA): to place the content of R2 into bus A.
2. MUX B selector (SELB): to place the content o f R 3 into bus B.
3. ALU operation selector (OPR): to provide the arithmetic addition A+ B.
4. Decoder destination selector (SELD): to transfer the content of the output bus into R 1

General Register Organization

Control Word

3	3	3	5
SLE A	SLE B	SLE D	OPR

There are 14 binary selection inputs in the unit, and their combined value specifies a control word

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

OPR Select	Operation	Symbol
00000	Transfer *A*	TSFA
00001	Increment *A*	INCA
00010	Add *A* + *B*	ADD
00101	Subtract *A* − *B*	SUB
00110	Decrement *A*	DECA
01000	AND *A* and *B*	AND
01010	OR *A* and *B*	OR
01100	XOR *A* and *B*	XOR
01110	Complement *A*	COMA
10000	Shift right *A*	SHRA
11000	Shift left *A*	SHLA

General Register Organization

Control Word

3	3	3	5
SLE A	**SLE B**	**SLE D**	**OPR**

Examples of Micro operations for the CPU

Microoperation	Symbolic Designation SELA	SELB	SELD	OPR	Control Word
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow$ shl $R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

Stack Organization:

- A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved (LIFO).
- The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.
- There are two types of stack organization
  1. Register stack – built using registers
  2. Memory stack – logical part of memory allocated as stack

Implementation of register Stack:

Two Operations are implemented on Stack:

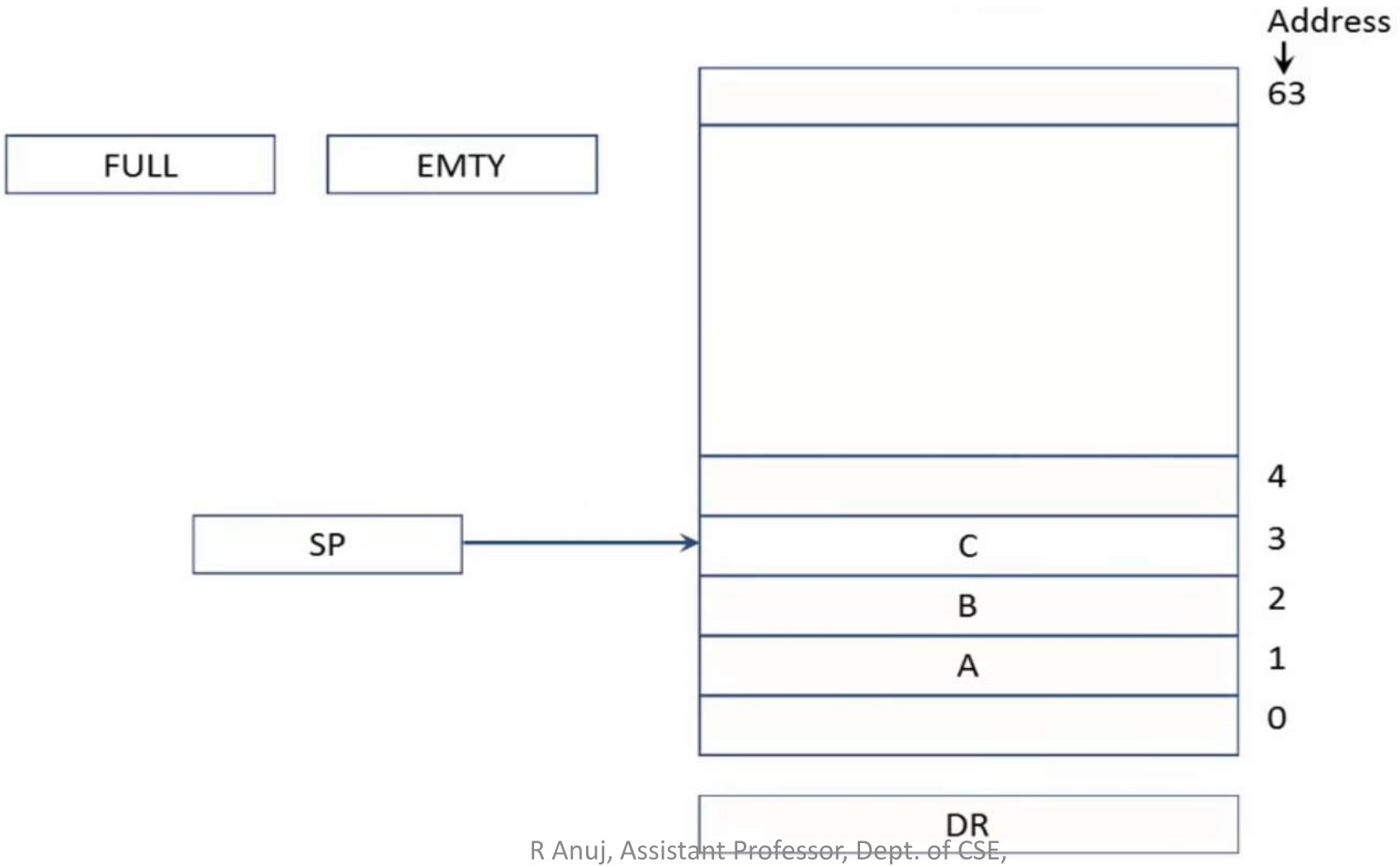
- PUSH Operation
- POP Operation

Applications of the stack:

- Evaluation of the expression using reverse polish notation
- For functions & Subroutines the current content of the registers are stored in to the stack

Register Stack:(64 word stack)

Initially SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0 so that SP points to the word at address 0



Register Stack:

- **PUSH Operation**

  SP ← SP + 1

  M[SP] ← DR

  IF (SP= 0) then (FULL ← 1)

  EMTY ← 0

- **POP Operation**

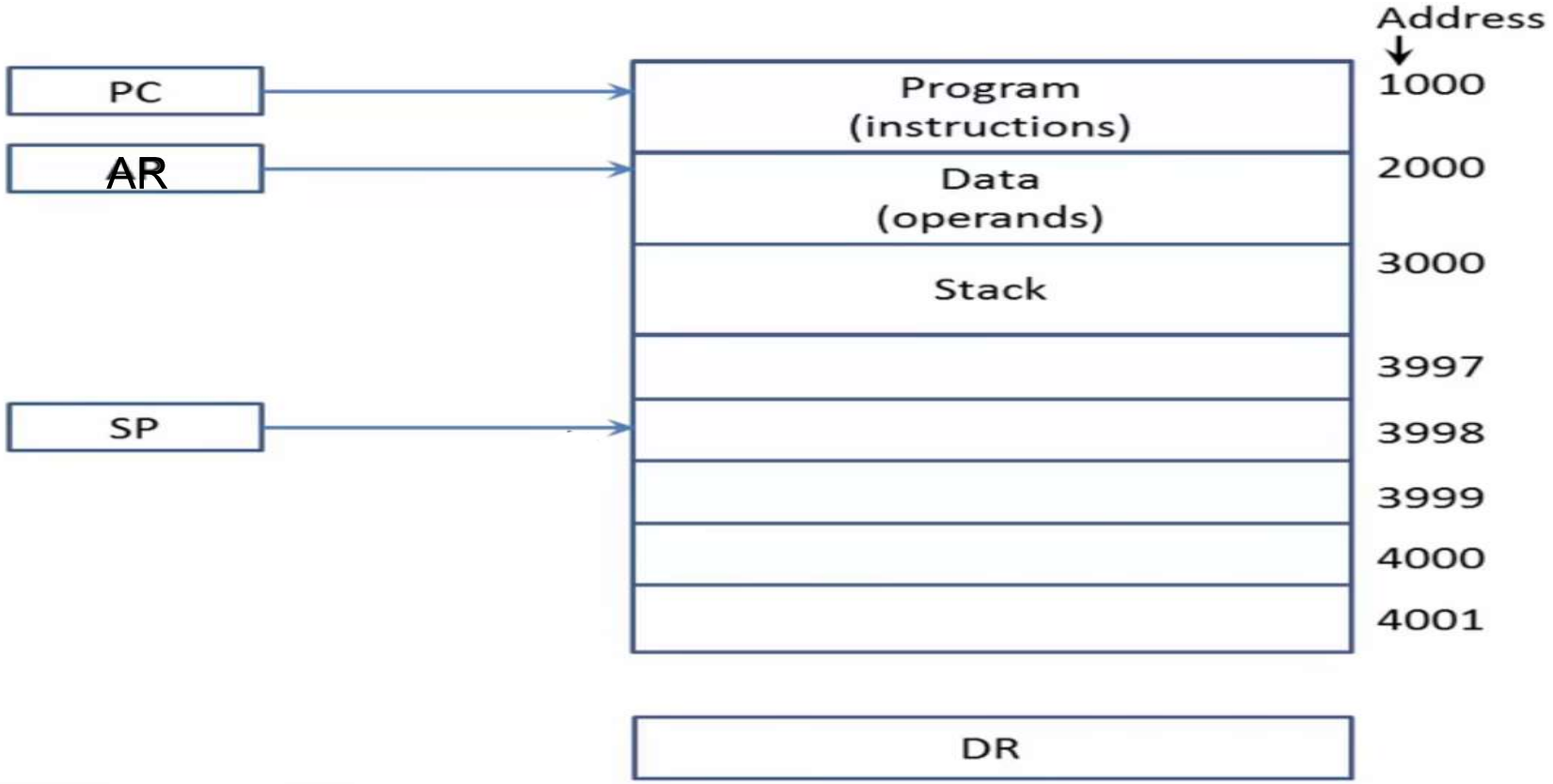
  DR ← M[SP]

  SP ← SP - 1

  IF (SP= 0) then (EMTY ← 1)

  FULL ← 0

Memory Stack:



- **PUSH Operation**

  SP ← SP - 1

  M[SP] ← DR

- **POP Operation**

  DR ← M[SP]

  SP ← SP + 1

Reverse Polish Notation:

- The common mathematical method of writing arithmetic expressions imposes difficulties when evaluated by a computer.
- The Polish mathematician Lukasiewicz showed that arithmetic expressions can be represented in prefix notation as well as postfix notation.

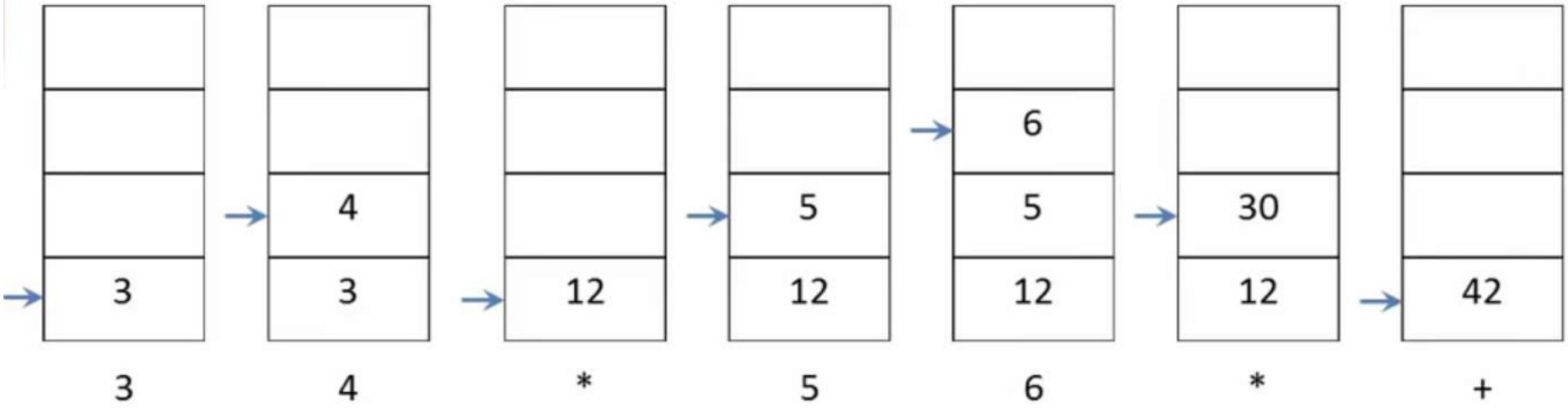
Infix	Prefix or Polish	Postfix or reverse Polish
$A + B$	$+ AB$	$AB +$

$A * B + C * D \longrightarrow AB * CD * +$

Reverse Polish

Evaluation of Arithmetic Expression:

$$(3 * 4) + (5 * 6) \longrightarrow 3\ 4\ *\ 5\ 6\ *\ + \longrightarrow 42$$



Instruction formats:

- Instructions are categorized into different formats with respect to the operand fields in the instructions.
  1. Three Address Instructions
  2. Two Address Instruction
  3. One Address Instruction
  4. Zero Address Instruction
  5. RISC Instructions

Three Address Instruction

- Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.
- The program in assembly language that evaluates X = (A + B) * (C + D) is shown below.

```
ADD    R1, A, B      R1←M[A]+M[B]
ADD    R2, C, D      R2←M[C]+M[D]
MUL    X, R1, R2     M[X]←R1 * R2
```

Two Address Instruction

- Two address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word.
- The program to evaluate X = (A + B) * (C + D) is as follows:

```
MOV    R1, A          R1←M[A]
ADD    R1, B          R1←R1+M[B]
MOV    R2, C          R2←M[C]
ADD    R2, D          R2←R2+M[D]
MUL    R1, R2         R1←R1 * R2
MOV    X, R1          M[X]←R1
```

One Address Instruction

- One address instructions use an implied accumulator (AC) register for all data manipulation.
- For multiplication and division these is a need for a second register.
- However, here we will neglect the second register and assume that the AC contains the result of all operations.
- The program to evaluate X = (A + B) * (C + D) is

```
LOAD   A        AC← M[A]
ADD    B        AC← AC+M[B]
STORE  T        M[T]←AC
LOAD   C        AC← M[C]
ADD    D        AC← AC+M[D]
MUL    T        AC← AC*M[T]
STORE  X        M[X]←AC
```

Zero Address Instruction

- A stack-organized computer does not use an address field for the instructions ADD and MUL.
- The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.
- The program to evaluate X = (A + B) * (C + D) will be written for a stack-organized computer.
- To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse polish notation.

```
PUSH   A        TOS← M[A]
PUSH   B        TOS← M[B]
ADD             TOS← (A+B)
PUSH   C        TOS← M[C]
PUSH   D        TOS← M[D]
ADD             TOS← (C+D)
MUL             TOS← (C+D) * (A+B)
POP    X        M[X] ← TOS
```

RISC(Reduced Instruction Set Computer) Instruction

- A program for a RISC type CPU consists of LOAD and STORE instructions that have one memory and one register address, and computational-type instructions that have three addresses with all three specifying processor registers.
- The following is a program to evaluate X = (A + B) * (C + D)

```
LOAD   R1, A          R1←M[A]
LOAD   R2, B          R2←M[B]
LOAD   R3, C          R3←M[C]
LOAD   R4, D          R4←M[D]
ADD    R1, R1, R2     R1←R1+R2
ADD    R3, R3, R4     R3←R3+R4
MUL    R1, R1, R3     R1←R1*R3
STORE  X, R1          M[X]←R1
```

Addressing Modes:

- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.
- Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:
  1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
  2. To reduce the number of bits in the addressing field of the instruction.
- There are basic 10 addressing modes supported by the computer.

Addressing Modes:

1. Implied Mode
2. Immediate Mode
3. Register Mode
4. Register Indirect Mode
5. Autoincrement or Autodecrement Mode
6. Direct Address Mode
7. Indirect Address Mode
8. Relative Addressing Mode
9. Indexed Addressing Mode
10. Base Register Addressing Mode

Addressing Modes: 1. Implied Mode

- Operands are specified *implicitly* in the definition of the instruction.
- For example, the instruction "complement accumulator (CMA)" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.
- In fact, all register reference instructions that use an accumulator and zero address instructions are implied mode instructions.

Addressing Modes: 2. immediate Mode

- Operand is specified in the instruction itself.
- In other words, an immediate-mode instruction has an operand field rather than an address field.
- The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
- Immediate mode of instructions is useful for initializing register to constant value.
- E.g. MOV R1, 05H

  instruction copies immediate number 05H to R1 register.

Addressing Modes: 3. Register Mode

- Operands are in registers that reside within the CPU.
- The particular register is selected from a register field in the instruction.
- E.g. MOV AX,BX

  move value from BX to AX register

Addressing Modes: 4. Register Indirect Mode

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
- Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.
- The advantage of this mode is that address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.
- E.g. MOV [R1], R2

  value of R2 is moved to the memory location specified in R1.

Addressing Modes: 5. Autoincrement 0r Autodecrement Mode

- This is similar to the register indirect mode expect that the register is incremented or decremented after (or before) its value is used to access memory.
- When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction.

Addressing Modes: 6. Direct Address Mode

- In this mode the effective address is equal to the address part of the instruction.
- The operand resides in memory and its address is given directly by the address field of the instruction.
- E.g. ADD 457

Addressing Modes: 7. Indirect Address Mode

- In this mode the address field of the instruction gives the address where the effective address is stored in memory.
- Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.
- The effective address in this mode is obtained from the following computational:

  Effective address = address part of instruction + content of CPU register

Addressing Modes: 8. Relative Address Mode

- In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.
- The address part of the instruction is usually a signed number which can be either positive or negative.

Effective address = address part of instruction + content of PC

Addressing Modes: 9. Indexed Addressing Mode

- In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
- The indexed register is a special CPU register that contain an index value.
- The address field of the instruction defines the beginning address of a data array in memory.
- Each operand in the array is stored in memory relative to the begging address.

Effective address = address part of instruction + content of index register

Addressing Modes: 10. Base Register Addressing Mode

- In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.
- A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address.
- The base register addressing mode is used in computers to facilitate the relocation of programs in memory.

Effective address = address part of instruction + content of base register

Data Transfer & data Manipulation instructions:

Data Transfer Instructions:

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data Transfer & data Manipulation instructions:

Data Manipulation Instructions:

- Data manipulation instructions perform operations on data and provide the computational capabilities for the computer.
- The data manipulation instructions in a typical computer are usually divided into three basic types:
  1. Arithmetic instructions
  2. Logical and bit manipulation instructions
  3. Shift instructions

Data Transfer & data Manipulation instructions:

1. Arithmetic Instructions:

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Data Transfer & data Manipulation instructions:

2. Logical & Bit manipulation Instructions:

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Data Transfer & data Manipulation instructions:

3. Shift Instructions:

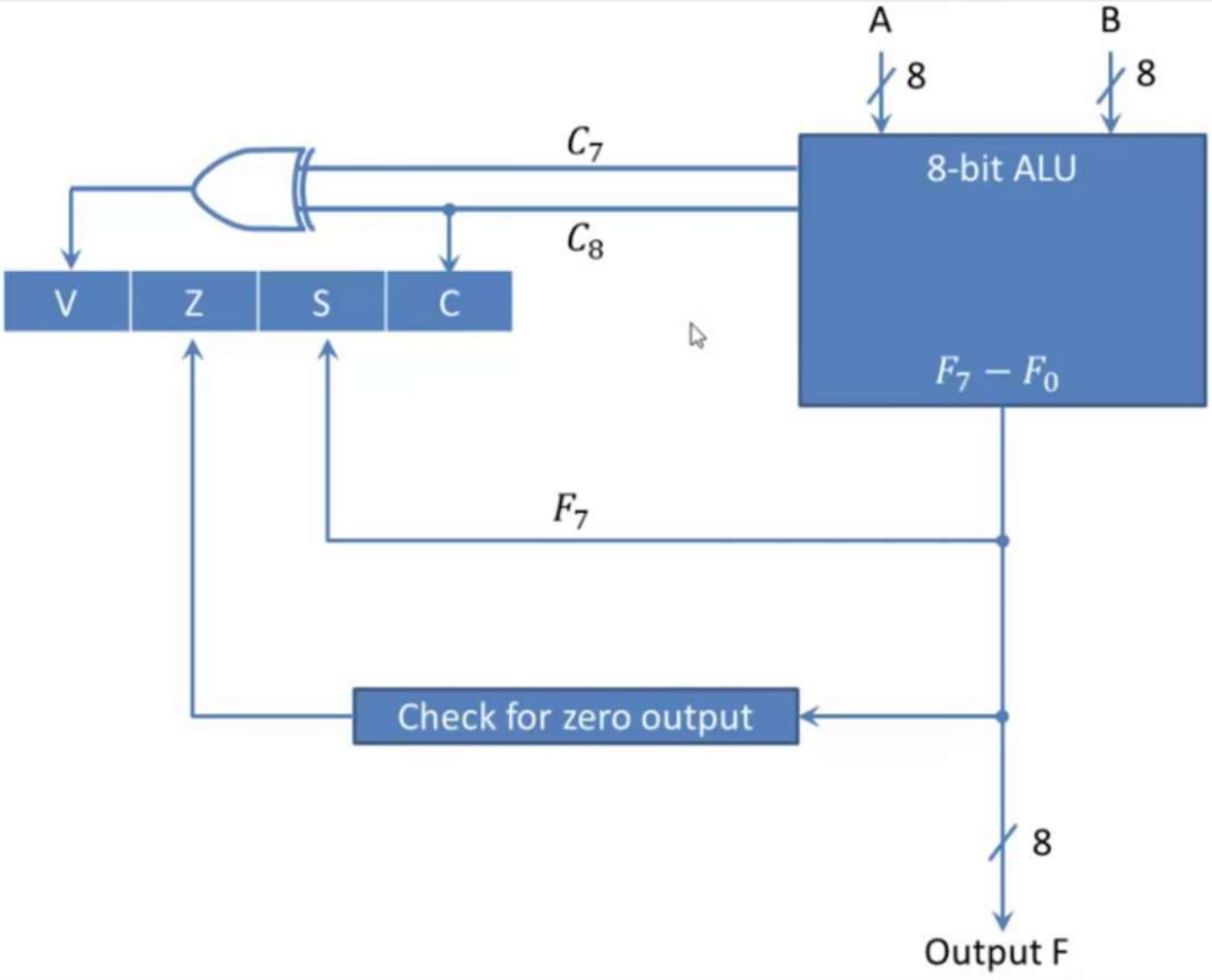
Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Program Control:

- A program control type of instruction, when executed, may change the address value in the program counter and cause the flow of control to be altered.
- The change in value of the program counter as a result of the execution of a program control instruction causes a break in the sequence of instruction execution.

Name	Mnemonic
Branch	BUN
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

Status Bit condition:



Conditional Branch Instructions:

Mnemonic	Branch Condition	Tested Condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0
Unsigned compare conditions (A – B)		
BHI	Branch if higher	A > B
BHE	Branch if higher or equal	A ≥ B
BLO	Branch if lower	A < B

Conditional Branch Instructions:

Mnemonic	Branch Condition	Tested Condition
BLOE	Branch if lower or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B
	Signed compare conditions (A – B)	
BGT	Branch if greater than	A > B
BGE	Branch if greater or equal	A ≥ B
BLT	Branch if less than	A < B
BLE	Branch if less or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B