Om Amriteshwaryai Namaha

Amrita Vishwa Vidyapeetham

Computer Organization and Architecture

Rajesh Kannan
Faculty, ECE
Amrita University
rajeshm@am.amrita.edu

I/O Organization

Rajesh Kannan
Faculty, ECE
Amrita University
rajeshm@amritapuri.amrita.edu

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

- Design of Basic Computer

- Design of Accumulator Logic

Amrita Vishwa Vidyapeetham

1/27/2010

INTRODUCTION

- Every different processor type has its own design (different registers, buses, microoperations, machine instructions, etc)
- Modern processor is a very complex device
- It contains
  - Many registers
  - Multiple arithmetic units, for both integer and floating point calculations
  - The ability to pipeline several consecutive instructions to speed execution
  - Etc.
- However, to understand how processors work, we will start with a simplified processor model
- This is similar to what real processors were like ~25 years ago
- M. Morris Mano introduces a simple processor model he calls the *Basic Computer*
- We will use this to introduce processor organization and the relationship of the RTL model to the higher level computer processor

THE BASIC COMPUTER

- **The Basic Computer has two components, a processor and memory**
- **The memory has 4096 words in it**
  - **$4096 = 2^{12}$, so it takes 12 bits to select a word in memory**
- **Each word is 16 bits long**

CPU   **RAM**

0

15   0

4095
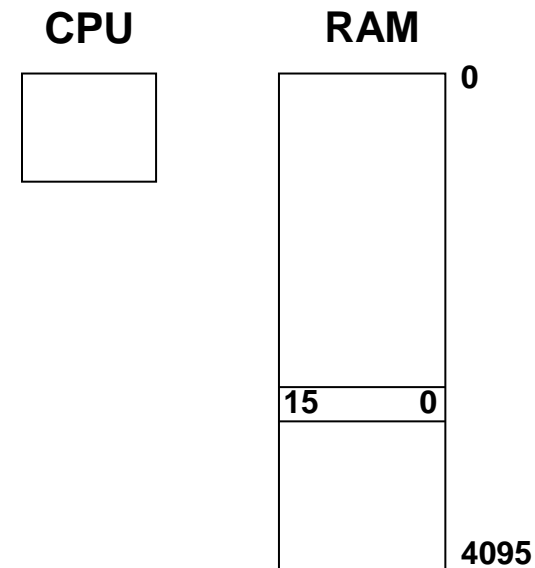
INSTRUCTIONS

- **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**

Amrita Vishwa Vidyapeetham

1/27/2010

INSTRUCTION FORMAT

- **A computer instruction is often divided into two parts**
  - **An *opcode* (Operation Code) that specifies the operation for that instruction**
  - **An *address* that specifies the registers and/or locations in memory to use for that operation**
- **In the Basic Computer, since the memory contains 4096 (= 2^{12}) words, we needs 12 bit to specify which memory address this instruction will use**
- **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**

Instruction Format

15	14	12	11		0
I	Opcode		Address		

Addressing mode

Amrita Vishwa Vidyapeetham
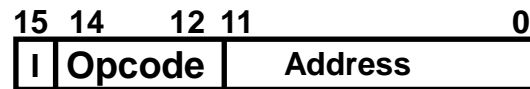
ADDRESSING MODES

- 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

Direct addressing

| 22 | 0 | ADD | 457 |
| 457 | | Operand | |

Indirect addressing

35	1	ADD	300
300		1350	
1350		Operand	

AC

AC
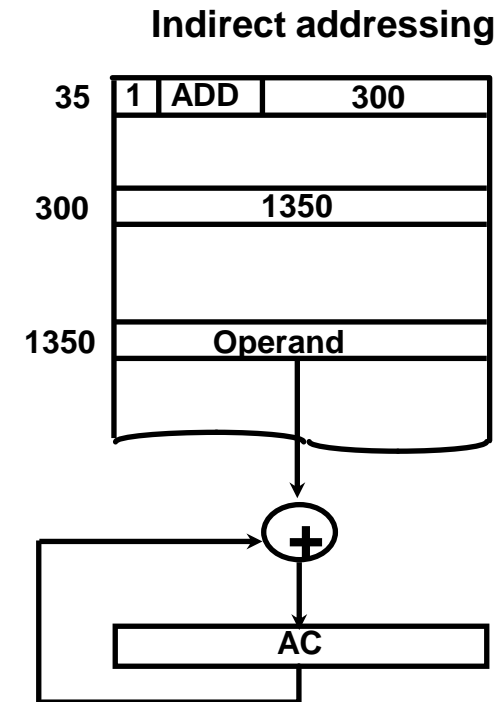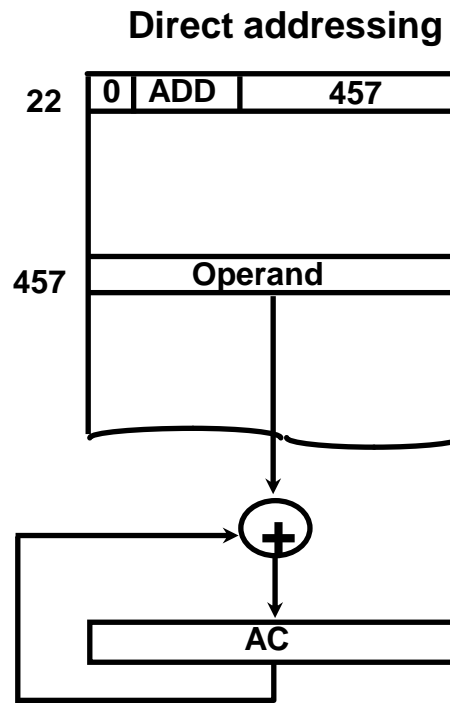
- Effective Address (EA)
  - The address, that can be directly used without modification to access an operand for a computation-type instruction, or as the target address for a branch-type instruction

1/27/2010

Amrita Vishwa Vidyapeetham

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 to get
  - Since the memory in the Basic Computer only has 4096 locations, the PC only needs 12 bits

- 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
  - The AR is a 12 bit register in the Basic Computer

- 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)
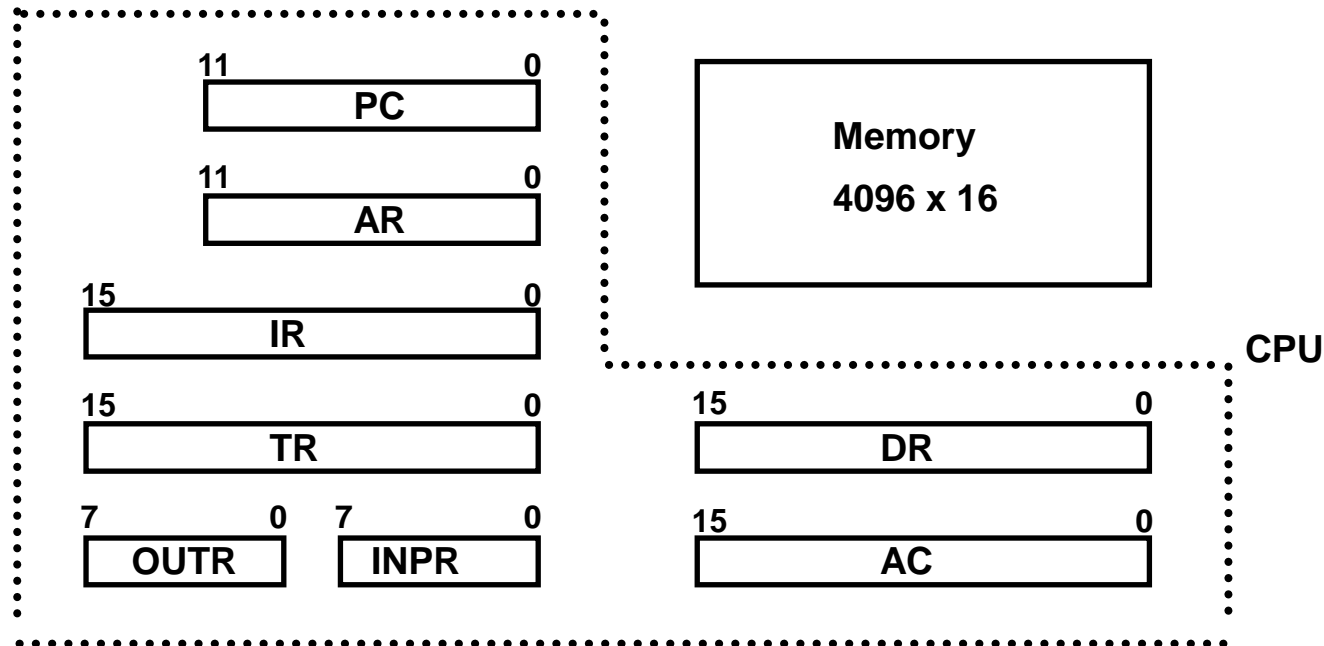
PROCESSOR REGISTERS

- The significance of a general purpose register is that it can be referred to in instructions
  - e.g. load AC with the contents of a specific memory location; store the contents of AC into a specified memory location

- 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 very simple model 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 BC Registers

DR	16-bits	Data Register	Holds memory operand
AR	12-bits	Address Register	Holds address for memory
AC	16-bits	Accumulator	Processor register
IR	16-bits	Instruction Register	Holds instruction code
PC	12-bits	Program Counter	Holds address of instruction
TR	16-bits	Temporary Register	Holds temporary data
INPR	8-bits	Input Register	Holds input character
OUTR	8-bits	Output Register	Holds output character

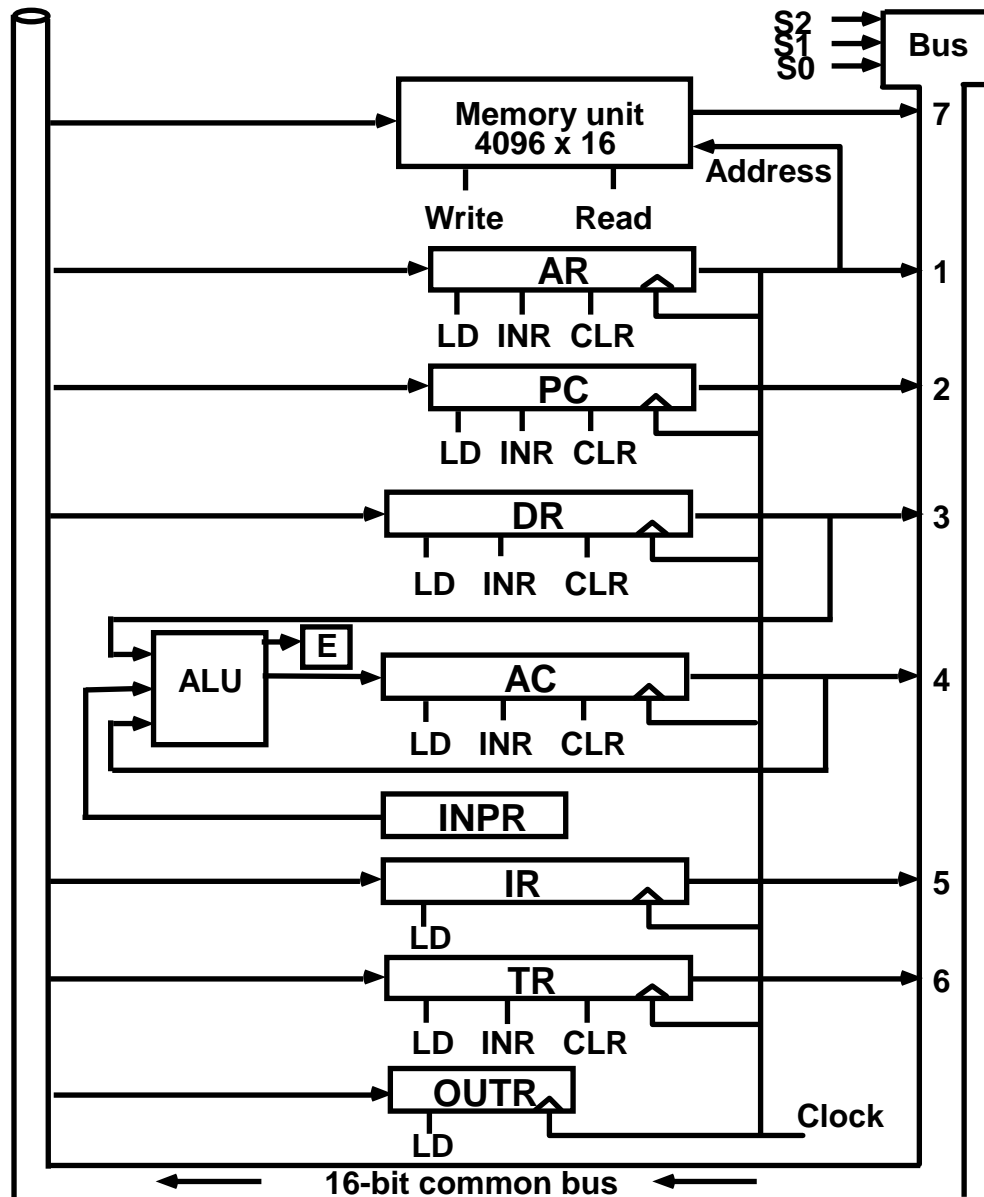1/27/2010

Amrita Vishwa Vidyapeetham

COMMON BUS SYSTEM

- **The registers in the Basic Computer are connected using a bus**

- **This gives a savings in circuitry over complete connections between registers**

COMMON BUS SYSTEM

1/27/2010

Amrita Vishwa Vidyapeetham
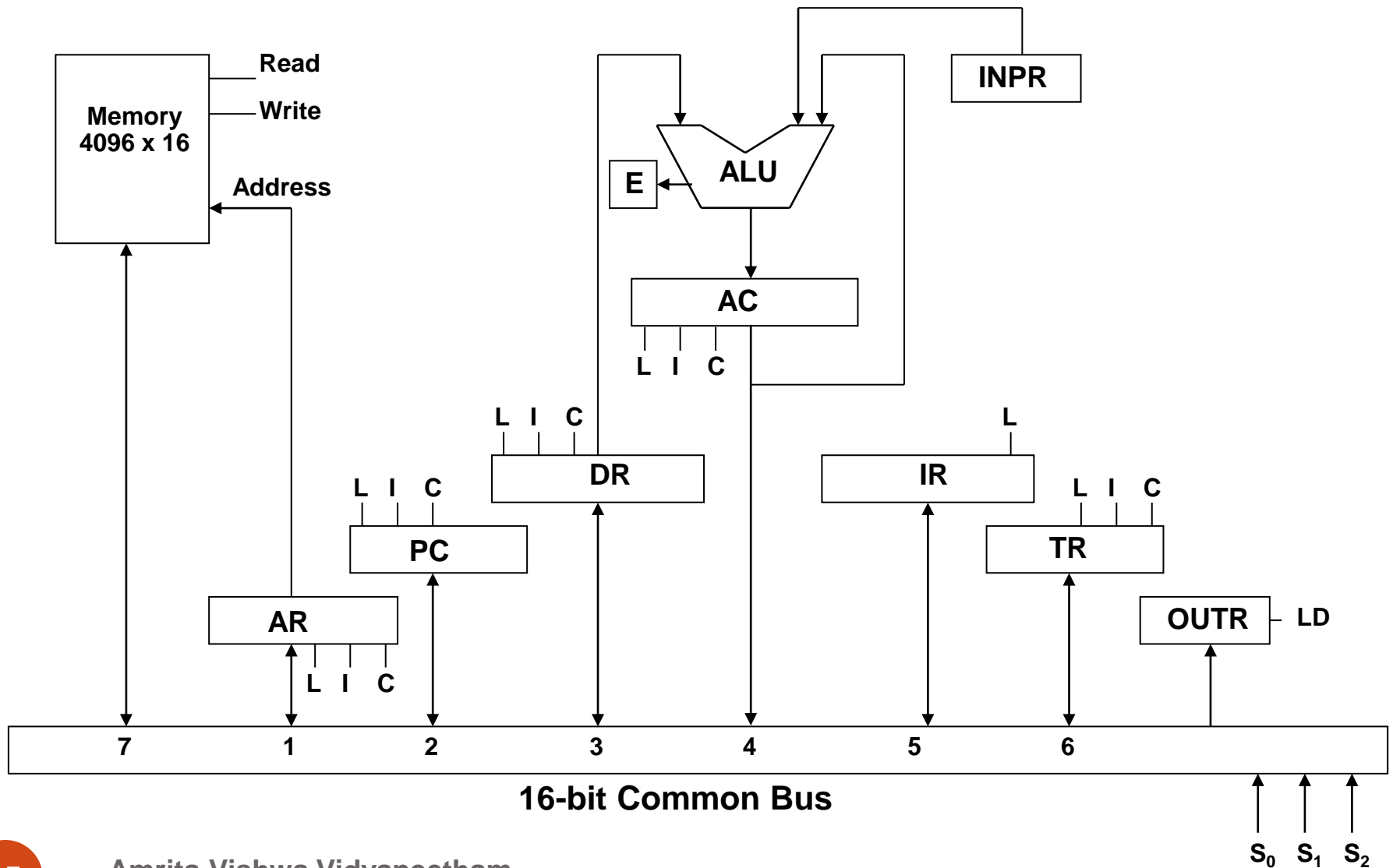
COMMON BUS SYSTEM

- 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

Amrita Vishwa Vidyapeetham

BASIC COMPUTER INSTRUCTIONS

• **Basic Computer Instruction Format**

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

15	14	12 11		0
I	Opcode		Address	

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

15			12 11		0
0	1	1	1	Register operation	

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

15			12 11		0
1	1	1	1	I/O operation	
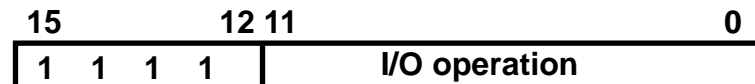
BASIC COMPUTER INSTRUCTIONS

Symbol	Hex Code		Description
	I = 0	I = 1	
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load 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

18

INSTRUCTION SET COMPLETENESS

A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function that is known to be computable.

- 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

Amrita Vishwa Vidyapeetham

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**

Control unit of Basic Computer
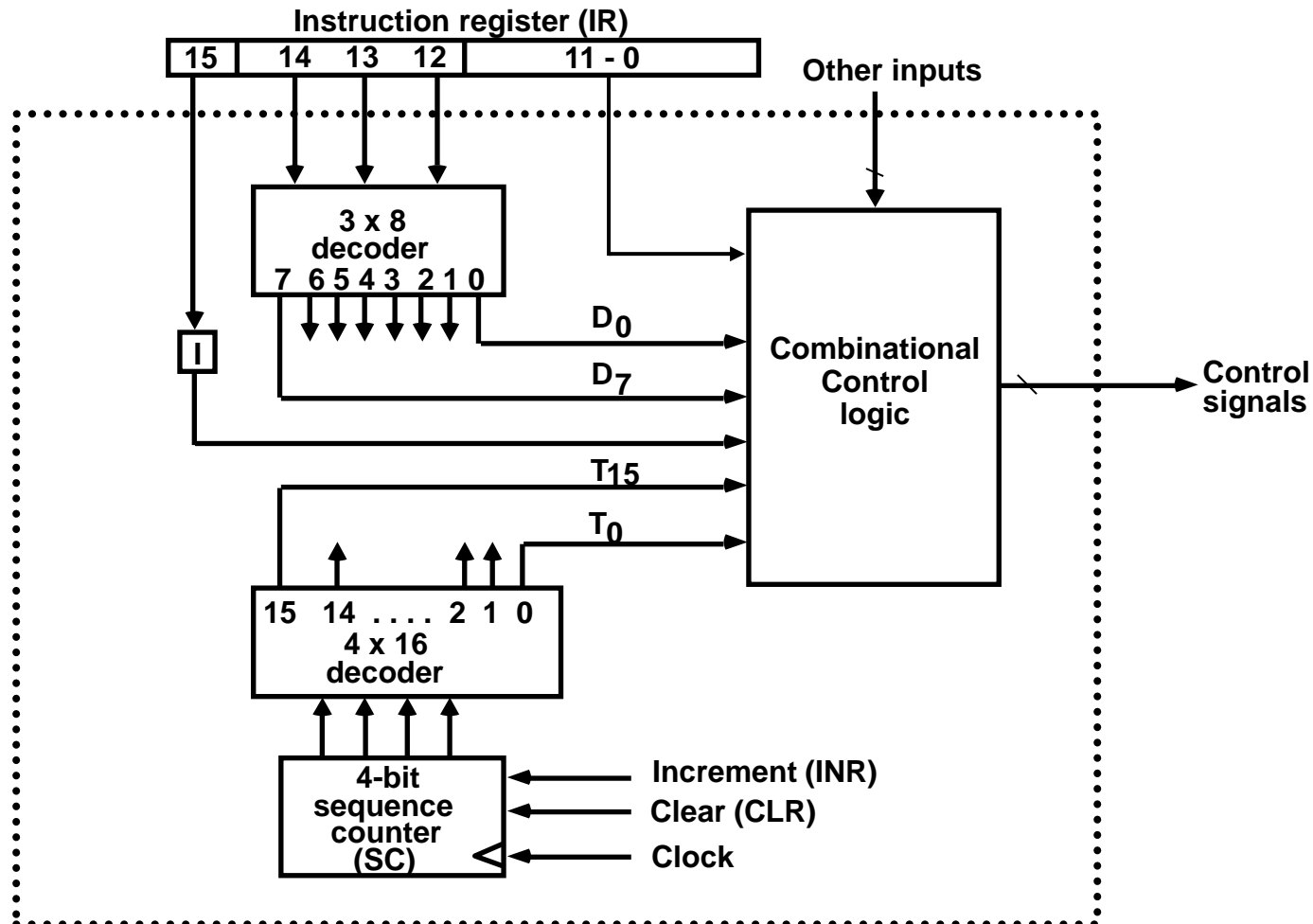
TIMING SIGNALS

- 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, . . .
    Assume: At time T_4, SC is cleared to 0 if decoder output D3 is active.

$$D_3 T_4: SC \leftarrow 0$$

Amrita Vishwa Vidyapeetham   **1/27/2010**
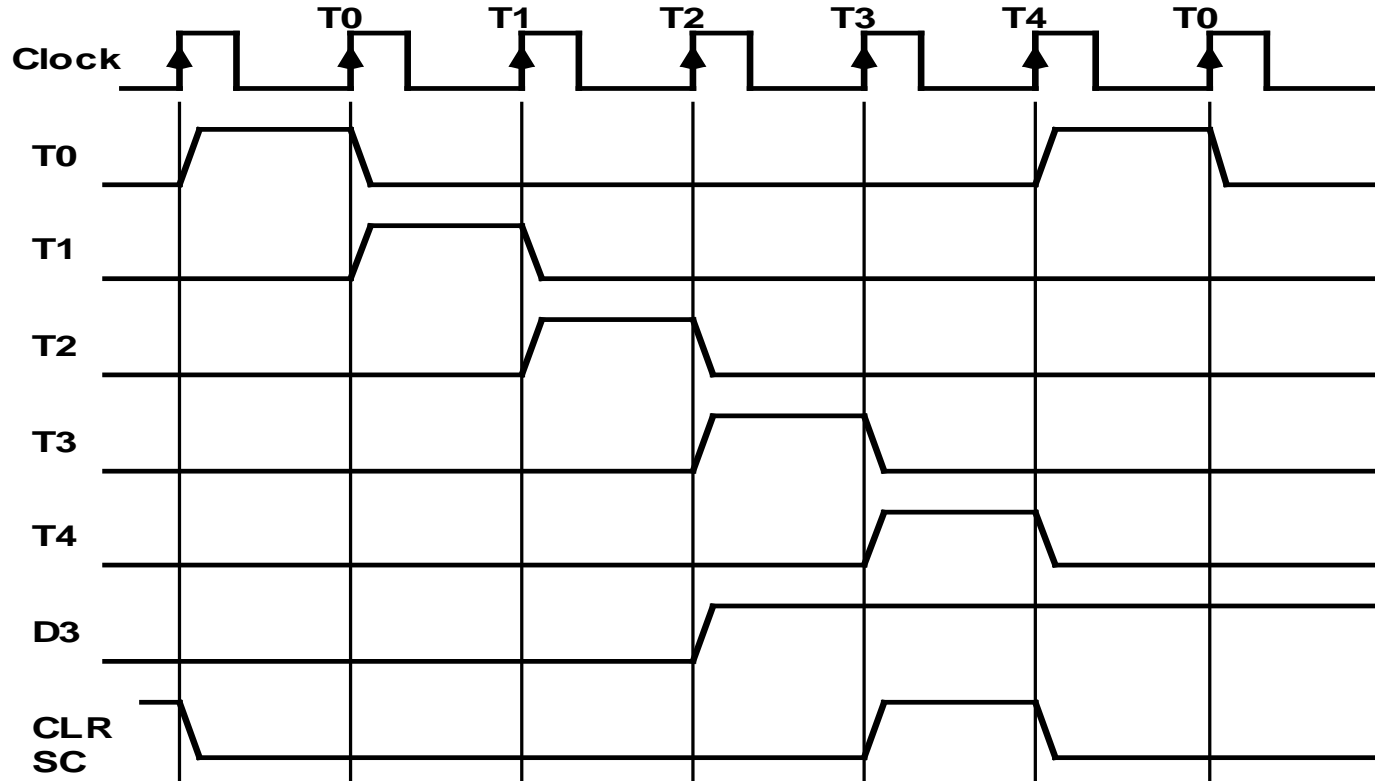
INSTRUCTION CYCLE

- 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

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)
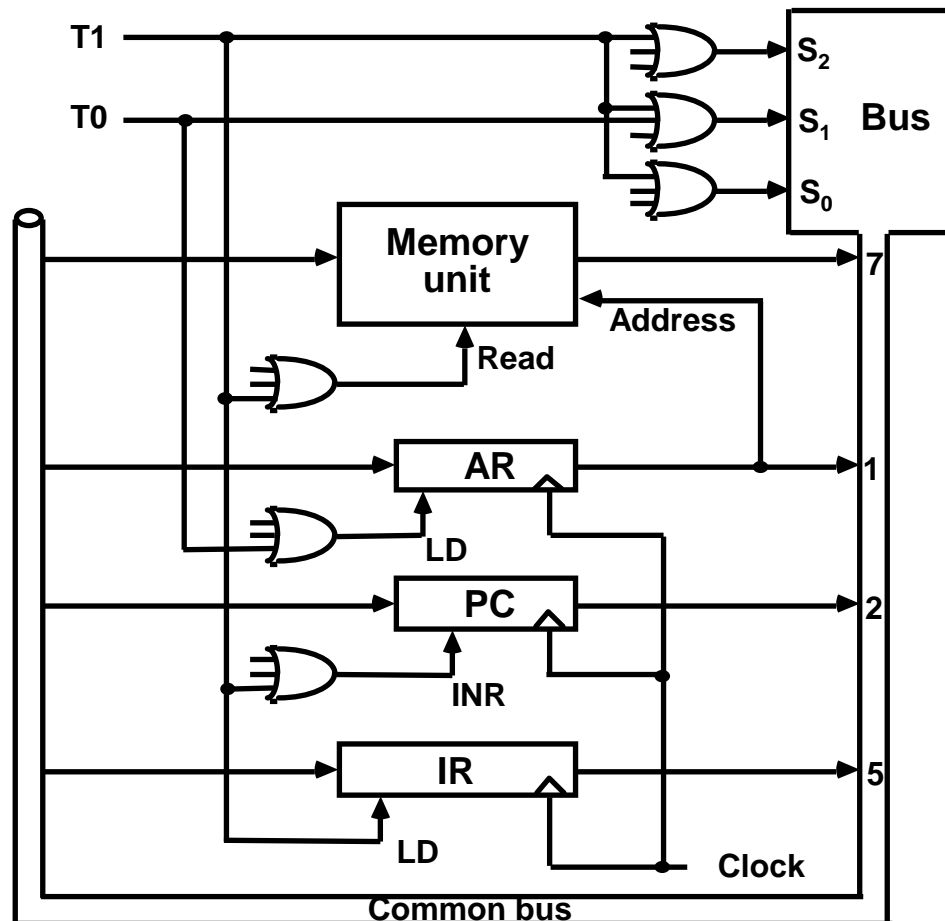
DETERMINE THE TYPE OF INSTRUCTION
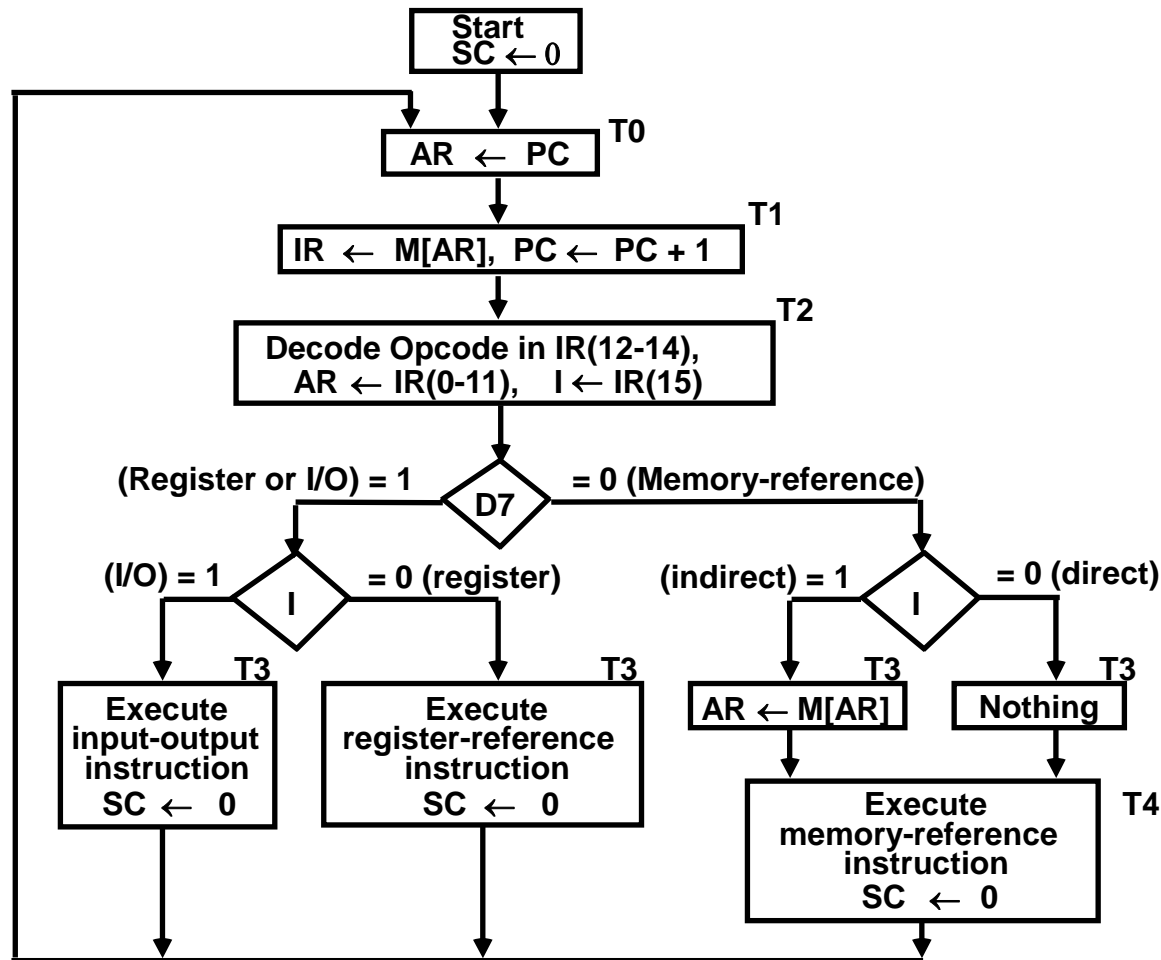
$D'_7 I T_3$: $AR \leftarrow M[AR]$
$D'_7 I' T_3$:   **Nothing**
$D_7 I' T_3$:   **Execute a register-reference instr.**
$D_7 I T_3$:   **Execute an input-output instr.**

Amrita Vishwa Vidyapeetham

1/27/2010

Register Reference Instructions are identified when

- $D_7 = 1,\ I = 0$
- 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:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	if $(AC(15) = 0)$ then $(PC \leftarrow PC+1)$
SNA	rB_3:	if $(AC(15) = 1)$ then $(PC \leftarrow PC+1)$
SZA	rB_2:	if $(AC = 0)$ then $(PC \leftarrow PC+1)$
SZE	rB_1:	if $(E = 0)$ then $(PC \leftarrow PC+1)$
HLT	rB_0:	$S \leftarrow 0$ (S is a start-stop flip-flop)

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 $\leftarrow$ M[AR]

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

STA: Store AC

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

BUN: Branch Unconditionally

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

BSA: Branch and Save Return Address

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

Memory, PC, AR at time T4

20	0 BSA 135
PC = 21	Next instruction
AR = 135	
136	Subroutine
	↓
	1 BUN 135

Memory

Memory, PC after execution

20	0 BSA 135
21	Next instruction
135	21
PC = 136	Subroutine
	↓
	1 BUN 135

Memory

MEMORY REFERENCE INSTRUCTIONS

BSA:

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

$\quad$ **D_5T_5:** $\quad$ **$PC \leftarrow AR, \; SC \leftarrow 0$**

ISZ: Increment and Skip-if-Zero

$\quad$ **D_6T_4:** $\quad$ **$DR \leftarrow M[AR]$**

$\quad$ **D_6T_5:** $\quad$ **$DR \leftarrow DR + 1$**

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

Memory-reference instruction

AND	ADD	LDA	STA
D_0T_4	D_1T_4	D_2T_4	D_3T_4
DR ← M[AR]	DR ← M[AR]	DR ← M[AR]	M[AR] ← AC SC ← 0
D_0T_5	D_1T_5	D_2T_5	
AC ← AC ∧ DR SC ← 0	AC ← AC + DR E ← Cout SC ← 0	AC ← DR SC ← 0	

BUN	BSA	ISZ
D_4T_4	D_5T_4	D_6T_4
PC ← AR SC ← 0	M[AR] ← PC AR ← AR + 1	DR ← M[AR]
	D_5T_5	D_6T_5
	PC ← AR SC ← 0	DR ← DR + 1

D_6T_6

M[AR] ← DR
If (DR = 0)
then (PC ← PC + 1)
SC ← 0
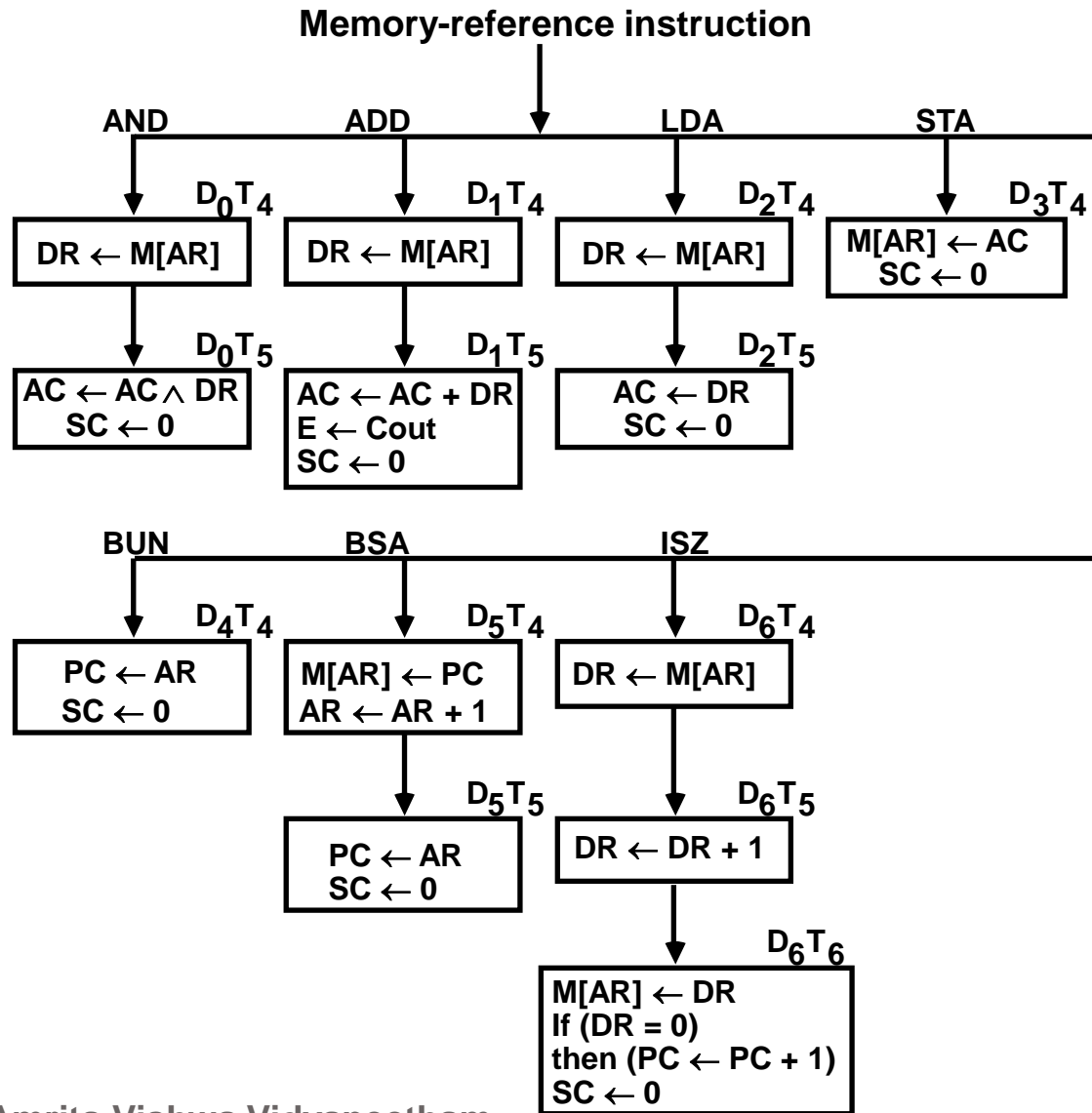
INPUT-OUTPUT AND INTERRUPT

A Terminal with a keyboard and a Printer

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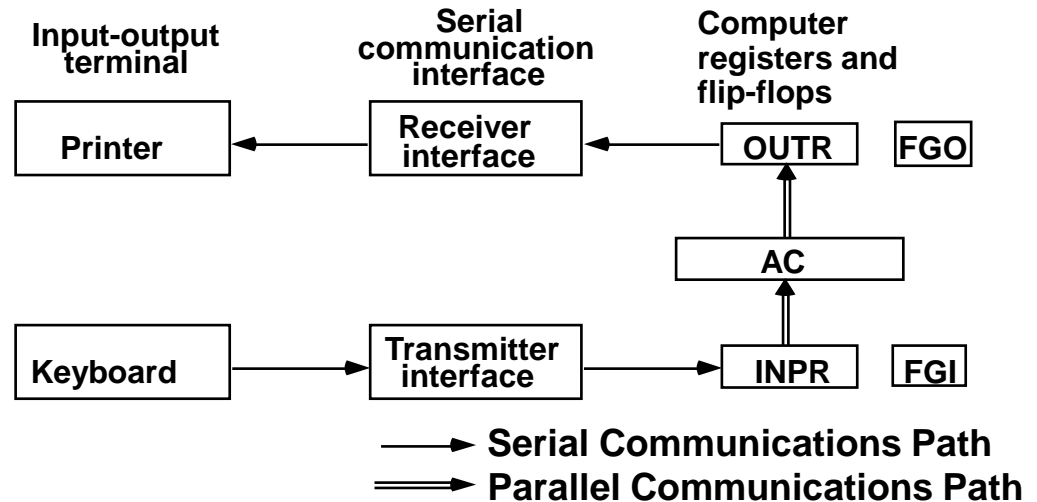
| INPR	Input register - 8 bits
OUTR	Output register - 8 bits
FGI	Input flag - 1 bit
FGO	Output flag - 1 bit
IEN	Interrupt enable - 1 bit

- **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**

-- CPU --

/* Input */   /* Initially FGI = 0 */
  loop: If FGI = 0 goto loop
          AC $\leftarrow$ INPR, FGI $\leftarrow$ 0

/* Output */   /* Initially FGO = 1 */
  loop: If FGO = 0 goto loop
          OUTR $\leftarrow$ AC, FGO $\leftarrow$ 0
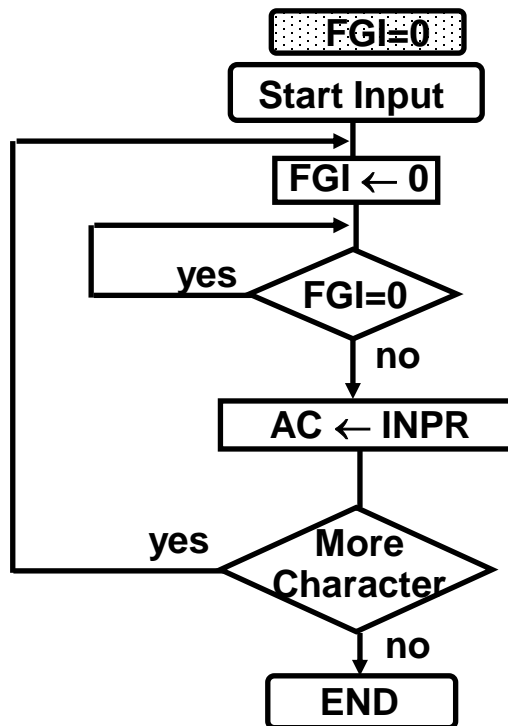
-- I/O Device --

loop: If FGI = 1 goto loop
      INPR $\leftarrow$ new data, FGI $\leftarrow$ 1

loop: If FGO = 1 goto loop
    consume OUTR, FGO $\leftarrow$ 1

CPU flowchart:

FGI=0 → Start Input → FGI $\leftarrow$ 0 → FGI=0 (yes: loop back; no) → AC $\leftarrow$ INPR → More Character (yes: loop back; no) → END

I/O Device flowchart:

FGO=1 → Start Output → AC $\leftarrow$ Data → FGO=0 (yes: loop back; no) → OUTR $\leftarrow$ AC → FGO $\leftarrow$ 0 → More Character (yes: loop back; no) → END
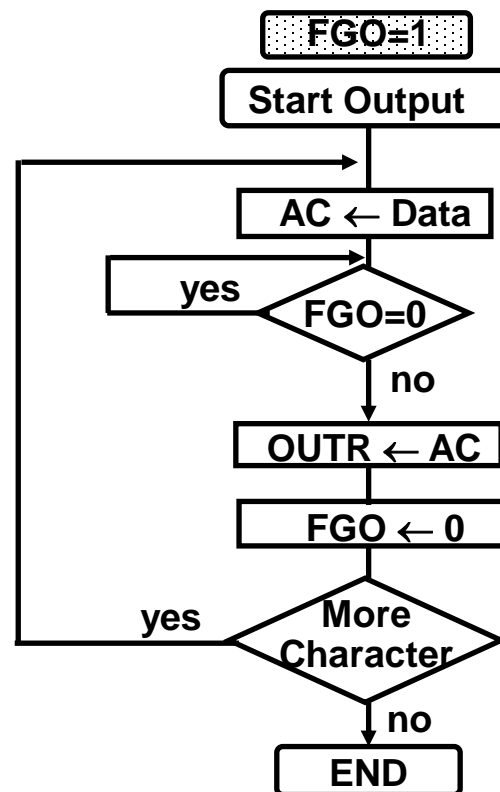
INPUT-OUTPUT INSTRUCTIONS

$D_7 IT_3 = p$

$IR(i) = B_i$, 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

PROGRAM-CONTROLLED INPUT/OUTPUT

- **Program-controlled I/O**
  - **- Continuous CPU involvement**
    - **I/O takes valuable CPU time**
  - **- CPU slowed down to I/O speed**
  - **- Simple**
  - **- Least hardware**

Input

```
LOOP,    SKI    DEV
          BUN   LOOP
          INP    DEV
```

Output

```
LOOP,    LDA    DATA
LOP,     SKO   DEV
          BUN   LOP
          OUT   DEV
```

Amrita Vishwa Vidyapeetham

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
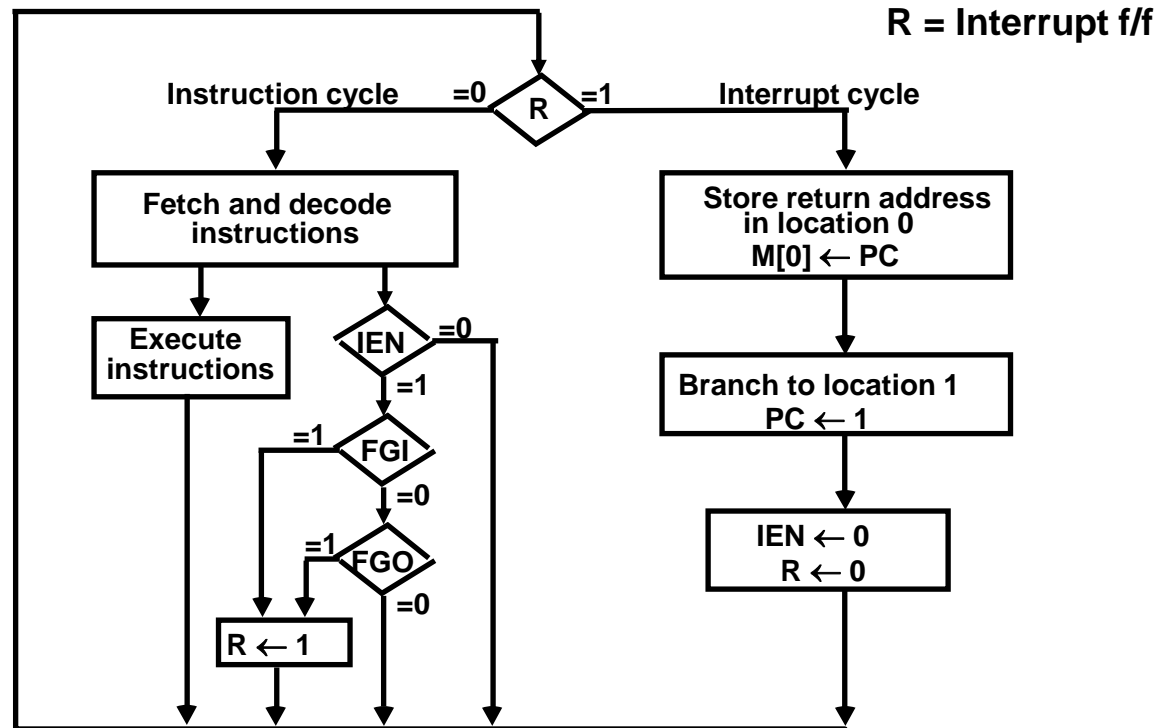
R = Interrupt f/f

- 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"

Memory

Before interrupt				**After interrupt cycle**		

0		
1	0 BUN 1120	
	Main Program	
255		
PC = 256		
1120	**I/O Program**	
	1 BUN 0	

0	256	
PC = 1	0 BUN 1120	
	Main Program	
255		
256		
1120	**I/O Program**	
	1 BUN 0	
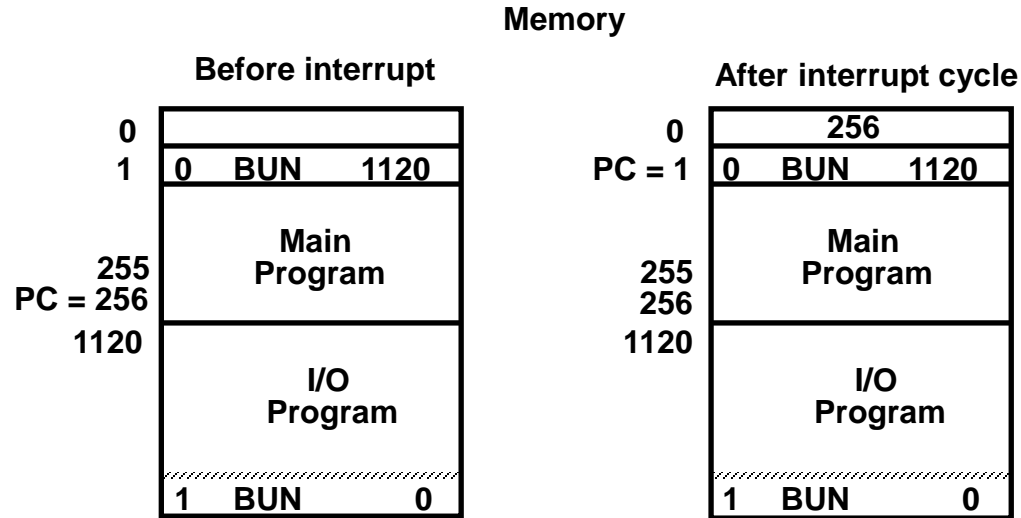
Register Transfer Statements for Interrupt Cycle

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

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

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

- The interrupt cycle :

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

FURTHER QUESTIONS ON INTERRUPT

How can the CPU recognize the device
requesting an interrupt ?
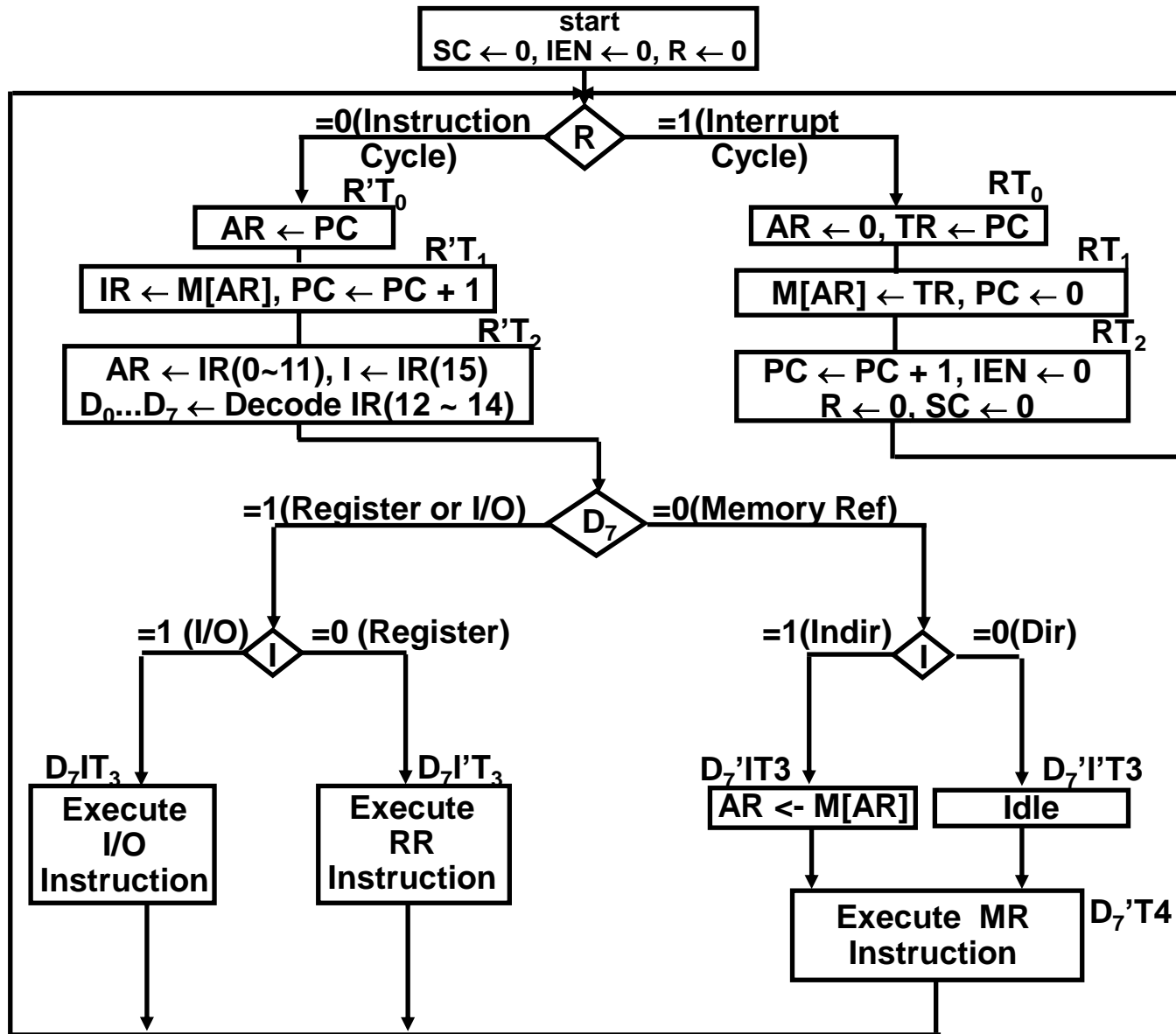
Since different devices are likely to require
different interrupt service routines, how can
the CPU obtain the starting address of the
appropriate routine in each case ?

Should any device be allowed to interrupt the
CPU while another interrupt is being serviced ?

How can the situation be handled when two or
more interrupt requests occur simultaneously ?

Amrita Vishwa Vidyapeetham

COMPLETE COMPUTER DESCRIPTION
Flowchart of Operations

start
$SC \leftarrow 0, IEN \leftarrow 0, R \leftarrow 0$

R

=0(Instruction Cycle)

=1(Interrupt Cycle)

$R'T_0$
$AR \leftarrow PC$

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

$R'T_2$
$AR \leftarrow IR(0\sim11), I \leftarrow IR(15)$
$D_0...D_7 \leftarrow$ Decode $IR(12 \sim 14)$

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

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

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

D_7

=1(Register or I/O)

=0(Memory Ref)

=1 (I/O) **I** =0 (Register)

=1(Indir) **I** =0(Dir)

D_7IT_3
Execute I/O Instruction

$D_7I'T_3$
Execute RR Instruction

$D_7'IT3$
$AR \leftarrow M[AR]$

$D_7'I'T3$
Idle

$D_7'T4$
Execute MR Instruction

Amrita Vishwa Vidyapeetham

COMPLETE COMPUTER DESCRIPTION
Microooperations

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 ($PC \leftarrow PC + 1$), $SC \leftarrow 0$

Amrita Vishwa Vidyapeetham

Microoperations

```
Register-Reference
                    D₇I'T₃ = r        (Common to all register-reference instr)
                    IR(i) = Bᵢ        (i = 0,1,2, ..., 11)
                     r:               SC ← 0
CLA                 rB₁₁:             AC ← 0
CLE                 rB₁₀:             E ← 0
CMA                 rB₉:              AC ← AC'
CME                 rB₈:              E ← E'
CIR                 rB₇:              AC ← shr AC, AC(15) ← E, E ← AC(0)
CIL                 rB₆:              AC ← shl AC, AC(0) ← E, E ← AC(15)
INC                 rB₅:              AC ← AC + 1
SPA                 rB₄:              If(AC(15) =0) then  (PC ← PC + 1)
SNA                 rB₃:              If(AC(15) =1) then  (PC ← PC + 1)
SZA                 rB₂:              If(AC = 0) then (PC ← PC + 1)
SZE                 rB₁:              If(E=0) then (PC ← PC + 1)
HLT                 rB₀:              S ← 0

Input-Output        D₇IT₃ = p         (Common to all input-output instructions)
                    IR(i) = Bᵢ        (i = 6,7,8,9,10,11)
                     p:               SC ← 0
INP                 pB₁₁:             AC(0-7) ← INPR, FGI ← 0
OUT                 pB₁₀:             OUTR ← AC(0-7), FGO ← 0
SKI                 pB₉:              If(FGI=1) then (PC ← PC + 1)
SKO                 pB₈:              If(FGO=1) then (PC ← PC + 1)
ION                 pB₇:              IEN ← 1
IOF                 pB₆:              IEN ← 0
```

Q & A
Thank You

Amrita Vishwa Vidyapeetham