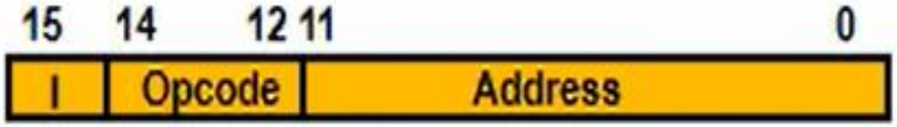
Computer System Architecture

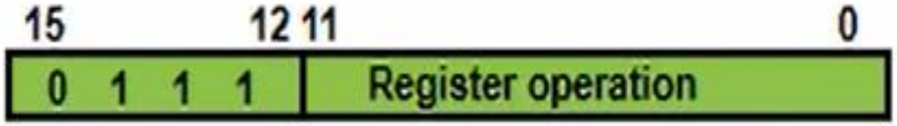
DR. Howida Youssry

- **Basic Computer Instruction Format**

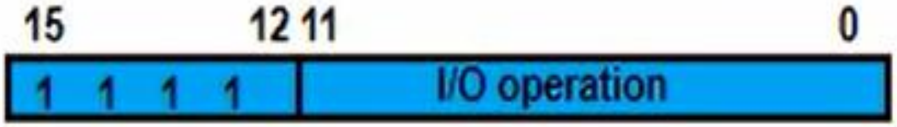
Memory-Reference Instructions (OP-code = 000 ~ 110)



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



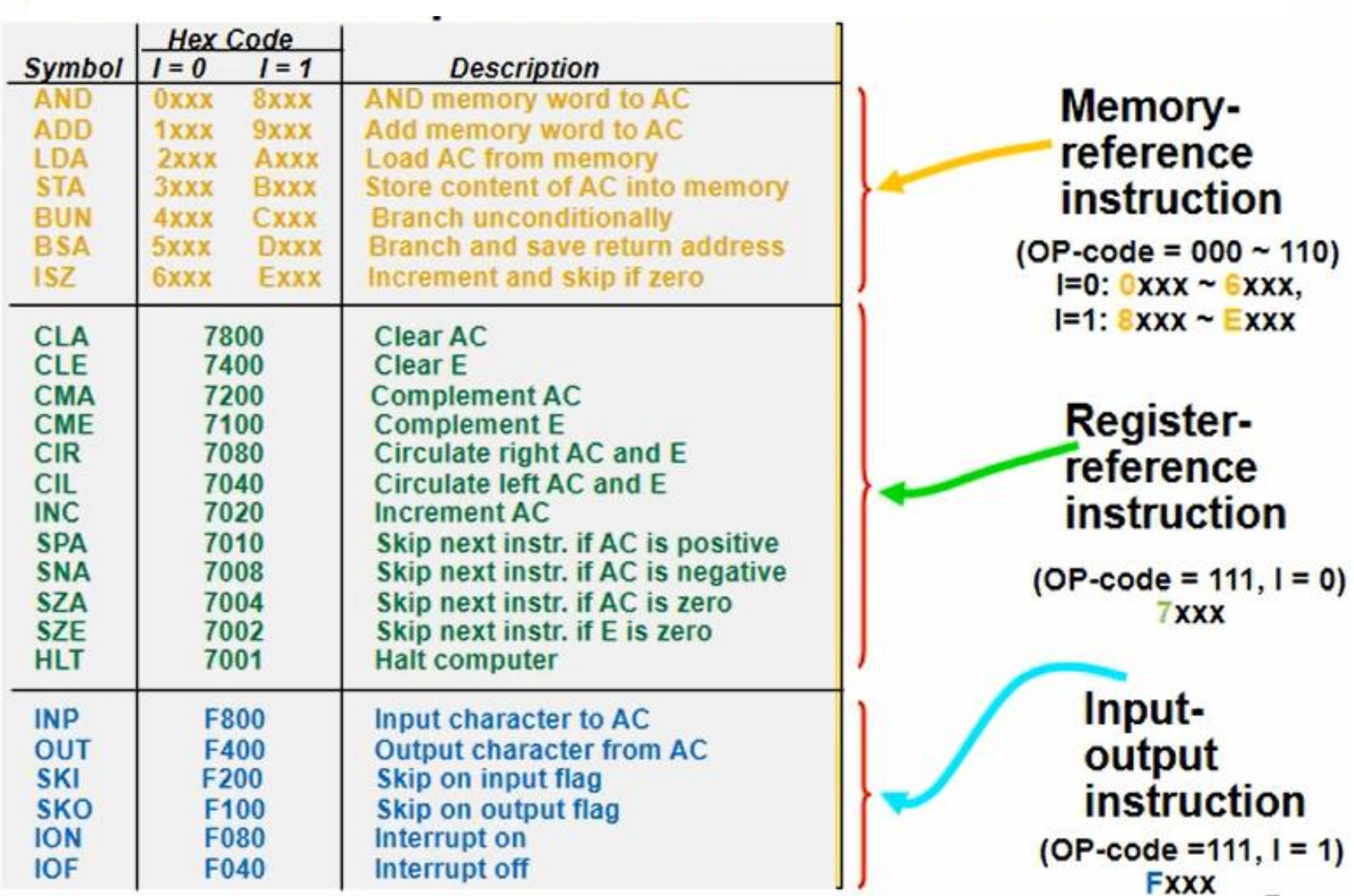
Input-Output Instructions (OP-code =111, I = 1)



Op-code (3 bits)

000
001
010
011
100
101
110
111

Basic Computer Instruction Format



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

Memory-reference instruction

(OP-code = 000 ~ 110)
I=0: 0xxx ~ 6xxx,
I=1: 8xxx ~ Exxx

Register-reference instruction

(OP-code = 111, I = 0)
7xxx

Input-output instruction

(OP-code =111, I = 1)
Fxxx

CONTROL UNIT

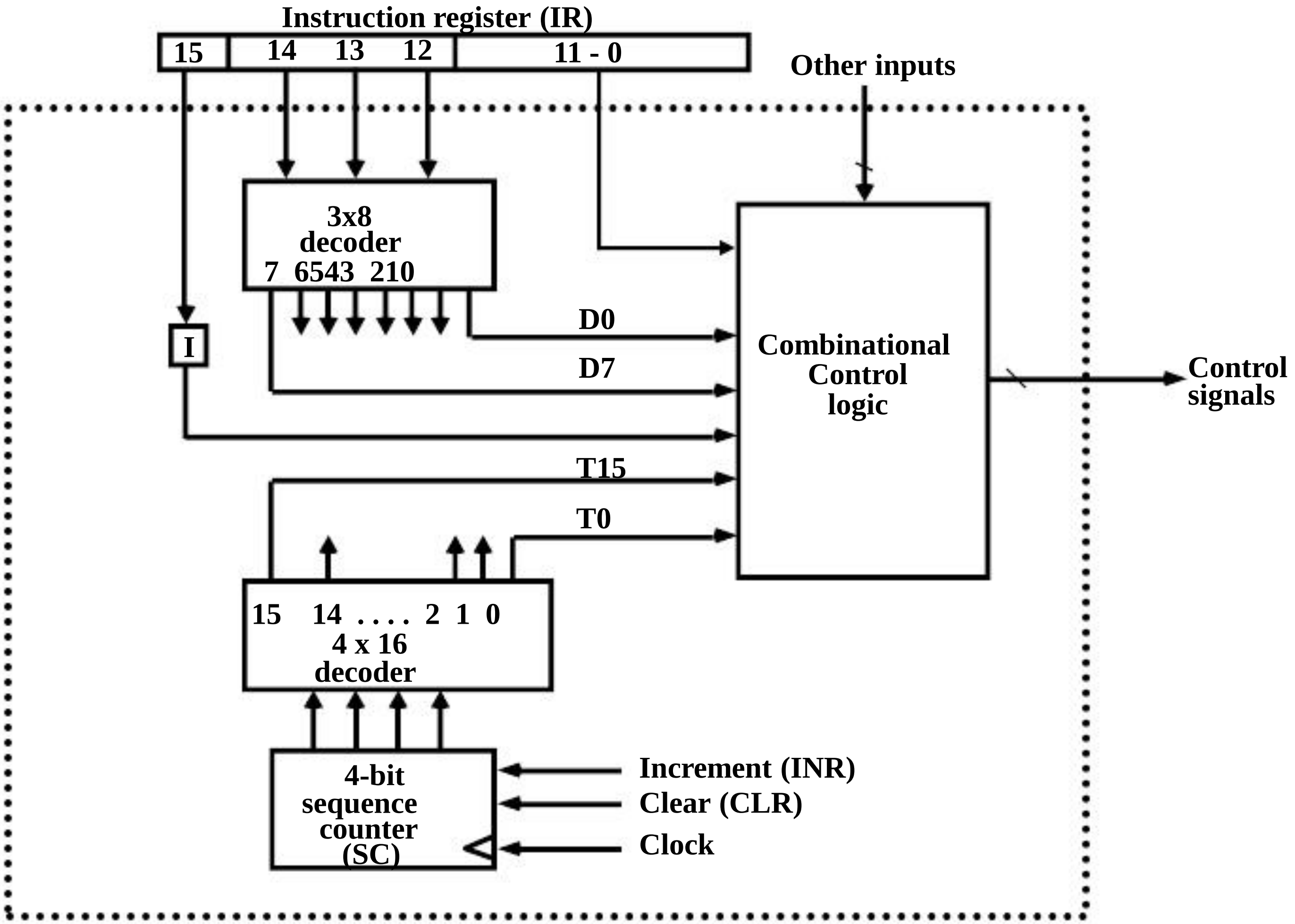
- **Control unit (CU) of a processor translates from machine instructions to the control signals for the microoperations that implement them**

- **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**

TIMING AND CONTROL

Control unit of Basic Computer

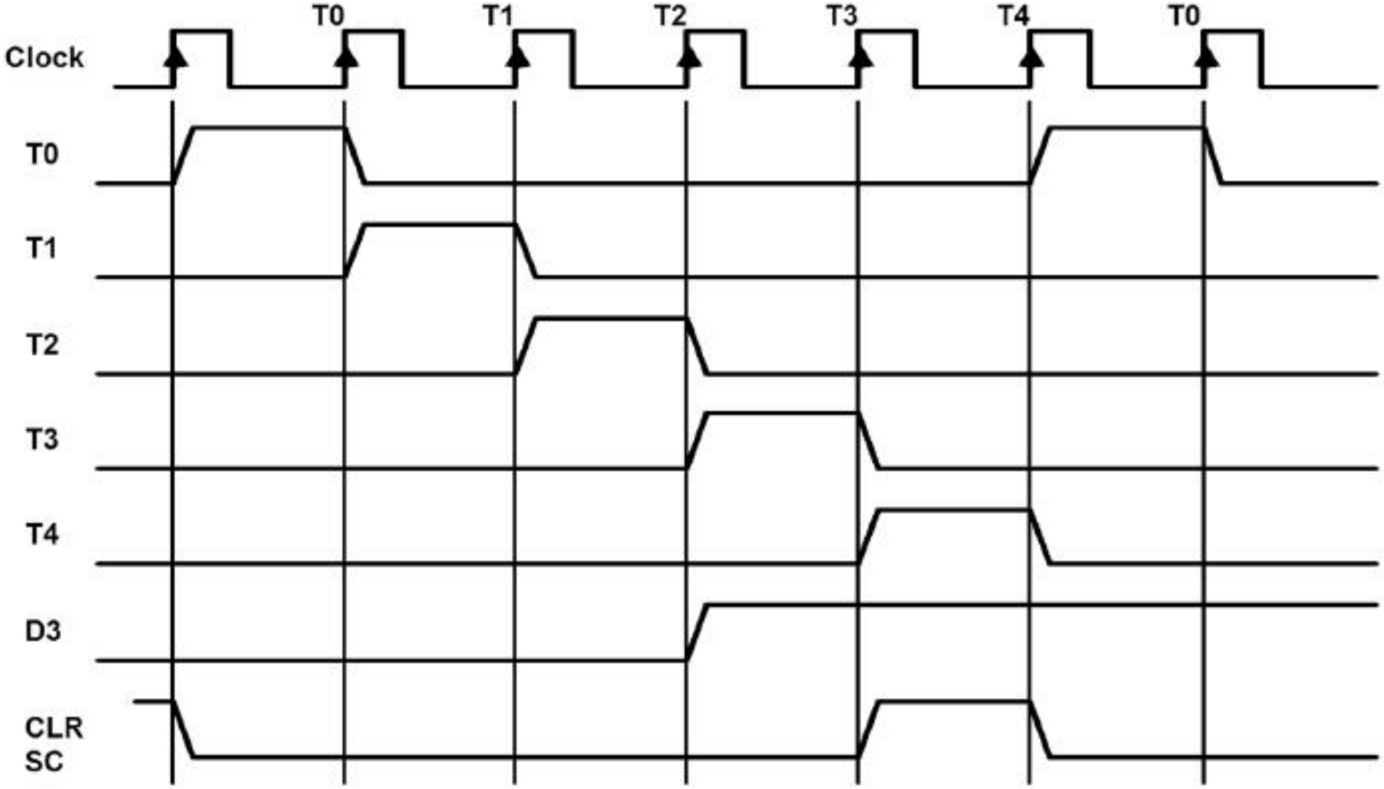


TIMING SIGNALS

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

$$D_3T_4: SC \leftarrow 0$$



INSTRUCTION CYCLE

- In Basic Computer, a machine instruction is executed in the following cycle:
  1. Fetch an instruction from memory
  2. Decode the instruction
  3. Read the effective address from memory if the instruction has an indirect address
  4. Execute the instruction

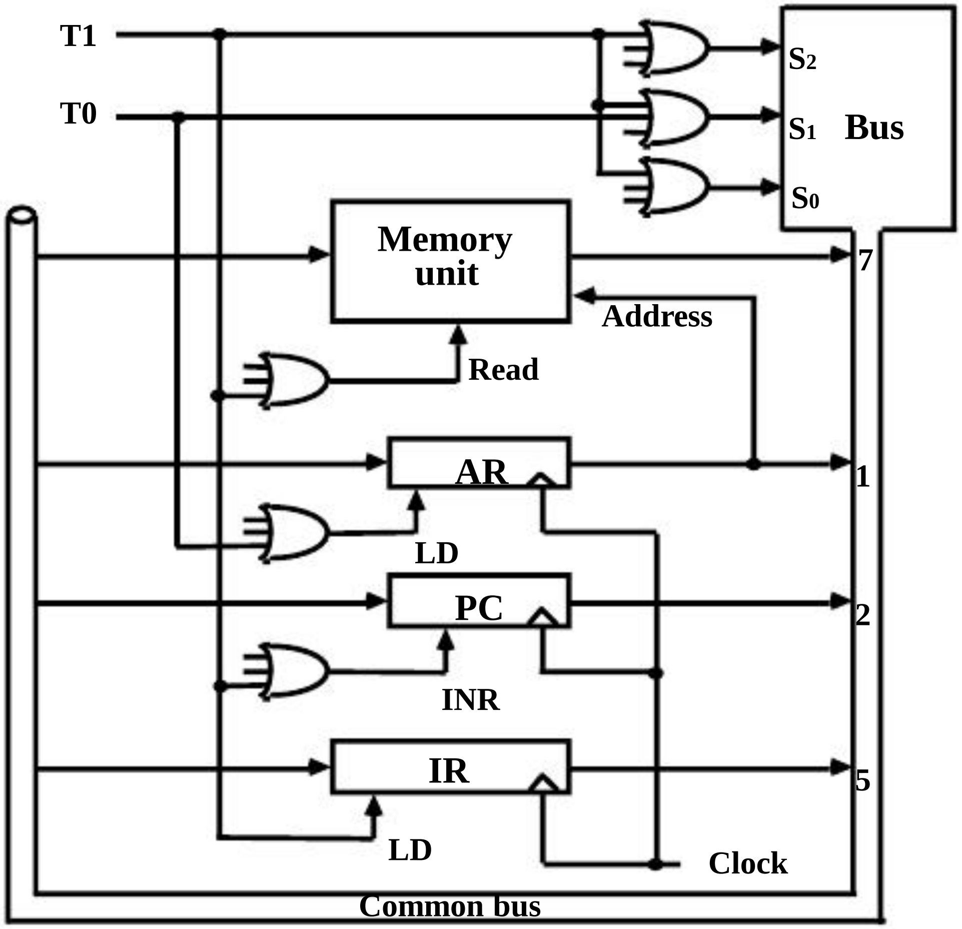
- After an instruction is executed, the cycle starts again at step 1, for the next instruction

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

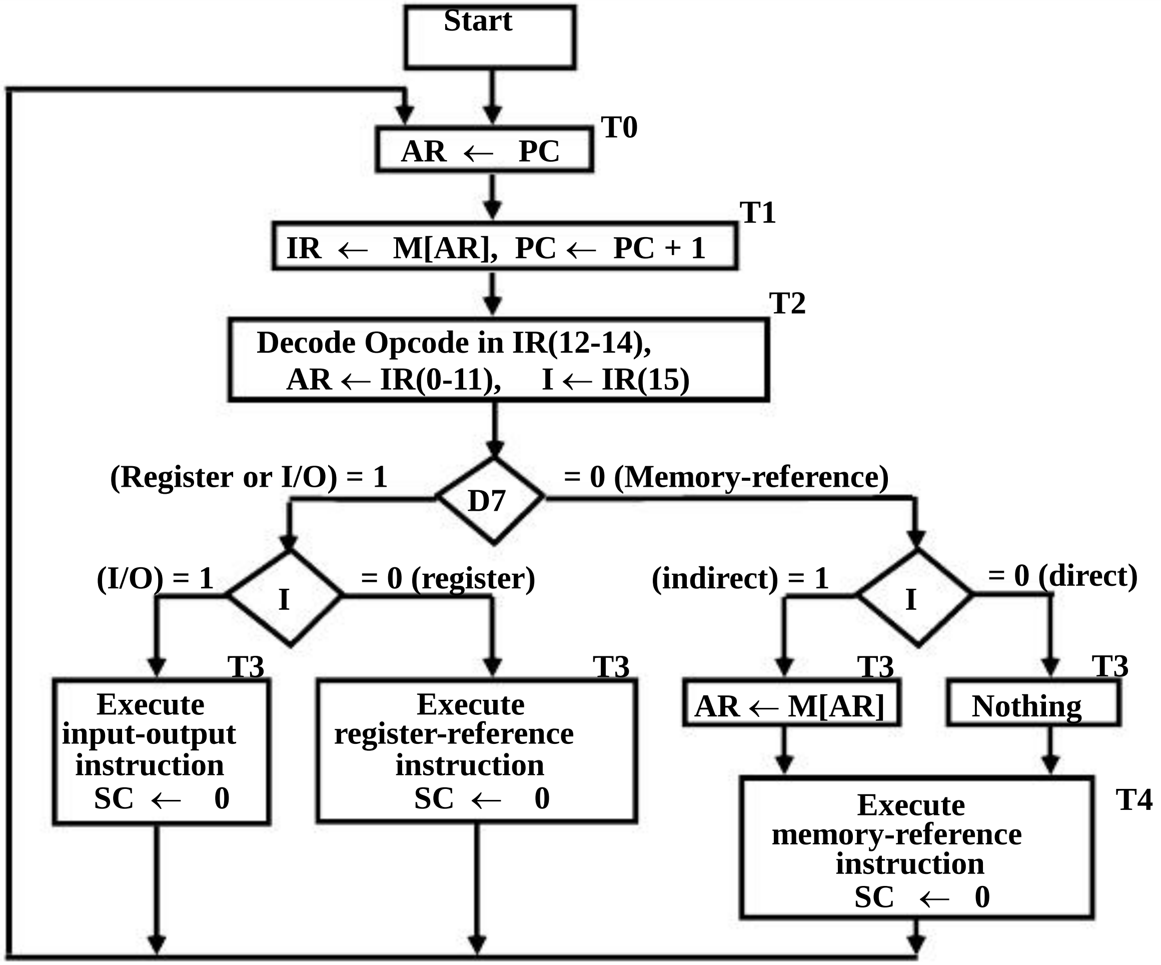
FETCH and DECODE

- **Fetch and Decode**

T0: AR ← PC (S$_0$S$_1$S$_2$=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


Start
T0
AR ← PC
T1
IR ← M[AR], PC ← PC + 1
T2
Decode Opcode in IR(12-14),
AR ← IR(0-11), I ← IR(15)
(Register or I/O) = 1
D7
= 0 (Memory-reference)
(I/O) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T3
T3
T3
T3
Execute input-output instruction SC ← 0
Execute register-reference instruction SC ← 0
AR ← M[AR]
Nothing
T4
Execute memory-reference instruction SC ← 0

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

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:	$SC \leftarrow 0$
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

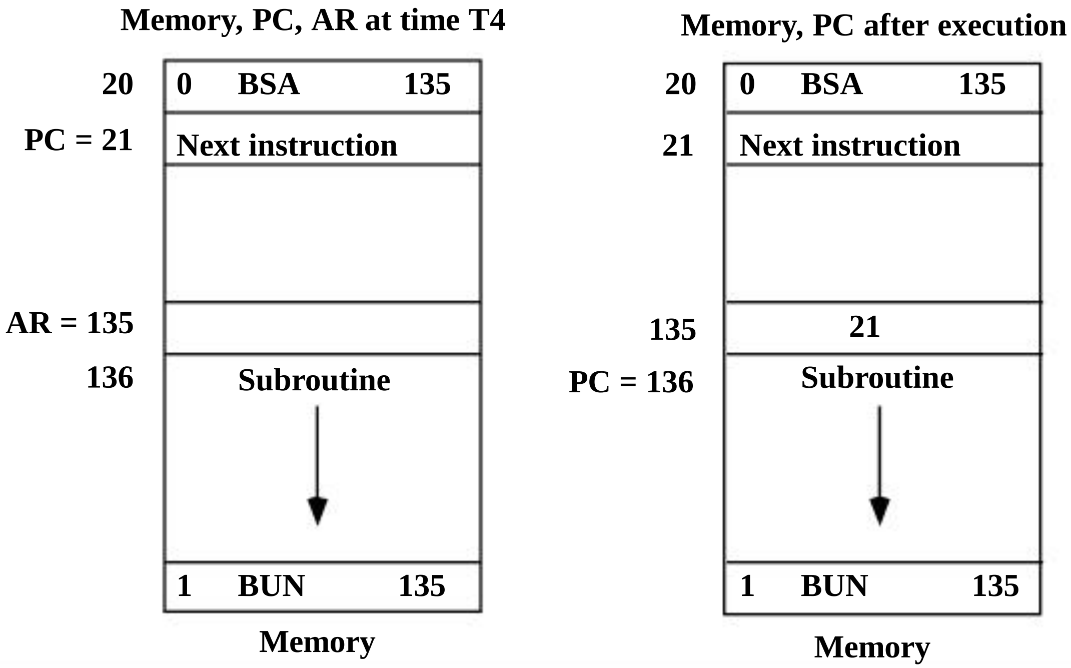
D_3T_4: $M[AR] \leftarrow AC, SC \leftarrow 0$

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

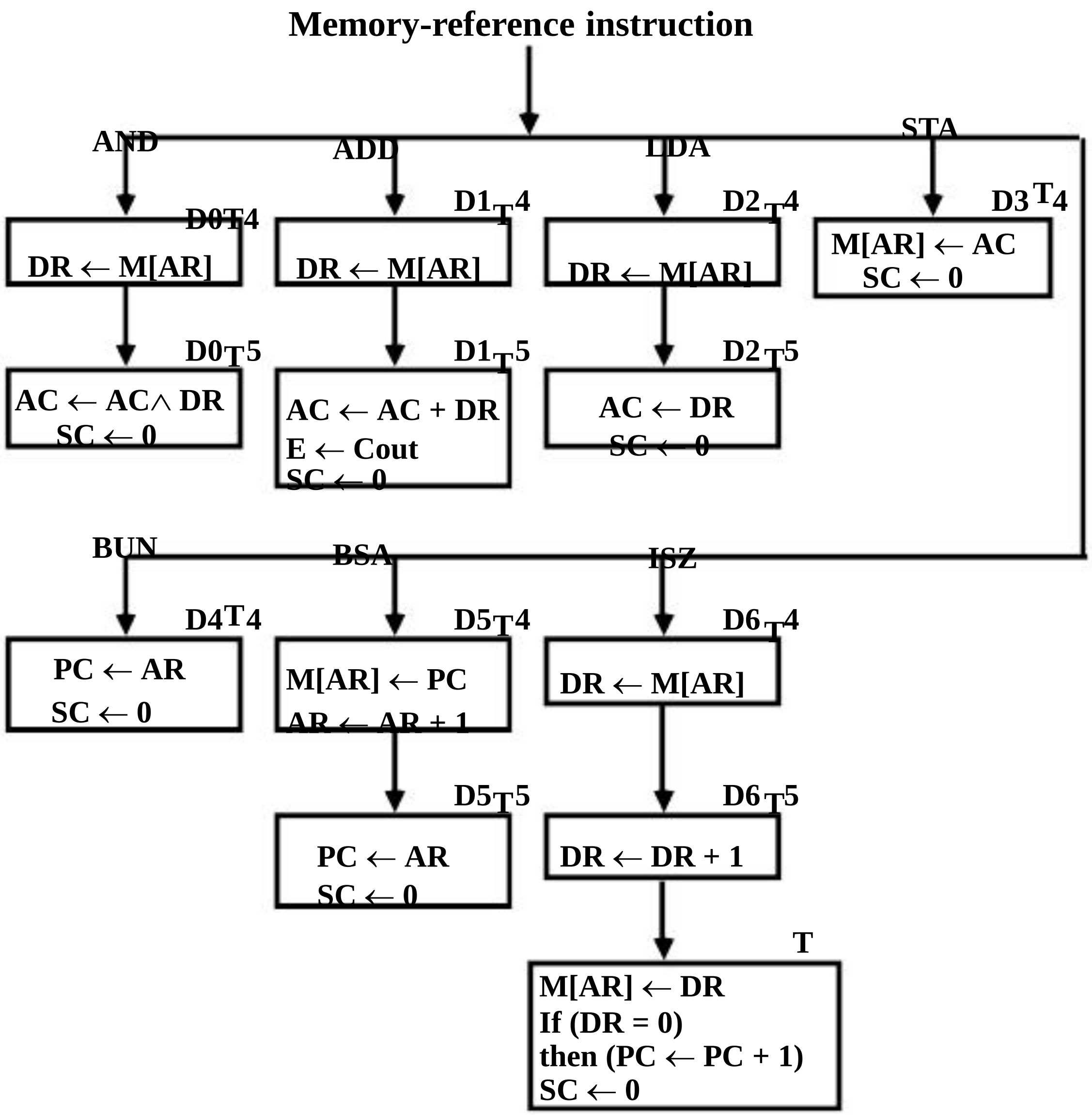
BSA:

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



Memory-reference instruction

AND

D0T4

DR ← M[AR]

D0T5

AC ← AC∧ DR
SC ← 0

ADD

D1T4

DR ← M[AR]

D1T5

AC ← AC + DR
E ← Cout
SC ← 0

LDA

D2T4

DR ← M[AR]

D2T5

AC ← DR
SC ← 0

STA

D3T4

M[AR] ← AC
SC ← 0

BUN

D4T4

PC ← AR
SC ← 0

BSA

D5T4

M[AR] ← PC
AR ← AR + 1

D5T5

PC ← AR
SC ← 0

ISZ

D6T4

DR ← M[AR]

D6T5

DR ← DR + 1

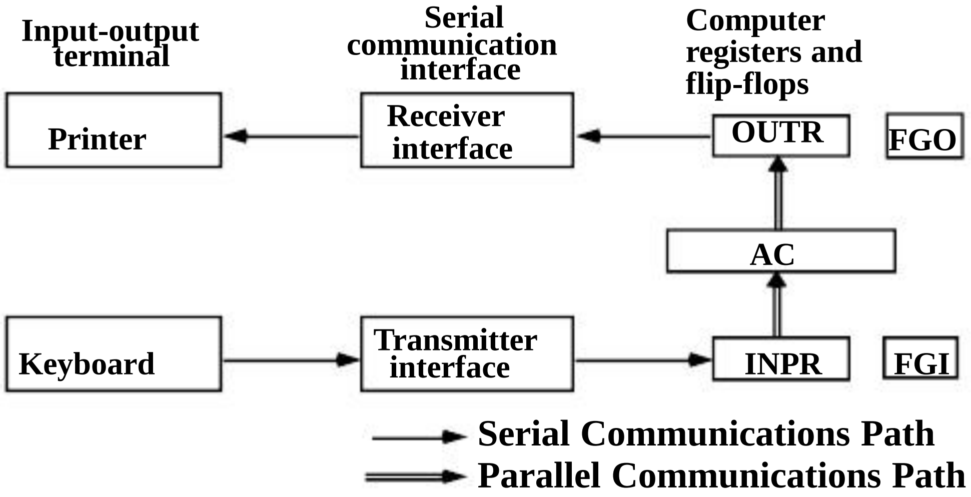
T

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**


Input-output terminal
Serial communication interface
Computer registers and flip-flops
Printer
Receiver interface
OUTR
FGO
AC
Keyboard
Transmitter interface
INPR
FGI
Serial Communications Path
Parallel Communications Path

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

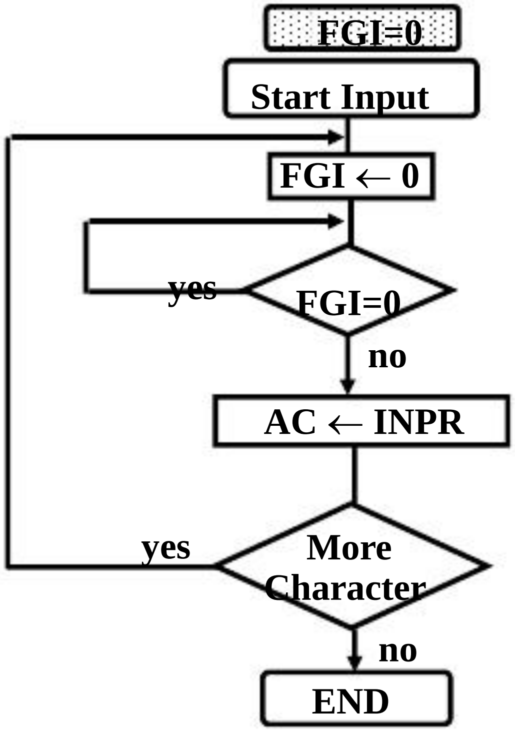
- The terminal sends and receives serial information
- The serial info. from the keyboard is shifted into INPR
- The serial info. for the printer is stored in the OUTR
- INPR and OUTR communicate with the terminal serially and with the AC in parallel.
- The flags are needed to *synchronize* the timing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

PROGRAM CONTROLLED DATA TRANSFER

-- CPU --

```
/* Input */          /* Initially FGI = 0 */
  loop:  If FGI = 0 goto loop
         AC ← INPR,  FGI ← 0

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



FGI=0

Start Input

FGI ← 0

yes

FGI=0

no

AC ← INPR

yes

More Character

no

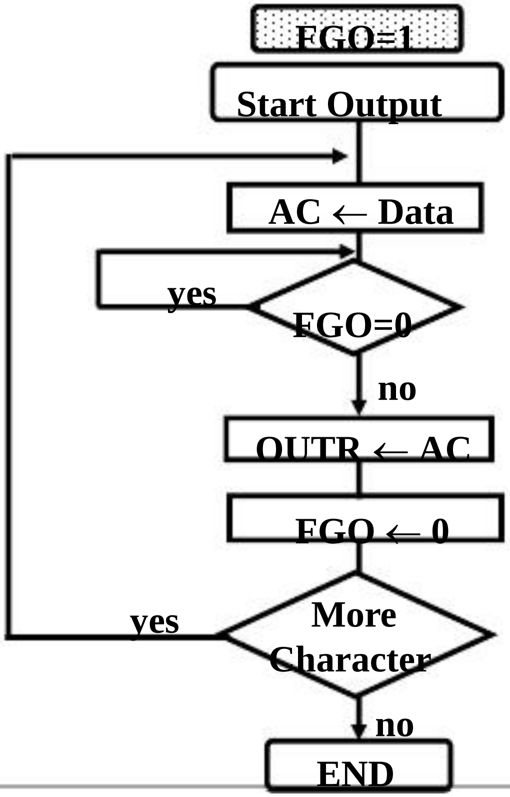
END

-- I/O Device --

```
loop: If FGI = 1 goto loop

      INPR ←  new data, FGI ← 1


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



FGO=1

Start Output

AC ← Data

yes

FGO=0

no

OUTR ← AC

FGO ← 0

yes

More Character

no

END

INPUT-OUTPUT INSTRUCTIONS

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

	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
```

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

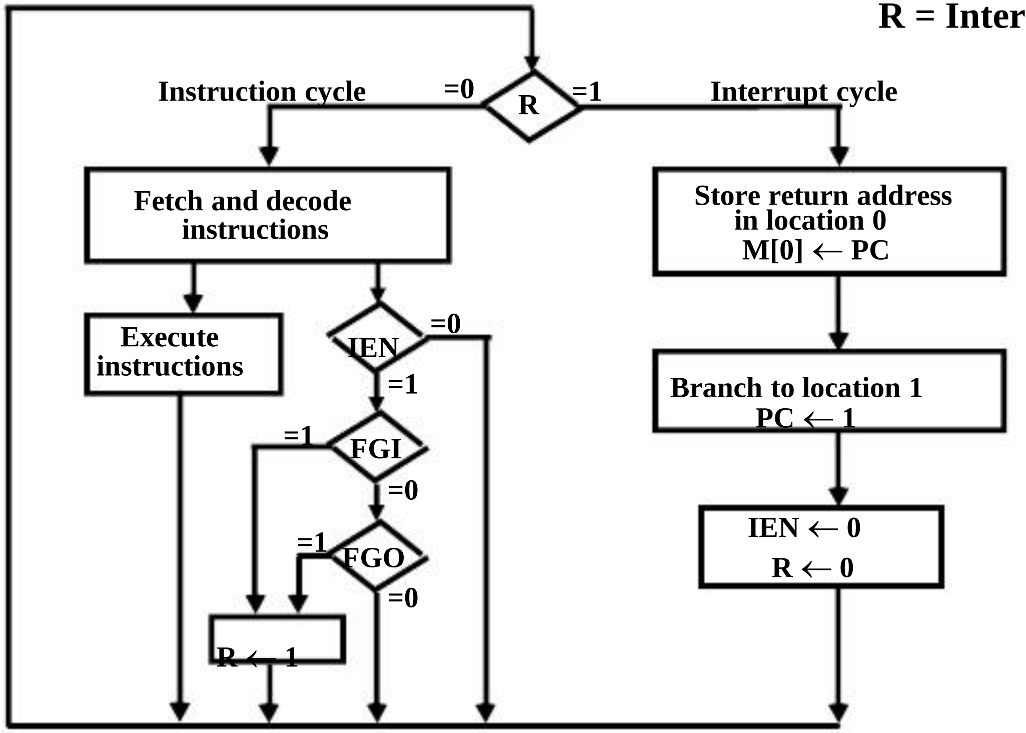
- Upon detecting an interrupt, the CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing.

*** IEN (Interrupt-enable flip-flop)**

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

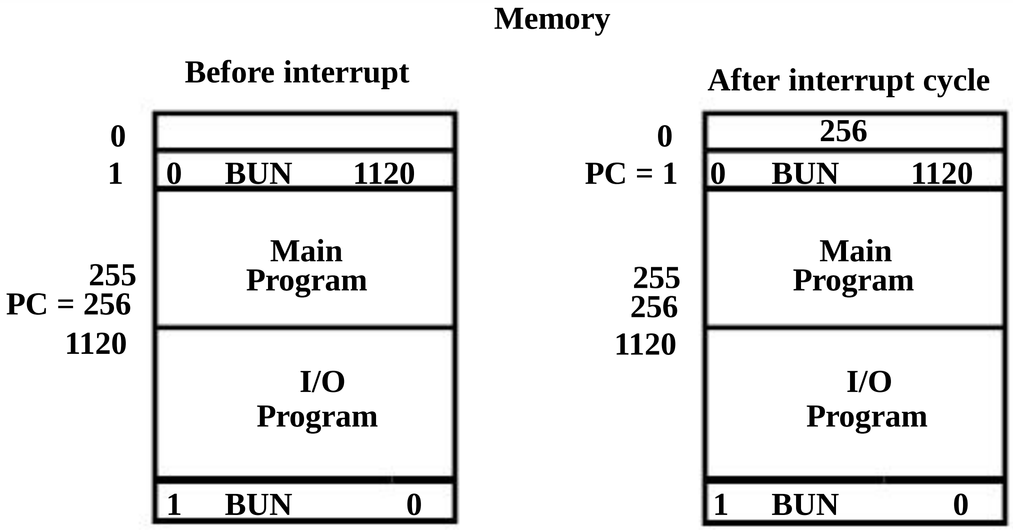
FLOWCHART FOR INTERRUPT CYCLE

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"

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

How can the CPU recognize the device requesting an interrupt ?

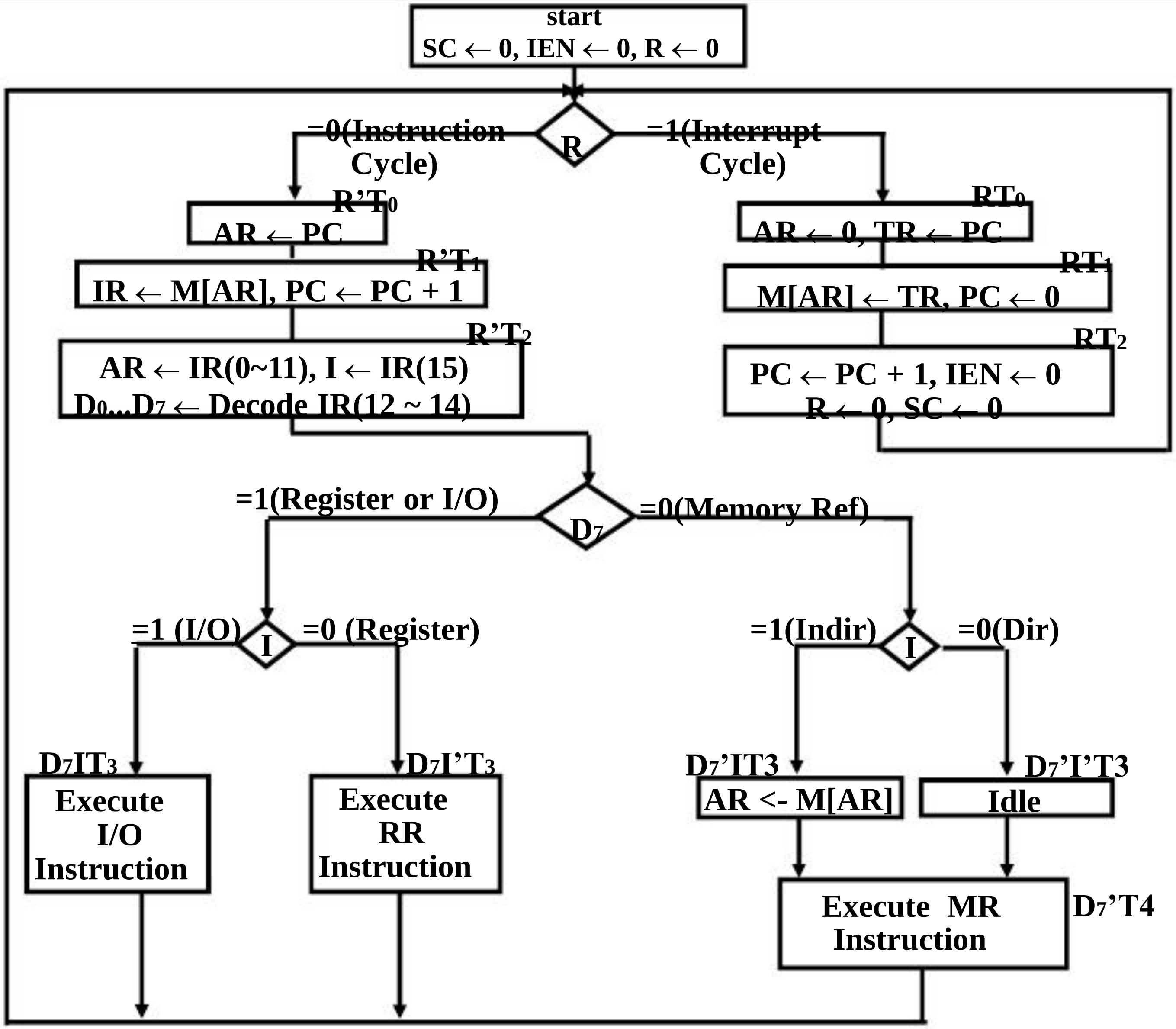
Since different devices are likely to require different interrupt service routines, how can the CPU obtain the starting address of the appropriate routine in each case ?

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



start

SC ← 0, IEN ← 0, R ← 0

R

=0(Instruction Cycle)

=1(Interrupt Cycle)

R’T_0

AR ← PC

R’T_1

IR ← M[AR], PC ← PC + 1

R’T_2

AR ← IR(0~11), I ← IR(15)

D_0...D_7 ← Decode IR(12 ~ 14)

RT_0

AR ← 0, TR ← PC

RT_1

M[AR] ← TR, PC ← 0

RT_2

PC ← PC + 1, IEN ← 0

R ← 0, SC ← 0

D_7

=1(Register or I/O)

=0(Memory Ref)

I

=1 (I/O)

=0 (Register)

D_7IT_3

Execute I/O Instruction

D_7I’T_3

Execute RR Instruction

I

=1(Indir)

=0(Dir)

D_7’IT3

AR <- M[AR]

D_7’I’T3

Idle

Execute MR Instruction

D_7’T4

COMPLETE COMPUTER DESCRIPTION

Microoperations

Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR],\ PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D0, ..., D7 \leftarrow \text{Decode } IR(12 \sim 14)$, $AR \leftarrow IR(0 \sim 11),\ I \leftarrow IR(15)$
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:	$M[AR] \leftarrow DR,\ \text{if}(DR=0) \text{ then } (PC \leftarrow PC + 1),$ $SC \leftarrow 0$

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