UNIT – I I

BASIC COMPUTER ORGANIZATION & DESIGN

BASIC COMPUTER ORGANIZATION AND DESIGN

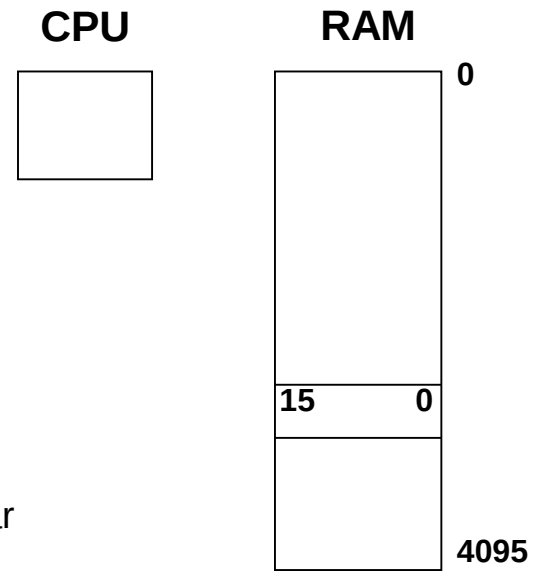
- **Instruction Codes**
- **Computer Registers**
- **Computer Instructions**
- **Timing and Control**
- **Instruction Cycle**
- **Memory Reference Instructions**
- **Input-Output and Interrupt**
- **Complete Computer Description**

INTRODUCTION

- Every different processor has its own design

  (different registers, buses, micro-operations, machine instructions, etc)

- Modern processor is a very complex device
- The Basic Computer has two components, a processor and memory
- The memory has 4096 words in it
- Each word is 16 bits long



INSTRUCTIONS

- Program
  - A sequence of (machine) instructions

- (Machine) Instruction
  - A group of bits that tell the computer to *perform a specific operation* (a sequence of micro-operation)

- The instructions of a program, along with any needed data are stored in memory

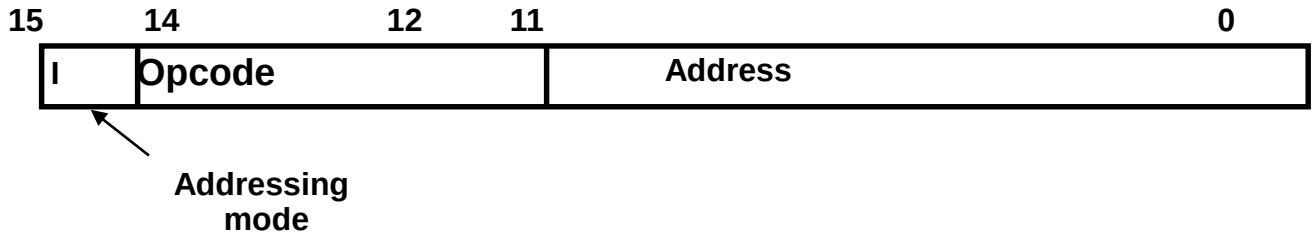
- The CPU reads the next instruction from memory
- It is placed in an *Instruction Register* (IR)

- Control circuitry in control unit then translates the instruction into the sequence of microoperations necessary to implement it

INSTRUCTION FORMAT

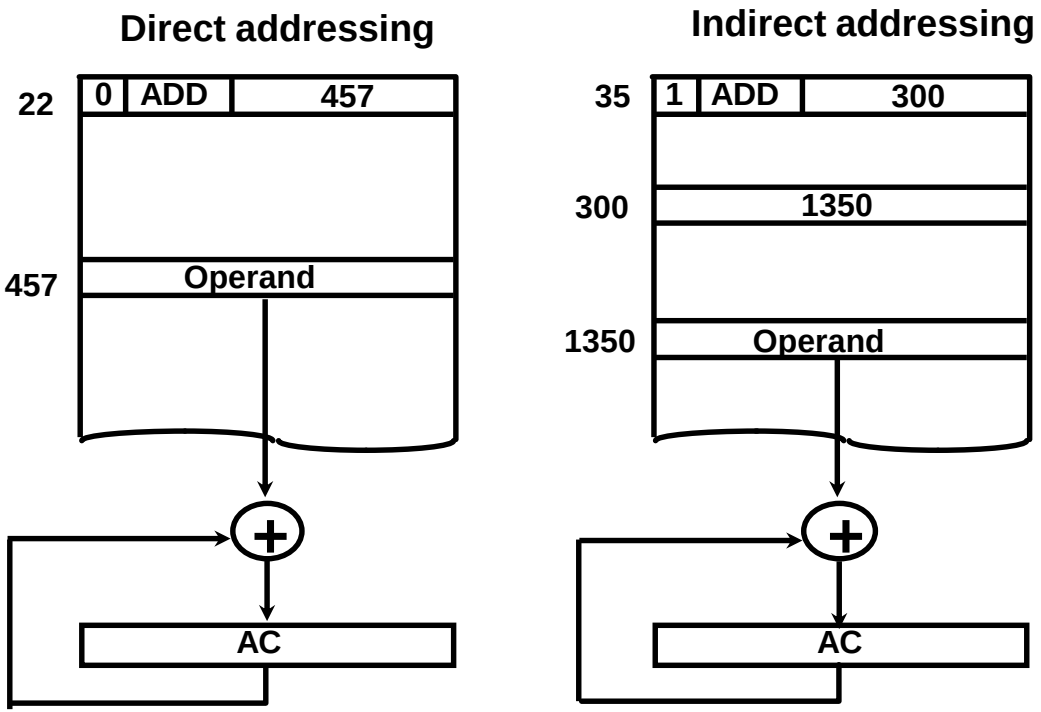
- In the Basic Computer, bit 15 of the instruction specifies the *addressing mode* (0: direct addressing, 1: indirect addressing)
- Since the memory words, and hence the instructions, are 16 bits long, that leaves 3 bits for the instruction's opcode
- Effective Address (EA)
  - The address, that can be directly used without modification to access an operand for a computation-type instruction, or as the target address for a branch-type instruction

Instruction Format



ADDRESSING MODES

- The address field of an instruction can represent either
  - Direct address: the address in memory of the data to use (the address of the operand), or
  - Indirect address: the address in memory of the address in memory of the data to use



PROCESSOR REGISTERS

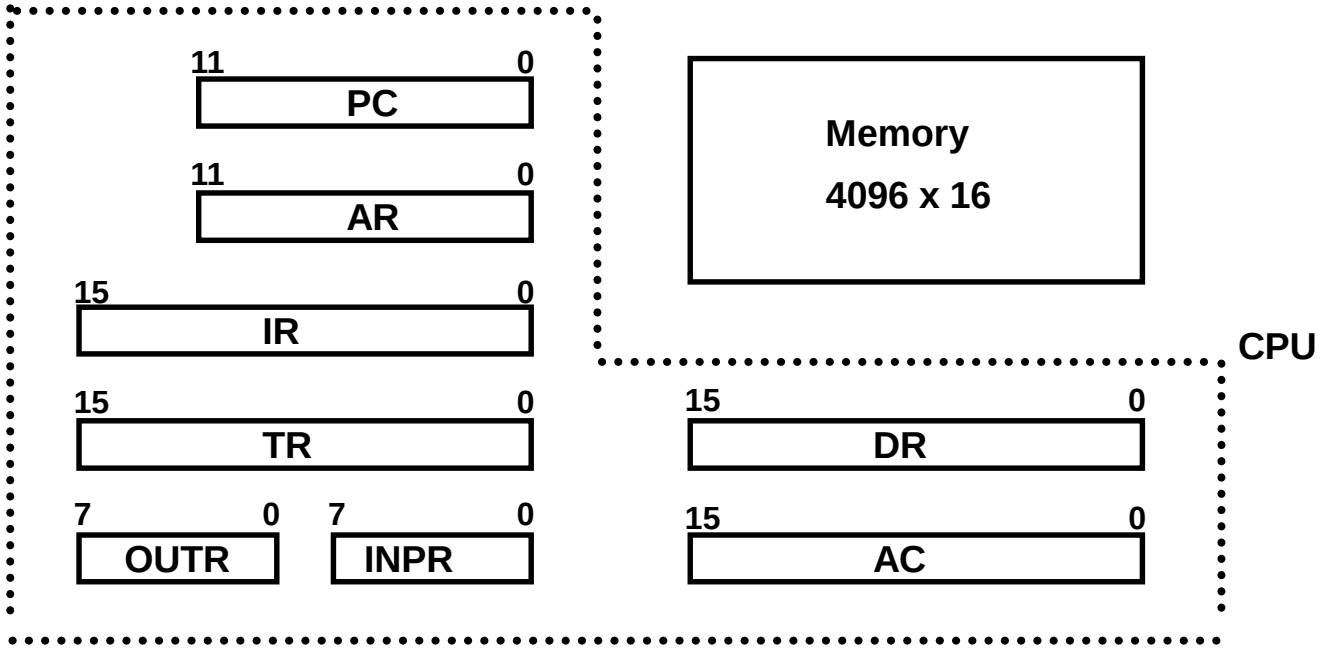
- A processor has many registers to hold instructions, addresses, data, etc
- The processor has a register, the *Program Counter* (PC) that holds the memory address of the next instruction
- In a direct or indirect addressing, the processor needs to keep track of what locations in memory it is addressing: The *Address Register* (AR) is used for this
- When an operand is found, using either direct or indirect addressing, it is placed in the *Data Register* (DR). The processor then uses this value as data for its operation
- The Basic Computer has a single *general purpose register* – the *Accumulator* (AC)

PROCESSOR REGISTERS

- Often a processor will need a scratch register to store intermediate results or other temporary data; in the Basic Computer this is the *Temporary Register* (TR)
- The Basic Computer uses a very simple model of input/output (I/O) operations
  - Input devices are considered to send 8 bits of character data to the processor
  - The processor can send 8 bits of character data to output devices
- The *Input Register* (INPR) holds an 8 bit character gotten from an input device
- The *Output Register* (OUTR) holds an 8 bit character to be send to an output device

BASIC COMPUTER REGISTERS

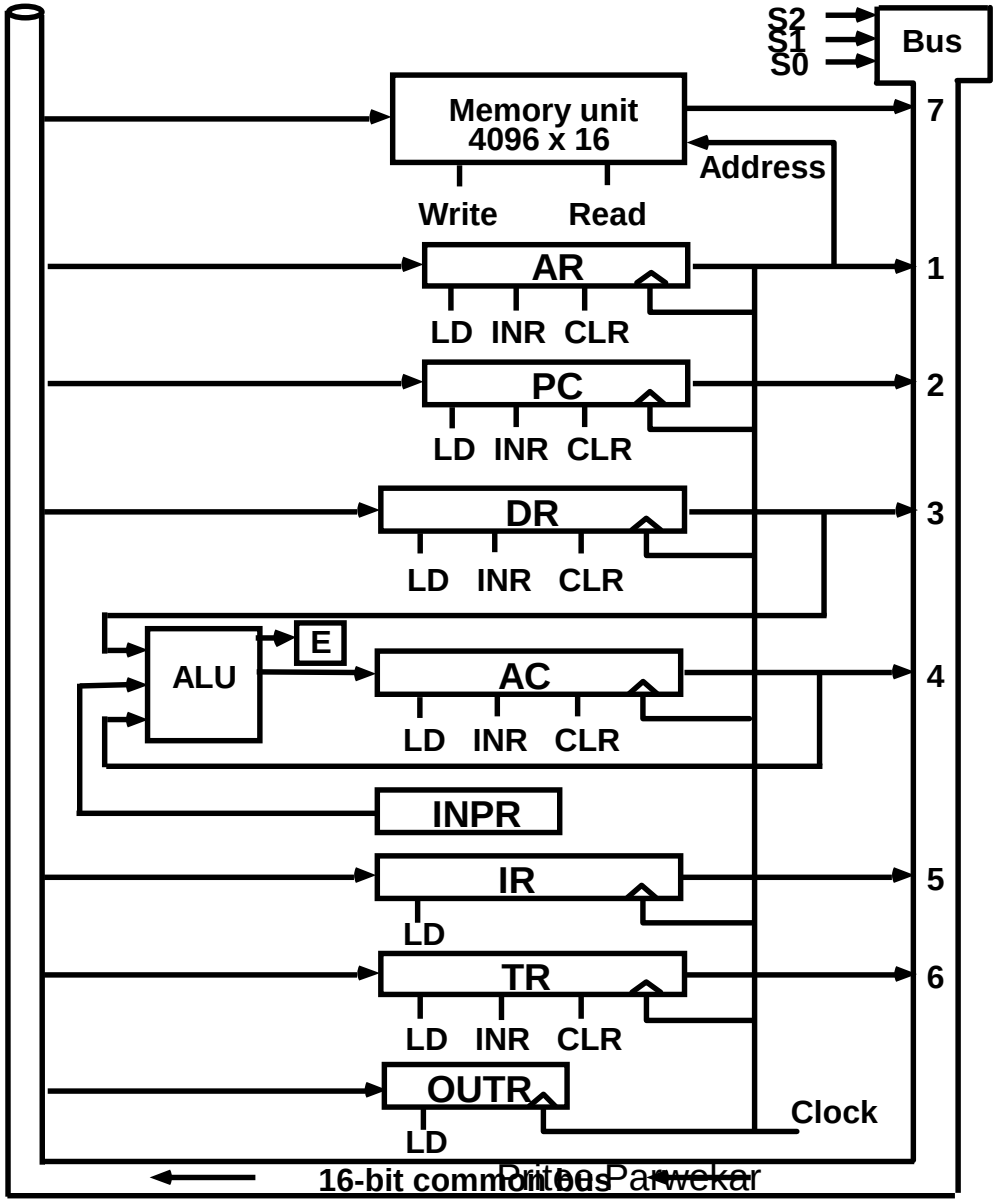
Registers in the Basic Computer



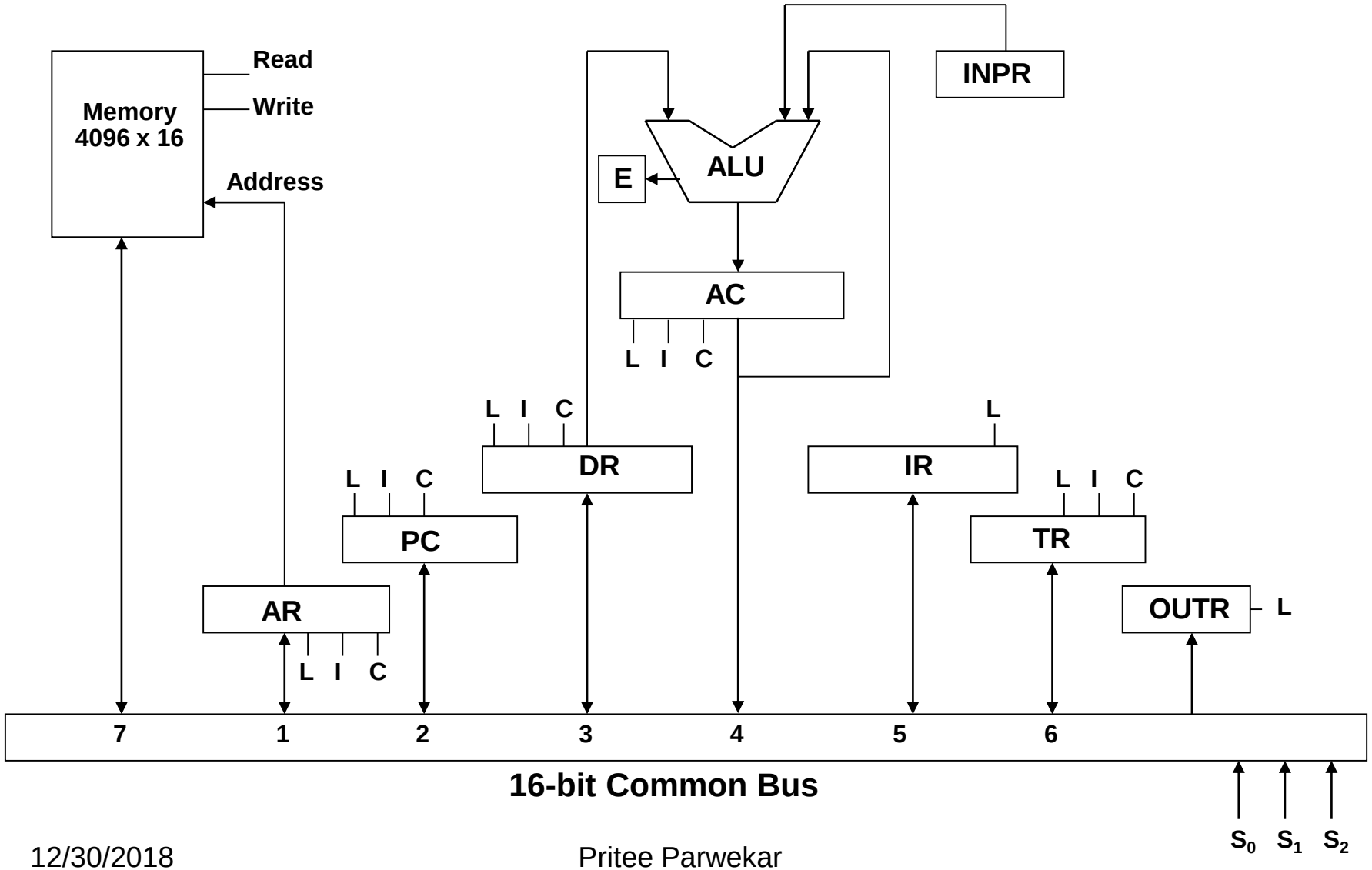
List of Registers

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

COMMON BUS SYSTEM



COMMON BUS SYSTEM



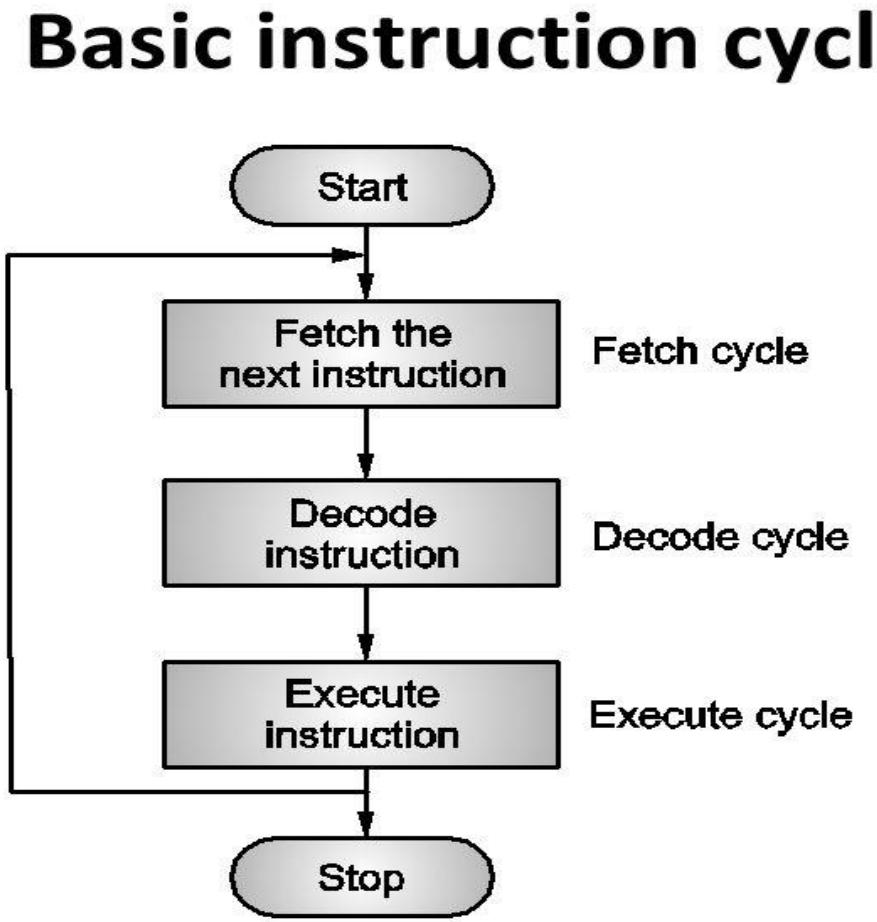
COMMON BUS SYSTEM

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

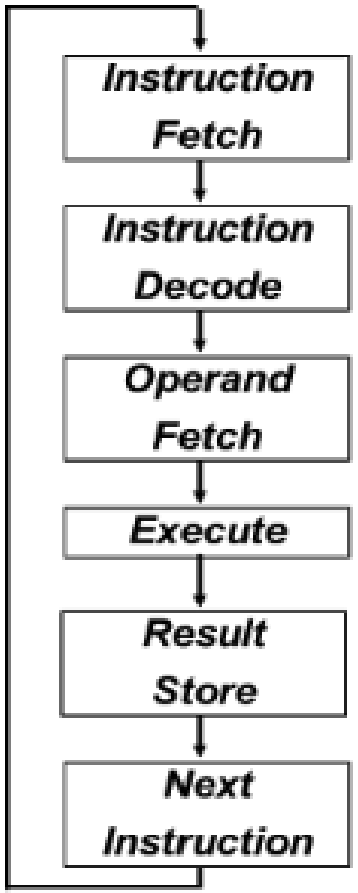
S_2	S_1	S_0	Register
0	0	0	x
0	0	1	AR
0	1	0	PC
0	1	1	DR
1	0	0	AC
1	0	1	IR
1	1	0	TR
1	1	1	Memory

- 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

INSTRUCTION CYCLE



INSTRUCTION CYCLE



Obtain instruction from program storage

Determine required actions and instruction size

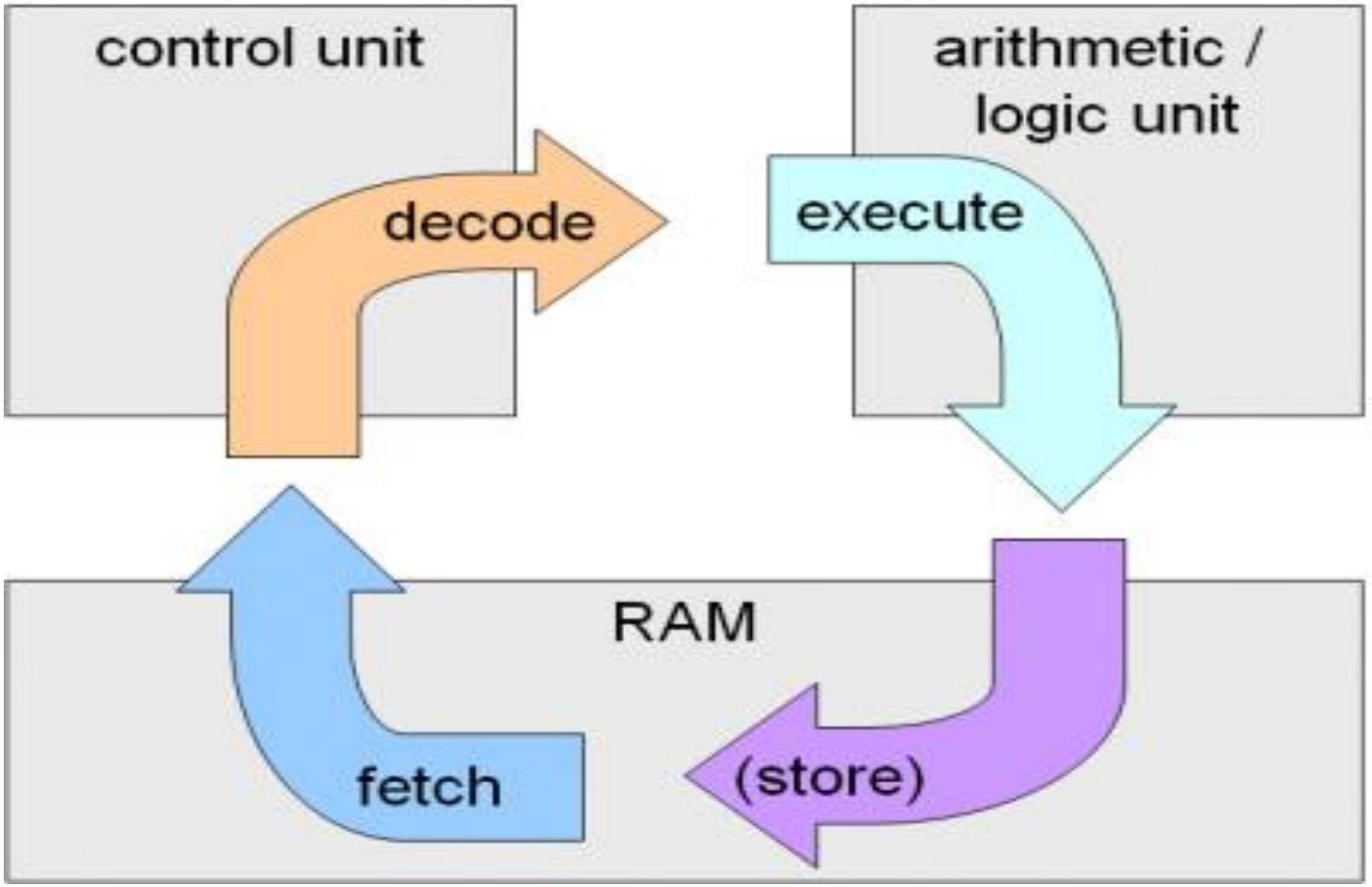
Locate and obtain operand data

Compute result value or status

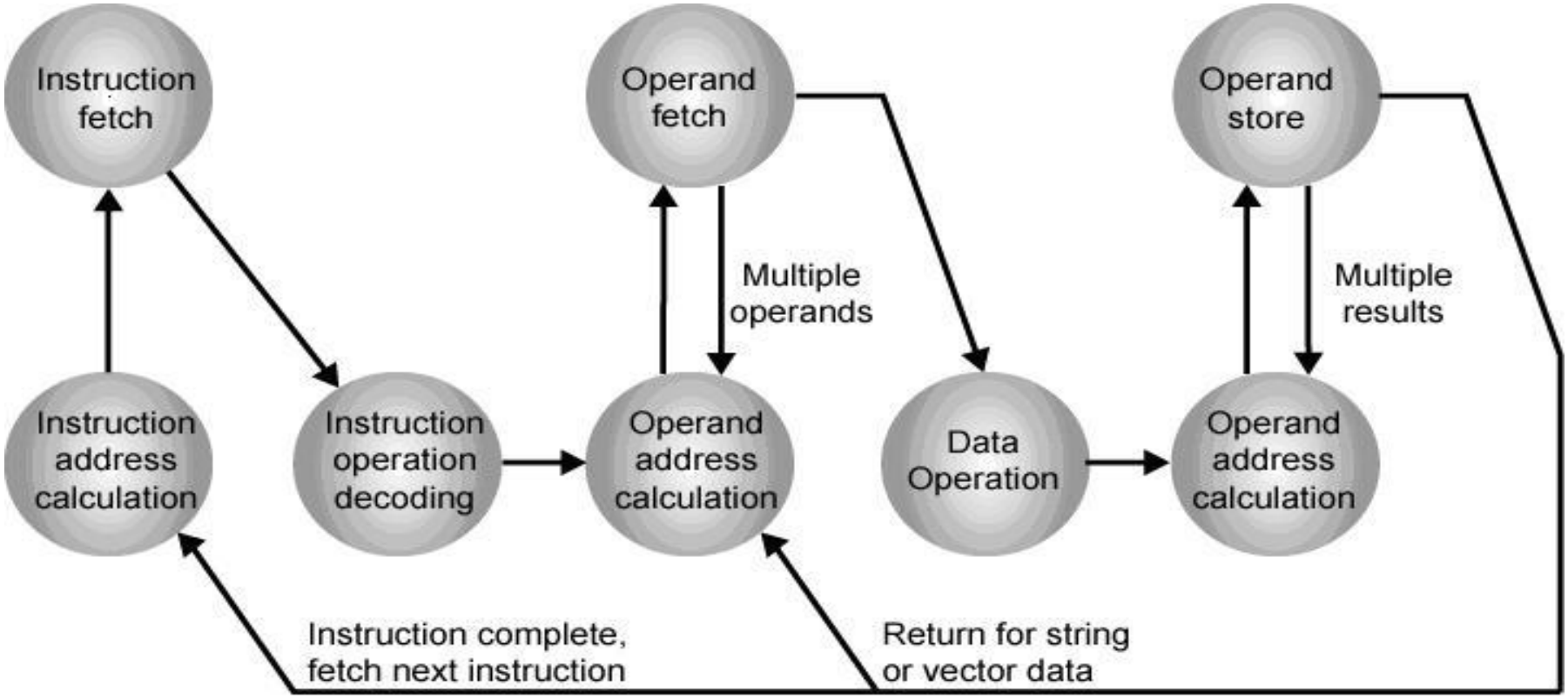
Deposit results in storage for later use

Determine successor instruction; can generally be combined w/ Decode

INSTRUCTION CYCLE



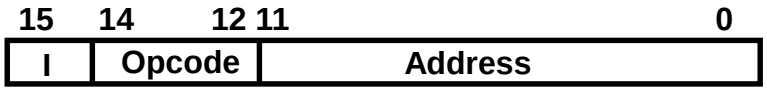
INSTRUCTION CYCLE



BASIC COMPUTER INSTRUCTIONS

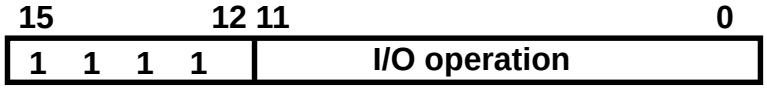
- **Basic Computer Instruction Format**

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



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

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



INSTRUCTION SET COMPLETENESS

Set of instructions using which user can construct machine language programs to evaluate any computable function.

- **Instruction Types**

  Functional Instructions
  - **Arithmetic, logic, and shift instructions**
  - **ADD, CMA, INC, CIR, CIL, AND, CLA**

  Transfer Instructions
  - **Data transfers between the main memory and the processor registers**
  - **LDA, STA**

  Control Instructions
  - **Program sequencing and control**
  - **BUN, BSA, ISZ**

  Input/Output Instructions
  - **Input and output**
  - **INP, OUT**

ADDRESS TYPES- instruction formats

A instruction is of various length depending upon the number of addresses it contain. Generally CPU organization are of three types on the basis of number of address fields:

- **Single Accumulator organization**
- **General register organization**
- **Stack organization**

ADDRESS TYPES- instruction formats

- **In first organization operation is done involving a special register called accumulator.**
- **In second on multiple registers are used for the computation purpose.**
- **In third organization the work on stack basis operation due to which it does not contain any address field.**
- **It is not necessary that only a single organization is applied a blend of various organization is mostly seen.**

ZERO ADDRESS INSTRUCTION

Stack organization Computer systems use zero address instructions.

There are two operations performed one stack – PUSH & POP.

To write the zero address instruction, a given expression is converted to revere Polish Notation i.e. Post fix Notation.

After conversion it will be easy to write zero address instructions

Example of zero address

Expression: X = (A+B)*(C+D)

Postfixed : X = AB+CD+*

TOP means top of stack

M[X] is any memory location

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

ONE ADDRESS INSTRUCTION

In one address instruction,address of one of the operand will be given along with the instruction.

This type of instruction use Accumulator as the implicit operand (use for calculation and result storage)

Example of one address

Expression: X = (A+B)*(C+D)

AC is accumulator

M[] is any memory location

M[T] is temporary location

LOAD	A	AC = M[A]
ADD	B	AC = A[C] + 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

TWO ADDRESS INSTRUCTION

Two addresses of operands will be given in the instruction.

The operands can be present in a register or in memory.

There will be know implicit storage of the result in accumulator as one address instruction.

Example of Two address

Expression: X = (A+B)*(C+D)

R1, R2 are registers

M[] is any memory location

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

THREE ADDRESS INSTRUCTIONS

In this type of instructions three addresses would be specified in the instruction.

This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

Example of Three Address Instruction

Expression: X = (A+B)*(C+D)

R1, R2 are registers

M[] is any memory location

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

Tutorial 4

- Write the code to implement following expressions on 3-, 2-, 1-, and 0- address machines

1. A = B * C + D * E
2. f = (a+b) /(c*d*e)

BASIC COMPUTER INSTRUCTIONS

Symbol	*Hex Code* I = 0	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

CONTROL UNIT

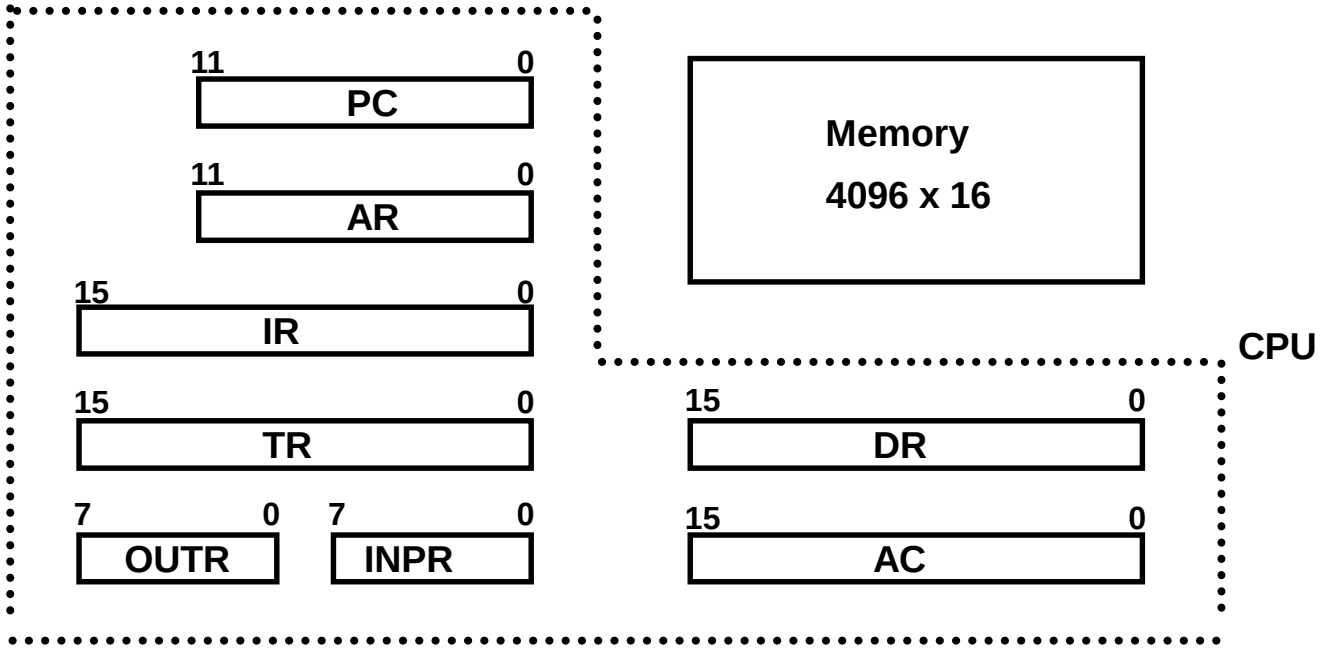
- **Control unit (CU) of a processor translates from machine instructions to the control signals (for the microoperations) that implement them**

- **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**

BASIC COMPUTER REGISTERS

Registers in the Basic Computer



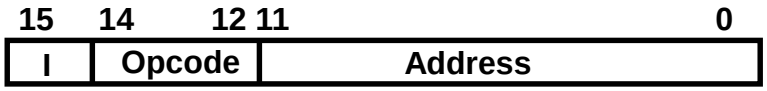
List of Registers

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

BASIC COMPUTER INSTRUCTIONS

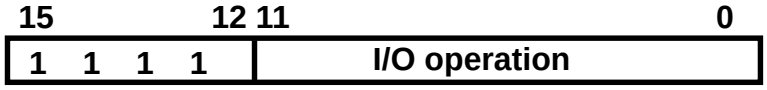
- **Basic Computer Instruction Format**

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



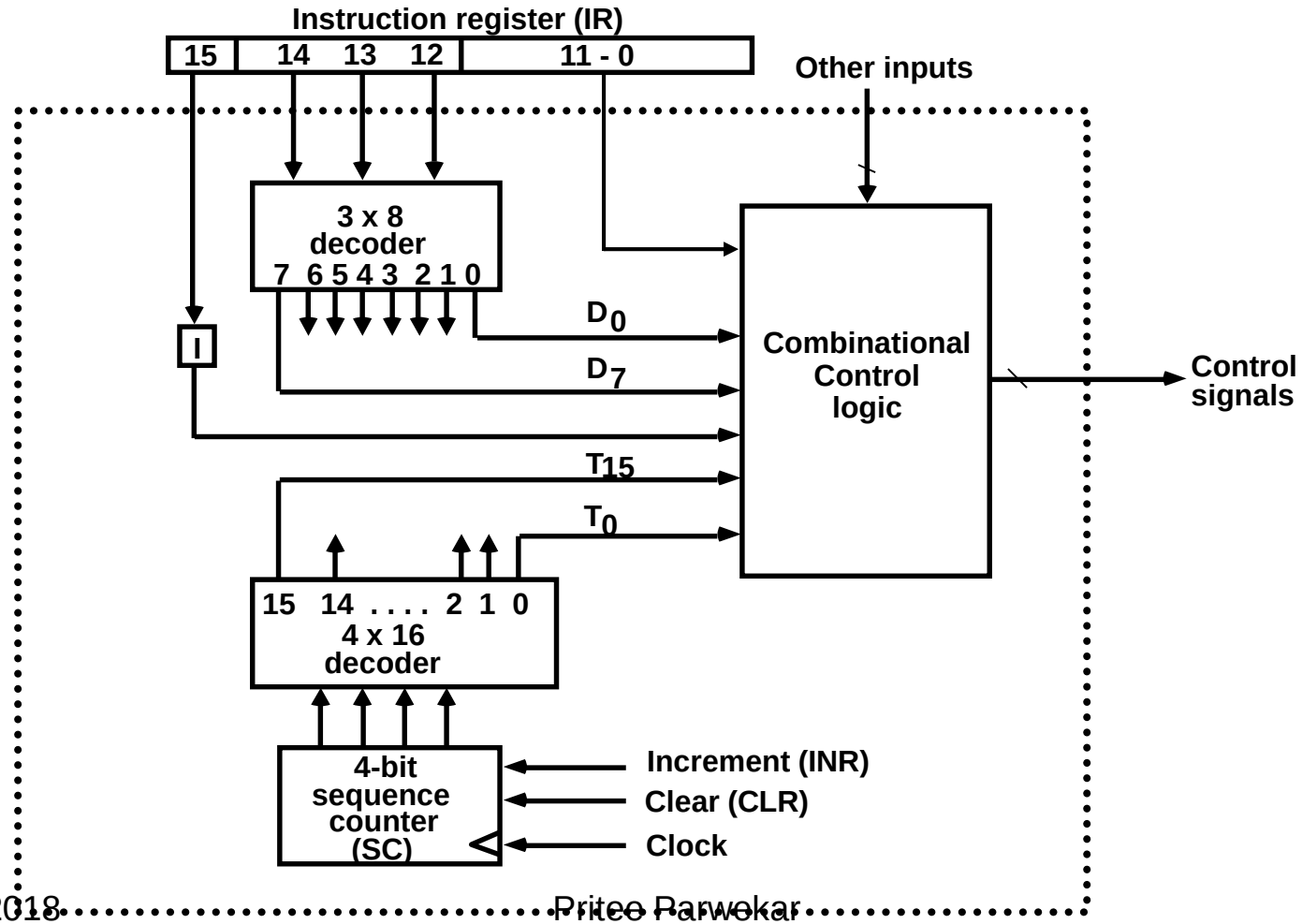
Register-Reference Instructions (OP-code = 111, I = 0)

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



TIMING AND CONTROL

Control unit of Basic Computer

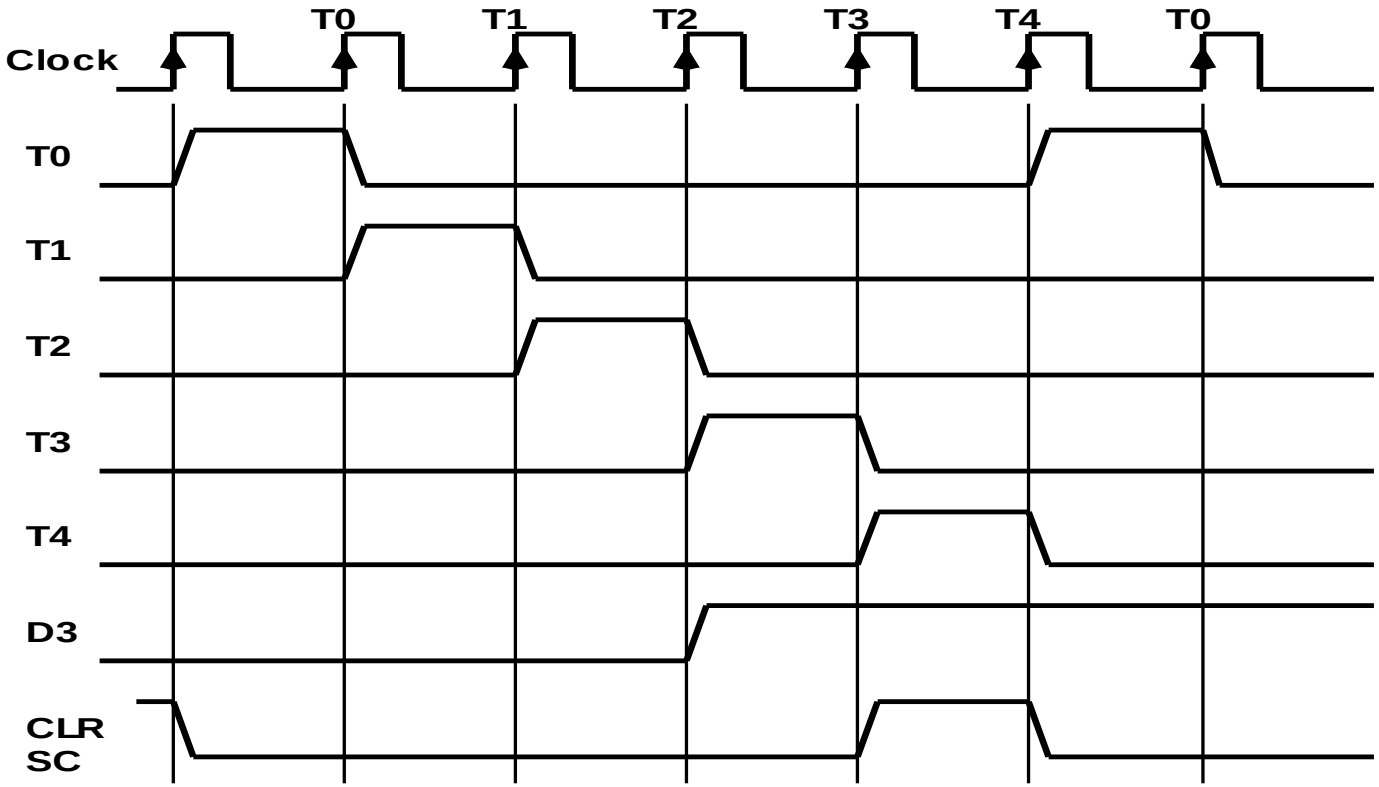


TIMING SIGNALS

- 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.

D_3T_4: SC ← 0



INSTRUCTION CYCLE

- **In Basic Computer, a machine instruction is executed in the following cycle:**
  1. **Fetch an instruction from memory**
  2. **Decode the instruction and calculate effective address (EA)**
  3. **Read the EA from memory if the instruction has an indirect address (Fetch operand)**
  1. **Execute the instruction**

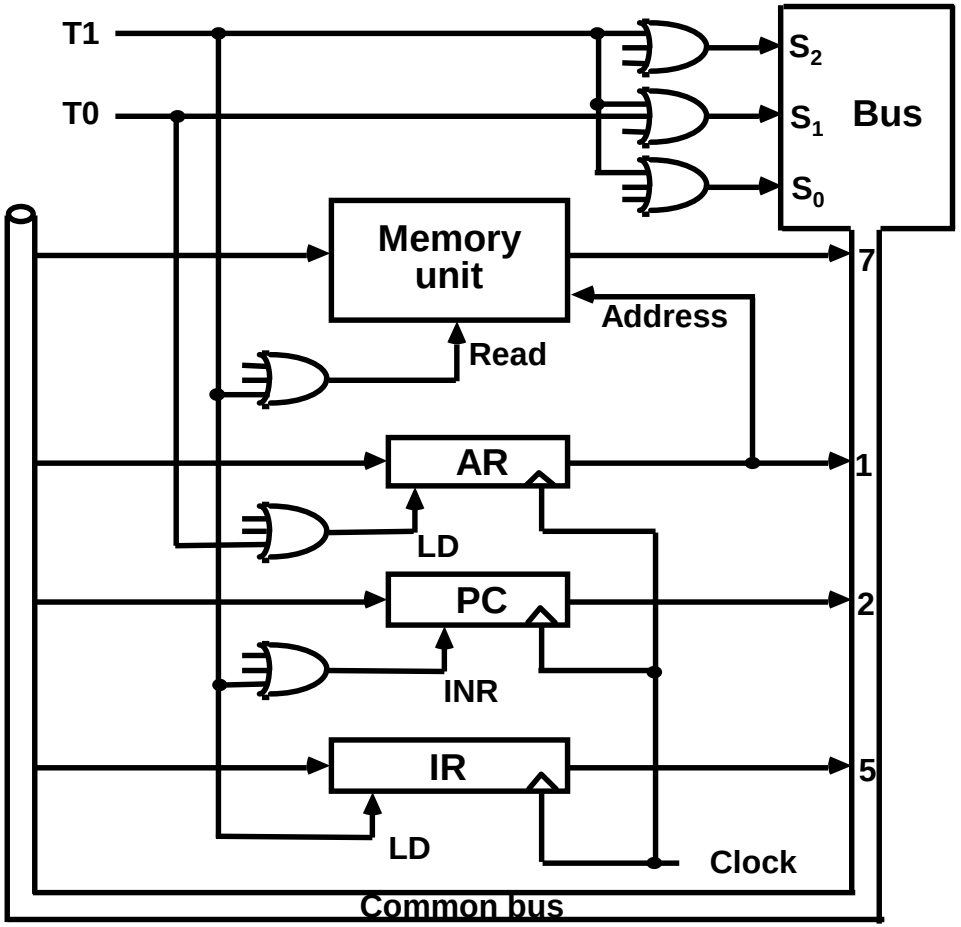
- **After an instruction is executed, the cycle starts again at step 1, for the next instruction**

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

FETCH and DECODE

- **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)



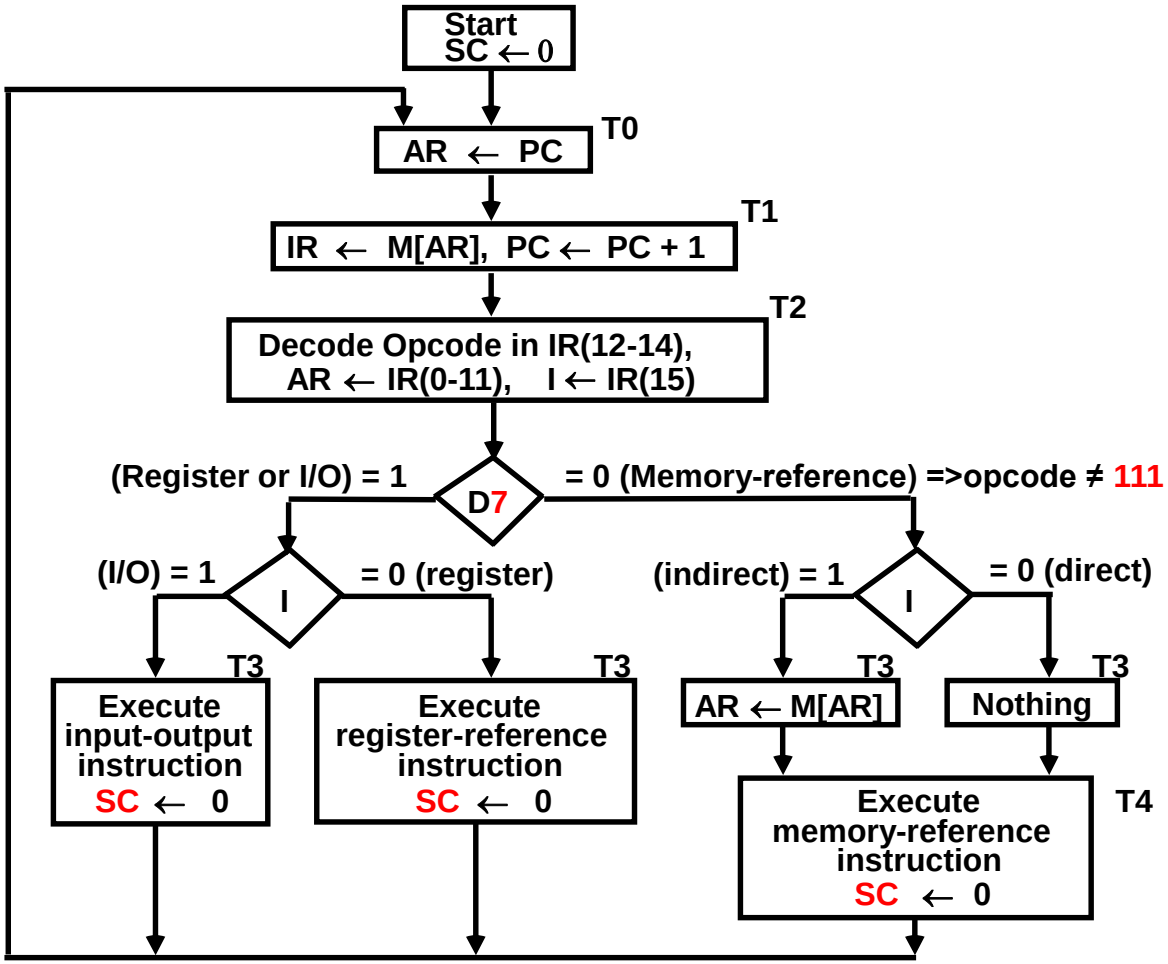
COMMON BUS SYSTEM

- **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**

DETERMINE THE 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.

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**

MEMORY REFERENCE INSTRUCTIONS

LDA: Load to AC

D_2T_4: DR ← M[AR]

D_2T_5: AC ← DR, SC ← 0

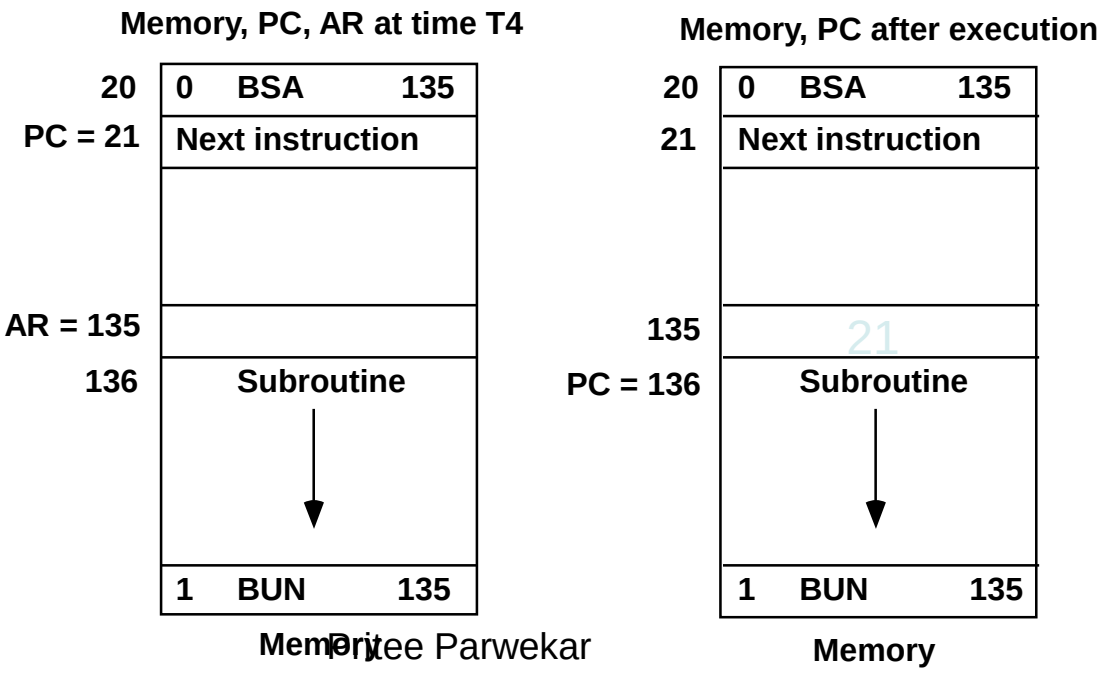
STA: Store AC

D_3T_4: M[AR] ← AC, SC ← 0

BUN: Branch Unconditionally

D_4T_4: PC ← AR, SC ← 0

BSA: Branch and Save Return Address



MEMORY REFERENCE INSTRUCTIONS

BSA:

D_5T_4: M[AR] ← PC, AR ← AR + 1

D_5T_5: PC ← AR, SC ← 0

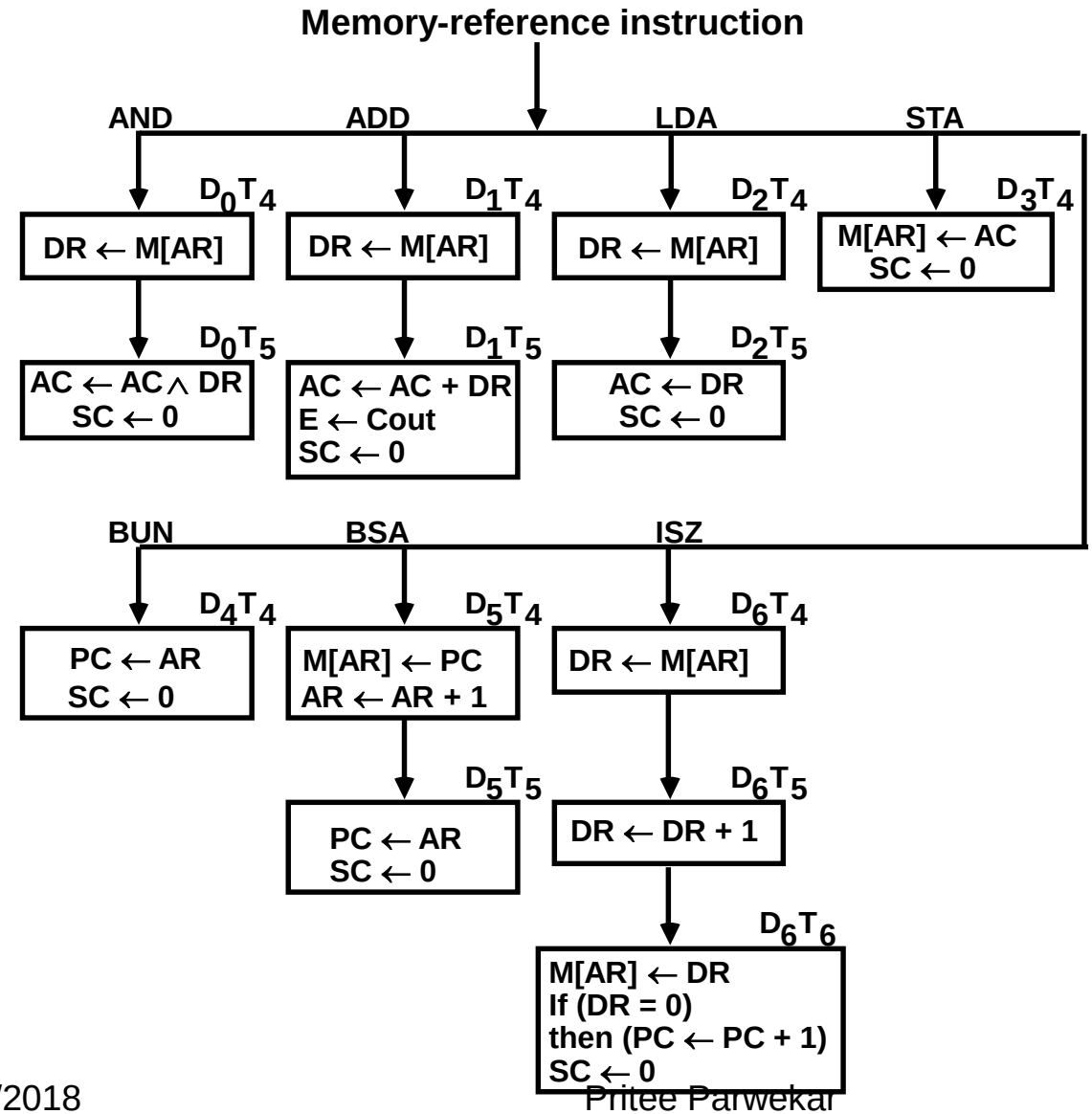
ISZ: Increment and Skip-if-Zero

D_6T_4: DR ← M[AR]

D_6T_5: DR ← DR + 1

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

FLOWCHART FOR MEMORY REFERENCE INSTRUCTIONS



REGISTER REFERENCE INSTRUCTIONS

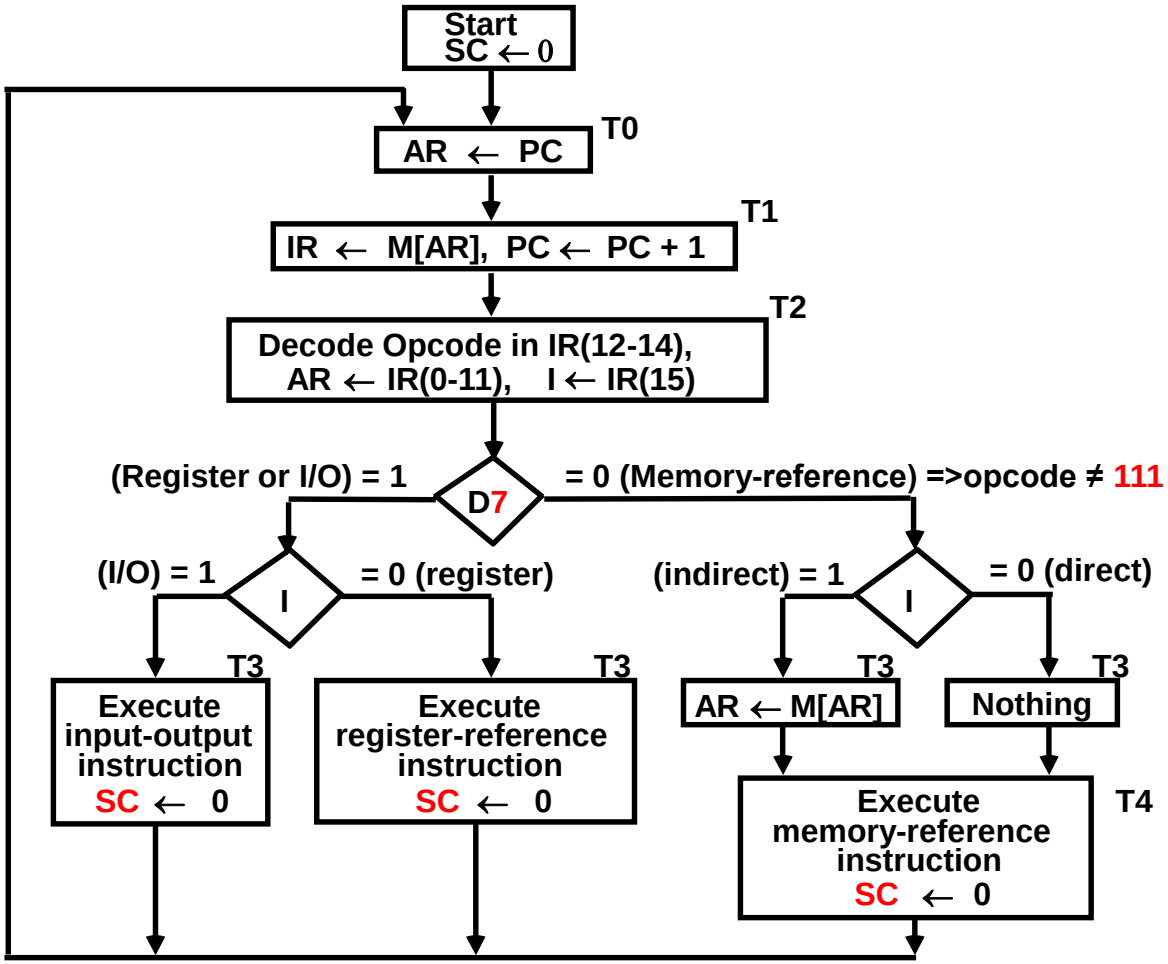
Register Reference Instructions are identified when

- **$D_7 = 1$, $I = 0$**
- **Register Ref. Instr. is specified in b_0 ~ 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)

DETERMINE THE TYPE OF INSTRUCTION

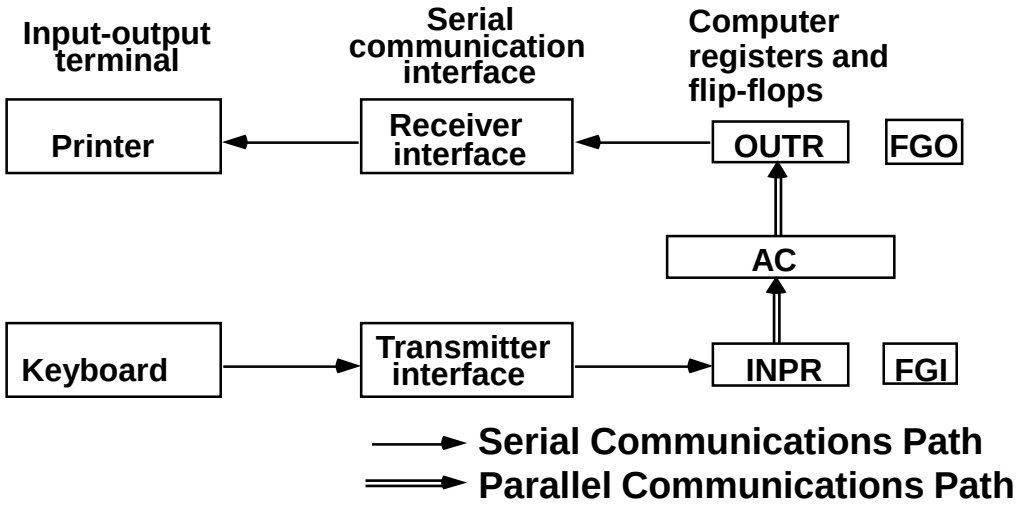


D'7IT3: AR ← M[AR]
D'7I'T3: Nothing
D7I'T3: Execute a register-reference instr.
D7IT3: Execute an input-output instr.

INPUT-OUTPUT AND 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

INPUT-OUTPUT INSTRUCTIONS

CPU Side

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

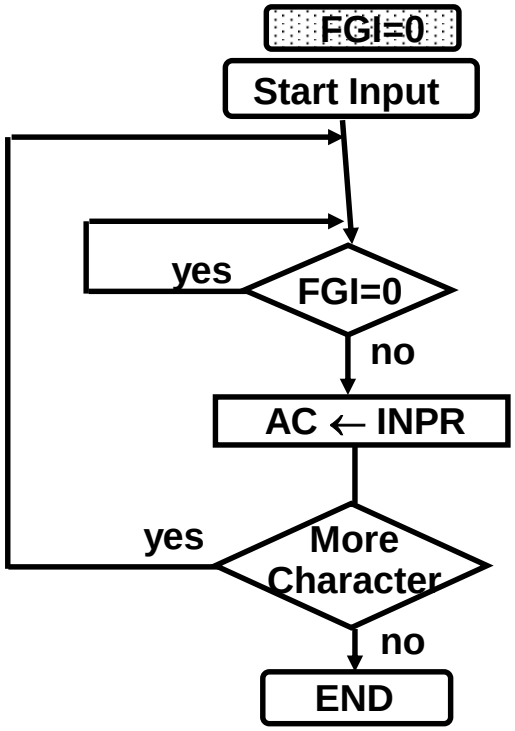
	p:	SC ← 0	Clear SC
INP	pB_{11}:	AC(0-7) ← INPR, FGI ← 0	Input char. to AC
OUT	pB_{10}:	OUTR ← AC(0-7), FGO ← 0	Output char. from AC
SKI	pB_9:	if(FGI = 1) then (PC ← PC + 1)	Skip on input flag
SKO	pB_8:	if(FGO = 1) then (PC ← PC + 1)	Skip on output flag
ION	pB_7:	IEN ← 1	Interrupt enable on
IOF	pB_6: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

PROGRAM CONTROLLED DATA TRANSFER

-- CPU --

loop: If FGI = 0 goto loop
AC ← INPR, FGI ← 0

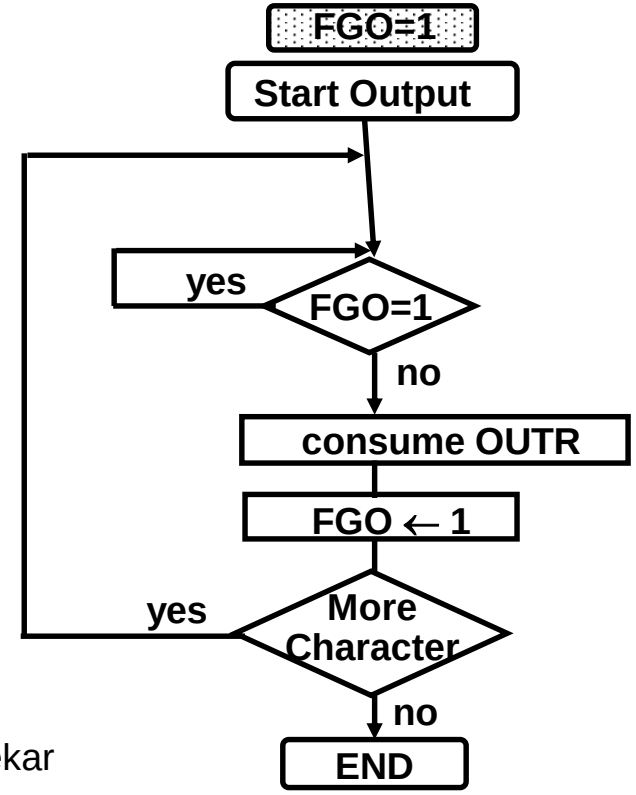
/* Output */ /* Initially FGO = 1 */
loop: If FGO = 0 goto loop
OUTR ← AC, FGO ← 0



-- I/O Device --

/* Input */ /* Initially FGI = 0 */
loop: If FGI = 1 goto loop
INPR ← new data, FGI ← 1

loop: If FGO = 1 goto loop
consume OUTR, FGO ← 1



INTERRUPT INITIATED INPUT/OUTPUT

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

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

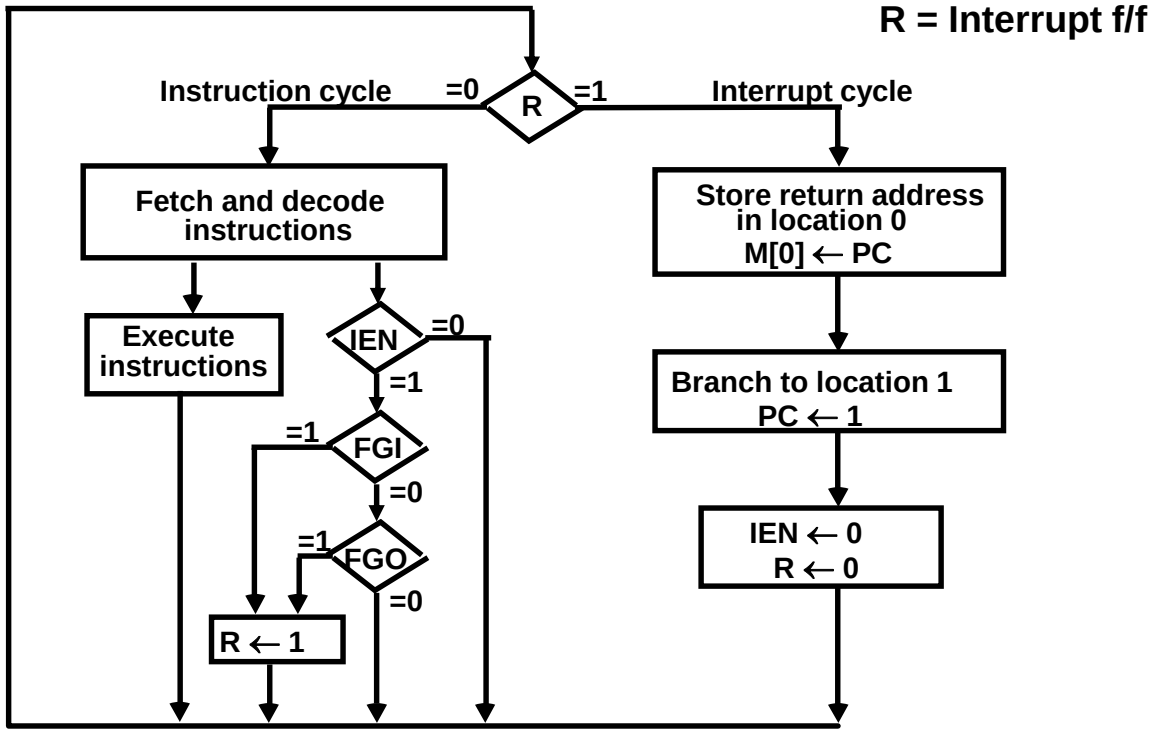
- When the interface founds that the I/O device is ready for data transfer, it generates an interrupt request to the CPU

- 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)**

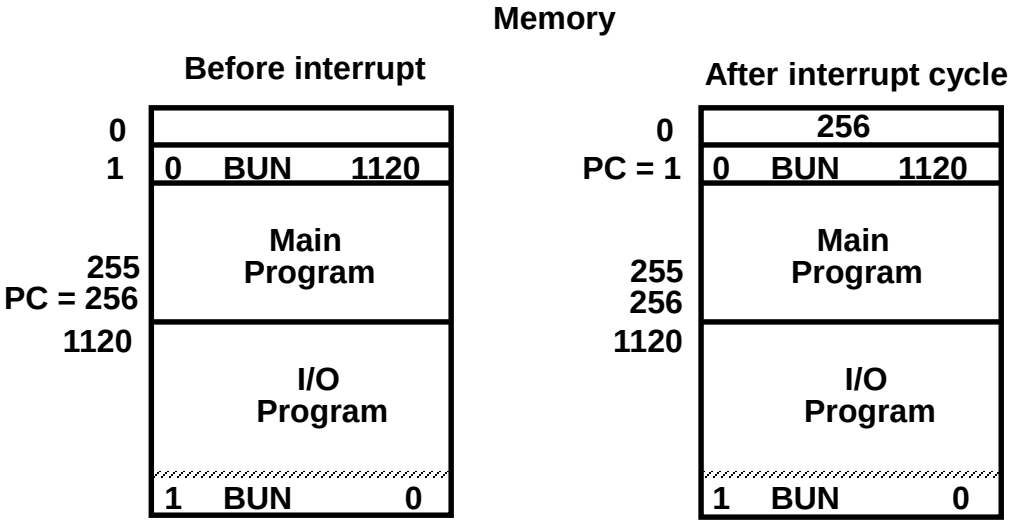
- can be set and cleared by instructions
- when cleared, the computer cannot be interrupted

FLOWCHART FOR 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"

REGISTER TRANSFER OPERATIONS 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 :

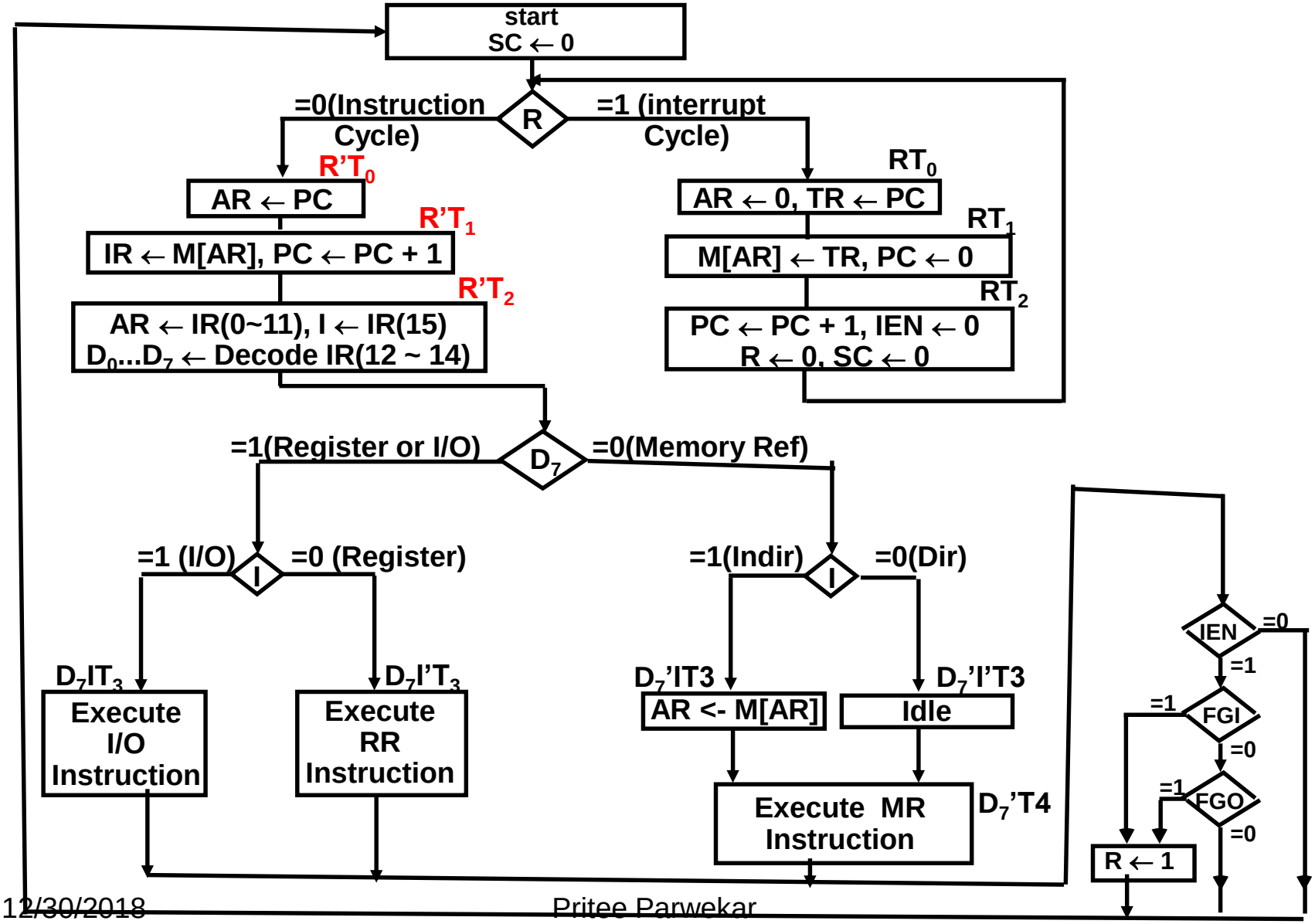
  RT_0: AR ← 0, TR ← PC

  RT_1: M[AR] ← TR, PC ← 0

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

COMPLETE COMPUTER DESCRIPTION

Flowchart of Operations



COMPLETE COMPUTER DESCRIPTION

Microoperations

Fetch	**$R'T_0$:**	**$AR \leftarrow PC$**
	$R'T_1$:	**$IR \leftarrow M[AR], PC \leftarrow PC + 1$**
Decode	**$R'T_2$:**	**$D0, ..., D7 \leftarrow$ Decode $IR(12 \sim 14)$, $AR \leftarrow IR(0 \sim 11), I \leftarrow IR(15)$**
Indirect	**$D_7'IT_3$:**	**$AR \leftarrow M[AR]$**
Interrupt		
$T_0'T_1'T_2'(IEN)(FGI + FGO)$:		**$R \leftarrow 1$**
	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-Reference		
AND	**D_0T_4:**	**$DR \leftarrow M[AR]$**
	D_0T_5:	**$AC \leftarrow AC \wedge DR, SC \leftarrow 0$**
ADD	**D_1T_4:**	**$DR \leftarrow M[AR]$**
	D_1T_5:	**$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$**
LDA	**D_2T_4:**	**$DR \leftarrow M[AR]$**
	D_2T_5:	**$AC \leftarrow DR, SC \leftarrow 0$**
STA	**D_3T_4:**	**$M[AR] \leftarrow AC, SC \leftarrow 0$**
BUN	**D_4T_4:**	**$PC \leftarrow AR, SC \leftarrow 0$**
BSA	**D_5T_4:**	**$M[AR] \leftarrow PC, AR \leftarrow AR + 1$**
	D_5T_5:	**$PC \leftarrow AR, SC \leftarrow 0$**
ISZ	**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 ($PC \leftarrow PC + 1$), $SC \leftarrow 0$**

COMPLETE COMPUTER DESCRIPTION

Microoperations

Register-Reference		
	$D_7I'T_3 = r$	**(Common to all register-reference instr)**
	$IR(i) = B_i$	**(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**
Input-Output	**$D_7IT_3 = p$**	**(Common to all input-output instructions)**
	$IR(i) = B_i$	**(i = 6,7,8,9,10,11)**
	p:	**SC ← 0**
INP	**pB_{11}:**	**AC(0-7) ← INPR, FGI ← 0**
OUT	**pB_{10}:**	**OUTR ← AC(0-7), FGO ← 0**
SKI	**pB_9:**	**If(FGI=1) then (PC ← PC + 1)**
SKO	**pB_8:**	**If(FGO=1) then (PC ← PC + 1)**
ION	**pB_7:**	**IEN ← 1**
IOF	**pB_6:**	**IEN ← 0**

End of UNIT 2