CHITKARA UNIVERSITY

Basic Computer Organization

CHITKARA UNIVERSITY

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**

INTRODUCTION

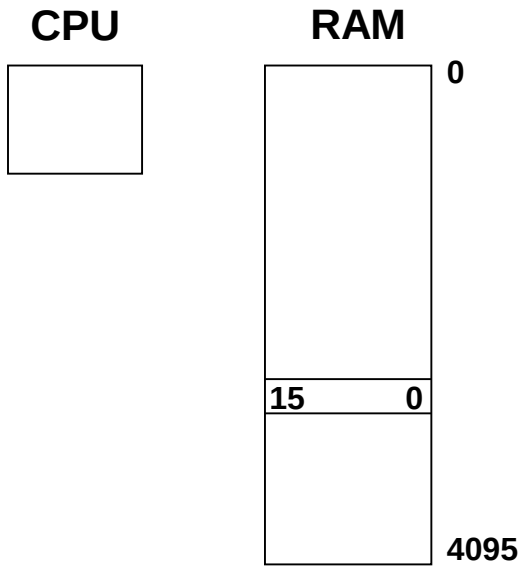
CHITKARA UNIVERSITY

- 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

CHITKARA UNIVERSITY

- 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



INSTRUCTIONS

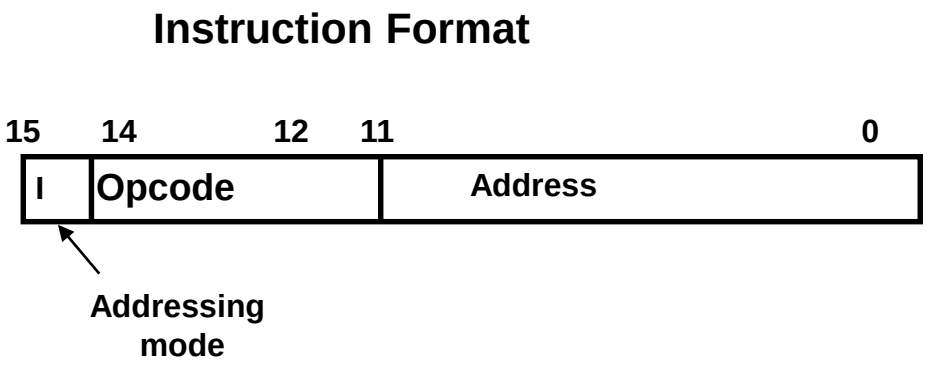
CHITKARA UNIVERSITY

- 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

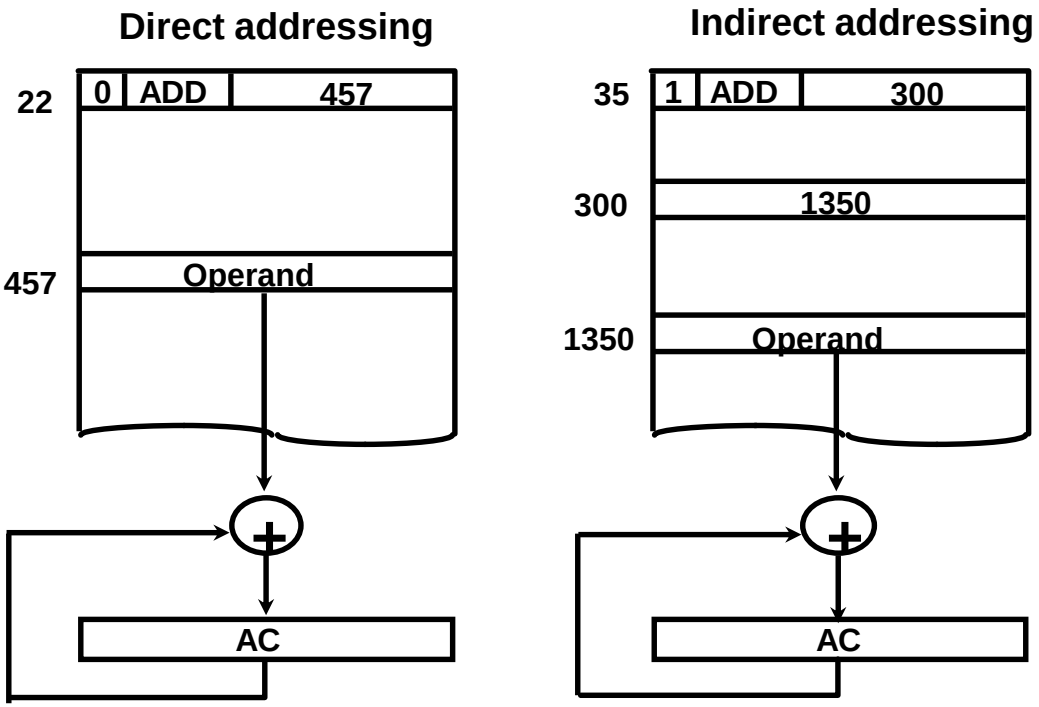
INSTRUCTION FORMAT

CHITKARA UNIVERSITY

- 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



- 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



- 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

PROCESSOR REGISTERS

CHITKARA UNIVERSITY

- 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

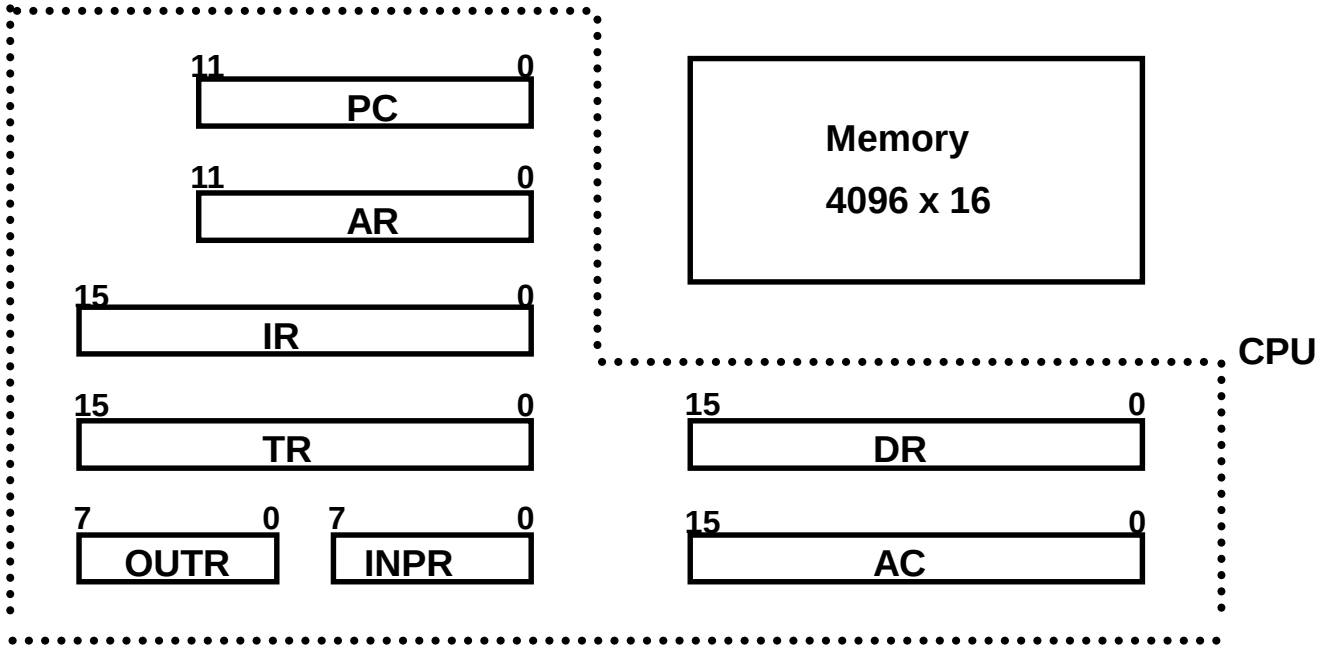
CHITKARA UNIVERSITY

- 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 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

CHITKARA UNIVERSITY

Registers in the Basic Computer(BC)



List of BC Registers

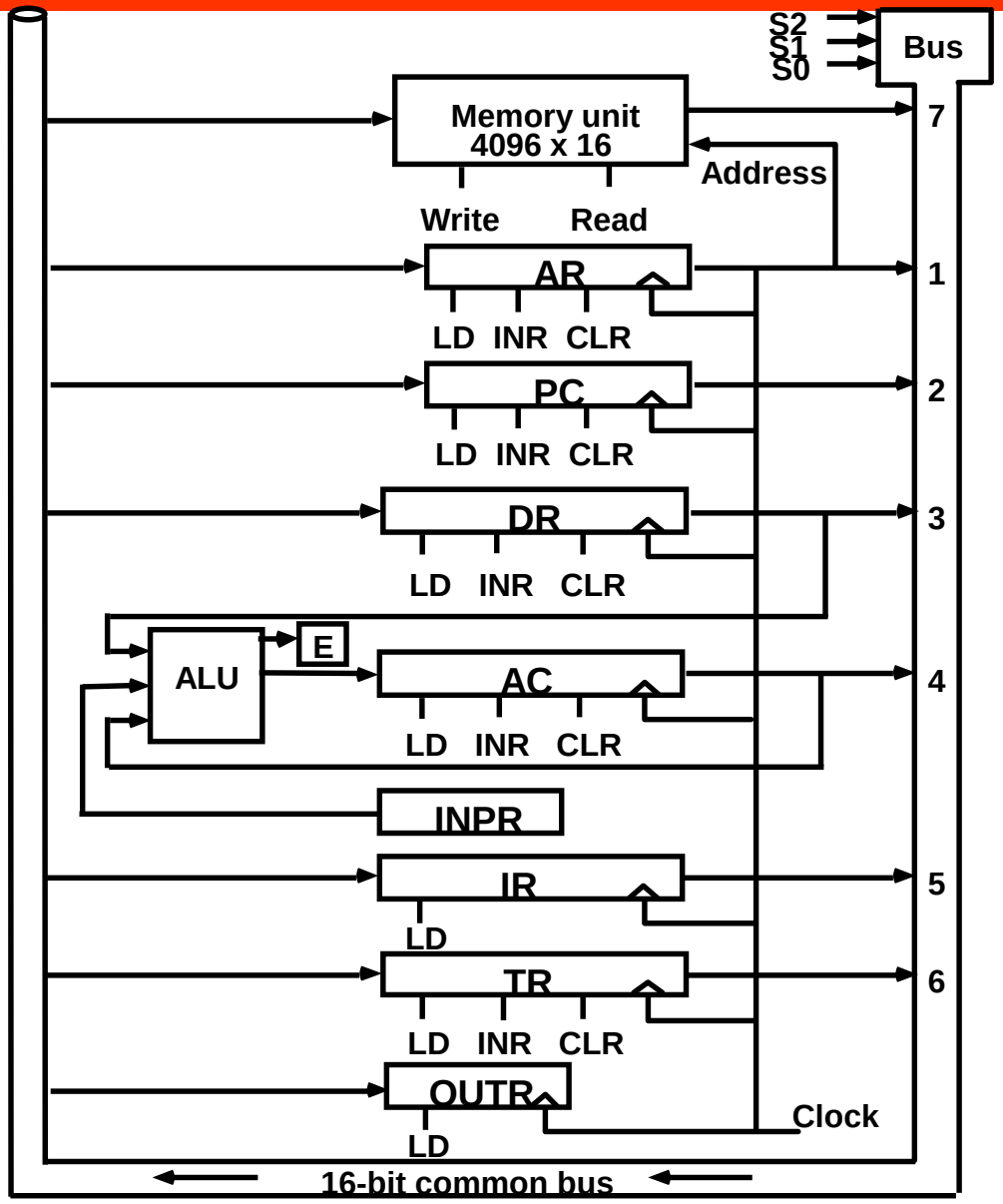
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction Register	Holds instruction code
PC	12	Program Counter	Holds address of instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds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

COMMON BUS SYSTEM

CHITKARA UNIVERSITY

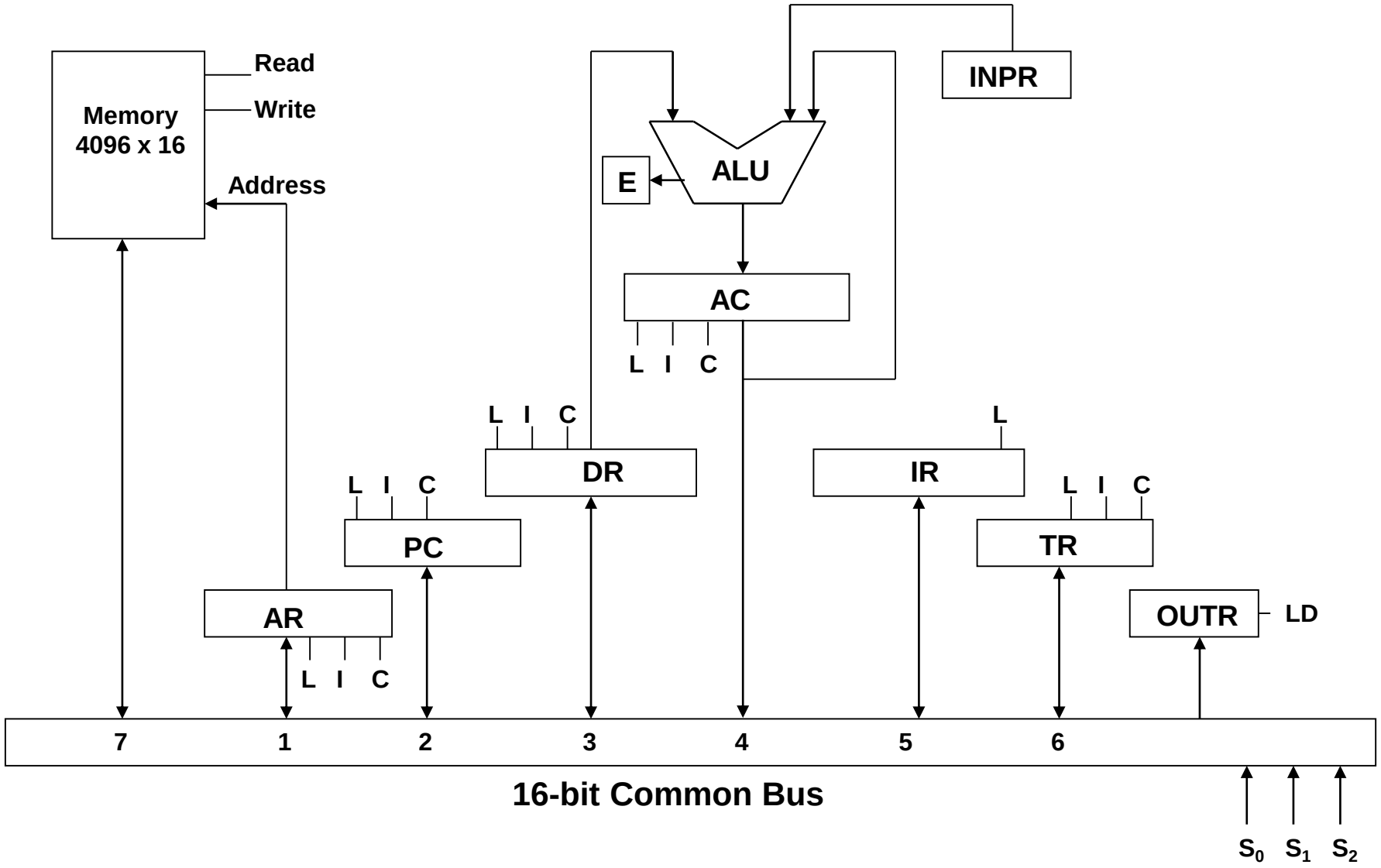
- The registers in the Basic Computer are connected using a bus
- This gives a savings in circuitry over complete connections between registers

COMMON BUS SYSTEM



COMMON BUS SYSTEM

CHITKARA UNIVERSITY



COMMON BUS SYSTEM

CHITKARA UNIVERSITY

- 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

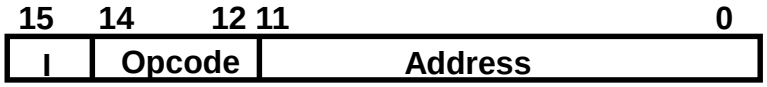
- Either one of the registers will have its load signal activated, or the memory will have its read signal activated
  - Will determine where the data from the bus gets loaded
- The 12-bit registers, AR and PC, have 0's loaded onto the bus in the high order 4 bit positions
- When the 8-bit register OUTR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

BASIC COMPUTER INSTRUCTIONS

CHITKARA UNIVERSITY

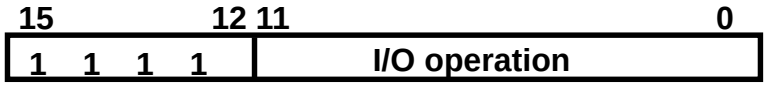
- **Basic Computer Instruction Format**

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



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

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



BASIC COMPUTER INSTRUCTIONS

CHITKARA UNIVERSITY

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

INSTRUCTION SET COMPLETENESS

CHITKARA UNIVERSITY

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**

CONTROL UNIT

The control unit (CU) is a component of a processor that interprets program instructions and **generates control signals** that **direct other parts** of the computer to perform specific micro-operations necessary to execute those instructions.

Two types:

- ***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

CONTROL UNIT

CHITKARA UNIVERSITY

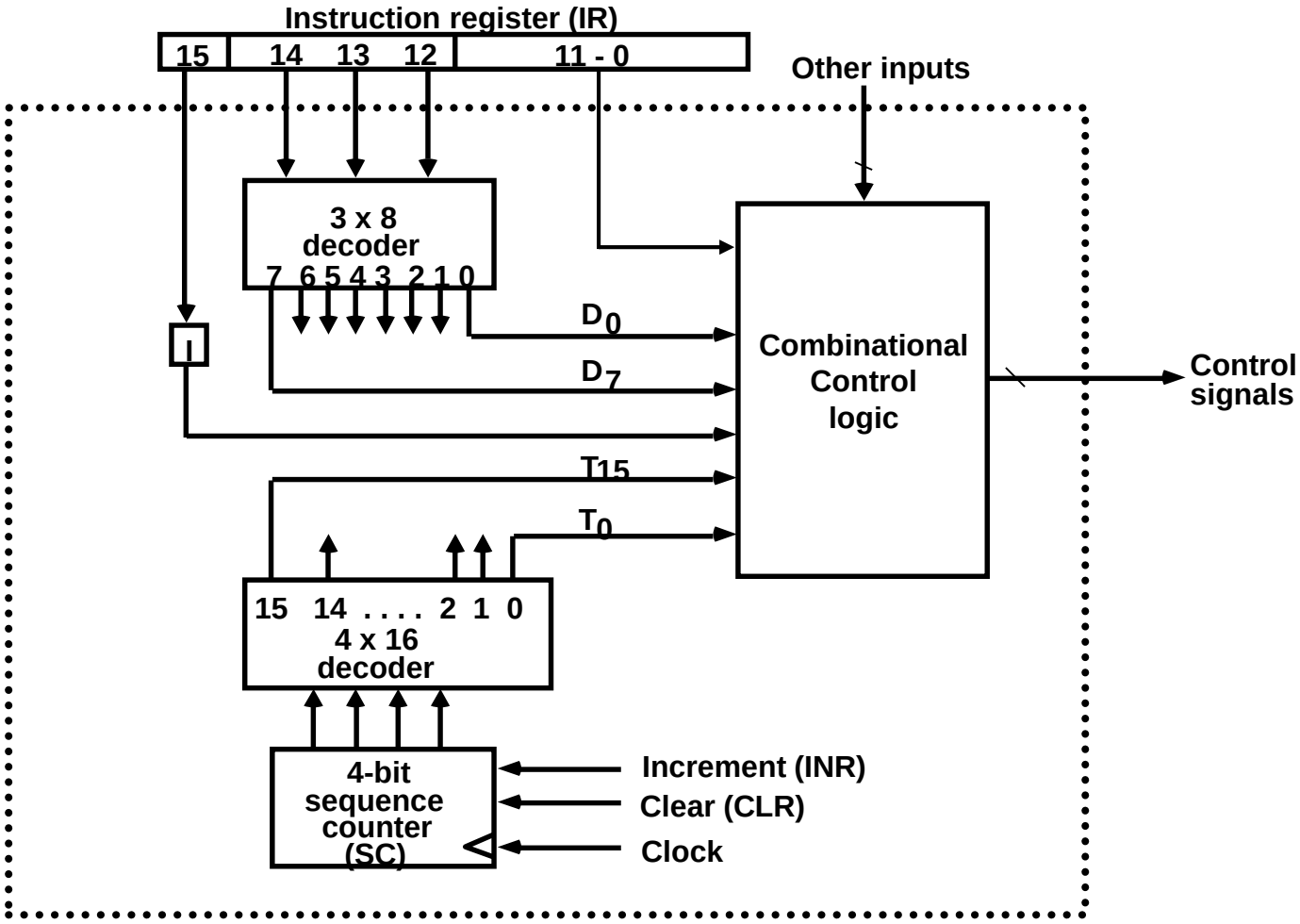
- Control unit (CU) of a processor translates from machine instructions to the control signals for the microoperations that implement them

- 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

CHITKARA UNIVERSITY

Control unit of Basic Computer



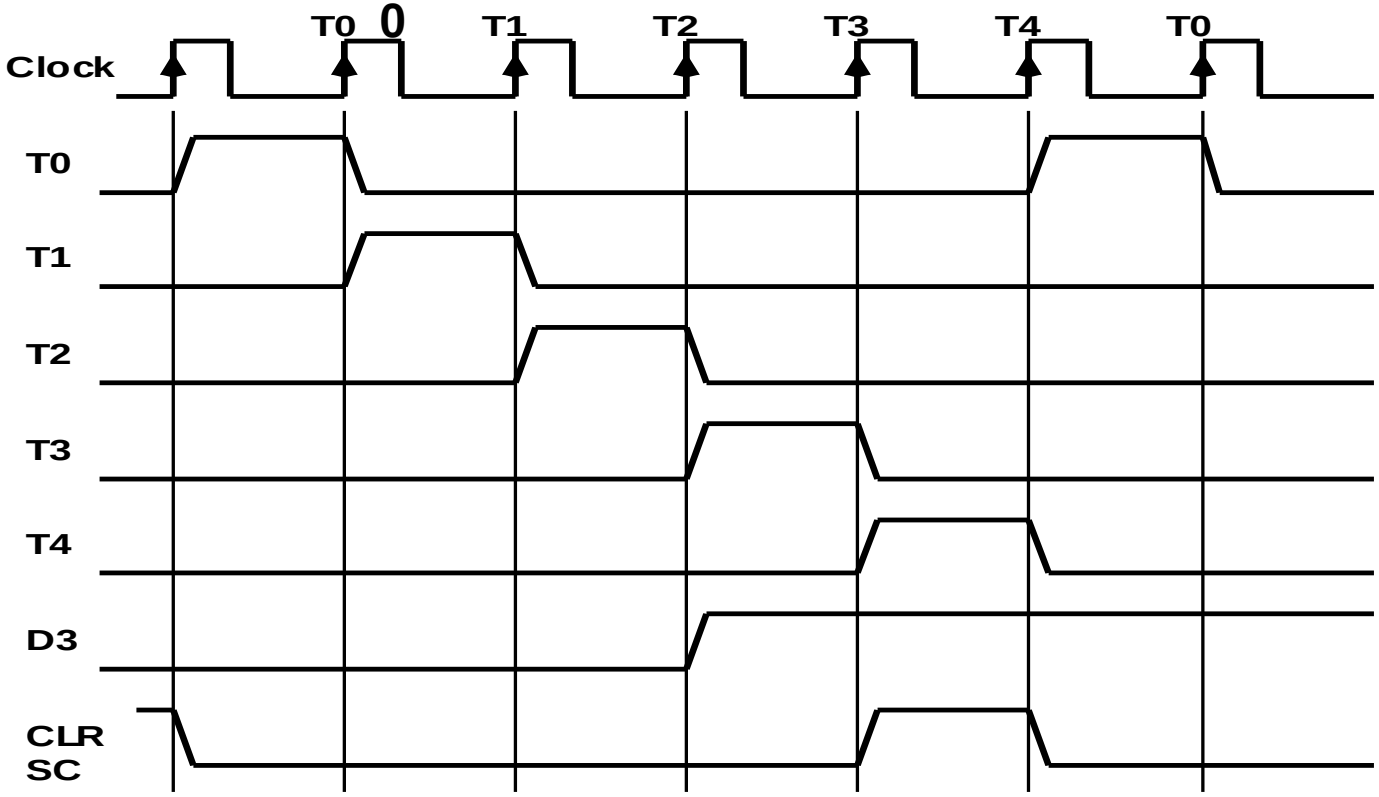
TIMING SIGNALS

- 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

CHITKARA UNIVERSITY

- 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

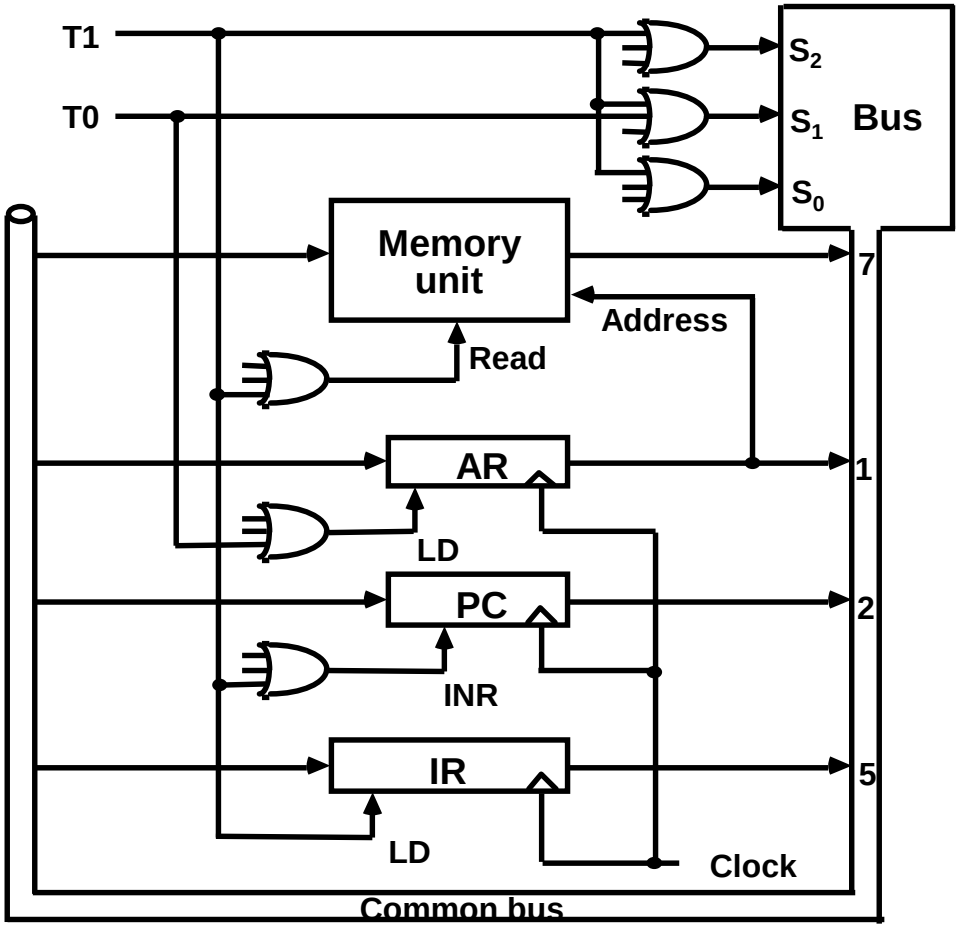
- *Note*: Every different processor has its own (different) instruction cycle

FETCH and DECODE

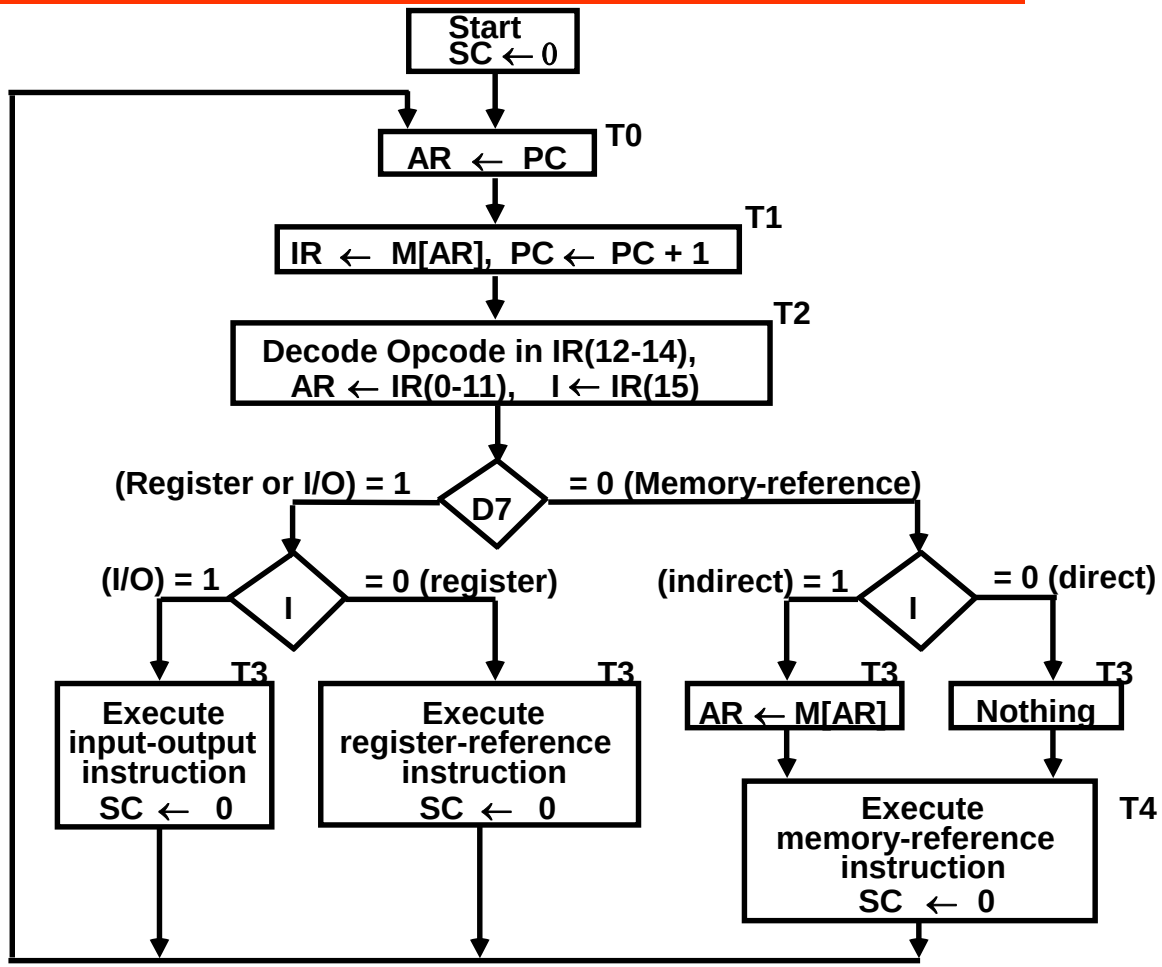
CHITKARA UNIVERSITY

- **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)



Flowchart of Instruction Cycle



D'_7IT_3: AR ← M[AR]
$D'_7I'T_3$: Nothing
$D_7I'T_3$: Execute a register-reference instr.
D_7IT_3: Execute an input-output instr.

REGISTER REFERENCE INSTRUCTIONS

CHITKARA UNIVERSITY

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 ← 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)

Memory Reference Instructions

CHITKARA UNIVERSITY

- In order to specify the microoperations needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely. Some instructions have an ambiguous description. This is because the explanation of an instruction in words is usually lengthy, and not enough space is available in the table for such a lengthy explanation.
- We will now show that the function of the memory-reference instructions can be defined precisely by means of register transfer notation.

- The decoded D; for i = 0, 1, 2, 3, 4, 5, and 6 from the operation decoder that belongs to each instruction is included in the table. The effective address of the instruction is in the address register AR and was placed there during timing signal T2 when I = 0, or during timing signal T3 when I = 1. The execution of the memory-reference instructions starts with timing signal T4• The symbolic description of each instruction is specified in the table in terms of register transfer notation.
- The actual execution of the instruction in the bus system will require a sequence of microoperations. This is because data stored in memory cannot be processed directly. The data must be read from memory to a register where they can be operated on with logic circuits. We now explain the operation of each instruction and list the control functions and microoperations needed for their execution.
  A flowchart that summarizes all the microoperations is presented at the end of this section

- Memory reference instructions are useful in order to perform operations on operands. Operands are located in main memory.
- We know that in Memory reference instruction, there are 3 bits for opcode. Means 2^3=8 operations are possible. For each operation we have a decoder assigned D0 to D6. D7 is reserved. Table 5-4 lists the seven memory-reference instructions. The seven memory reference instructions are:

MEMORY REFERENCE INSTRUCTIONS

CHITKARA UNIVERSITY

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

1. AND to AC

- This is an instruction that perform 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 . The microoperations that execute this instruction are:

D0T4: DR <- M [AR]
D0T5: AC <- AC /\ DR, SC <--- 0

The control function for this instruction uses the operation decoder D0 since this output of the decoder is active when the instruction has an AND operation whose binary code value 000. Two timing signals are needed to execute the instruction. The clock transition associated with timing signal T4 transfers the operand from memory into DR . The clock transition associated with the next timing signal T5 transfers to AC the result of the AND logic operation between the contents of DR and AC. The same clock transition clears SC to 0, transferring control to timing signal T0 to start a new instruction cycle.

2. 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. The microoperations needed to execute this instruction are

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

- Same Two timing signals, T, and T5, are used again but with operation decoder D1 instead of D0, which was used for the AND instruction. After the instruction is fetched from memory and decoded, only one output of the operation decoder will be active, and that output determines the sequence of microoperations that the control follows during the execution of a memory-reference instruction.

3. LDA: Load to AC

- This instruction transfers the memory word specified by the effective address to AC . The microoperations needed to execute this instruction are

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

Looking back at the bus system shown in Fig. 5-4 we note that there is no direct path from the bus into AC . The adder and logic circuit receive information from DR which can be transferred into AC . Therefore, it is necessary to read the memory word into DR first and then transfer the content of DR into AC . The reason for not connecting the bus to the inputs of AC is the delay encountered in the adder and logic circuit. It is assumed that the time it takes to read from memory and transfer the word through the bus as well as the adder and logic circuit is more than the time of one clock cycle. By not connecting the bus to the inputs of AC we can maintain one clock cycle per microoperation.

4. STA: Store AC

CHITKARA UNIVERSITY

- This instruction stores the content of AC into the memory word specified by the effective address. Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation:
  D3T4: M [AR] ← AC, SC ← 0

5. BUN: Branch Unconditionally

- This instruction transfers the program to the instruction specified by the effective address. Remember that PC holds the address of the instruction to be read from memory in the next instruction cycle. PC is incremented at time T1 to prepare it for the address of the next instruction in the program sequence. The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally. The instruction is executed with one microoperation:
  D4T4: PC ← AR, SC ← 0
- The effective address frpom AR is transferred through the common bus to PC .Resetting SC to 0 transfers control to T0• The next instruction is then fetched and executed from the memory address given by the new value in PC .

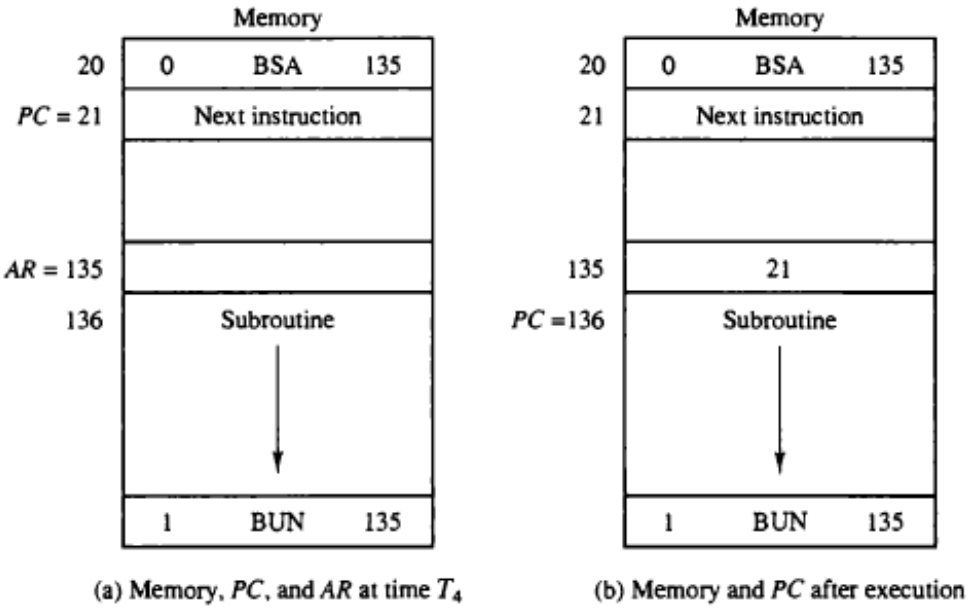
6. BSA: Branch and Save Return Address

CHITKARA UNIVERSITY

- 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. The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine. This operation was specified in Table 5-4 with the following register transfer:
- M[AR] <-- PC, PC <-- AR + I
- A numerical example that demonstrates how this instruction is used with a subroutine is shown in Fig. 5-10. The BSA instruction is assumed to be in memory at address 20. The I bit is 0 and the address part of the instruction has the binary equivalent of 135. After the fetch and decode phases, PC contains 21, which is the address of the next instruction in the program (referred to as the return address). AR holds the effective address 135. This is shown in part (a) of the figure. The BSA instruction performs the following numerical operation:

- M[135] <-- 21, PC <-- 135 + 1 = 136
- The result of this operation is shown in part (b) of the figure. The return address 21 is stored in memory location 135 and control continues with the subroutine program starting from address 136. The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine. When this instruction is executed, control goes to the indirect phase to read the effective address at location 135, where it finds the previously saved address 21. When the BUN instruction is executed, the effective address 21 is transferred to PC . The next instruction cycle finds PC with the value 21, so control continues to execute the instruction at the return address.

Figure 5-10 Example of BSA instruction execution.



7. ISZ: Increment and Skip if Zero

- This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1. The programmer usually stores a negative number (in 2's complement) in the memory word. As this negative number is repeatedly incremented by one, it eventually reaches the value of zero. At that time PC is incremented by one in order to skip the next instruction in the program.
  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. This is done with the following sequence of microoperations:
  D6T4: DR <-- M [AR]
  D6T5: DR <-- DR + 1
  D,T,: M [AR] <-- DR,
  if (DR = 0) then (PC ← PC + 1), SC ← 0

MEMORY REFERENCE INSTRUCTIONS

CHITKARA UNIVERSITY

LDA: Load to AC

D_2T_4: DR ← M[AR]

D_2T_5: AC ← DR, SC ← 0

STA: Store AC

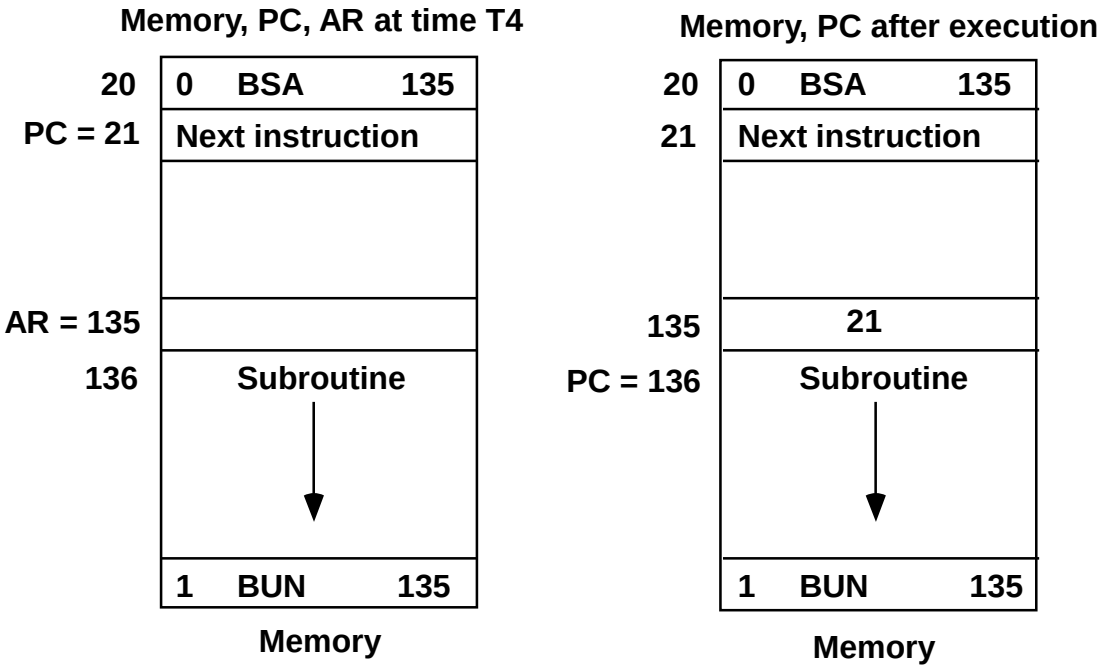
D_3T_4: M[AR] ← AC, SC ← 0

BUN: Branch Unconditionally

D_4T_4: PC ← AR, SC ← 0

BSA: Branch and Save Return Address

M[AR] ← PC, PC ← AR + 1



MEMORY REFERENCE INSTRUCTIONS

CHITKARA UNIVERSITY

BSA:

D_5T_4: M[AR] ← PC, AR ← AR + 1

D_5T_5: PC ← AR, SC ← 0

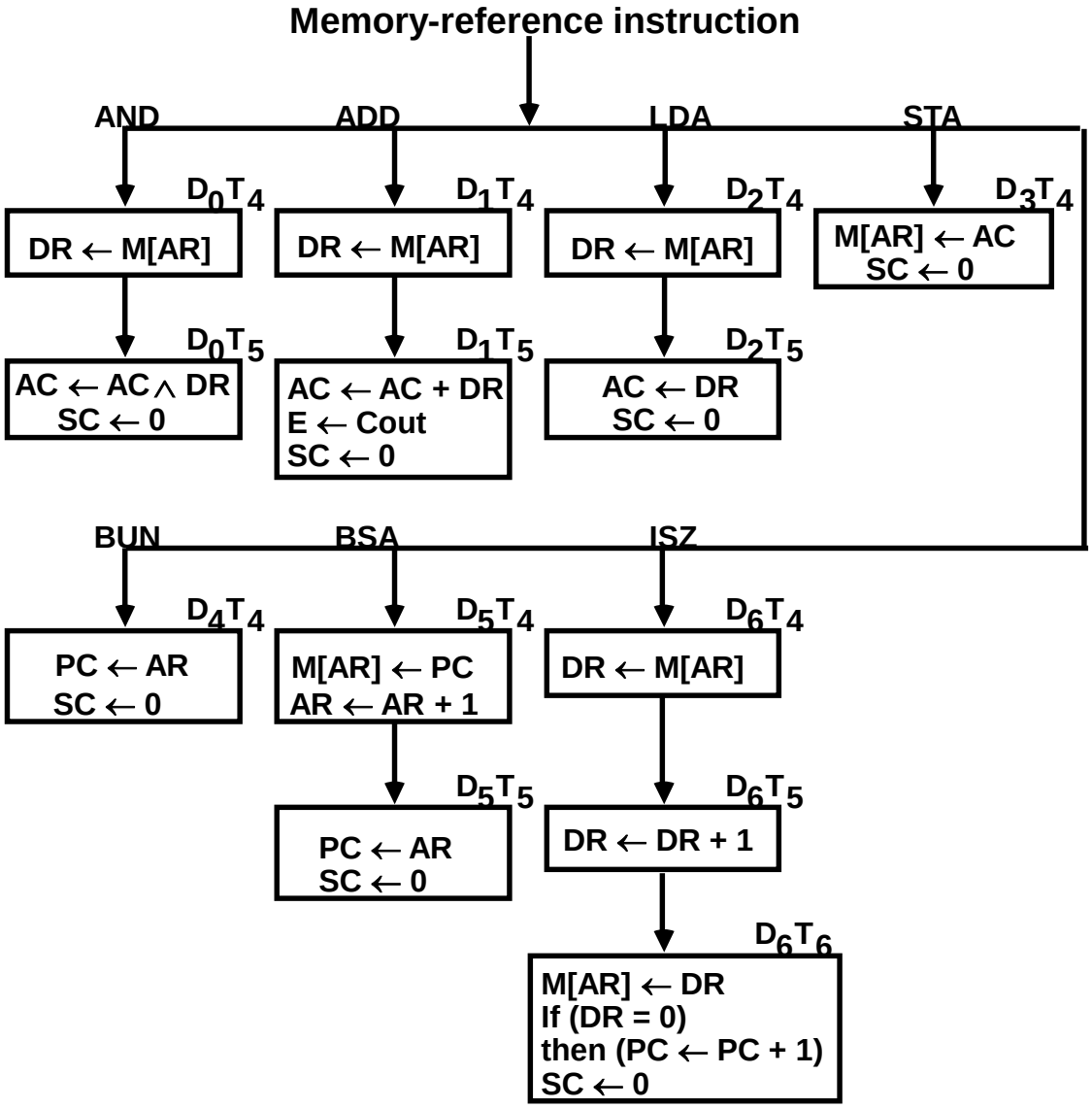
ISZ: Increment and Skip-if-Zero

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

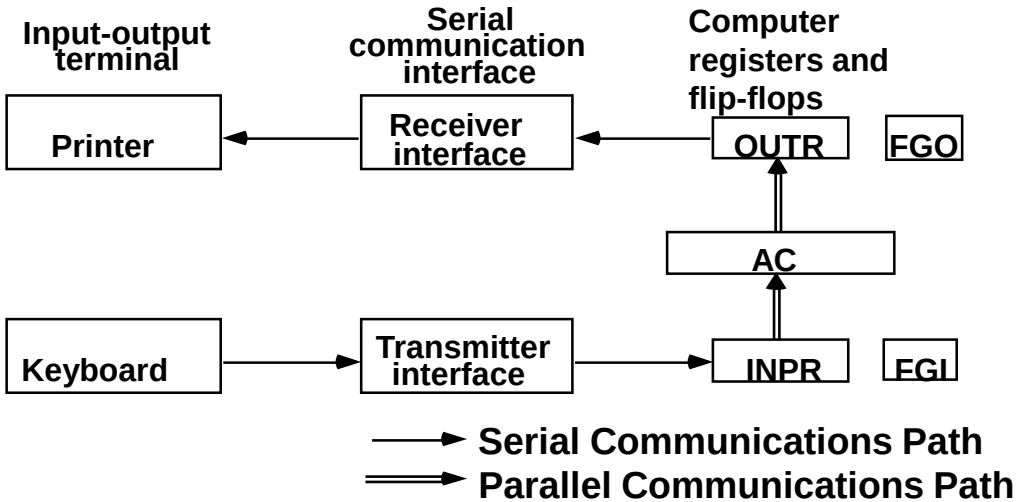


INPUT-OUTPUT AND INTERRUPT

CHITKARA UNIVERSITY

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

CHITKARA UNIVERSITY

$D_7IT_3 = p$
$IR(i) = B_i$, 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

INTERRUPT INITIATED INPUT/OUTPUT

CHITKARA UNIVERSITY

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

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

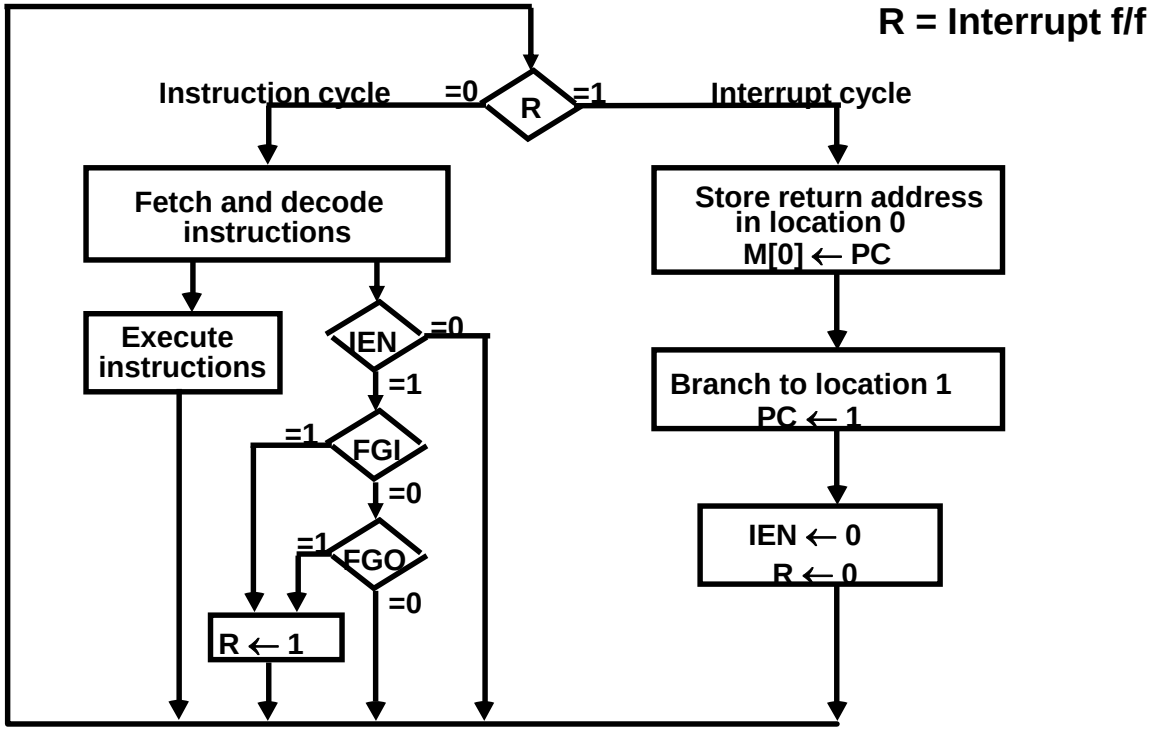
- When the interface founds that the I/O device is ready for data transfer, it generates an interrupt request to the CPU

- 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)

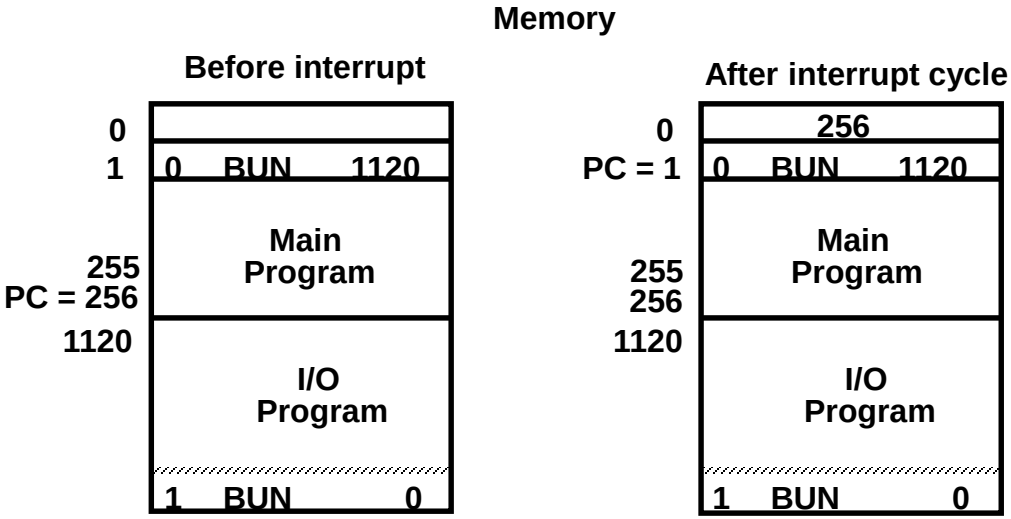
  - can be set and cleared by instructions
  - when cleared, the computer cannot be interrupted

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

FURTHER QUESTIONS ON INTERRUPT

CHITKARA UNIVERSITY

How can the CPU recognize the device requesting an interrupt ?

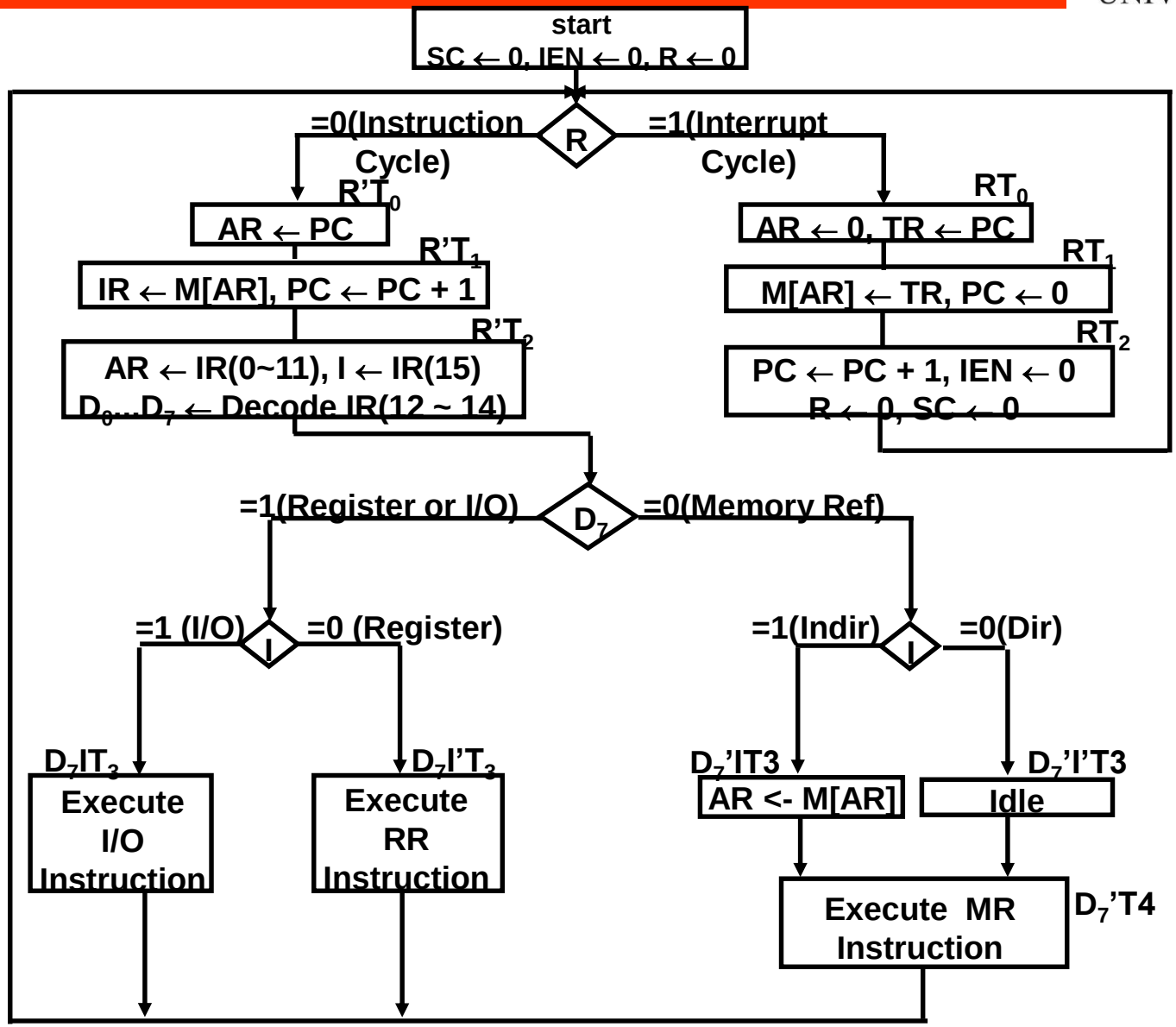
Since different devices are likely to require different interrupt service routines, how can the CPU obtain the starting address of the appropriate routine in each case ?

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 ?

COMPLETE COMPUTER DESCRIPTION

Flowchart of Operations



COMPLETE COMPUTER DESCRIPTION

CHITKARA UNIVERSITY

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

CHITKARA UNIVERSITY

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

DESIGN OF BASIC COMPUTER

CHITKARA UNIVERSITY

Hardware Components of BC

A memory unit: 4096 x 16.
Registers:
AR, PC, DR, AC, IR, TR, OUTR, INPR, and SC
Flip-Flops(Status):
I, S, E, R, IEN, FGI, and FGO
Decoders: a 3x8 Opcode decoder
a 4x16 timing decoder
Common bus: 16 bits
Control logic gates:
Adder and Logic circuit: Connected to AC

Control Logic Gates

- Input Controls of the nine registers

- Read and Write Controls of memory

- Set, Clear, or Complement Controls of the flip-flops

- S_2, S_1, S_0 Controls to select a register for the bus

- AC, and Adder and Logic circuit

CONTROL OF REGISTERS AND MEMORY

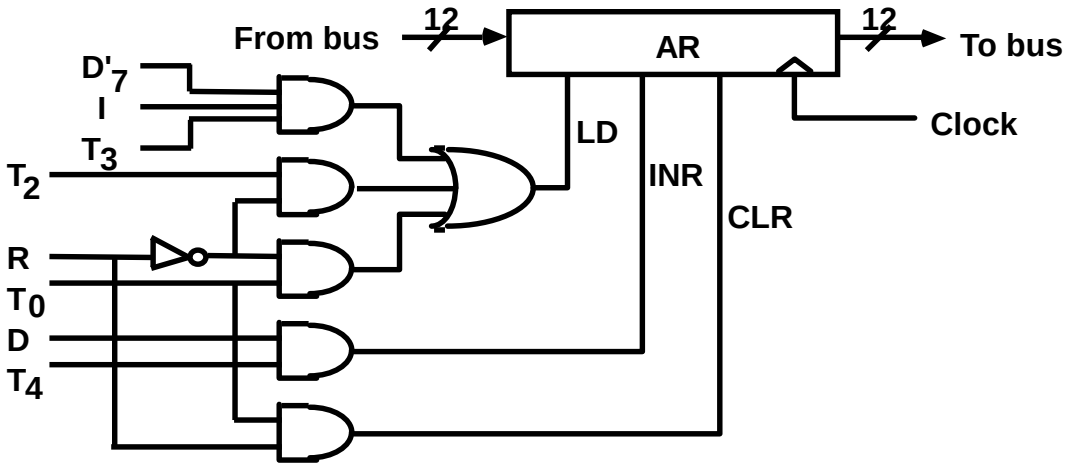
CHITKARA UNIVERSITY

Address Register; AR

Scan all of the register transfer statements that change the content of AR:

$R'T_0$:	$AR \leftarrow PC$	$LD(AR)$
$R'T_2$:	$AR \leftarrow IR(0\text{-}11)$	$LD(AR)$
D'_7IT_3:	$AR \leftarrow M[AR]$	$LD(AR)$
RT_0:	$AR \leftarrow 0$	$CLR(AR)$
D_5T_4:	$AR \leftarrow AR + 1$	$INR(AR)$

$$LD(AR) = R'T_0 + R'T_2 + D'_7IT_3$$
$$CLR(AR) = RT_0$$
$$INR(AR) = D_5T_4$$



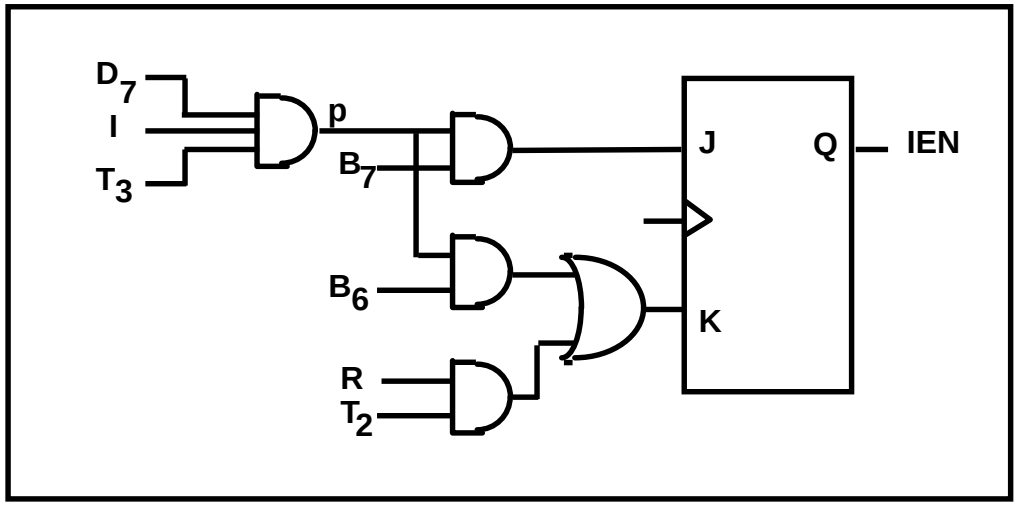
CONTROL OF FLAGS

CHITKARA UNIVERSITY

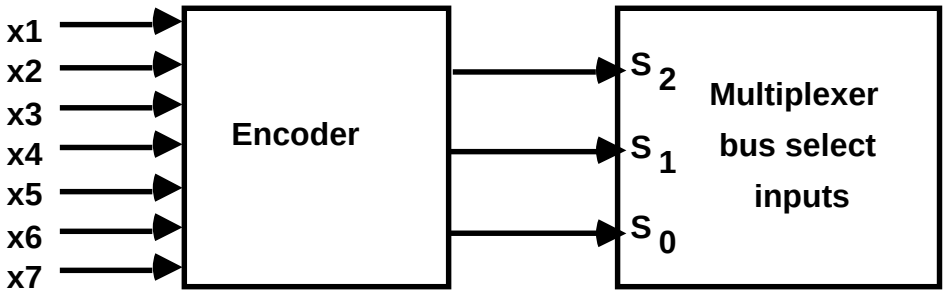
IEN: Interrupt Enable Flag

pB_7: IEN ← 1 (I/O Instruction)
pB_6: IEN ← 0 (I/O Instruction)
RT_2: IEN ← 0 (Interrupt)

$p = D_7IT_3$ (Input/Output Instruction)



CONTROL OF COMMON BUS



x1	x2	x3	x4	x5	x6	x7	S2	S1	S0	selected register
0	0	0	0	0	0	0	0	0	0	none
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

For AR

D_4T_4: PC ← AR
D_5T_5: PC ← AR

$x1 = D_4T_4 + D_5T_5$

Microprogrammed

To execute an instruction, there are two types of control units Hardwired Control unit and Micro-programmed control unit.

1.Hardwired control units are generally faster than microprogrammed designs. In hardwired control, we saw how all the control signals required inside the CPU can be generated using a state counter and a PLA circuit.

2.A microprogrammed control unit is a relatively simple logic circuit that is capable of (1) sequencing through microinstructions and (2) generating control signals to execute each microinstruction.

The control unit's implementation, whether hardwired or micro-programmed, affects the performance and flexibility of the CPU.

Hardwired Control Unit	Microprogrammed Control Unit
Hardwired control unit generates the control signals needed for the processor using logic circuits	Microprogrammed control unit generates the control signals with the help of micro instructions stored in control memory
Hardwired control unit is faster when compared to microprogrammed control unit as the required control signals are generated with the help of hardwares	This is slower than the other as micro instructions are used for generating signals here
Difficult to modify as the control signals that need to be generated are hard wired	Easy to modify as the modification need to be done only at the instruction level
More costlier as everything has to be realized in terms of logic gates	Less costlier than hardwired control as only micro instructions are used for generating control signals
It cannot handle complex instructions as the circuit design for it becomes complex	It can handle complex instructions
Only limited number of instructions are used due to the hardware implementation	Control signals for many instructions can be generated
Used in computer that makes use of Reduced Instruction Set Computers(RISC)	Used in computer that makes use of Complex Instruction Set Computers(CISC)

Figure 7-1 Microprogrammed control organization.