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

- **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**

  yani ye shomare mese address vase oon word dashte bashim.

- **Each word is 16 bits long**

CPU

RAM

0

15　　　　0

4095
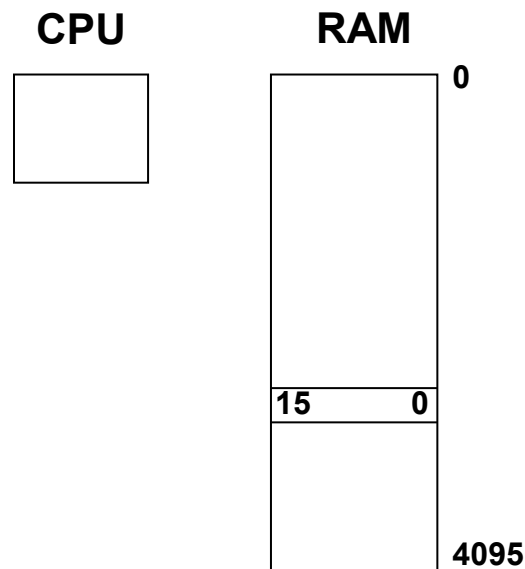
INSTRUCTIONS

- **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**
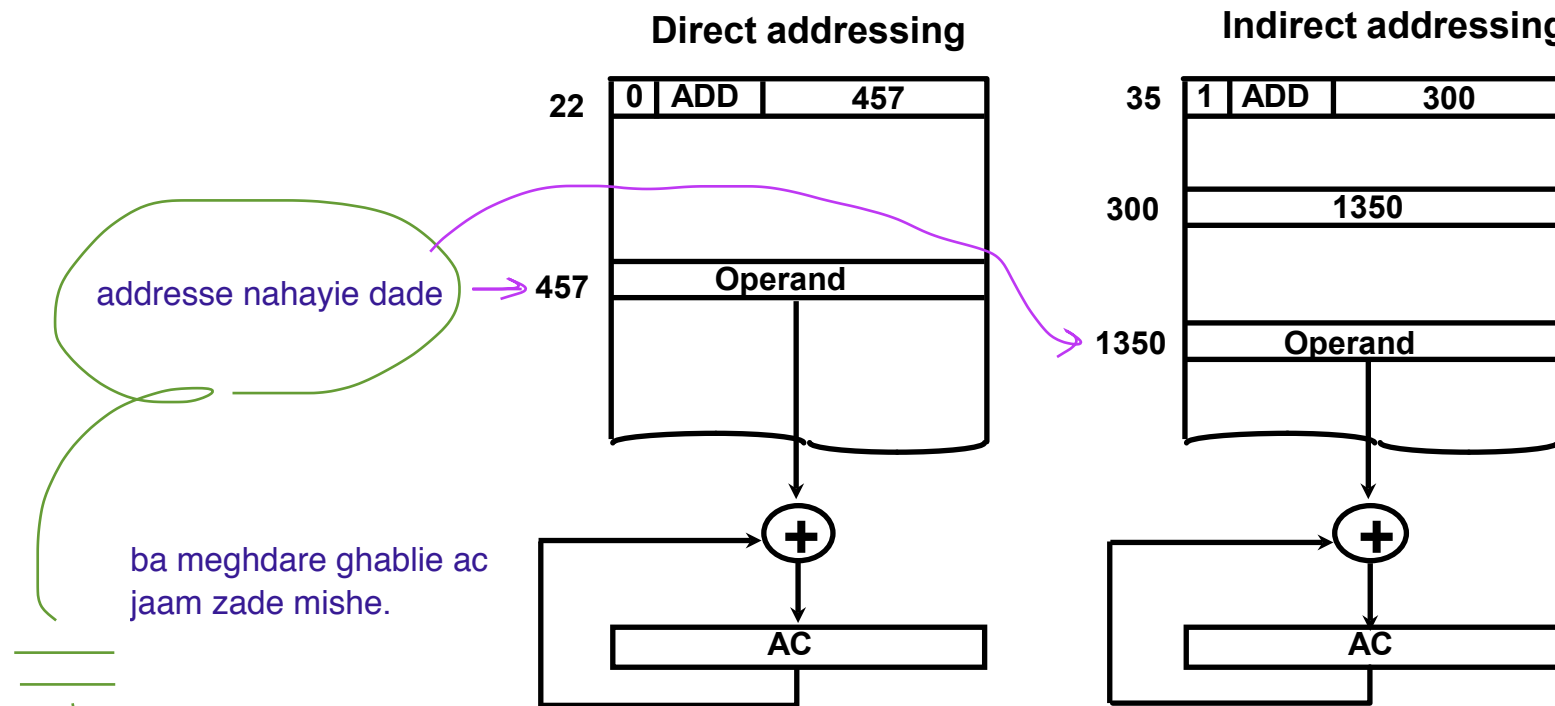
INSTRUCTION FORMAT

- ## 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 —> 12bit

- ## 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
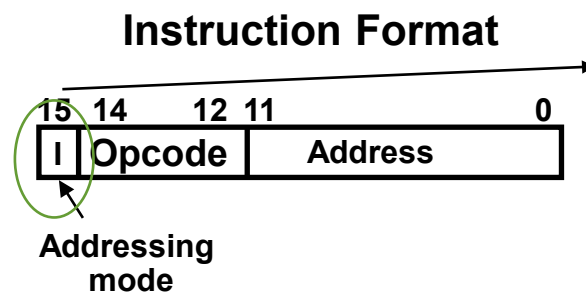
inja faqat address dahi hafezeyi mostaghim va gheire mostaghim made nazar boode be khatere hamin ye bit vase addressing mode gozashte.

15	14	12	11		0
I	Opcode		Address		

Addressing mode

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

addresse nahayie dade → 457 | Operand |

ba meghdare ghablie ac jaam zade mishe.

+

AC

Indirect addressing

35	1	ADD	300

| 300 | 1350 |

| 1350 | Operand |

+

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**

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)**
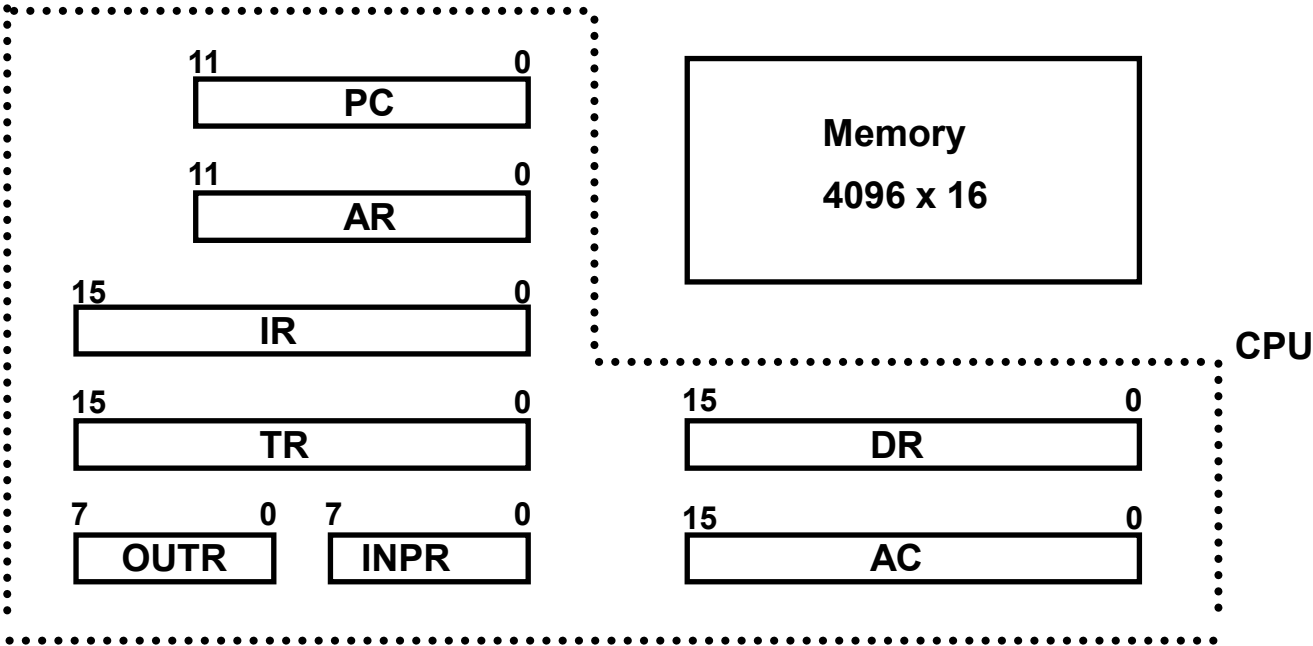
PROCESSOR REGISTERS

- **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 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

11	PC	0

11	AR	0

Memory

4096 x 16

15	IR	0

CPU

15	TR	0

15	DR	0

7	OUTR	0

7	INPR	0

15	AC	0

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

- **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
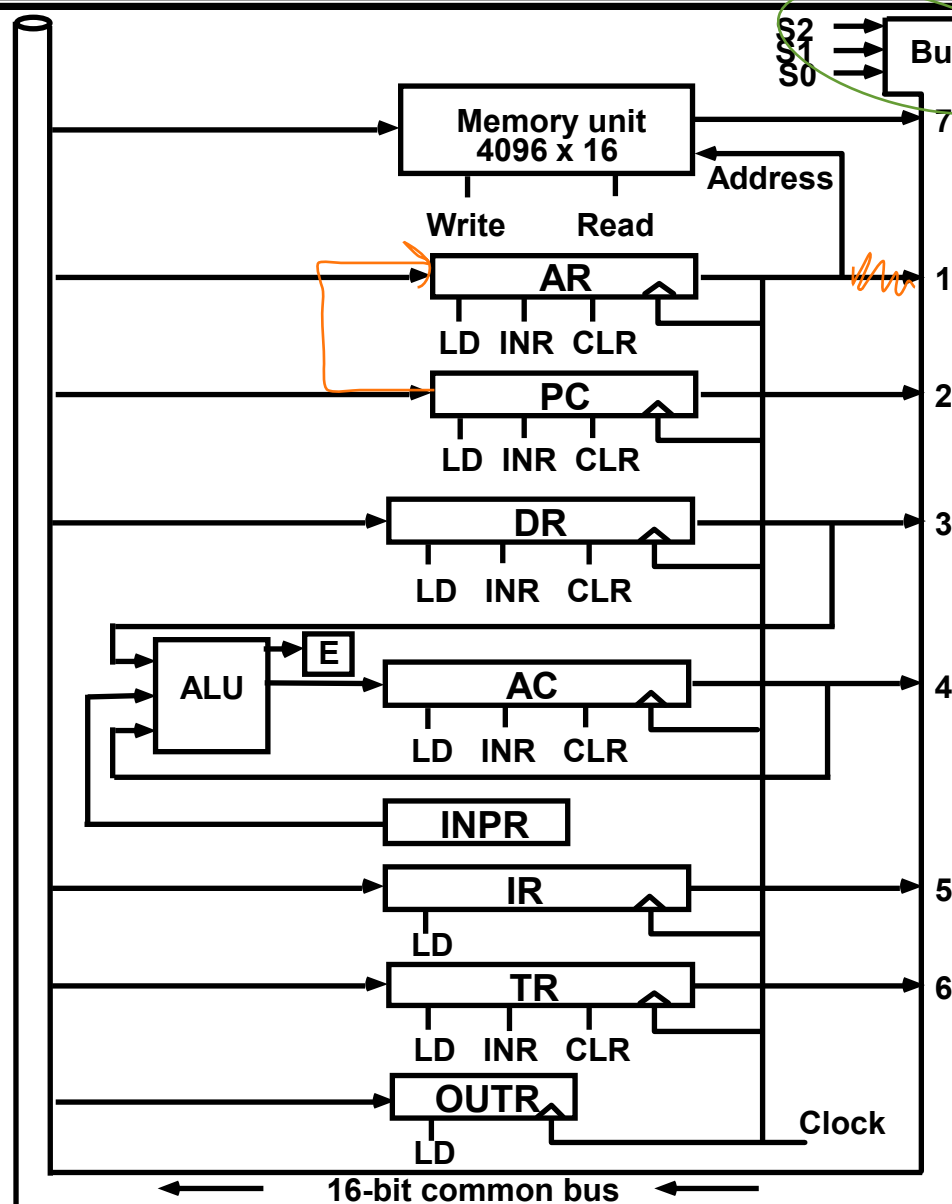
tamamie sabat ha be bus meghdar mirizan az bus meghdar migiran.

tamamie sabbat ha
paye load = LOAD
clear = CLR (sefr shodan)
increment = INR (afzayesh)
daran.
tamamie sabbat ha clk daran va hassas be labe hastan.
tak take in signal haro cpu moshakhas mikone.
va signal haye read write va khotote selecte mux.

cpu —> madare kamelan tarkibi

be in ghesmat migan data path.(be kole in meshkie)

in adad hayi ke joloshon neveshtim , baraye muxe.

COMMON BUS SYSTEM
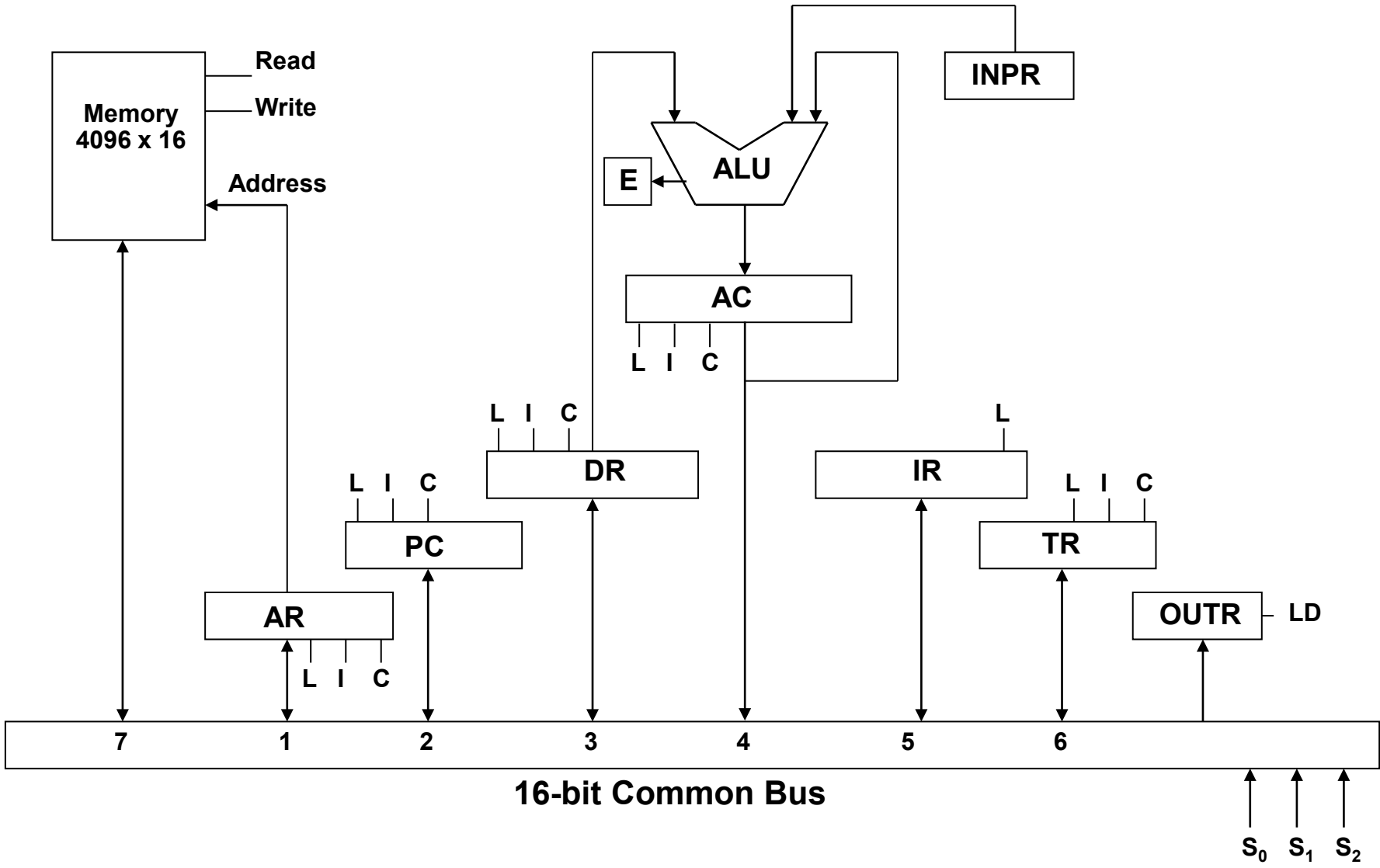
COMMON BUS SYSTEM

- **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**

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	
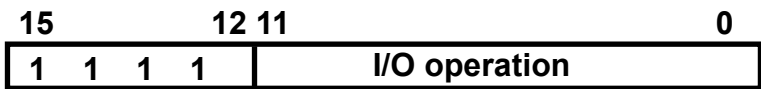
BASIC COMPUTER INSTRUCTIONS

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

hafezei

sabati

IO

7
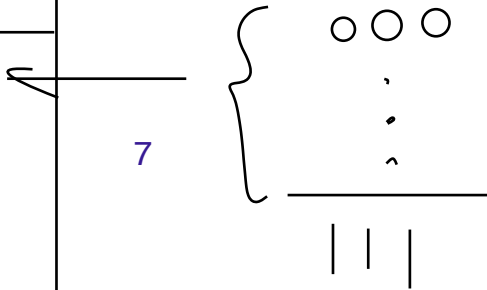
12

6

pas kolllllan 25 ta dastoorl darim.

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**

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**   sim bandi shode   sorate bala + tavan kam
    - **CU is made up of sequential and combinational circuits to generate the control signals**

- *Microprogrammed* **Control**   rizbarname pazir --> sorat kam + tavan ziad + ghabele barname pazir
    - **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

be sharte dastoore hafezeyi boodan:

Instruction register (IR)

15	14	13	12	11 - 0

Other inputs

3 x 8 decoder

7 6 5 4 3 2 1 0

I

D_0

D_7 :

Combinational Control logic

T_{15}

T_0

madare

tarkibi

Control signals

tamamie signal haye

controli

15	14	. . . .	2	1	0

4 x 16 decoder

4-bit sequence counter (SC)

Increment (INR)

Clear (CLR)

Clock

sequence counter

tavali gar - shomarande

tavali
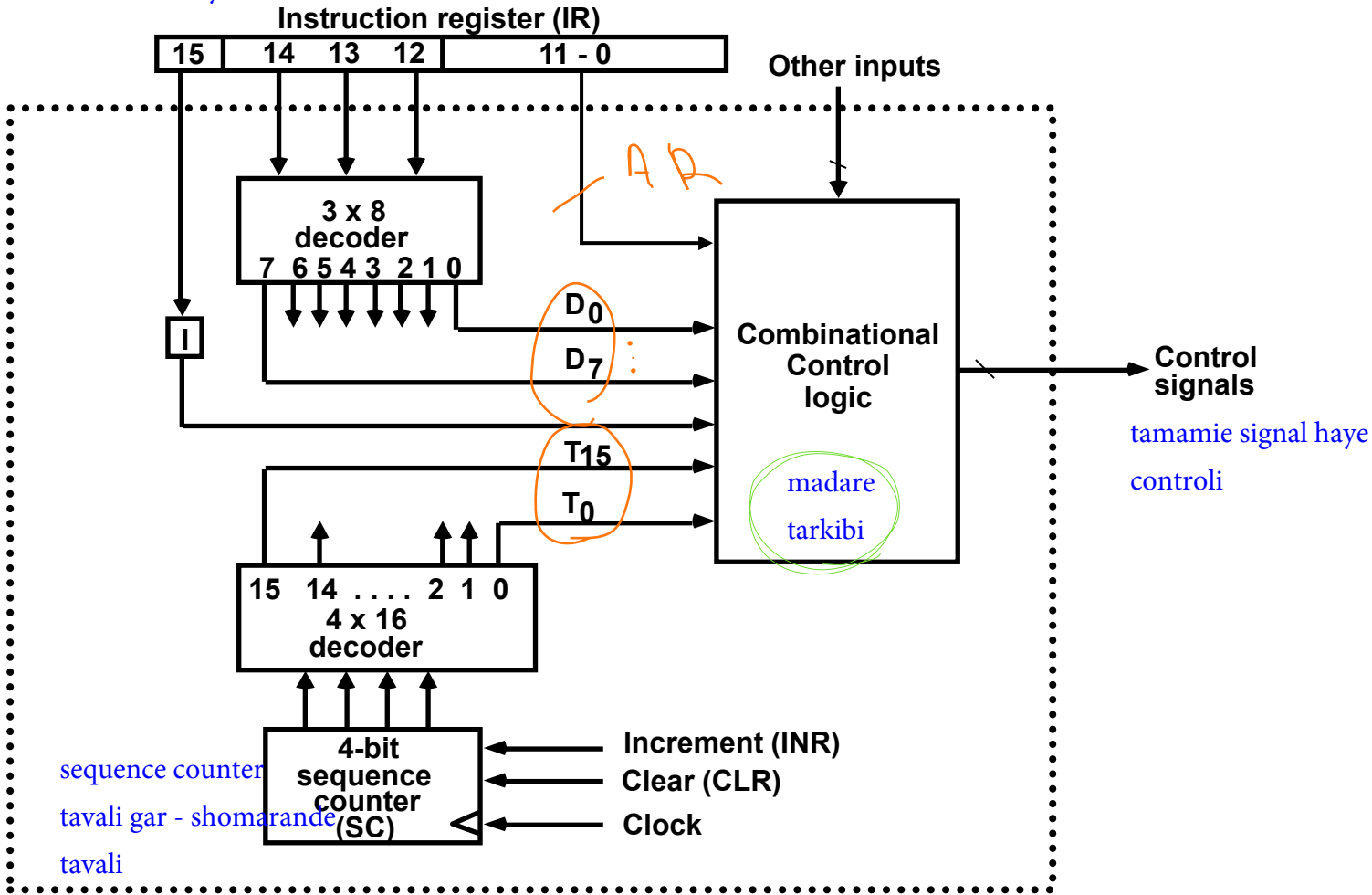
TIMING SIGNALS

- **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.

$$D_3T_4: SC \leftarrow 0$$
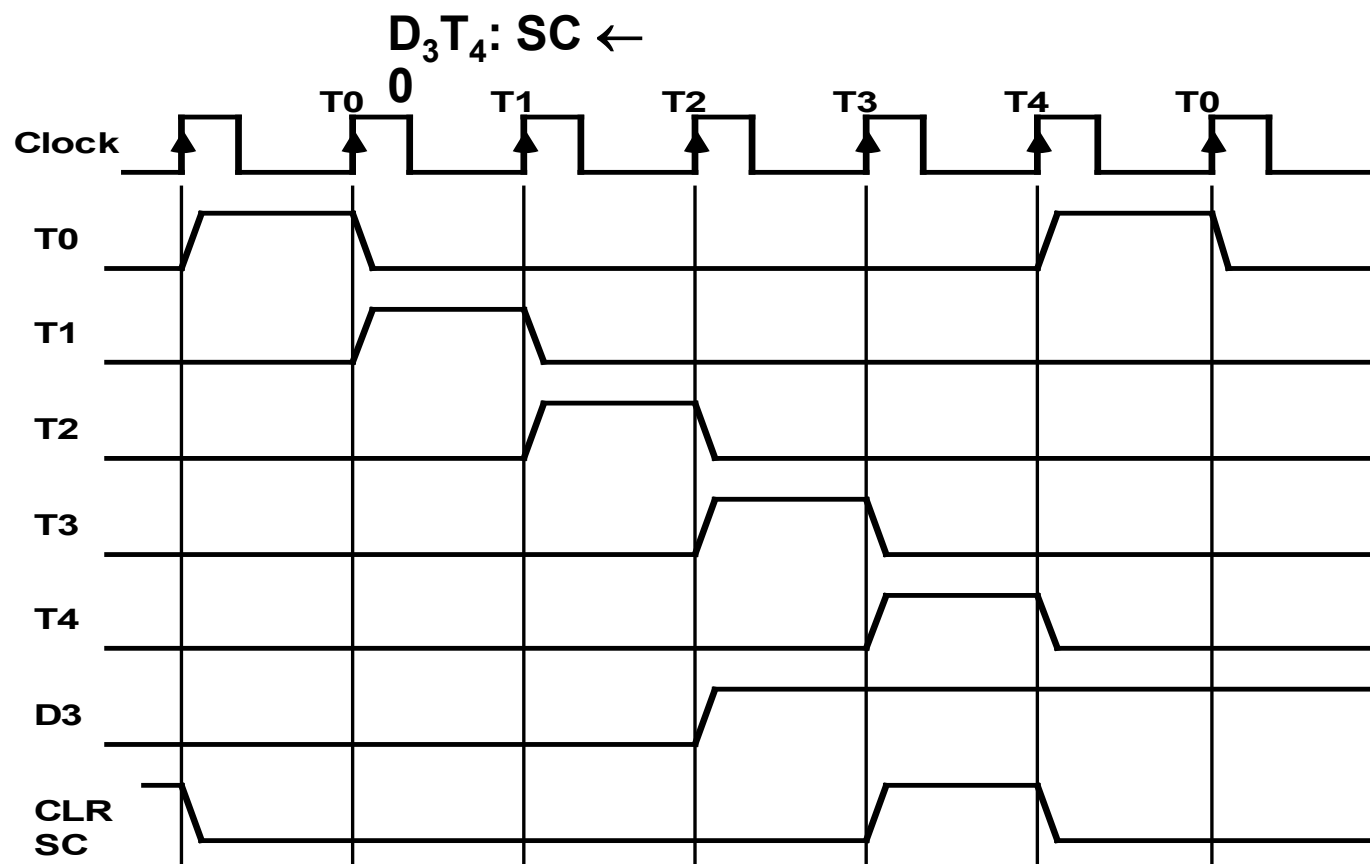
INSTRUCTION CYCLE

- **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)

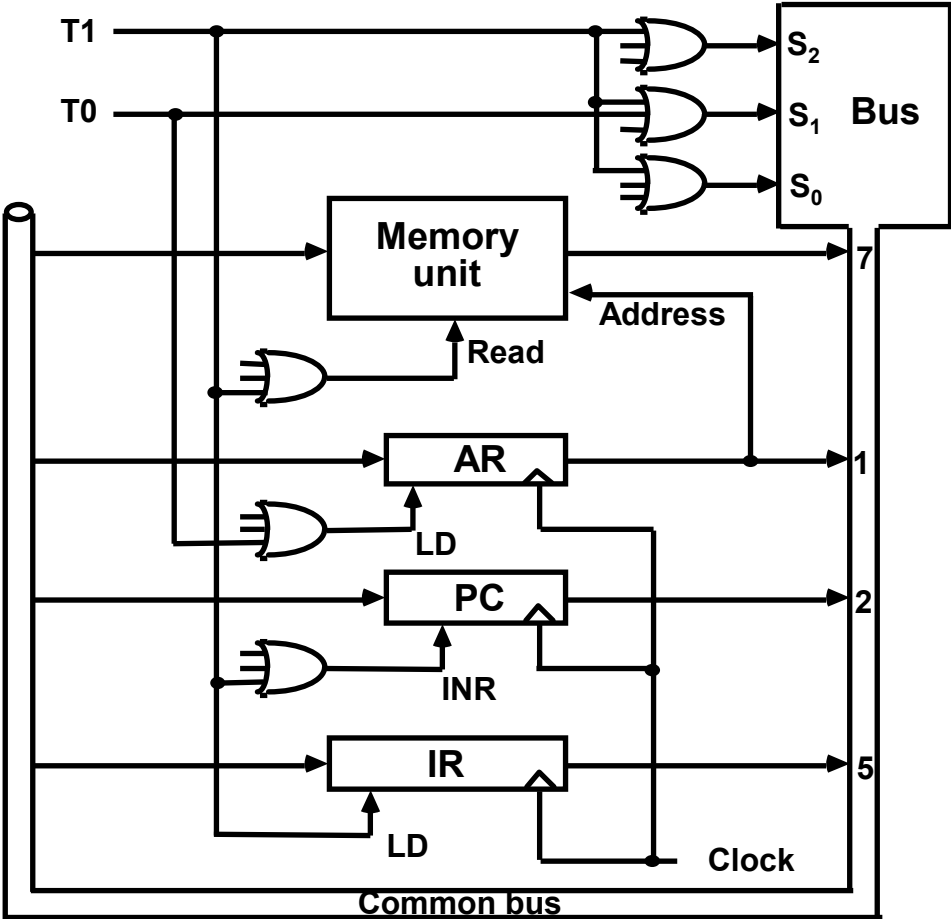T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)

IR

hamoon gam 6 ro anjam dade

vali zoodtar

addressing mode

tavazi darim , 3 ta movazi

DETERMINE THE TYPE OF INSTRUCTION

```
        Start
        SC ← 0
```

jahayi ke moraba ya mostatil darim --> hamzamani

T0   `AR ← PC`   *sc shode T0*

goosheye mostatil ha : zaman ha

dastoorate dakhele mostatil : movazi

T1   `IR ← M[AR], PC ← PC + 1`

T2   `Decode Opcode in IR(12-14), AR ← IR(0-11), I ← IR(15)`

no e dastoor olamal

(Register or I/O) = 1   D7   **= 0 (Memory-reference)**

yani pattern haye dgaro didim

yani 111...1 didim

(I/O) = 1   I   **= 0 (register)**   **(indirect) = 1**   I   **= 0 (direct)**

T3   Execute input-output instruction   `SC ← 0`

T3   Execute register-reference instruction   `SC ← 0`

T3   `AR ← M[AR]`

T3   Nothing

ba in kare bihoode , zaman ro gerefte--> be khatere sadegi dar tarahi

T4   Execute memory-reference instruction   `SC ← 0`

ejraye dastoorol amal

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.

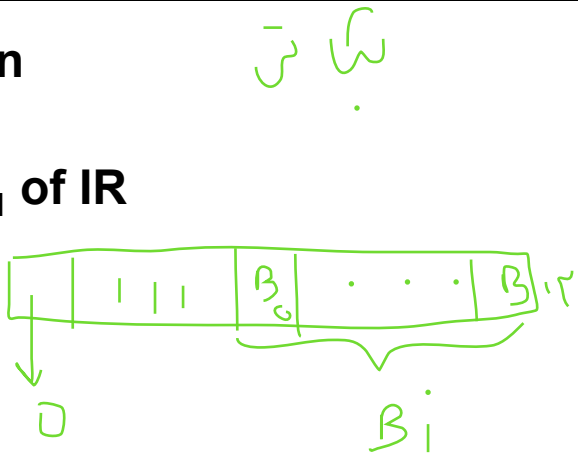*hafezayie gheire mostaghim yebar khoondane bishtar dare.*

lozi = if

REGISTER REFERENCE INSTRUCTIONS

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

d7 ro ba I' and koni yani inst sabbatie.

$r = D_7 I'T_3$ => 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

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 $PC \leftarrow PC+1$

operand

- **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   ghabele movazi sazi nistan.

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

   M[AR] ← PC, PC ← 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
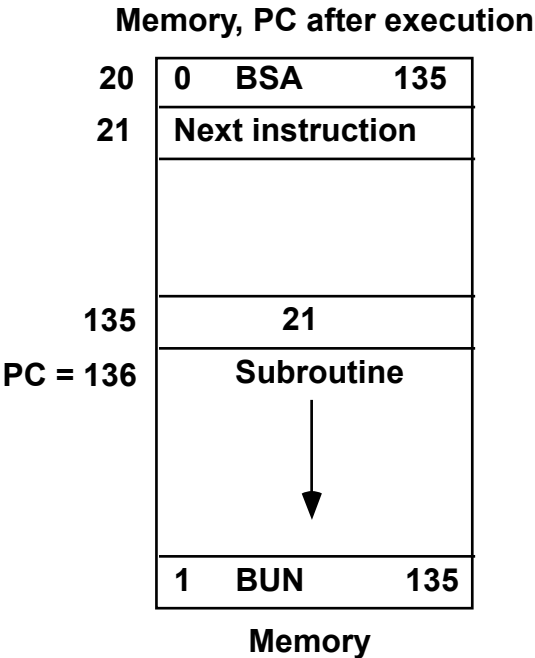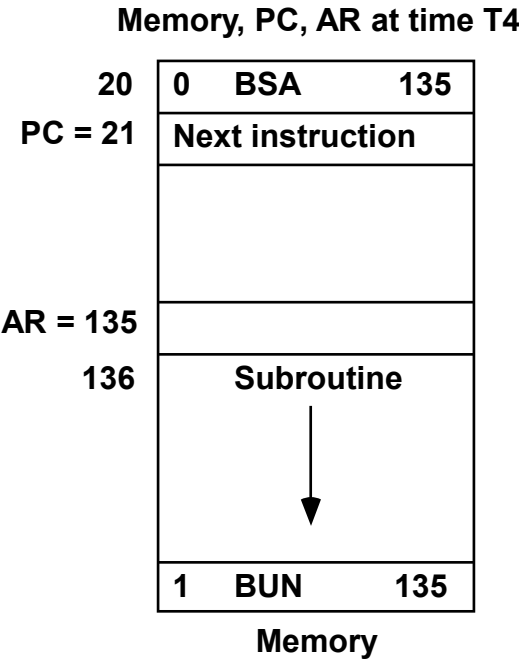
Memory

MEMORY REFERENCE INSTRUCTIONS

BSA:

D_5T_4: $M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
D_5T_5: $PC \leftarrow AR$, $SC \leftarrow 0$

ISZ: Increment and Skip-if-Zero

D_6T_4: $DR \leftarrow M[AR]$
D_6T_5: $DR \leftarrow DR + 1$
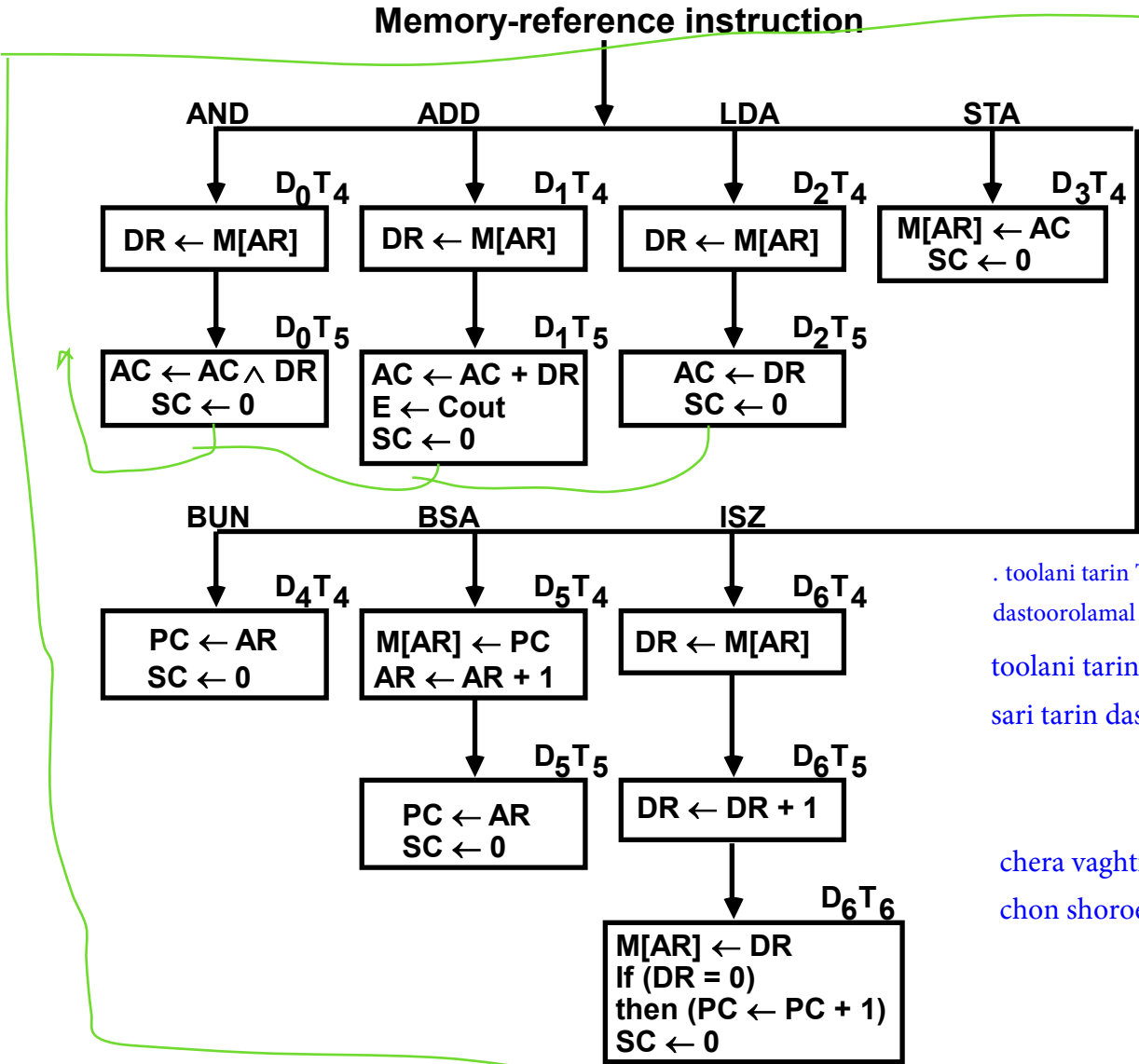D_6T_4: $M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

FLOWCHART FOR MEMORY REFERENCE INSTRUCTIONS

Memory-reference instruction

AND   **ADD**   **LDA**   **STA**

$D_0 T_4$
```
DR ← M[AR]
```

$D_1 T_4$
```
DR ← M[AR]
```

$D_2 T_4$
```
DR ← M[AR]
```

$D_3 T_4$
```
M[AR] ← AC
SC ← 0
```

$D_0 T_5$
```
AC ← AC ∧ DR
SC ← 0
```

$D_1 T_5$
```
AC ← AC + DR
E ← Cout
SC ← 0
```

$D_2 T_5$
```
AC ← DR
SC ← 0
```

BUN   **BSA**   **ISZ**

$D_4 T_4$
```
PC ← AR
SC ← 0
```

$D_5 T_4$
```
M[AR] ← PC
AR ← AR + 1
```

$D_6 T_4$
```
DR ← M[AR]
```

$D_5 T_5$
```
PC ← AR
SC ← 0
```

$D_6 T_5$
```
DR ← DR + 1
```

$D_6 T_6$
```
M[AR] ← DR
If (DR = 0)
then (PC ← PC + 1)
SC ← 0
```
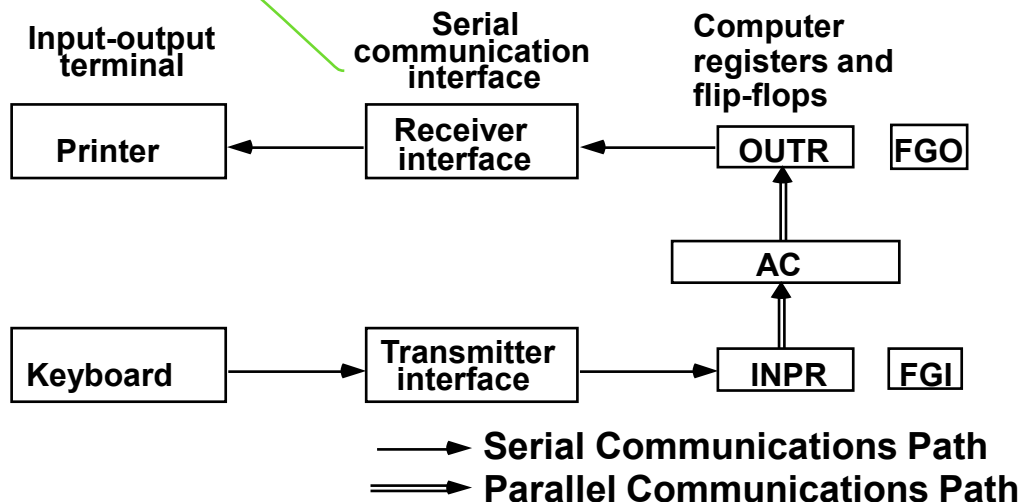
(handwritten annotations in blue:)

memory
instruction
excecute

. toolani tarin T = tedade clocke lazem baraye ejraye yek
dastoorolamal

toolani tarin dastoorol amale hafezeyi : ISZ

sari tarin dastoor = STA

chera vaghti 4 clocke migim 5 clock ??
chon shoroesh az t0 e.

(handwritten green:) clok   7   5   or bun

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
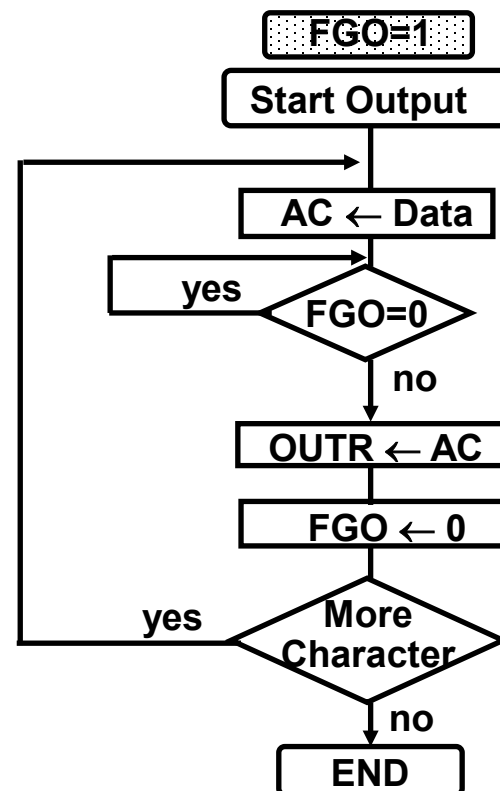　　　　　AC ← INPR, FGI ← 0

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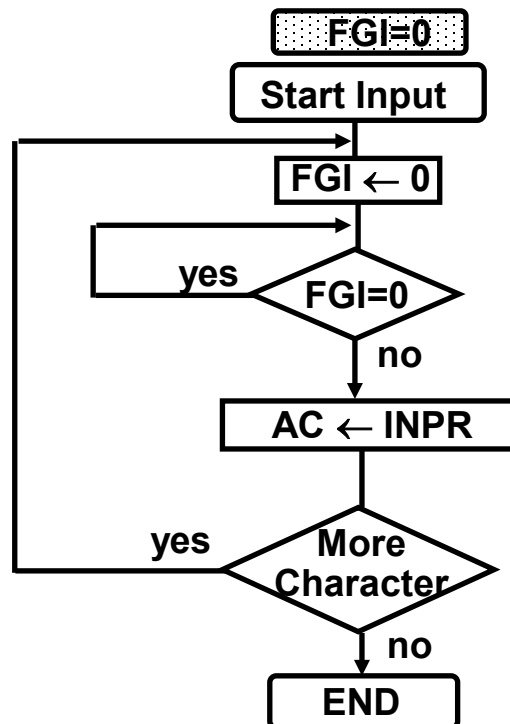
-- I/O Device --

loop: If FGI = 1 goto loop
　　　INPR ← new data, FGI ← 1

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

INPUT-OUTPUT INSTRUCTIONS

$D_7 I T_3 = p$

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

	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
```
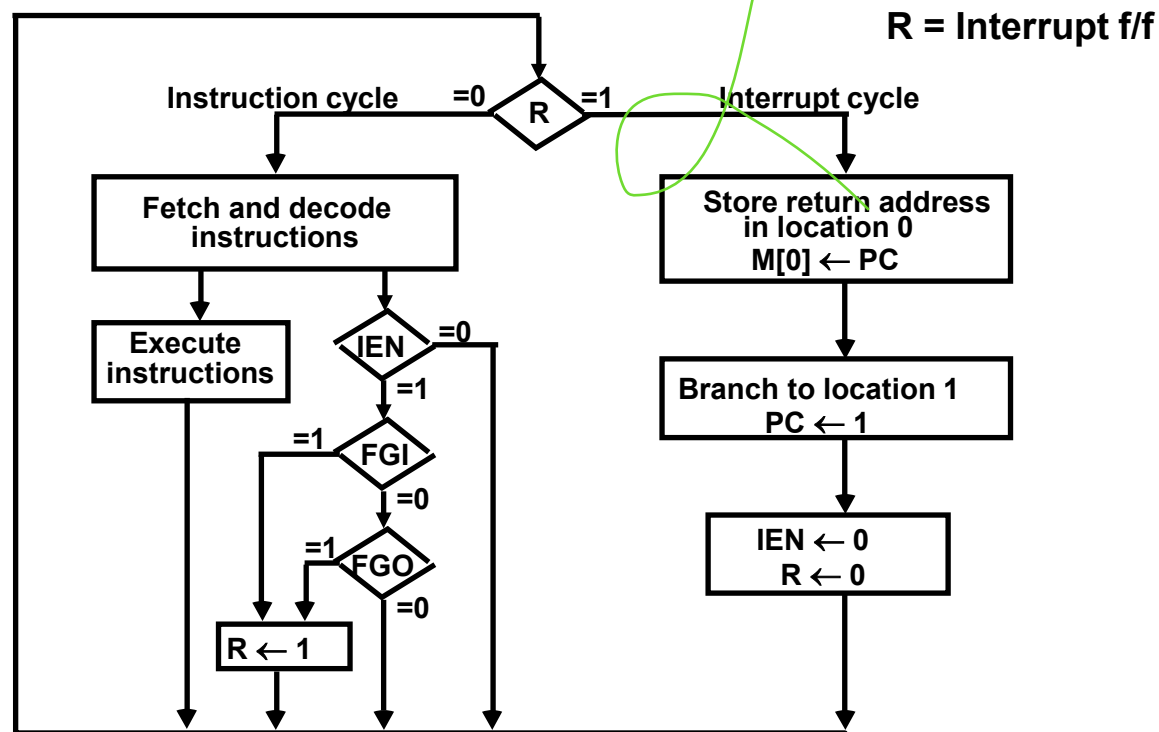
INTERRUPT INITIATED INPUT/OUTPUT

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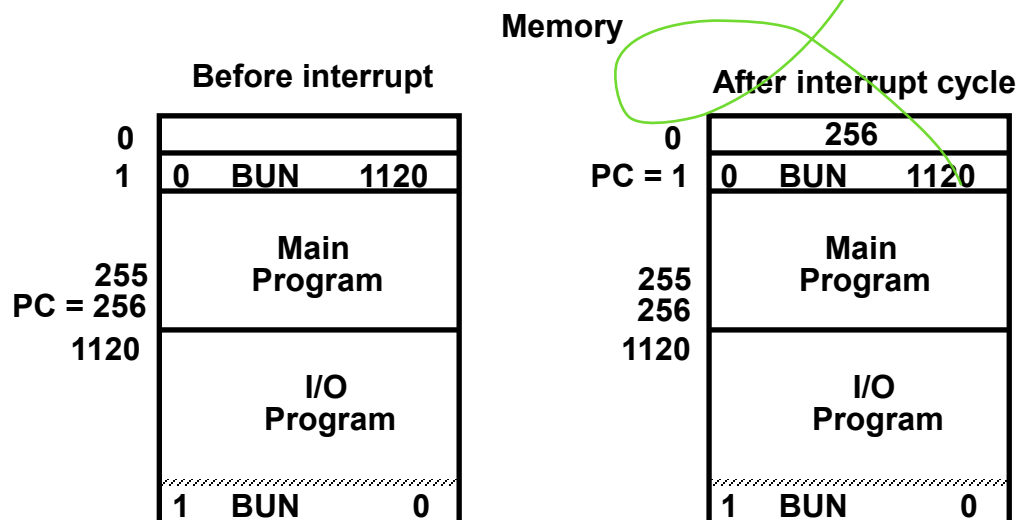
* IEN (Interrupt-enable flip-flop)

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

FLOWCHART FOR INTERRUPT CYCLE

R = Interrupt f/f

Instruction cycle   =0   **R**   =1   Interrupt cycle

Fetch and decode instructions

Execute instructions

IEN =0

=1

=1 **FGI**

=0

=1 **FGO**

=0

R ← 1

Store return address in location 0
M[0] ← PC

Branch to location 1
PC ← 1

IEN ← 0
R ← 0

- 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

Memory

Before interrupt	After interrupt cycle

0			0	256	
1	0 BUN 1120	PC = 1	0 BUN 1120		

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

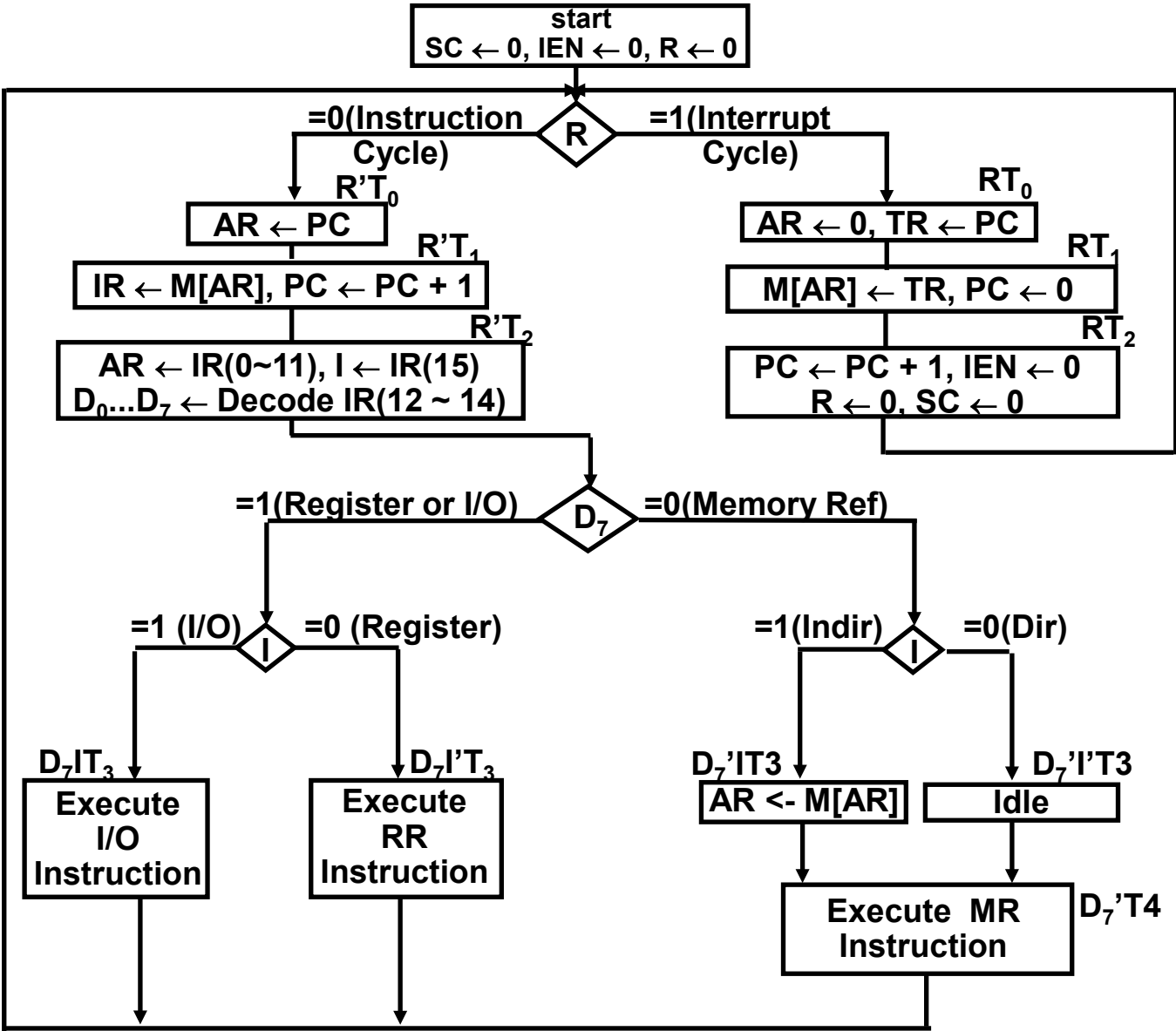How can the CPU recognize the device
requesting an interrupt ?

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 ?

COMPLETE COMPUTER DESCRIPTION
Flowchart of Operations

```
                          ┌─────────────────────────┐
                          │          start          │
                          │ SC ← 0, IEN ← 0, R ← 0   │
                          └─────────────────────────┘
```

start
SC ← 0, IEN ← 0, R ← 0

=0(Instruction Cycle)   **R**   =1(Interrupt Cycle)

$R'T_0$

AR ← PC

RT_0

AR ← 0, TR ← PC

$R'T_1$

IR ← M[AR], PC ← PC + 1

RT_1

M[AR] ← TR, PC ← 0

$R'T_2$

AR ← IR(0~11), I ← IR(15)
$D_0...D_7$ ← Decode IR(12 ~ 14)

RT_2

PC ← PC + 1, IEN ← 0
R ← 0, SC ← 0

=1(Register or I/O) D_7 =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 <- 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	$R'T_2$:	$D0, ..., D7 \leftarrow$ Decode IR(12 ~ 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:	$SC \leftarrow 0$
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:	$S \leftarrow 0$

Input-Output

	$D_7IT_3 = p$	(Common to all input-output instructions)
	$IR(i) = B_i$	(i = 6,7,8,9,10,11)
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR,\ FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7),\ FGO \leftarrow 0$
SKI	pB_9:	If(FGI=1) then $(PC \leftarrow PC + 1)$
SKO	pB_8:	If(FGO=1) then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

DESIGN OF BASIC COMPUTER(BC)

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

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)

⇩

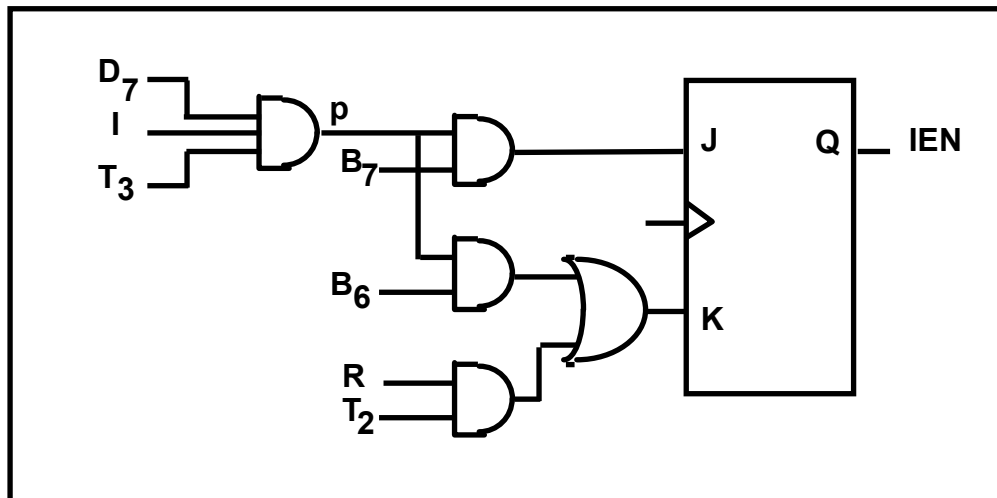$LD(AR) = R'T_0 + R'T_2 + D'_7IT_3$
$CLR(AR) = RT_0$
$INR(AR) = D_5T_4$
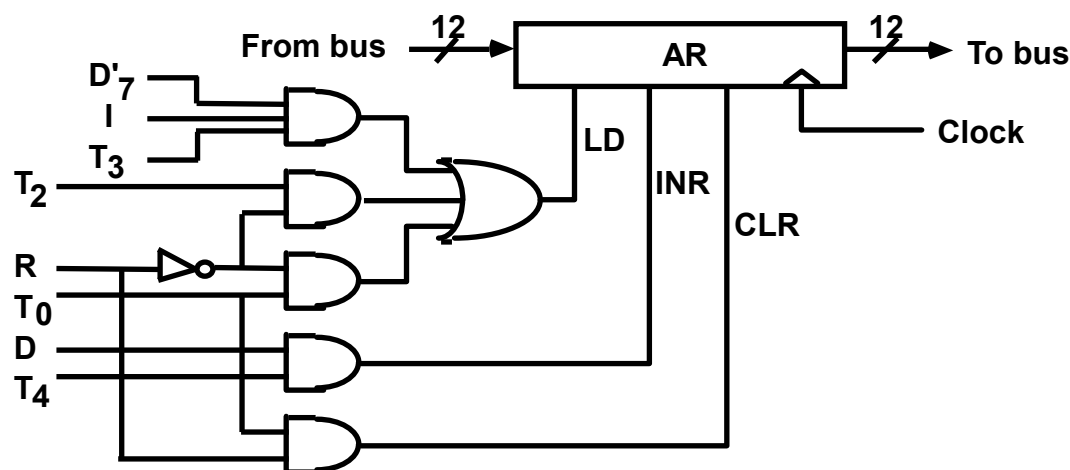
CONTROL OF FLAGS

IEN: Interrupt Enable Flag
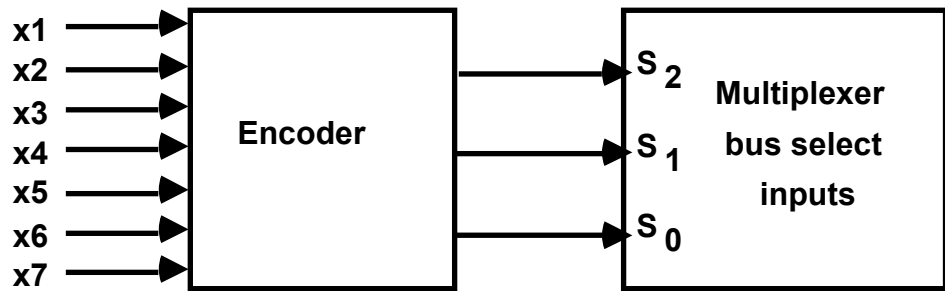
pB_7: $IEN \leftarrow 1$ **(I/O Instruction)**

pB_6: $IEN \leftarrow 0$ **(I/O Instruction)**

RT_2: $IEN \leftarrow 0$ **(Interrupt)**
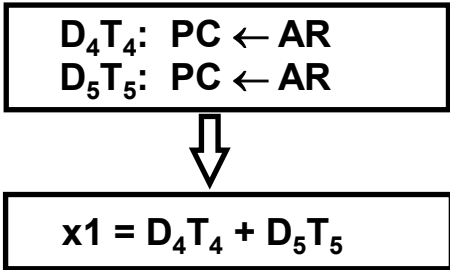
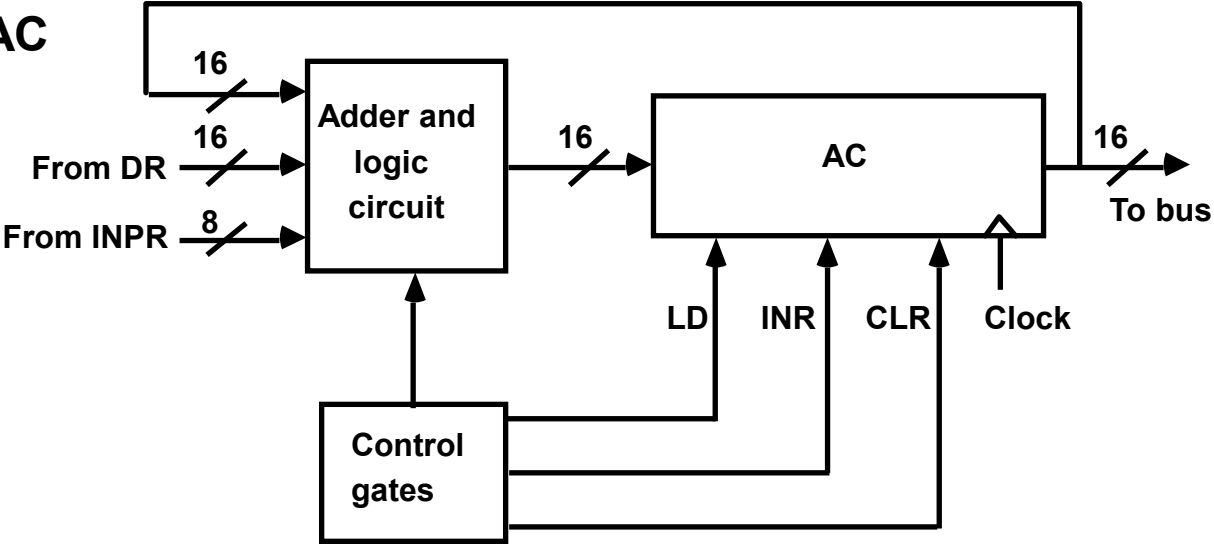$p = D_7 I T_3$ **(Input/Output Instruction)**

CONTROL OF COMMON BUS

x1
x2
x3
x4
x5
x6
x7

Encoder

S_2
S_1
S_0

Multiplexer
bus select
inputs

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

⇓

$x1 = D_4T_4 + D_5T_5$

DESIGN OF ACCUMULATOR LOGIC

Circuits associated with AC

16

From DR → 16

From INPR → 8

Adder and logic circuit

16

AC

16

To bus

LD   INR   CLR   Clock

Control gates

All the statements that change the content of AC

D_0T_5:	$AC \leftarrow AC \wedge DR$	AND with DR
D_1T_5:	$AC \leftarrow AC + DR$	Add with DR
D_2T_5:	$AC \leftarrow DR$	Transfer from DR
pB_{11}:	$AC(0\text{-}7) \leftarrow INPR$	Transfer from INPR
rB_9:	$AC \leftarrow AC'$	Complement
rB_7 :	$AC \leftarrow shr\ AC, AC(15) \leftarrow E$	Shift right
rB_6 :	$AC \leftarrow shl\ AC, AC(0) \leftarrow E$	Shift left
rB_{11} :	$AC \leftarrow 0$	Clear
rB_5 :	$AC \leftarrow AC + 1$	Increment

CONTROL OF AC REGISTER

**Gate structures for controlling
the LD, INR, and CLR of AC**
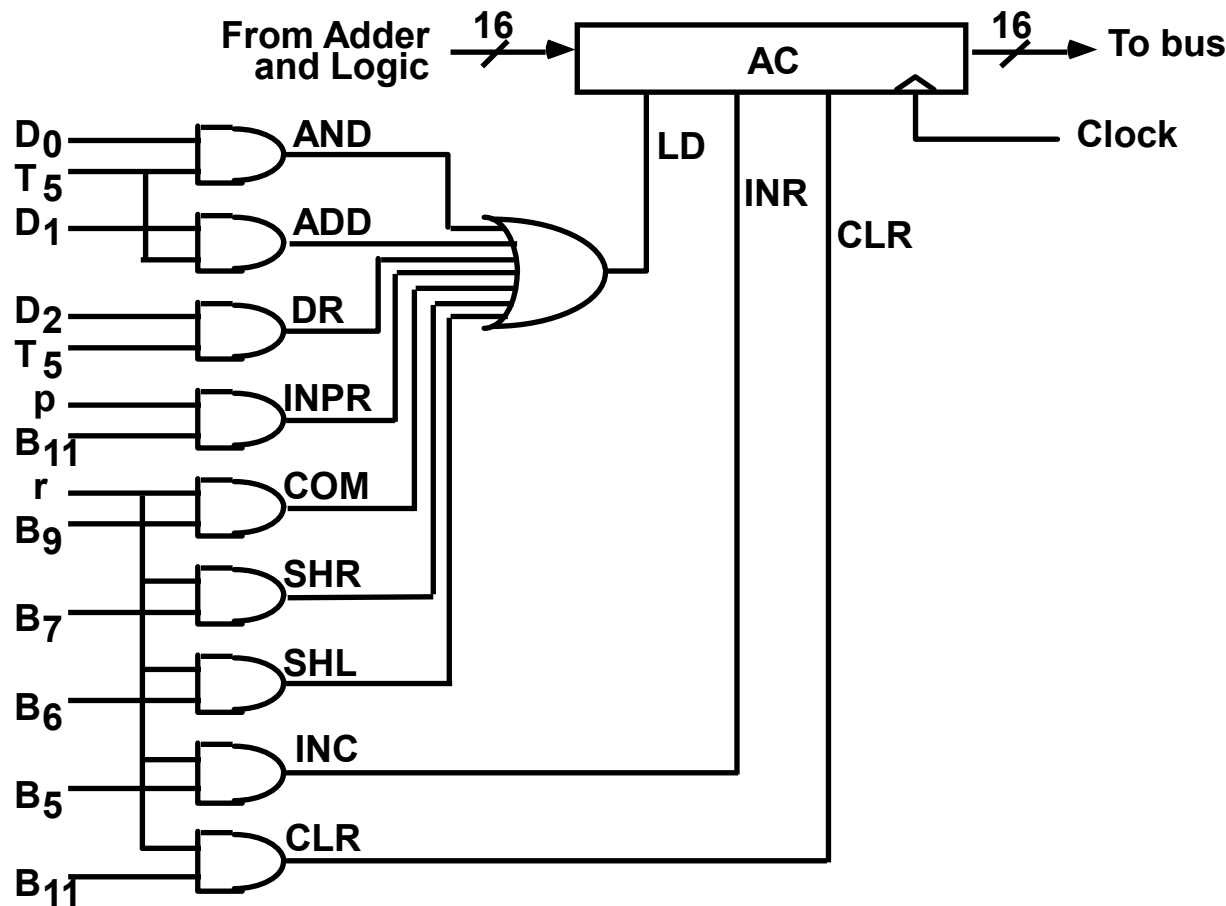
ALU (ADDER AND LOGIC CIRCUIT)

One stage of Adder and Logic circuit