Computer Organisation and Architecture

Course Code: CO206

Module-2: Control Unit

Contents of Module 2

- Instruction types, Formats, Instruction cycles and sub cycles.
- Micro-operations, execution of a complete instruction.
- Hardwired and micro programmed control.
- Microprogrammed sequencing.
- Wide branch addressing.
- Micro- instructions with next address field.
- Pre-fetching micro instructions.
- Concept of horizontal and vertical micro-programming.

Introduction

- Every **different processor type has its own design** (different registers, buses, micro-operations, 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**
- However, **to understand howprocessors work, we will start with a simplified processor** model
- This is **similar to what real processors were like ~25 years ago**
- **M. Morris Mano introduced a simple processor model** he calls the ***Basic Computer***
  - *The memory has* ***4096 =2^{12} words*** *in it (i.e. 12 bits to select a word in memory)*
  - *Each word is* ***16 bits long***.
- **We will use this** to **introduce processor organization** and the **relationship of the RTL model to the higher level computer processor**

Instruction

- **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 the memory**
- It is **placed in an *Instruction Register* (IR)**
- **Control circuitry** in control unit then **translates the instruction into the sequence of micro-operations** necessary to **implement it.**

Instruction Types

- **A computer has a set of instructions** which can be classified as
  - Data transfer Instructions
  - Arithmetic Instructions
  - Logical Instructions
  - Shift and Rotate Instructions
  - Program control Instructions
  - Input/Output Instructions

Data Transfer Instruction

Instruction	Mnemonic	Description
Load	LDA	It transfers data from specified location to the processor register usually accumulator
Store	STA	It transfers data from the processor register(usually accumulator) to the specified memory location
Move	MOV	It transfers data between processor register and memory or between two memory words
Exchange	XCH	It swaps data betweentwo registers or a register and a memory word

Arithmetic Instruction

Instruction	Mnemonic	Description
Add	ADD	It adds the content of two operands
Subtract	SUB	It subtracts the contents of two operands
Increment	INC	It adds 1 to the value stored in the register or a memory word
Decrement	DEC	It subtracts 1 from the value stored in the register or a memory word
Multiply	MUL	It multiplies the content of two operands
Divide	DIV	It divides the content of two operands
Negate	NEG	It gives the 2's Complement of the specified operand

Logical Instruction

Instruction	Mnemonic	Description
AND	AND	It logically ANDs the individual bits of the operands
OR	OR	It logically ORs the individual bits of the operands
Exclusive OR	XOR	It logically Ex-ORs the individual bits of the operands
Clear	CLR	It causes the specified operand to be replaced by 0s
Complement	COM	It gives the 1’s complement of the specified operand

Shift and Rotate Instruction

Instruction	Mnemonic	Description
Logical Shift Right	SHR	It shifts the contents of the specified register 1-bit position towards right and fills the vacant bit with zero
Logical Shift Left	SHL	It shifts the contents of the specified register 1-bit position towards left and fills the vacant bit with zero
Arithmetic Shift Right	ASHR	It shifts the contents of the specified register 1-bit position towards right and fills the vacant bit with previous sign bit
Arithmetic Shift Left	ASHL	It shifts the contents of the specified register 1-bit position towards left and fills the vacant bit with zero
Rotate Left	ROL	It rotates (circular shifts) left the contents of the specified register
Rotate Right	ROR	It rotates (circular shifts) right the contents of the specified register

Program Control Instruction

Instruction	Mnemonic	Description
Branch	BR	It transfers program control to the specified address
Jump	JMP	It transfers program control to the specified address
Compare	CMP	It compares the content of two registers
Test	TST	It performs bitwise AND operation on two operands
Skip	SKP	It skips the next instruction

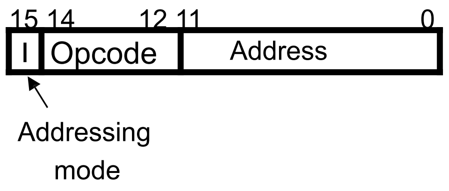
Input/Output Instruction

Instruction	Mnemonic	Description
Input	INP	Load a character in Accumulator register from input port
Output	OUT	Send a character to output port from Accumulator register
Skipping instruction from input	SKI	It skips the next instruction if input flag=1
Skipping instruction from output	SKO	It skips the next instruction if output flag=1

Instruction Format

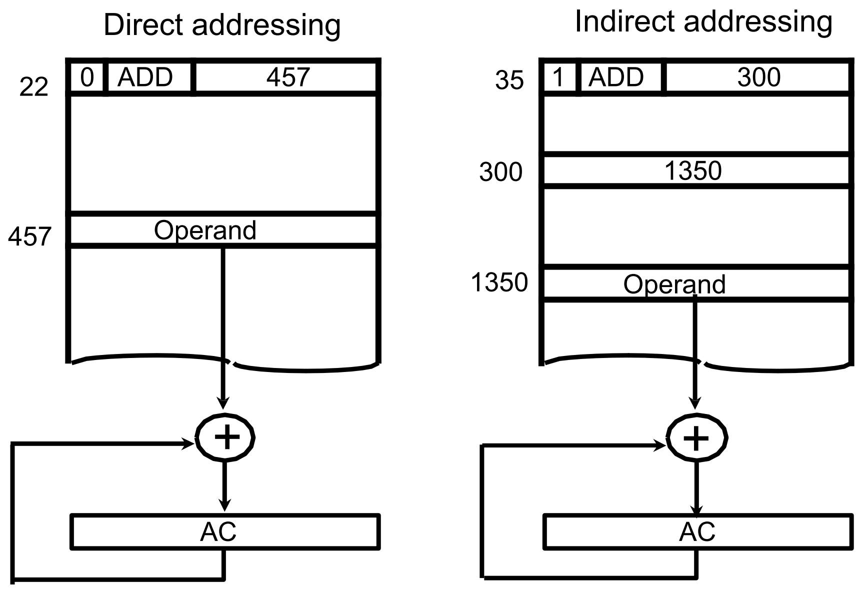
- A Computer **instruction is often divided into two parts**
  - An ***opcode*** (Operation Code) that **specifies the operation for that instruction**
  - An ***address*** that **specifies the registers and/or locations in memory to use** for that operation
- In the **Basic Computer**, since the **memory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



Addressing

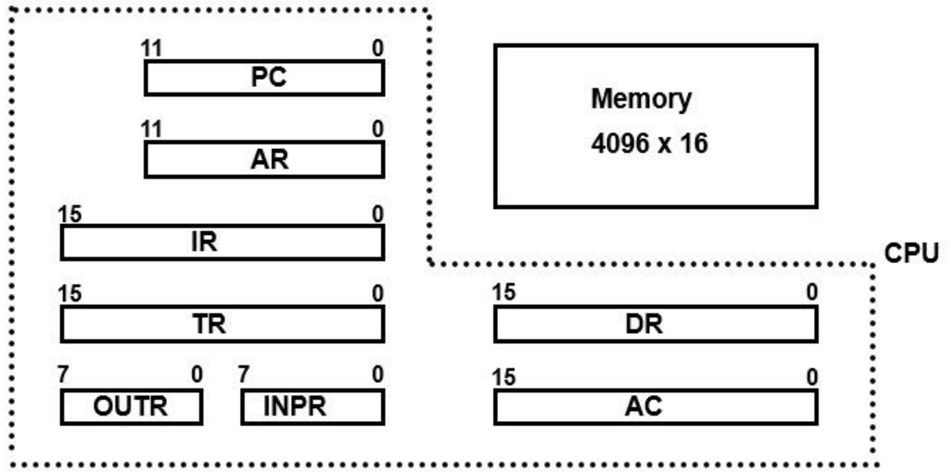
- The address field of an instruction can represent either
  - **Direct address**: it has the address of the operand (direct access to operand)
  - **Indirect address**: It is the address in memory from which the address of the operand can be fetched.



- 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

Basic Computer Registers

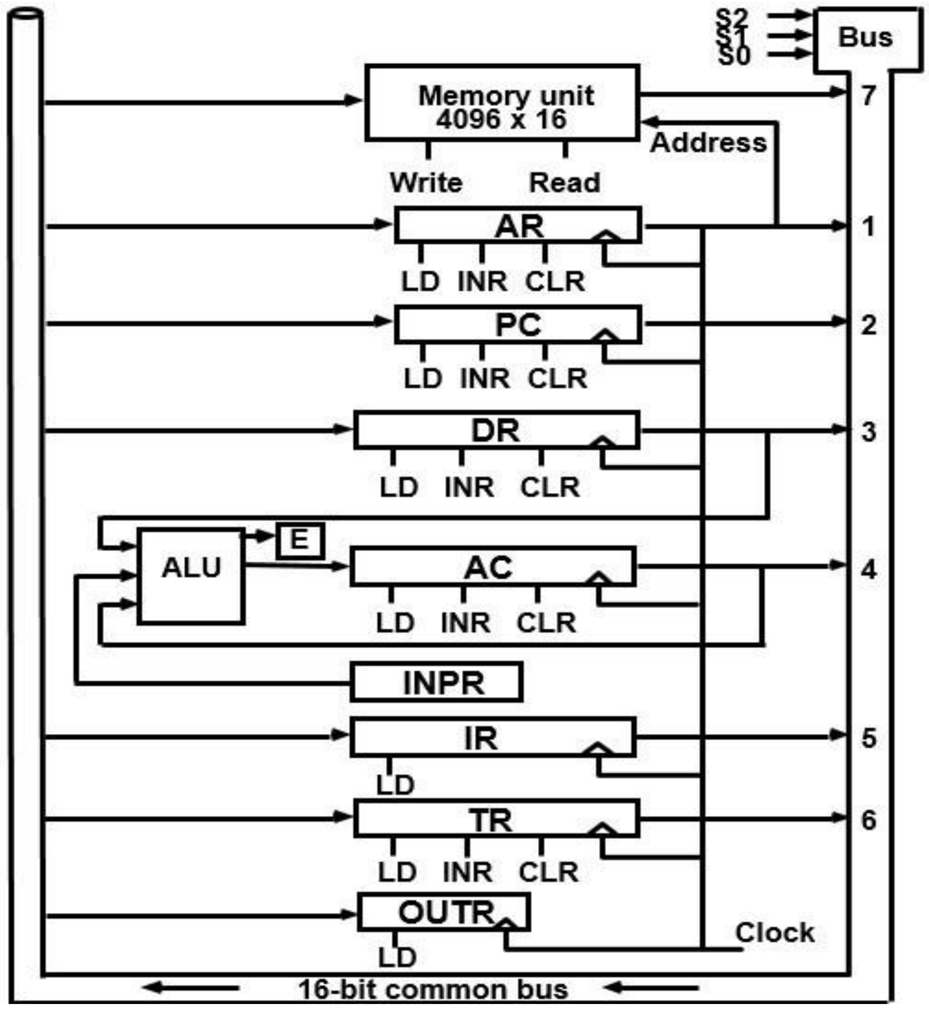
Registers in the Basic Computer



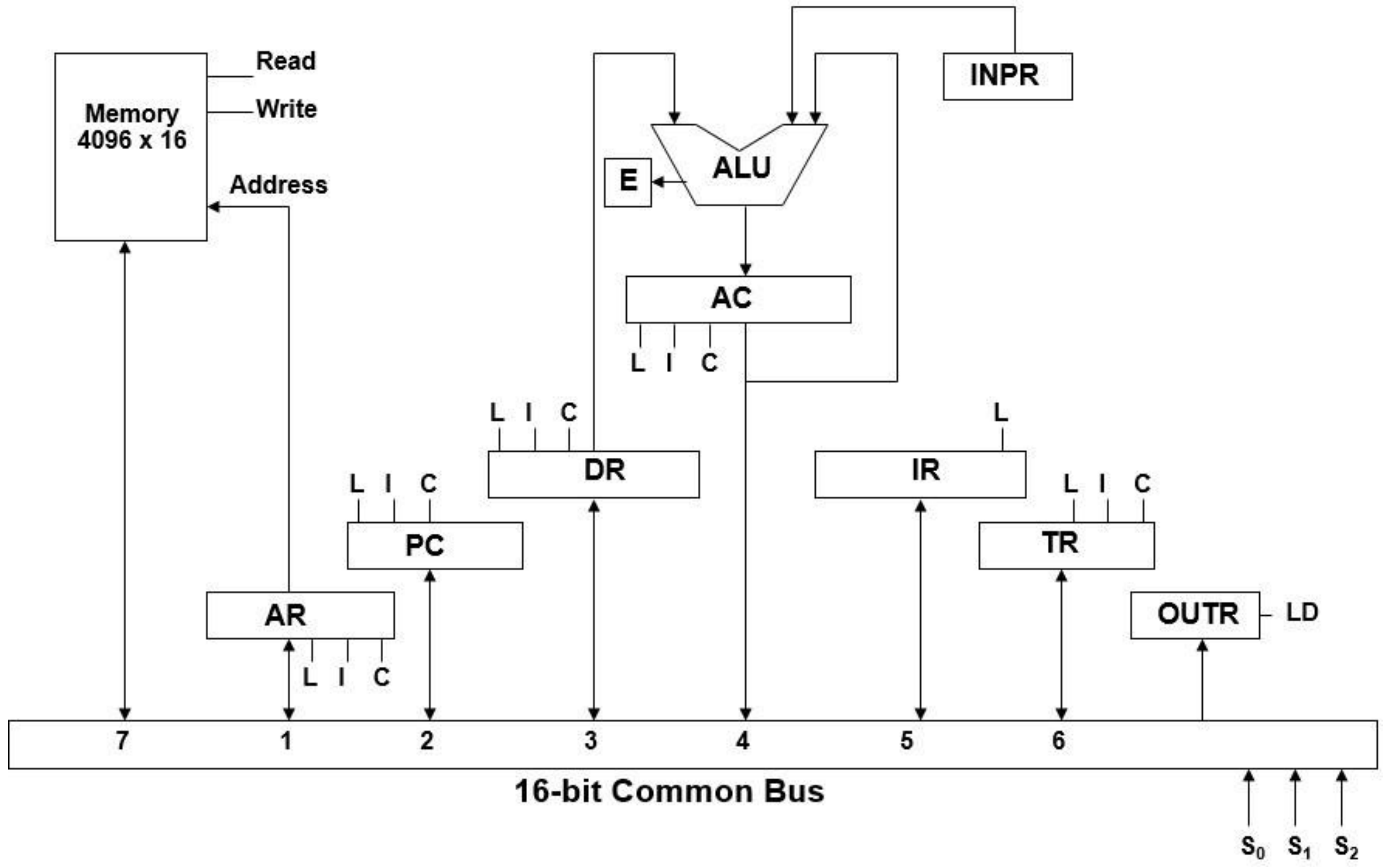
List of basic computer registers

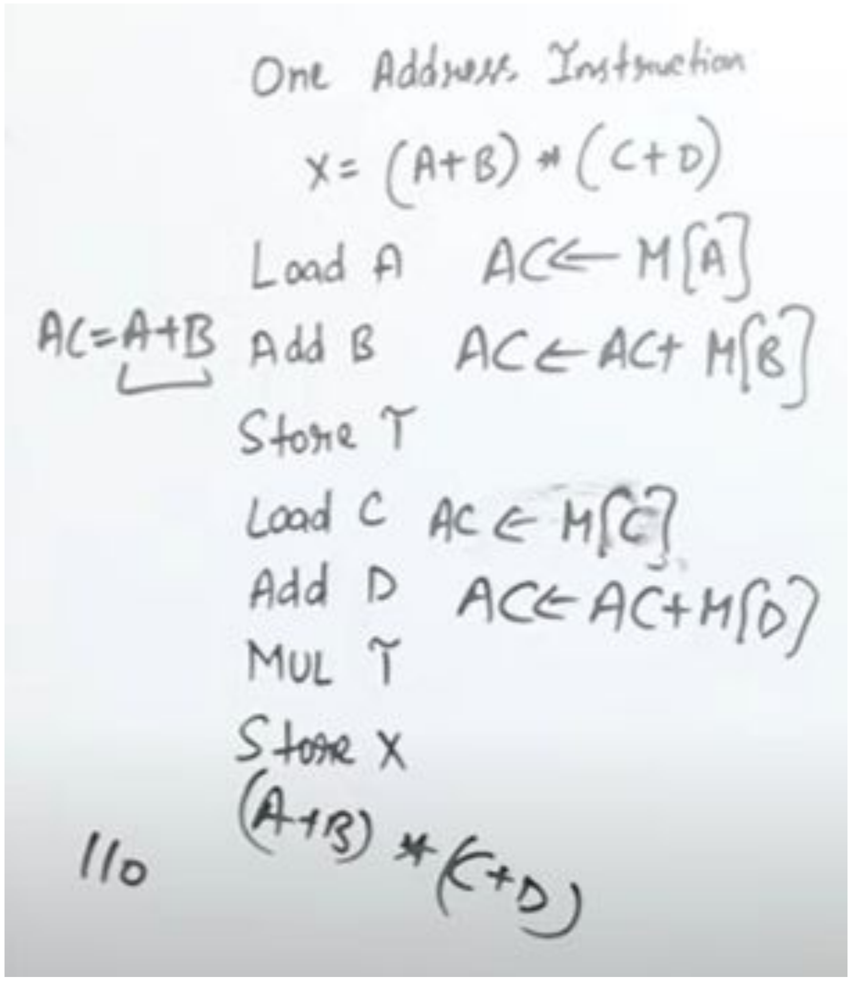
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction Register	Holds instruction code
PC	12	Program Counter	Holds address of instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds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

Common Bus System



Common Bus System





One Address Instruction

$X = (A+B) * (C+D)$

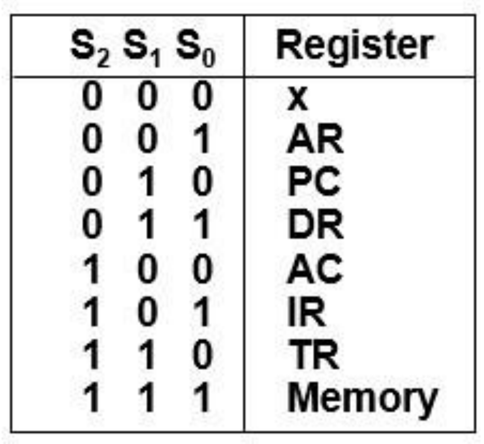
```
        Load A     AC ← M[A]
AC=A+B  Add B      AC ← AC + M[B]
        Store T
        Load C     AC ← M[C]
        Add D      AC ← AC + M[D]
        MUL T
        Store X
```

$(A+B) * (C+D)$

110

Common Bus System

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

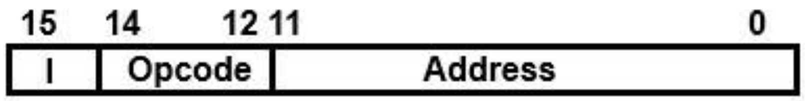


S_2	S_1	S_0	Register
0	0	0	x
0	0	1	AR
0	1	0	PC
0	1	1	DR
1	0	0	AC
1	0	1	IR
1	1	0	TR
1	1	1	Memory

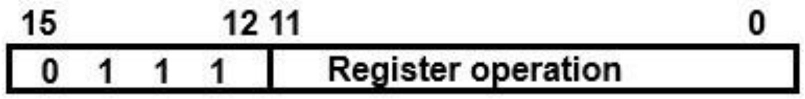
- Either **one of the registers will have its load signal activated**, or the **memory will have its read signal activated**
  - Will determine where the data from the bus gets loaded
- The 12-bit registers, AR and PC**, have 0's loaded onto the bus in the high order 4 bit positions**
- **When** the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

Basic Computer Instruction Format

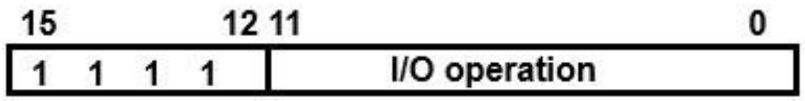
Memory-Reference Instructions **(OP-code = 000 ~ 110)**



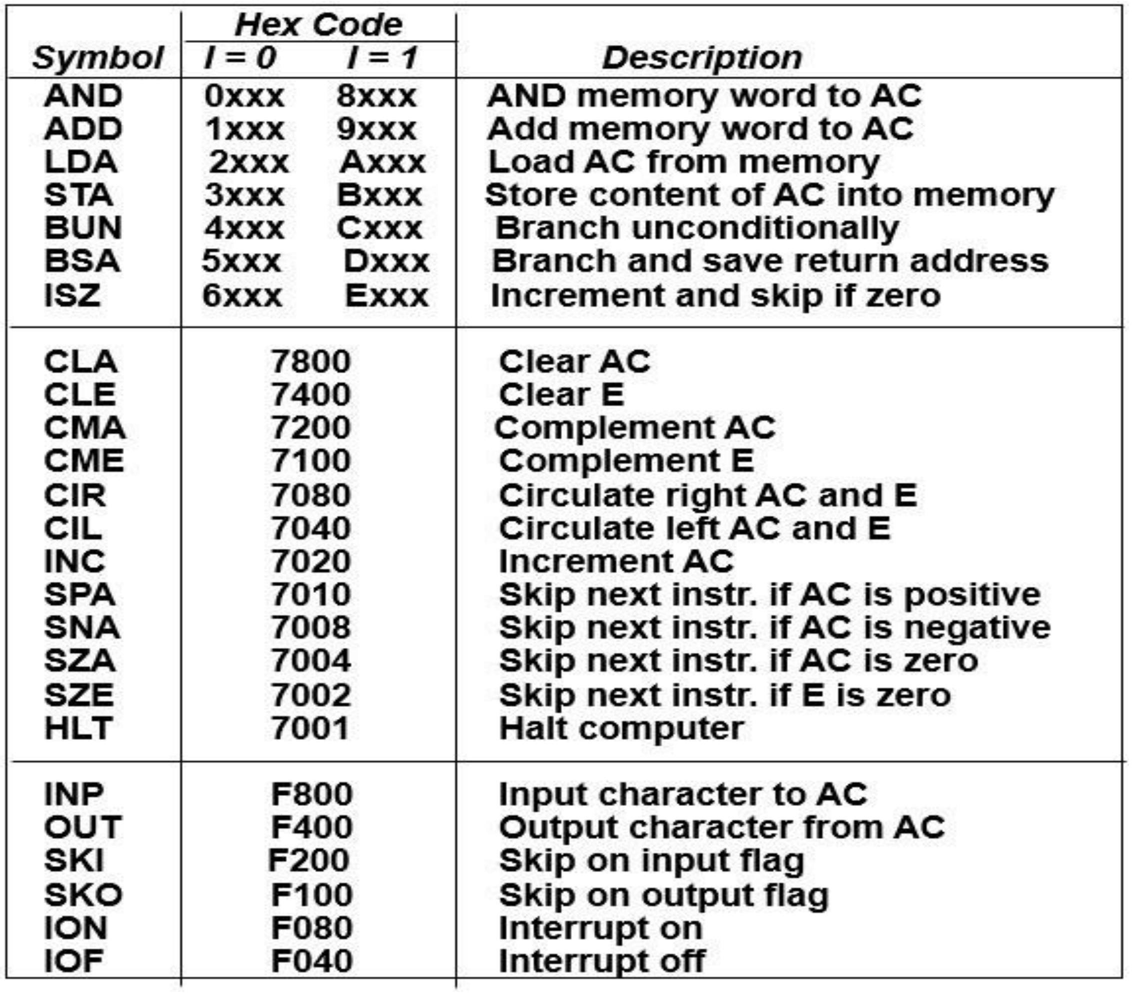
Register-Reference Instructions **(OP-code = 111, I = 0)**



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

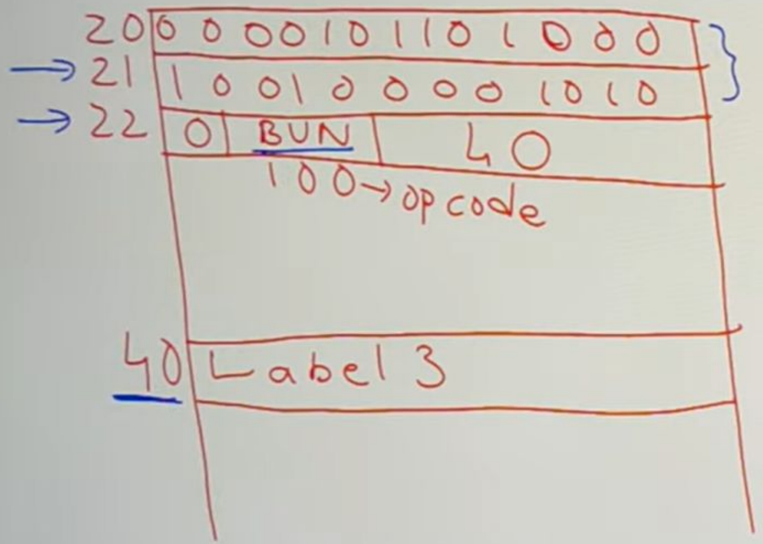


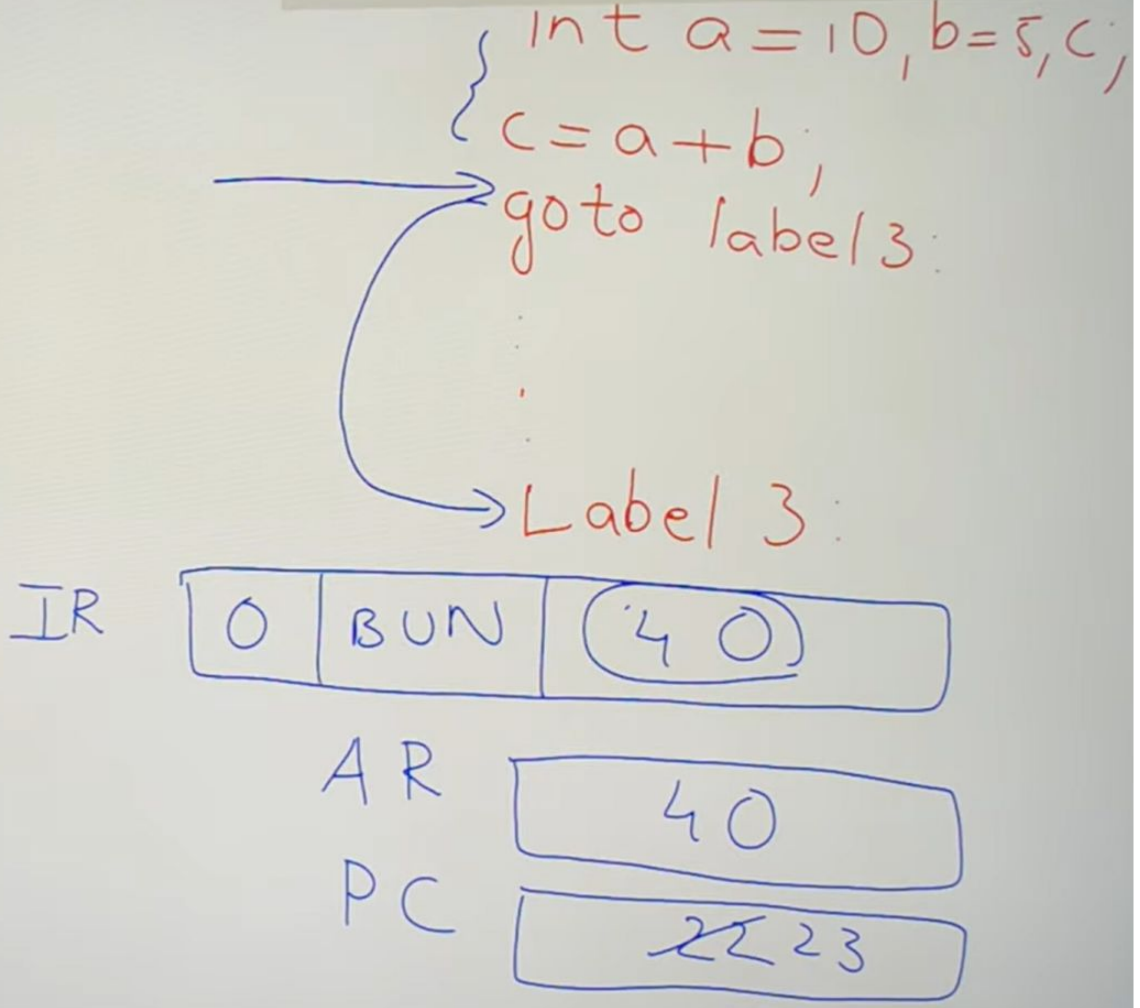
Basic Computer Instructions



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

Symbol	$I = 0$	$I = 1$	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
~~BSA~~	~~5xxx~~	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero

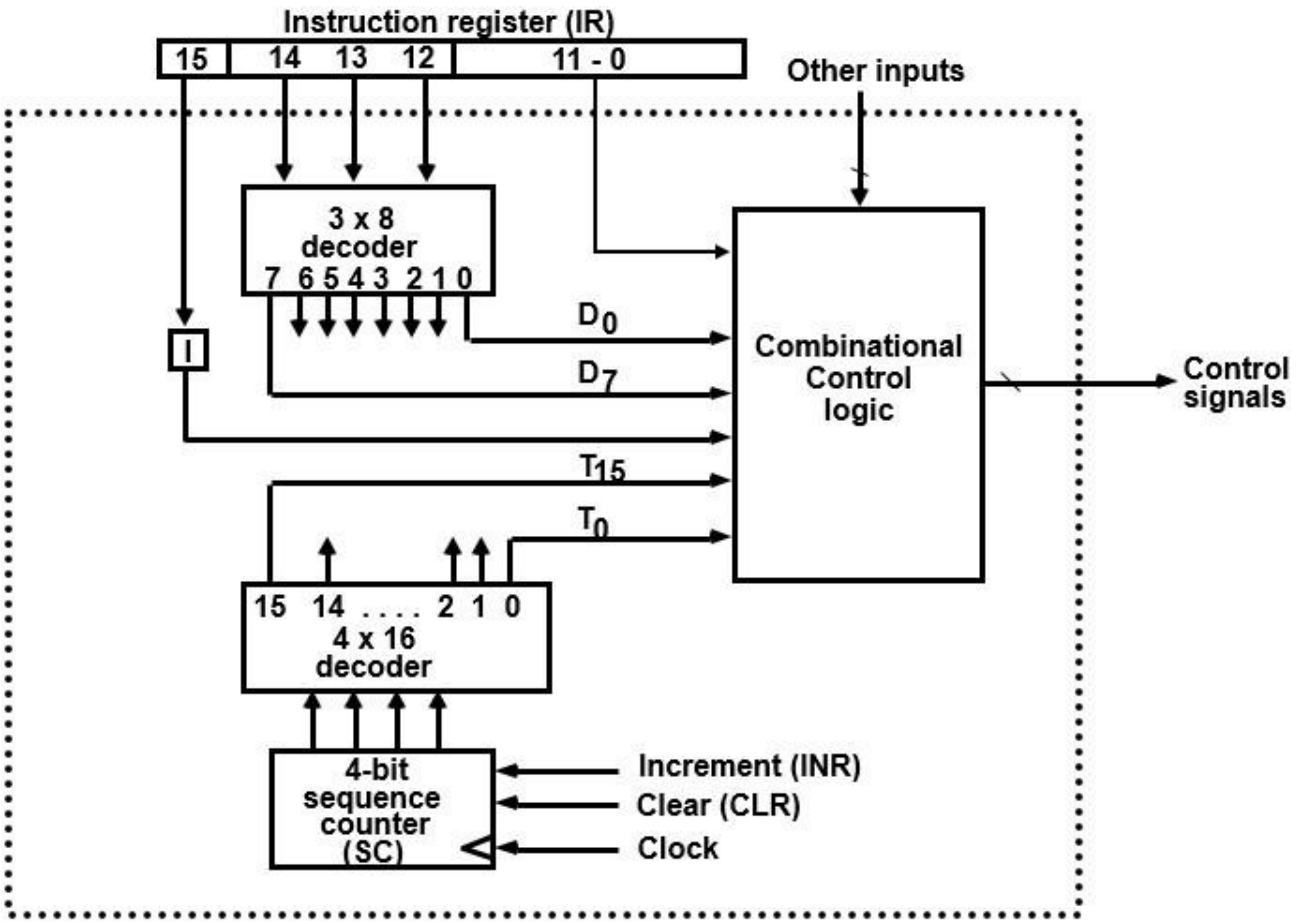




BUN: Branch Unconditionally

$\overline{D_4}T_4$: $PC \leftarrow AR$, $SC \leftarrow 0$

Control Unit of Basic Computer

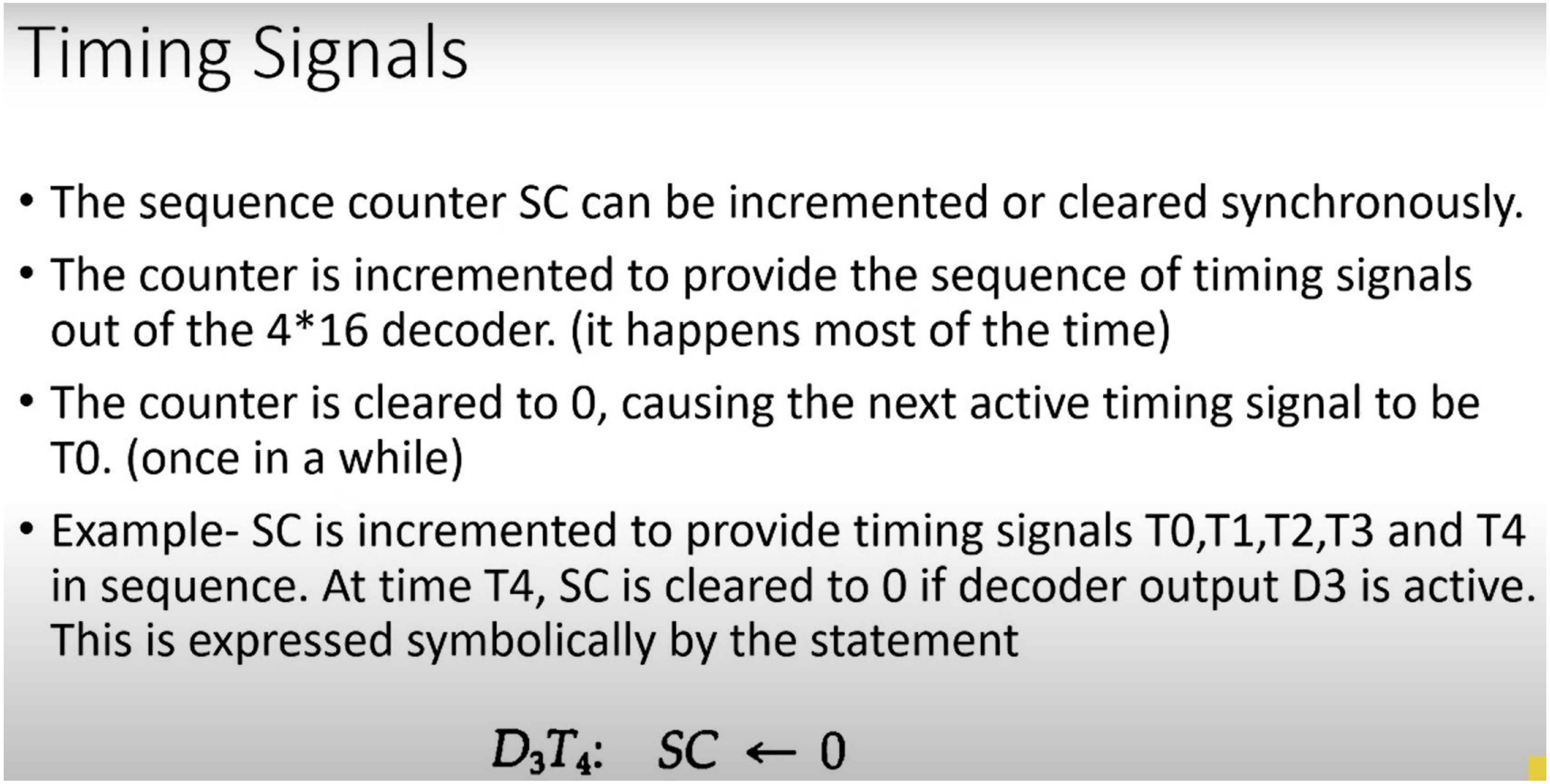


Timing and control

- The timing for all registers is controlled by a master clock generator.
- Clock pulses are applied to all flip-flops and register in the system , including all flip-flops and registers in the control unit.
- Clock pulse do not change the state of a register unless the register is enabled by a control signal.

Control organizations

- Hardwired control
  - The control logic is implemented by gates, flip-flops, decoders, and other digital circuits.
  - It requires changes in the wiring among the various components if the design has to be modified or changed.
- Microprogrammed control
  - Control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations.
  - Any modification or changes can be done by updating the microprogram in control memory.



Timing Signals

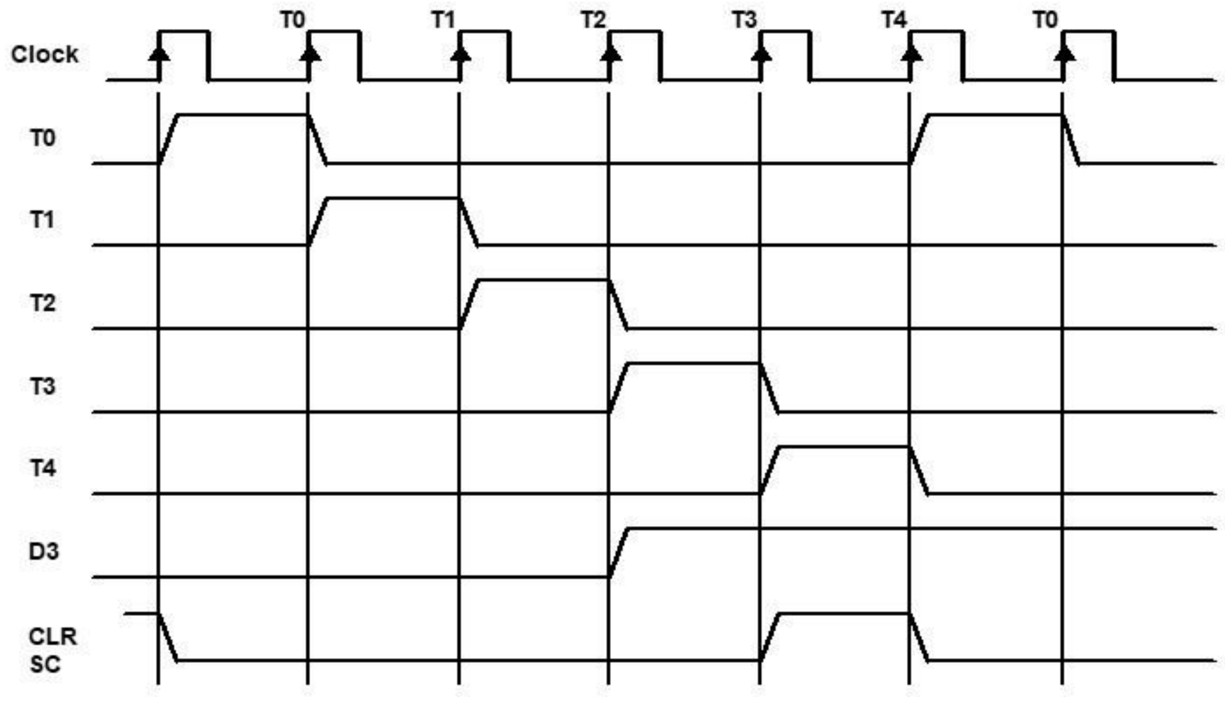
- The sequence counter SC can be incremented or cleared synchronously.
- The counter is incremented to provide the sequence of timing signals out of the 4*16 decoder. (it happens most of the time)
- The counter is cleared to 0, causing the next active timing signal to be T0. (once in a while)
- Example- SC is incremented to provide timing signals T0,T1,T2,T3 and T4 in sequence. At time T4, SC is cleared to 0 if decoder output D3 is active. This is expressed symbolically by the statement

$$D_3T_4: \quad SC \leftarrow 0$$

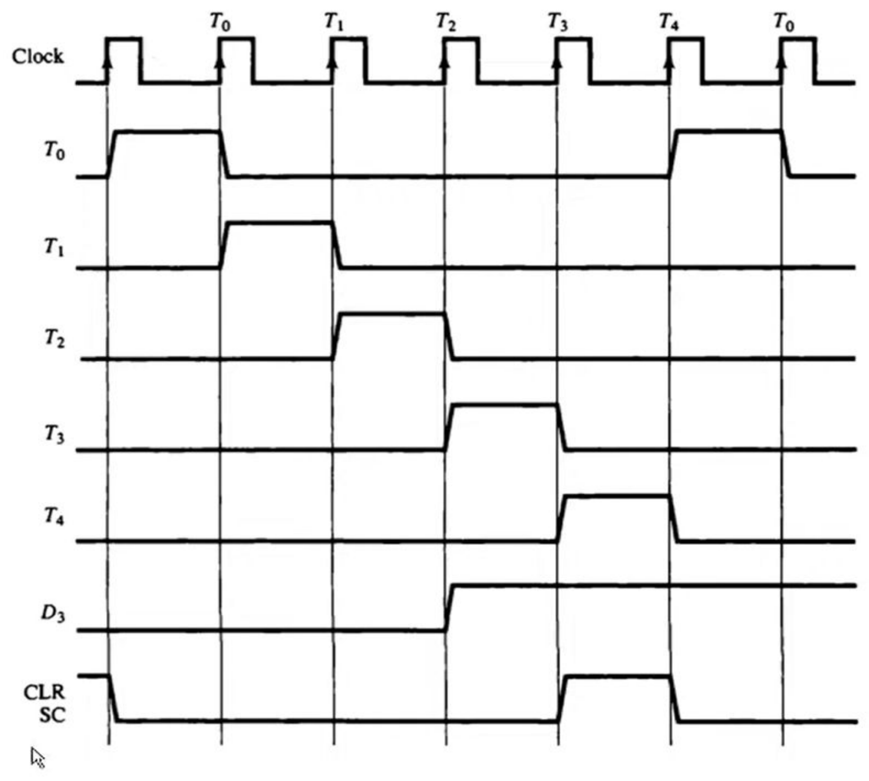
Timing Signal

- Generated by 4-bit sequence counter and 4×16 decoder
- The Sequence Counter 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.

$$D_3T_4: SC \leftarrow 0$$



Example of control timing signal



- The SC responds to the positive transition of the clock.
- Initially, CLR input of SC is active
- The first positive transition of the clock clears SC to 0, which in tern activates the timing signal T0 out of the decoder.
- The positive clock transition labelled T0 in the diagram will trigger only those registers whose control inputs are connected to timing signal T0.
- SC is incremented with every positive clock transition unless its clear input is active. This produces the sequence of timing signals T0,T1,T2,T3,T4 and so on.
- If SC is not cleared, the timing signals will continue with T5,T6 up to T15 and back to T0.
- Output D3 from the operation decoder become active at the end of timing signal T2.
- When timing signal T4 become active the output of the AND gate that implements the control function D3T4 becomes active.
- This signal is applied to the CLR input of SC.
- On the next positive clock transition the counter is cleared to 0, this causes the T0 to become active instead of T5.

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

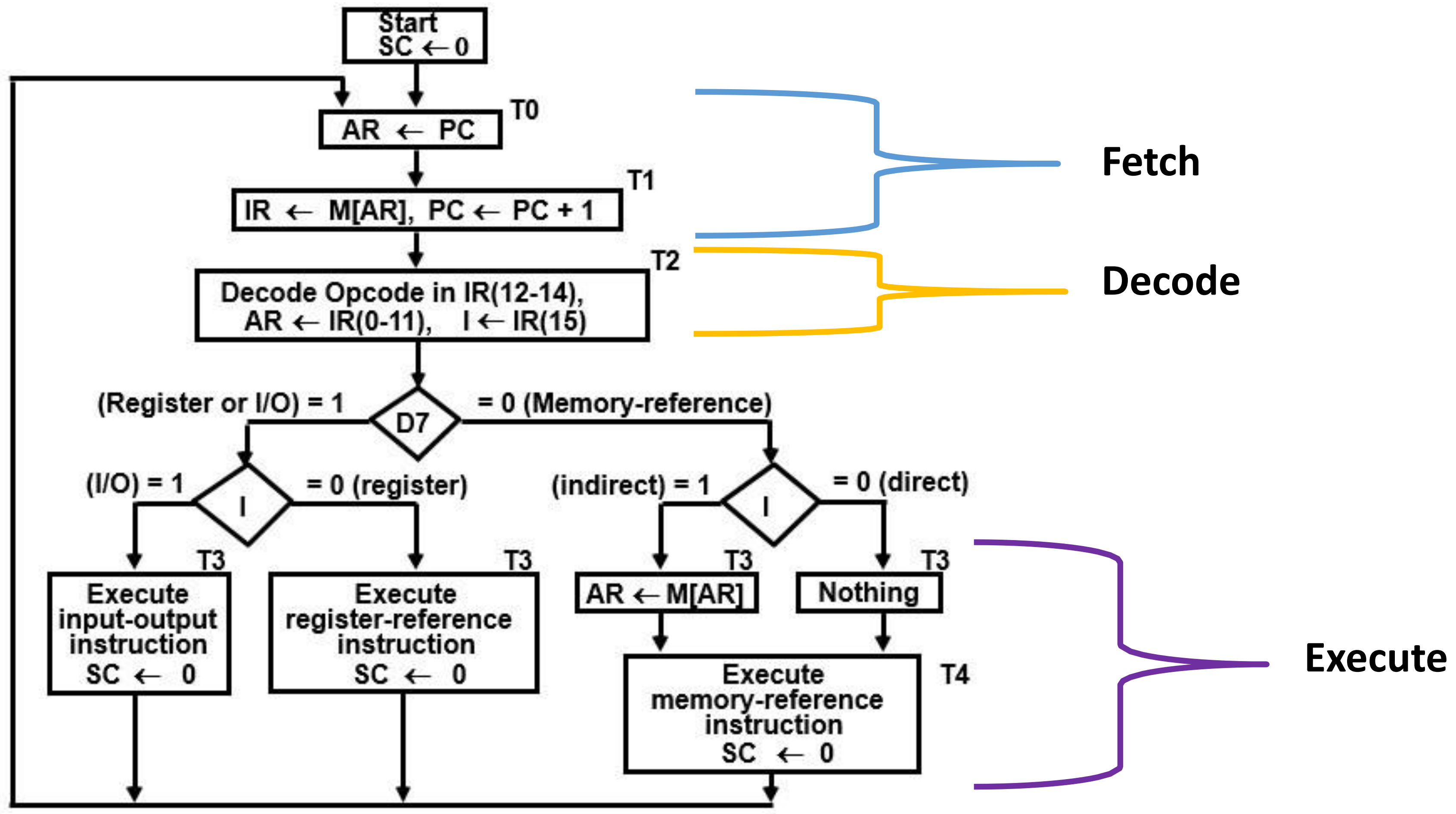
- **After an instruction is executed**, **the cycle starts again at step 1**, for the next instruction

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

Elements of Instruction

- **Operation Code(OPCODE)**
  - **Specifies the operation to be performed**
  - The operation is specified by a binary code, known as the operation code, opcode do ADD, SUB,DIV,LOAD.
- **Source Operand reference**
  - The operation **may involve one or more source operands**, that is, operands that are inputs for the operation
- **Result Operand reference**
  - The **operation may produce a result**
- **Next Instruction reference**
  - This **tells the processor where to fetch the next instruction** after the execution of this instruction is complete

Instruction Cycle



Fetch Cycle

- The address of next instruction is fetched from the program counter and stored in address register.

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

- The instruction read from memory is then placed in the instruction register.
- Program counter is incremented by 1.

Decode Cycle

- In decode operation processor decode the instruction which is fetched from memory
- $I \leftarrow IR(15)$, $Decode \leftarrow IR(12\text{-}14)$, $AR \leftarrow IR(0\text{-}11)$
- 0-11 bits in the instruction format are store in the address register.
- 12-14 bits are decoded.
- 15 bit is I means direct or indirect address.

Execute Cycle

- After decoding the instruction, the third timing signal is active.
- There are three types of instruction:-
  - Memory Reference
  - Register Reference
  - I/O Reference
- If D=1 and IR(15)=0, that means the instruction is register reference.
- If D=1 and IR(15)=1, that means the instruction is I/O reference.

Execute Cycle

- If D=0 and IR(15)=0, that means memory reference instruction is direct address instruction.
- If D=0 and IR(15)=1, that means memory reference instruction is indirect address instruction
- AR $\leftarrow$ M[AR], AR holds the address part of the instruction
- After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase.

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

$r = D_7 I' T_3$ => Register Reference Instruction

$B_i = IR(i)$, $i=0,1,2,\ldots,11$

	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:	$AC \leftarrow \text{shr } AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow \text{shl } AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
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)

CLA:	Clear Accumulator
CLE:	Clear E
CMA:	Complement AC
CME:	Complement E
CIR:	Circulate right
CIL:	Circulate Left
INC:	Increment AC
SPA:	Skip if positive
SNA:	Skip if negative
SZA:	Skip if AC zero
SZE:	Skip if E zero
HLT:	Halt Computer

Memory Reference Instructions

-The effective address of the instruction is in AR and is 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

Symbol	Operation Decoder	Symbolic Description
AND	D_0	AC <- AC ^ M[AR]
ADD	D_1	AC <- AC + M[AR], E <- C_{out}
LDA	D_2	AC <- M[AR]
STA	D_3	M[AR] <- AC
BUN	D_4	PC <- AR
BSA	D_5	M[AR] <- PC, PC <- AR + 1
ISZ	D_6	M[AR] <- M[AR] + 1, if M[AR] + 1 = 0 then PC <- PC+1

BUN: Branch Unconditionally

BSA: Branch and save return address

ISZ: Increment and Skip if Zero

Input Output Instructions

$D_7IT_3 = p$ => Input Output Instruction

$IR(i) = B_i$, 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

Instruction Types(on the basis of addresses)

- **Stack type organization**
  - 0-Address instruction
- **Accumulator type organization**
  - 1-Address Instructions
- **General Register Organization**
  - 2-Address instructions
  - 3-Address instructions

Zero Address Instruction

- ❑ All addresses implicit, e.g. ADD
- ❑ Uses a stack, e.g. POP A, POP B

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

```
PUSH A
PUSH B
ADD
PUSH C
PUSH D
ADD
MUL
POP X
```

1- Address Instruction

One address instruction use an implied accumulator (AC) register for all data manipulation now we see the same example

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

```
LOAD A
ADD B
STORE T
LOAD C
ADD D
MUL T
STORE
X
```

All operation are done between the AC register and a memory operand

T is the address of a temporary memory location for storing intermediate result

2- Address Instructions

One address is used as operand and result both.

Most common in commercial in computers . Each address field can specify either a processes register or a memory word

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

```
MOV  R1,  A
ADD  R1,  B
MOV  R2,  C
ADD  R2,  D
MUL  R1,R2
MOV X, R1
```

Reduces length of instruction

Requires some extra work , temporary storage

MOV instruction moves or transfers the operand to and from memory and processor registers.

First symbol listed in an instruction is assumed to be both a source and destination .

3-Address Instructions

Computer with three address instructions formats can use each address field to specify either a processor register or a memory operand .

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

```
ADD   R1,A,B
ADD   R2,C,D
MUL X,R1,R2
```

Needs very long words to hold everything

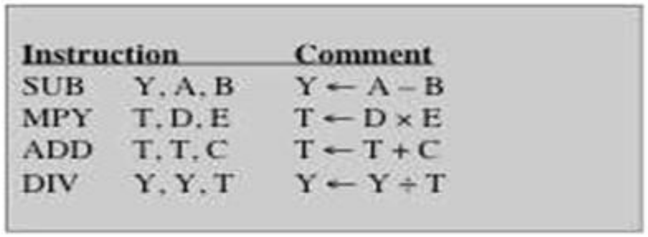
It is assumed that the computer has two processor registers R1 andR2

The advantage of three address format is that it results in short programs when evaluating arithmetic expression .

The disadvantage is the binary coded instructions require too many bits to specify

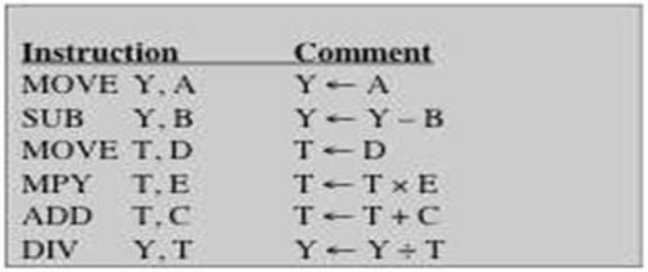
Instances for instructions

$$Y = \frac{A - B}{C + (D \times E)}$$



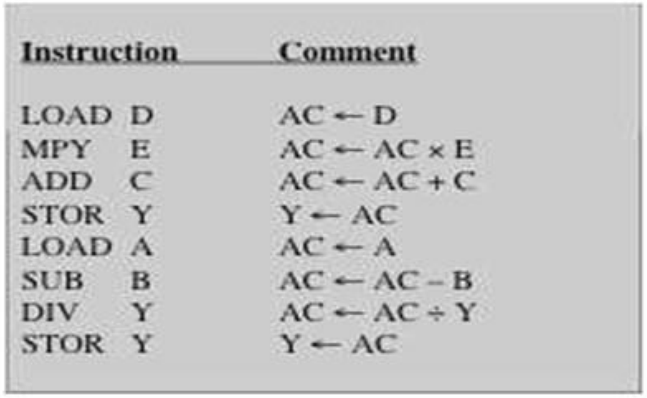
Instruction		Comment
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-address instructions



Instruction		Comment
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

(b) Two-address instructions



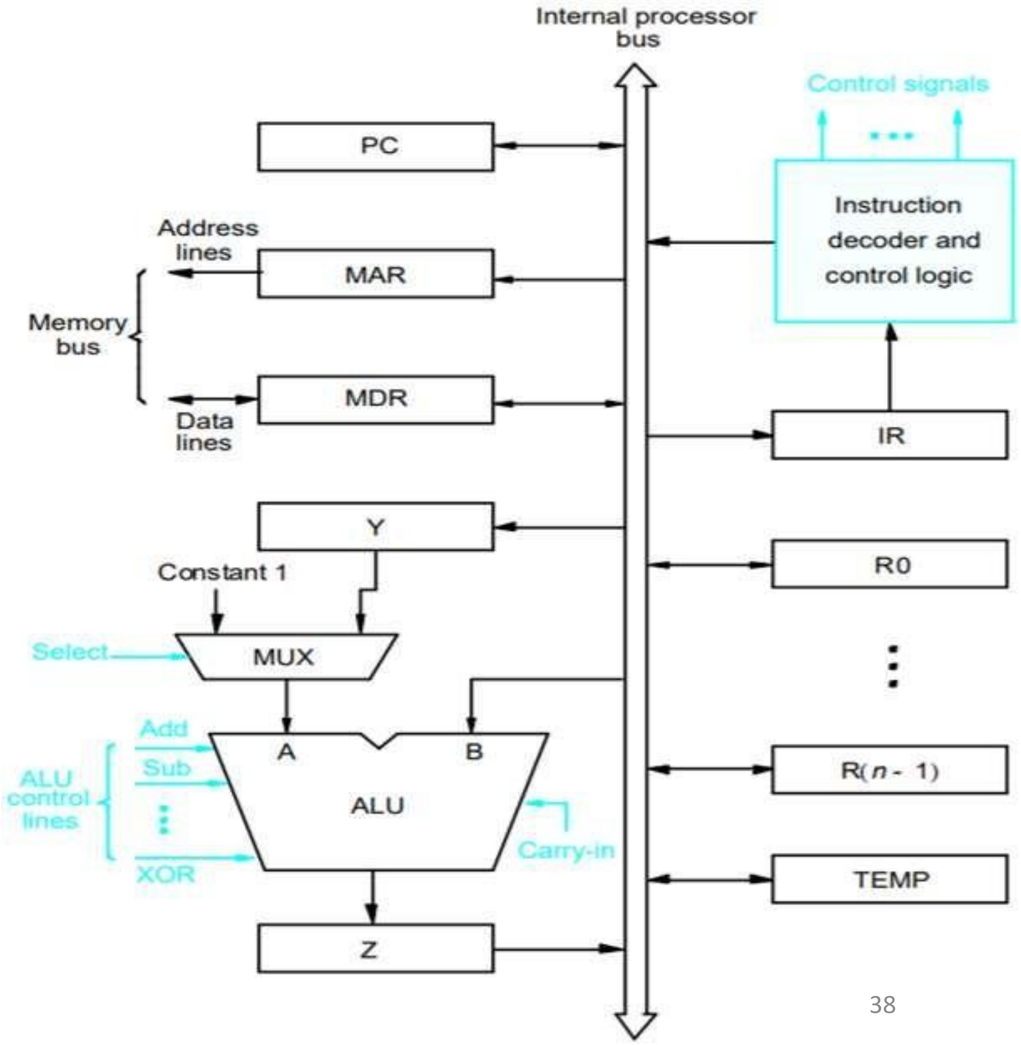
Instruction		Comment
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

(c) One-address instructions

Execution of Complete Instruction

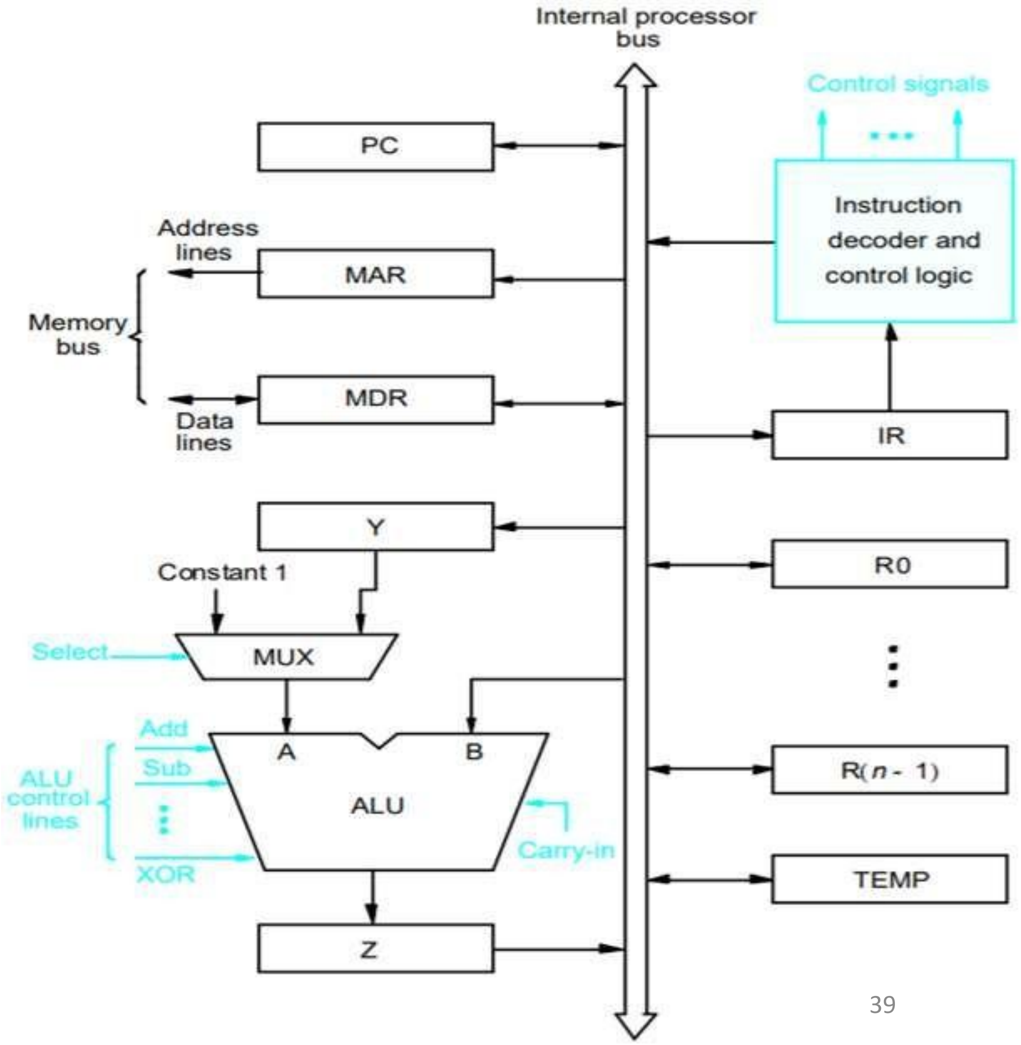
- ADD R1 (R3)
  - Fetch the instruction from memory-IR
  - Fetch the operand from memory
  - Perform Addition operation
  - Load the result into R1

STEP	ACTION
1	PCout, MARin, Read, Select Constant 1, Add, Zin
2	Zout, PCin, Yin ,WMFC
3	MDRout, IRin
4	R3out, MARin, Read
5	R1out, Yin, WMFC
6	MDRout, select Y, Add, Zin
7	Zout, R1in, END



Unconditional Branch

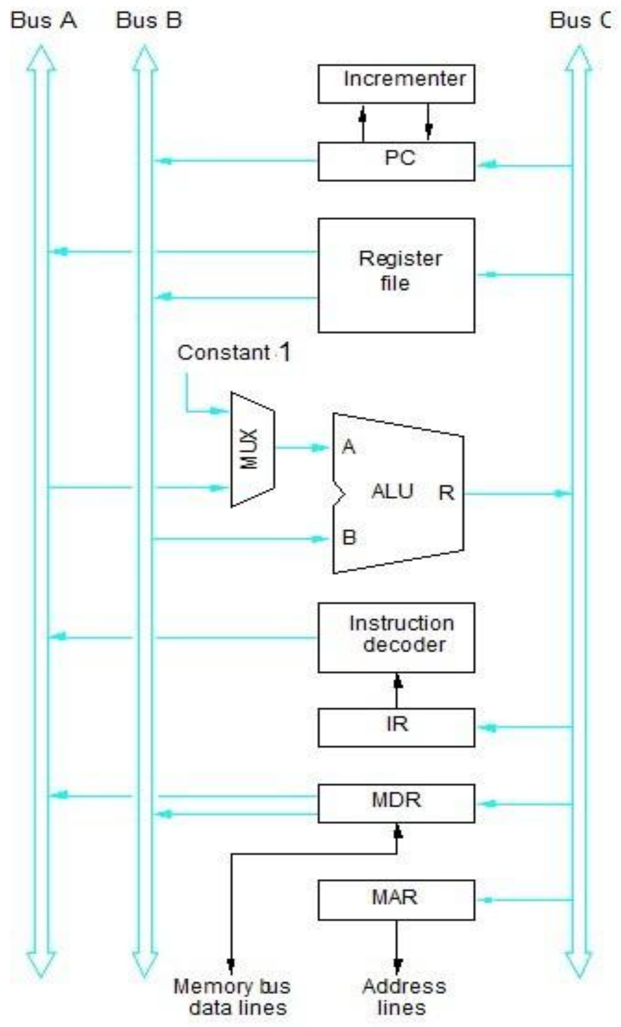
STEP	ACTION
1	PCout, MARin, Read, Select Constant 1, Add, Zin
2	Zout, PCin, Yin ,WMFC
3	MDRout, IRin
4	Offset-field-of-IRout, Add, Zin
5	Zout, PCin, END



Multiple Bus Organisation

Add R4, R5, R6

Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End



Generation of Control Signal

- To **execute instructions**, the **processor must have some means of generating the control signals** needed **in** the **proper sequence**.

- Two categories:
  - **Hardwired Control**
  - **Micro-programmed control**

  Hardwired system can operate at high speed but with little flexibility

This type of CU is designed using a nmber of combinational and sequential circuits like gates, FFs, decoder, encoder and other digital circuis.

For CU to perform it's function, it has some **inputs** that allows it to determine the state of the system and outputs that allows it to control the behaviour of the system.

These inputs are the **external specification** of the CU.

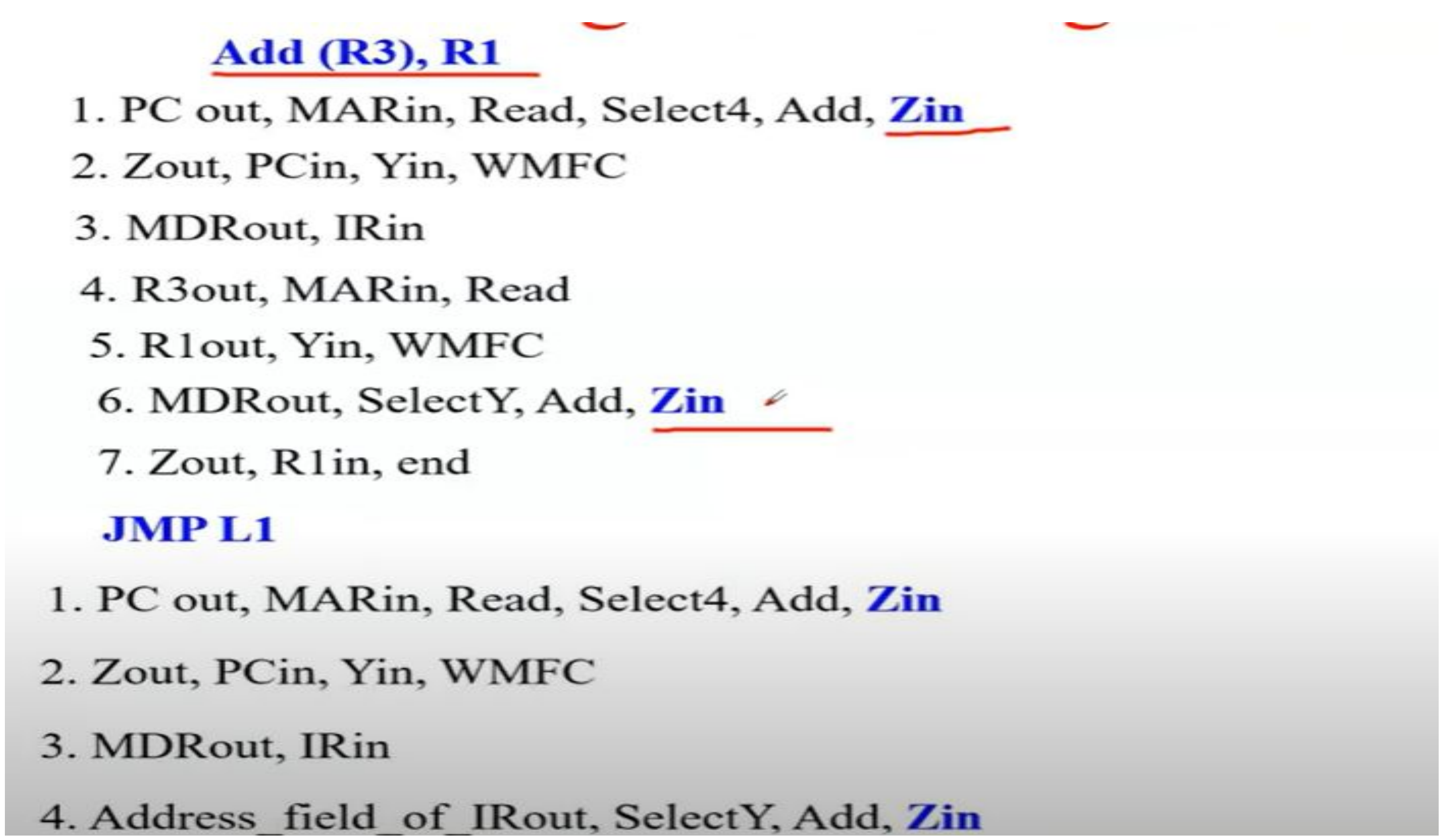
Internally, the CU must have some logic to perform its sequencing and execution function.

Steps used in generating a control signal: (e.g., Zin)

1. Find out the control sequence for the instructions supported by the ISA.

2. Next find out , in which instructions the signal is appearing and then find out the step number of **that instruction** the signal is appearing.

3. Say, Zin is appearing in the **6th step** of the instruction**Add (R3), R1.** It means Zin signal need to be generated for the step no 6 of ADD instruction, so when **both** the cases are true, Zin need to be generated, like that for **JMP L1 instruction ,** Zin is generated in step no 4. i.e., in **either of the two** instructions Zin signal need to be generated.

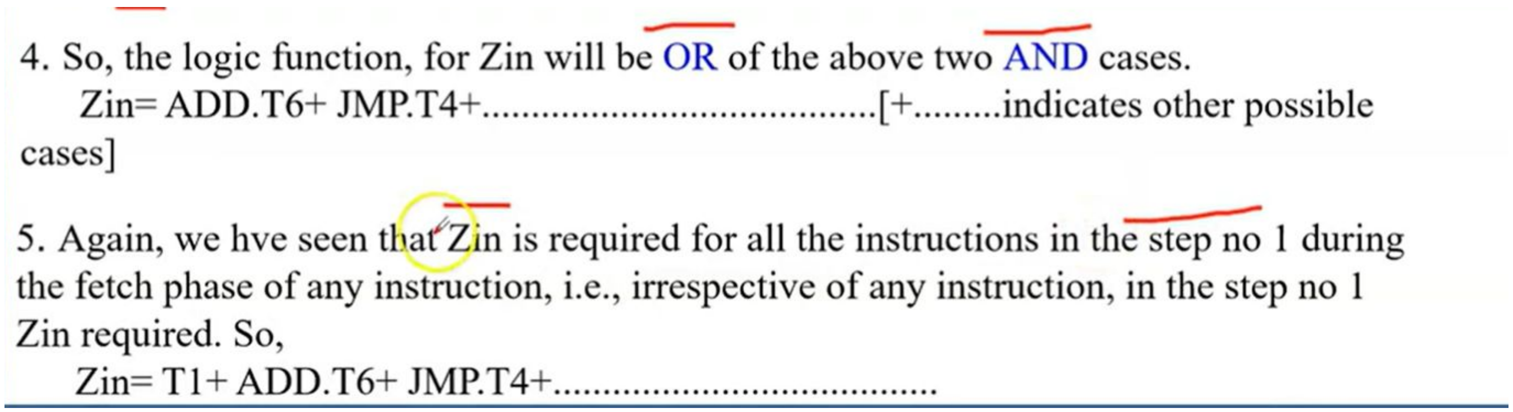


Add (R3), R1

1. PC out, MARin, Read, Select4, Add, **Zin**
2. Zout, PCin, Yin, WMFC
3. MDRout, IRin
4. R3out, MARin, Read
5. R1out, Yin, WMFC
6. MDRout, SelectY, Add, **Zin**
7. Zout, R1in, end

JMP L1

1. PC out, MARin, Read, Select4, Add, **Zin**
2. Zout, PCin, Yin, WMFC
3. MDRout, IRin
4. Address_field_of_IRout, SelectY, Add, **Zin**

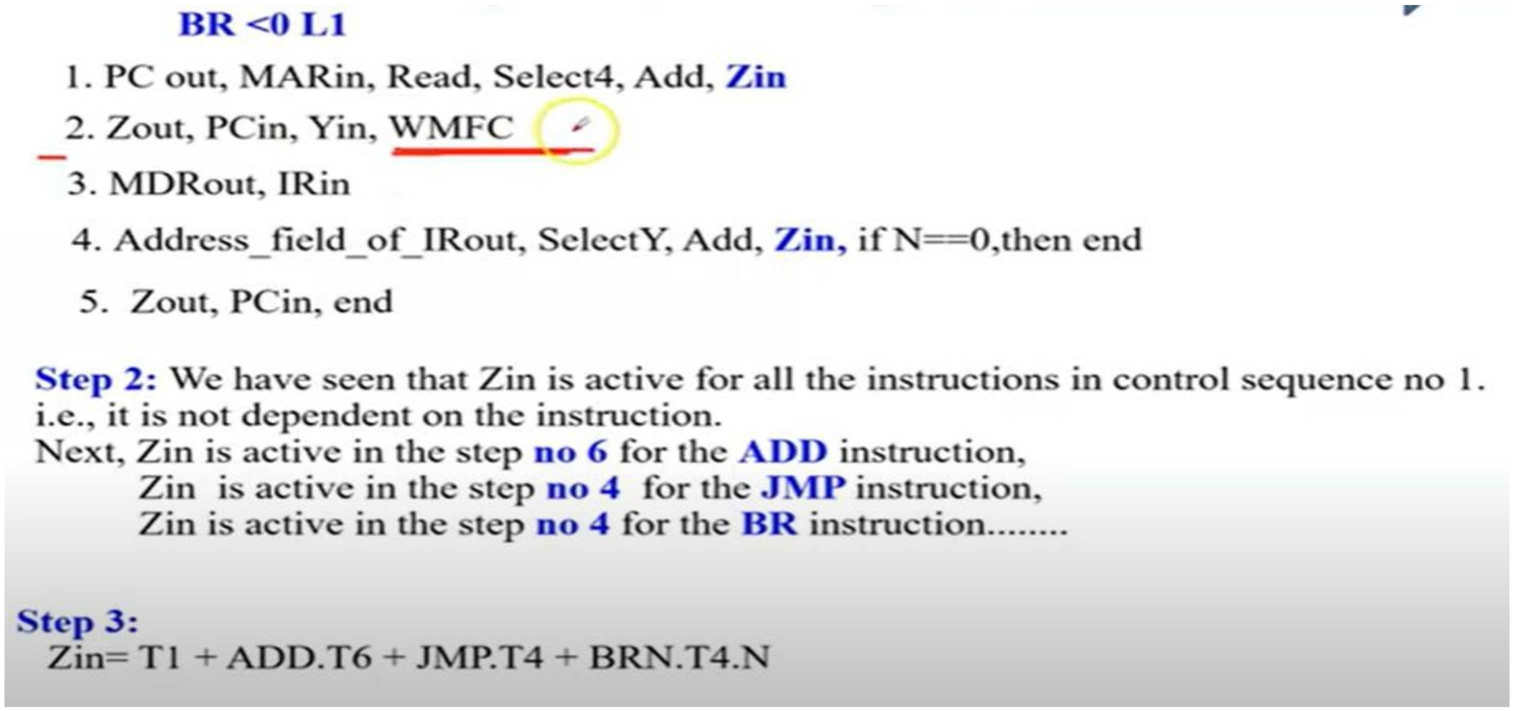


4. So, the logic function, for Zin will be OR of the above two AND cases.

Zin= ADD.T6+ JMP.T4+.......................................[+.........indicates other possible cases]

5. Again, we hve seen that Zin is required for all the instructions in the step no 1 during the fetch phase of any instruction, i.e., irrespective of any instruction, in the step no 1 Zin required. So,

Zin= T1+ ADD.T6+ JMP.T4+.......................................

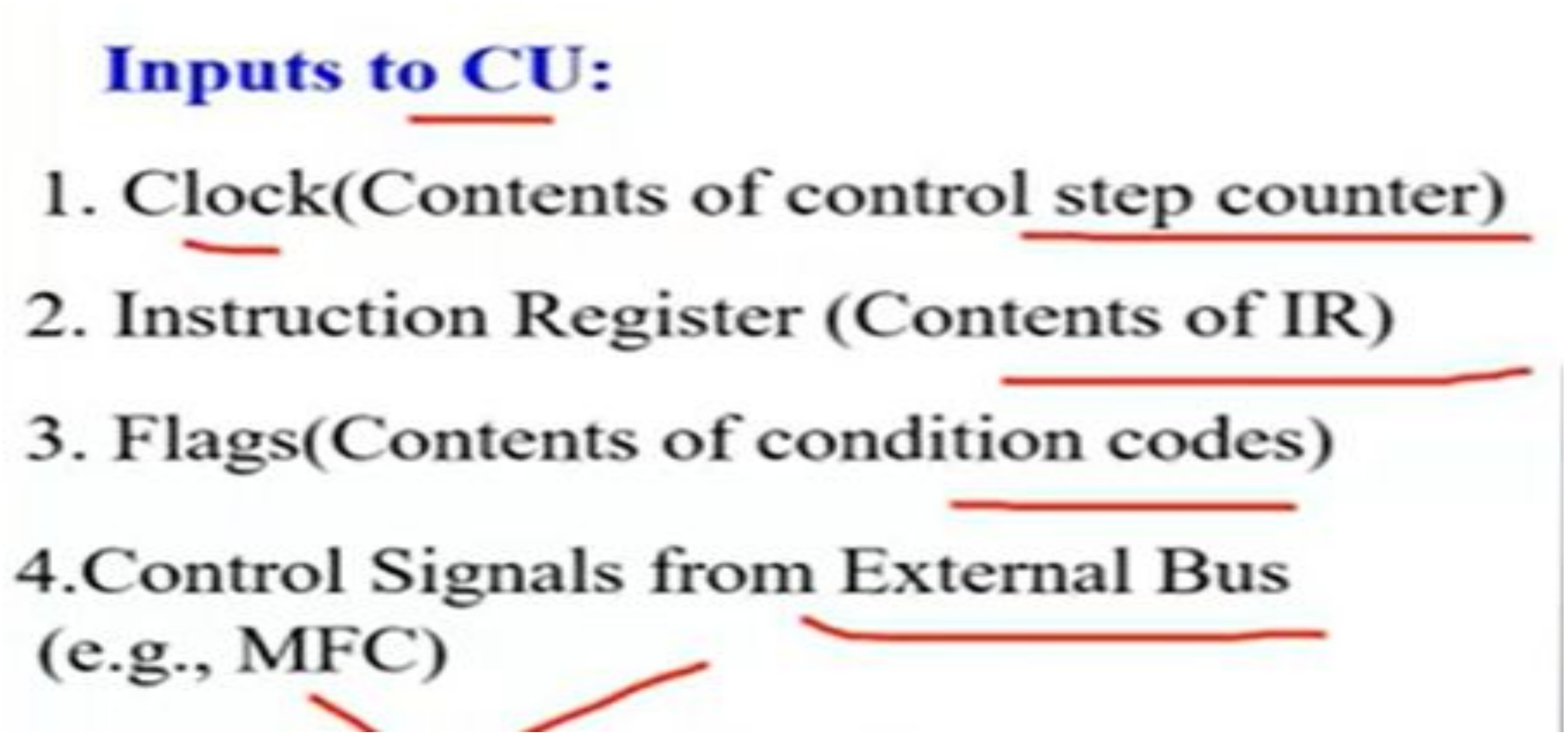


BR <0 L1

1. PC out, MARin, Read, Select4, Add, **Zin**
2. Zout, PCin, Yin, WMFC
3. MDRout, IRin
4. Address_field_of_IRout, SelectY, Add, **Zin,** if N==0,then end
5. Zout, PCin, end

Step 2: We have seen that Zin is active for all the instructions in control sequence no 1. i.e., it is not dependent on the instruction.
Next, Zin is active in the step **no 6** for the **ADD** instruction,
Zin is active in the step **no 4** for the **JMP** instruction,
Zin is active in the step **no 4** for the **BR** instruction........

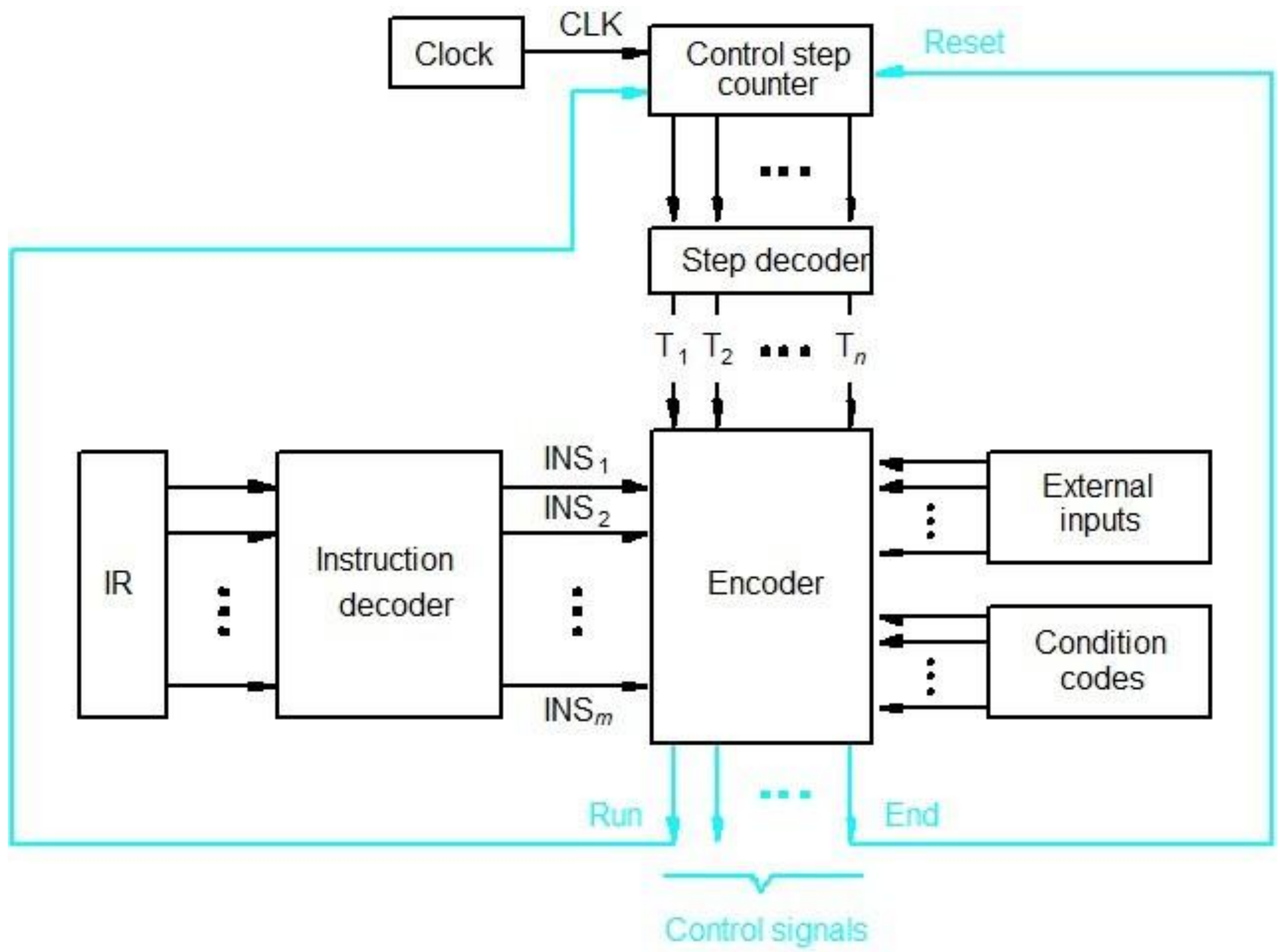
Step 3:
Zin= T1 + ADD.T6 + JMP.T4 + BRN.T4.N



Inputs to CU:

1. Clock(Contents of control step counter)
2. Instruction Register (Contents of IR)
3. Flags(Contents of condition codes)
4. Control Signals from External Bus (e.g., MFC)

Hardwired Control

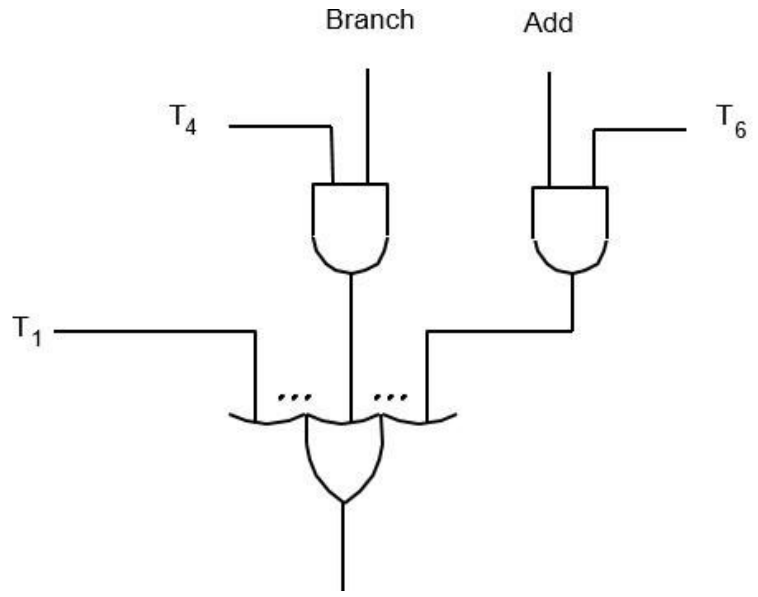


Hardwired Control

- The control unit **uses fixed logic circuits to interpret instructions** and **generate control signals** for them
- The **fixed logic circuits use contents** of the **control step counter, contents of the instruction register, contents of conditional code flag** and the **external input** signals such as MFC and **interrupts requests** to **generate control signal**.

Hardwired Control Signal Example

- $Z_{in} = T_1 + T_6 \bullet ADD + T_4 \bullet BR + \ldots$



Logic function for End signal

$End = ADD.T7 + BRN.T5.N + BRN.T4.\overline{N} + BR.T5 + \ldots\ldots\ldots\ldots\ldots\ldots$