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 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 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 between two 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

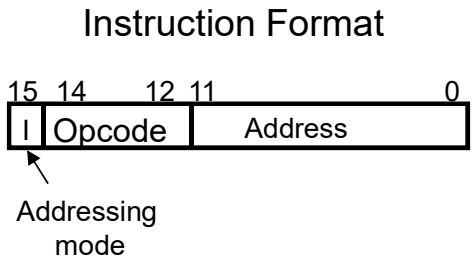
Instruction	Mnemonic	Description
Branch	BR	It transfers program control to the specified address
Jump	JMP	It transfers program control to the specified address
Compare	CMP	It compares the content of two registers
Test	TST	It performs bitwise AND operation on two operands
Skip	SKP	It skips the next instruction

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

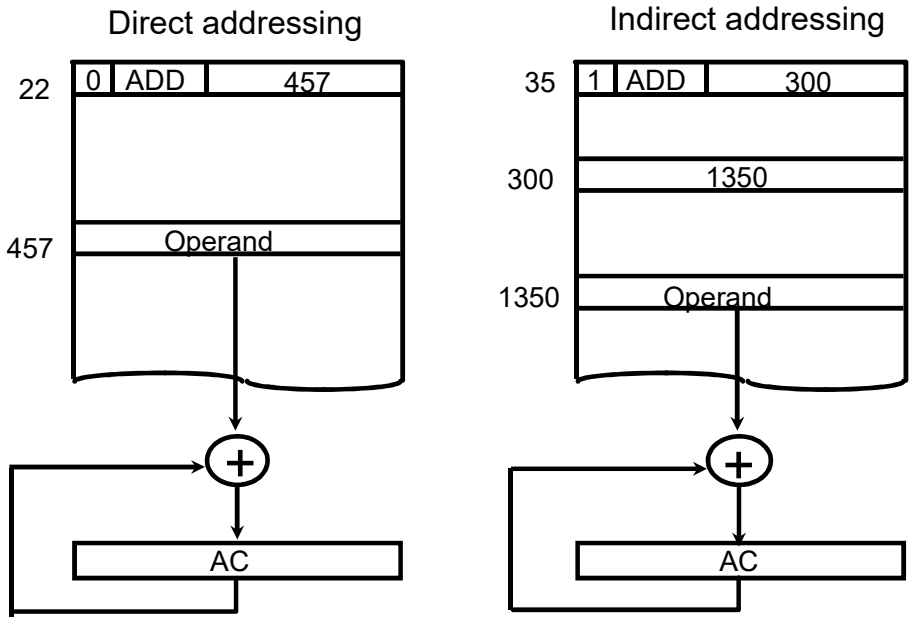
Instruction Format

- 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



Addressing Modes

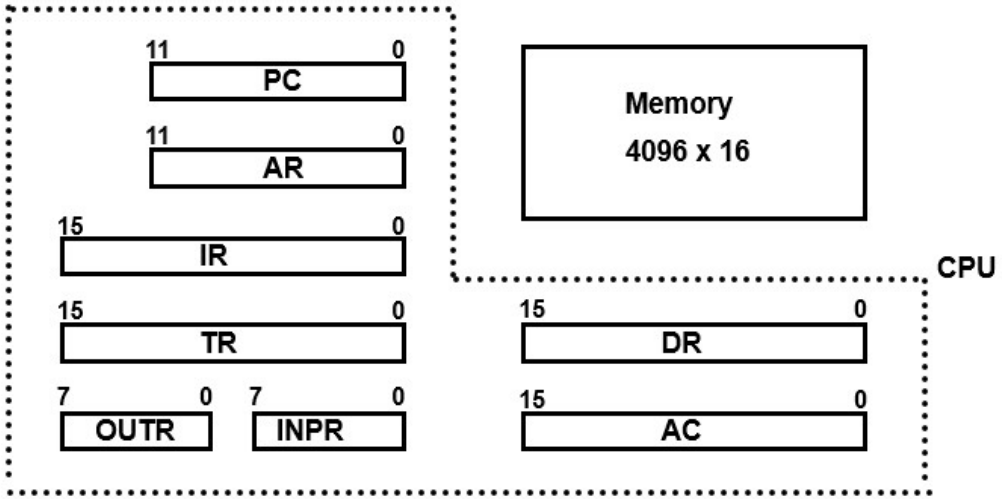
- The address field of an instruction can represent either
  - **Direct address**: it has the address of the operand (direct access to operand)
  - **Indirect address**: It is the address in memory from the address of the operand can be fetched.



- 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

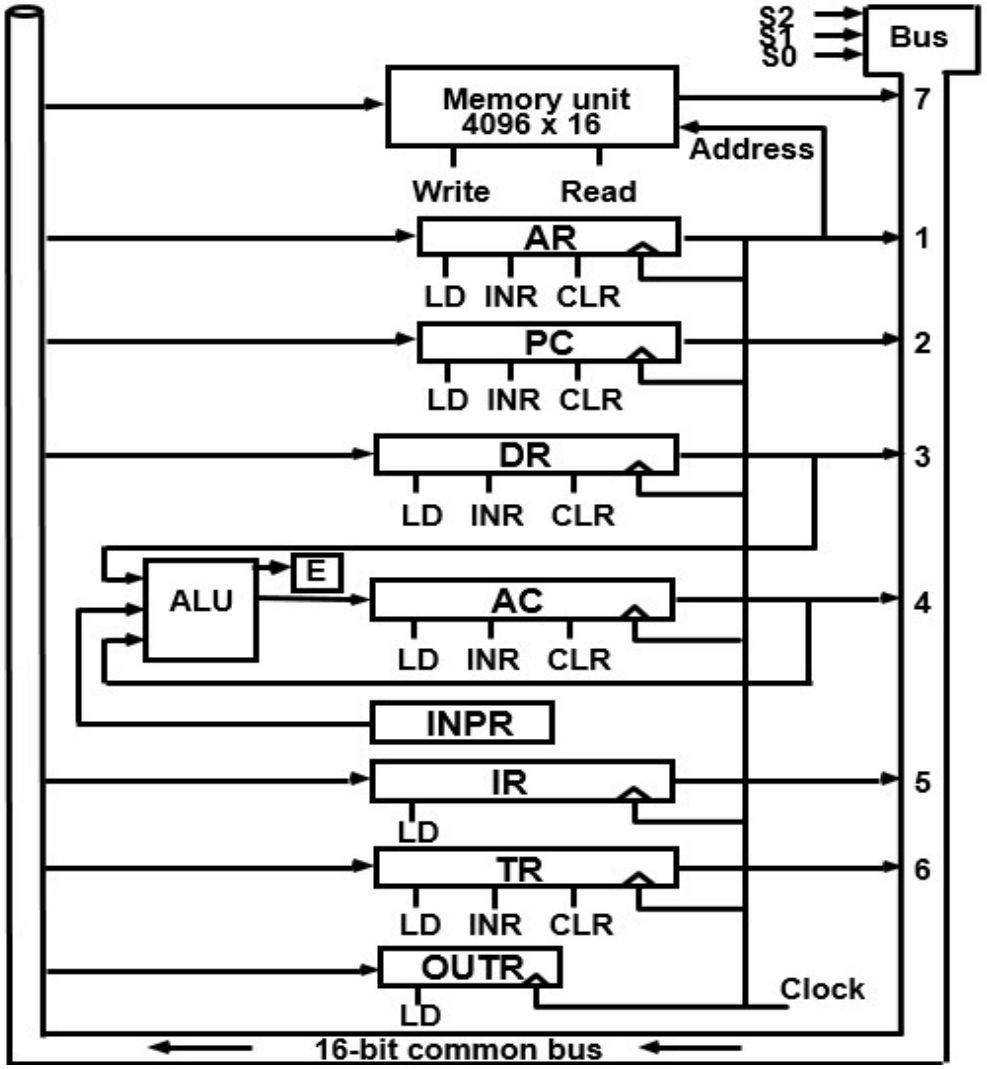
Registers in the Basic Computer



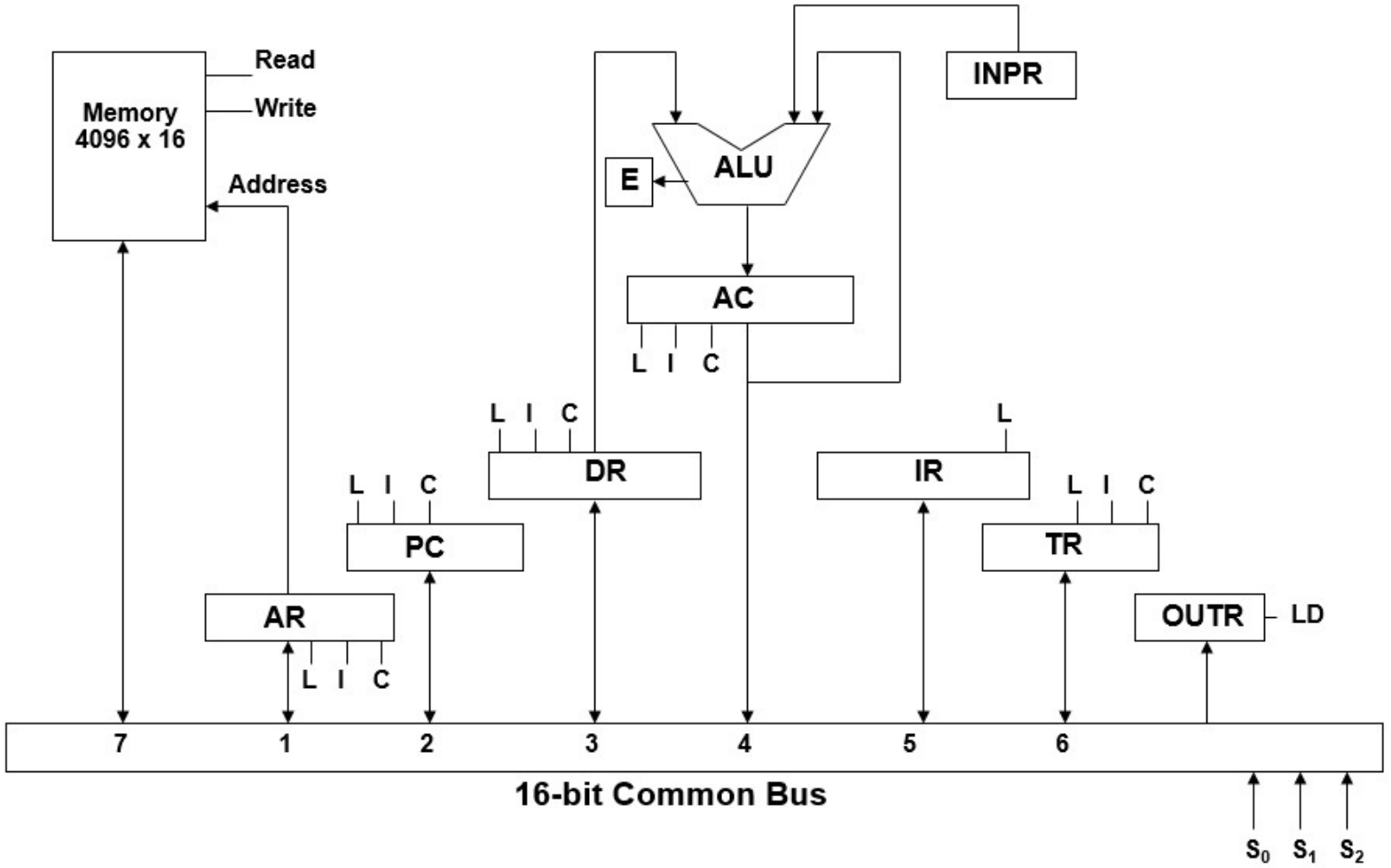
List of basic computer registers

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



Common Bus System

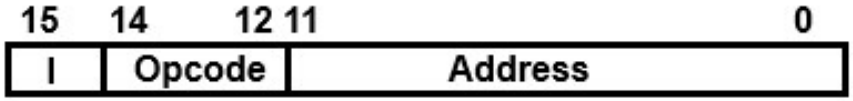
- Three control lines, S_2, S_1, and S_0 control which register the bus selects as its input

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**

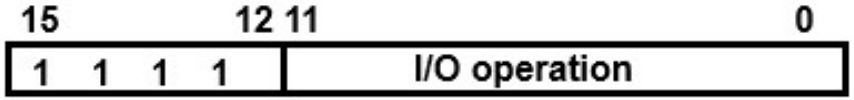
Basic Computer Instruction Format

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



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

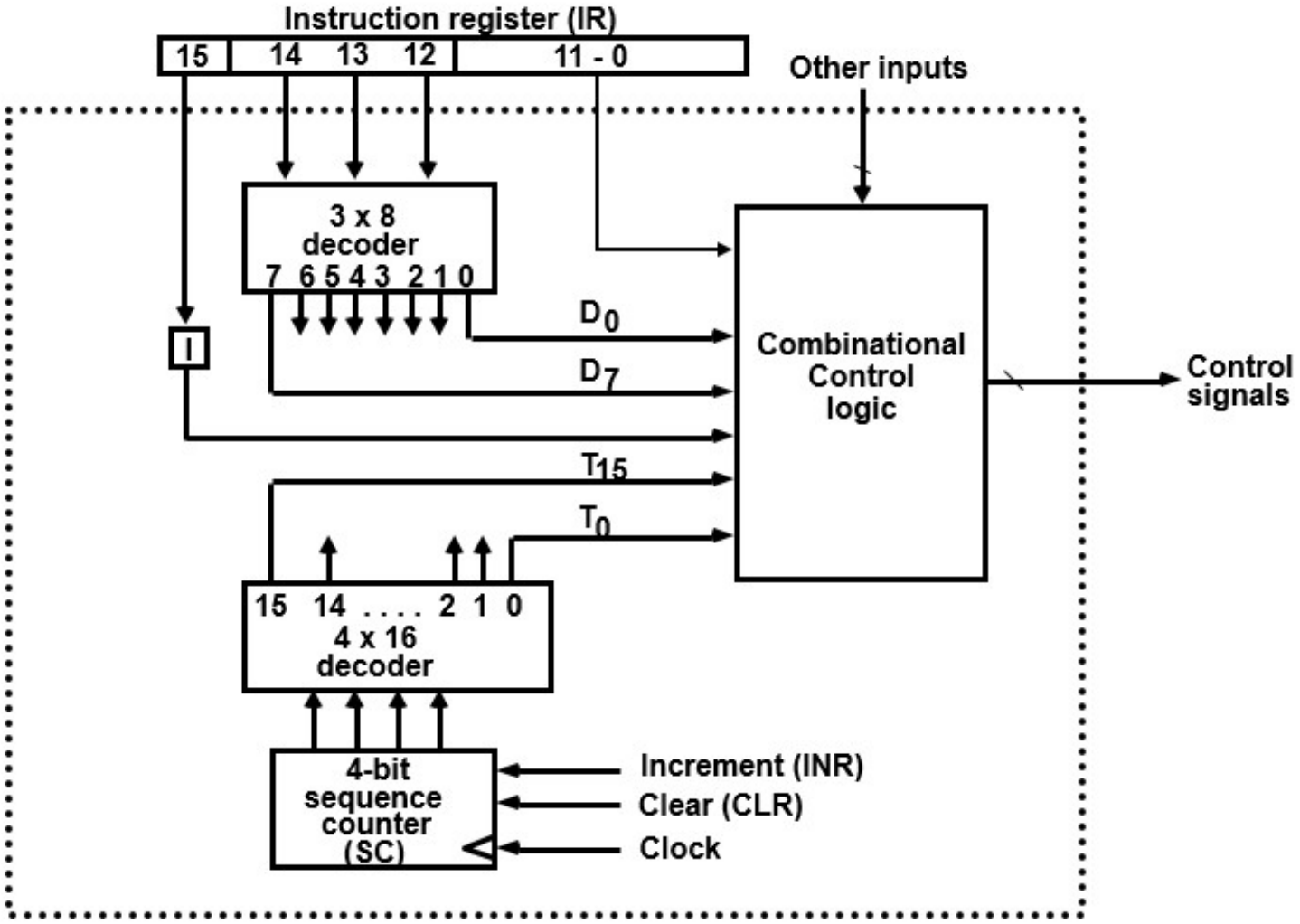
Input-Output Instructions (OP-code =111, I = 1)



Basic Computer Instructions

Symbol	Hex Code 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

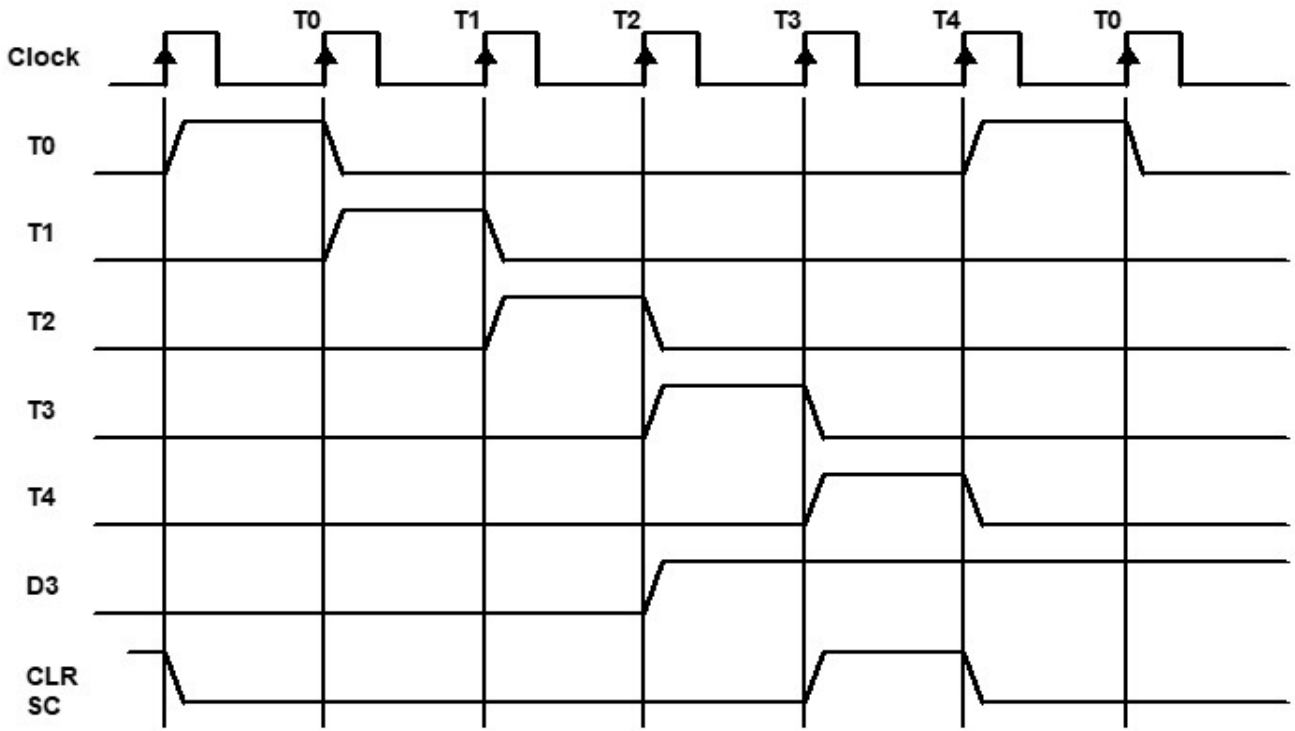
Control Unit of Basic Computer



Timing Signal

- Generated by 4-bit sequence counter and 4×16 decoder
- The Sequence Counter can be incremented or cleared.
- Example: $T_0, T_1, T_2, T_3, T_4, T_0, T_1, \ldots$
- Assume: At time T_4, SC is cleared to 0 if decoder output D3 is active.

D_3T_4: SC ← 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

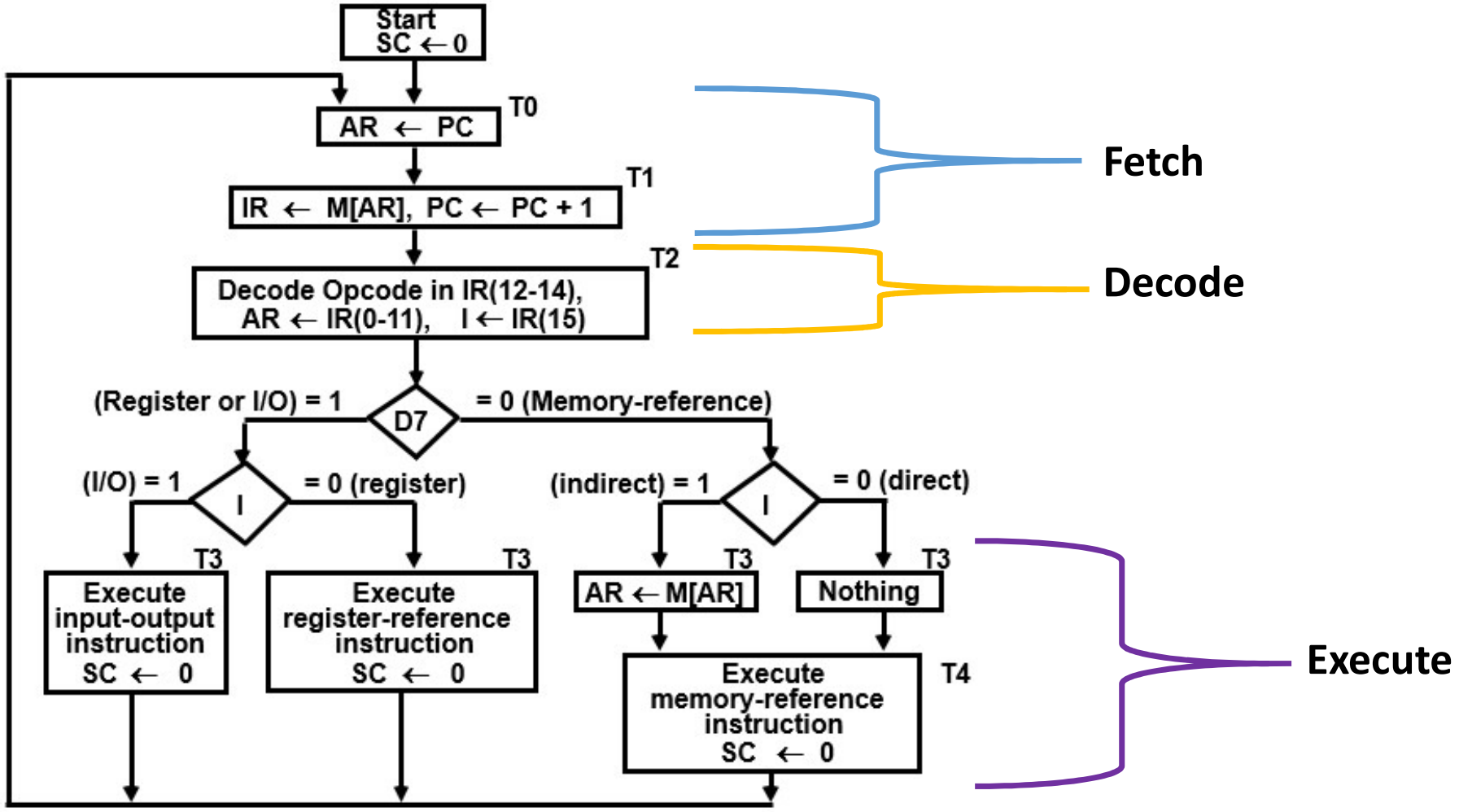
- **After an instruction is executed**, **the cycle starts again at step 1**, for the next instruction

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

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 ← PC
- IR ← M[AR], PC ← 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 ← IR(15), Decode ← IR(12-14) , AR ← IR(0-11)
- 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 ← 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,...,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)

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	$AC \leftarrow AC \wedge M[AR]$
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 $PC \leftarrow PC+1$

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 on 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 ← A – B
MPY	T, D, E	T ← D × E
ADD	T, T, C	T ← T + C
DIV	Y, Y, T	Y ← Y ÷ T

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	Y ← A
SUB	Y, B	Y ← Y – B
MOVE	T, D	T ← D
MPY	T, E	T ← T × E
ADD	T, C	T ← T + C
DIV	Y, T	Y ← Y ÷ T

(b) Two-address instructions

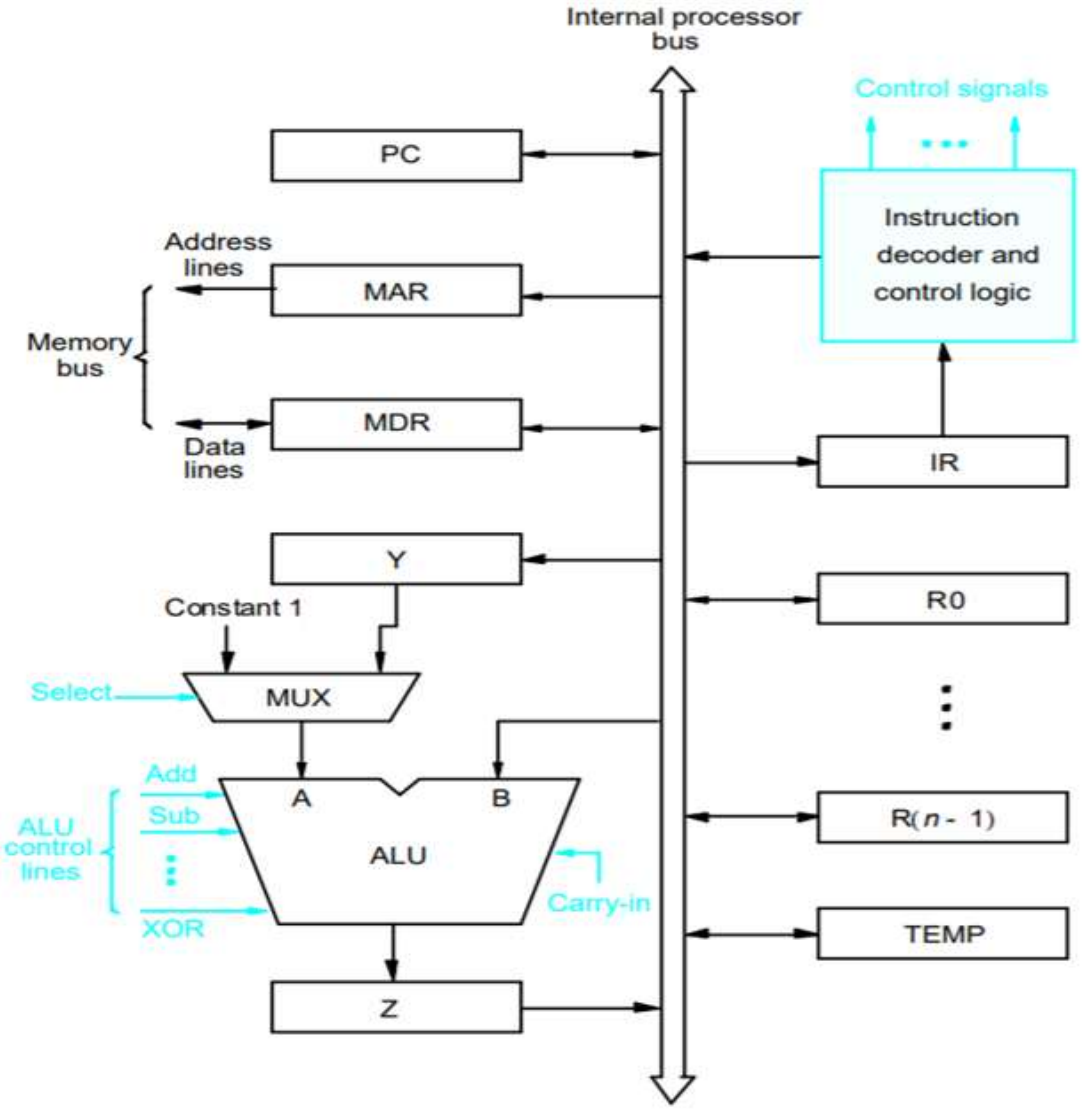
Instruction		Comment
LOAD	D	AC ← D
MPY	E	AC ← AC × E
ADD	C	AC ← AC + C
STOR	Y	Y ← AC
LOAD	A	AC ← A
SUB	B	AC ← AC – B
DIV	Y	AC ← AC ÷ Y
STOR	Y	Y ← AC

(c) One-address instructions

Execution of Complete Instruction

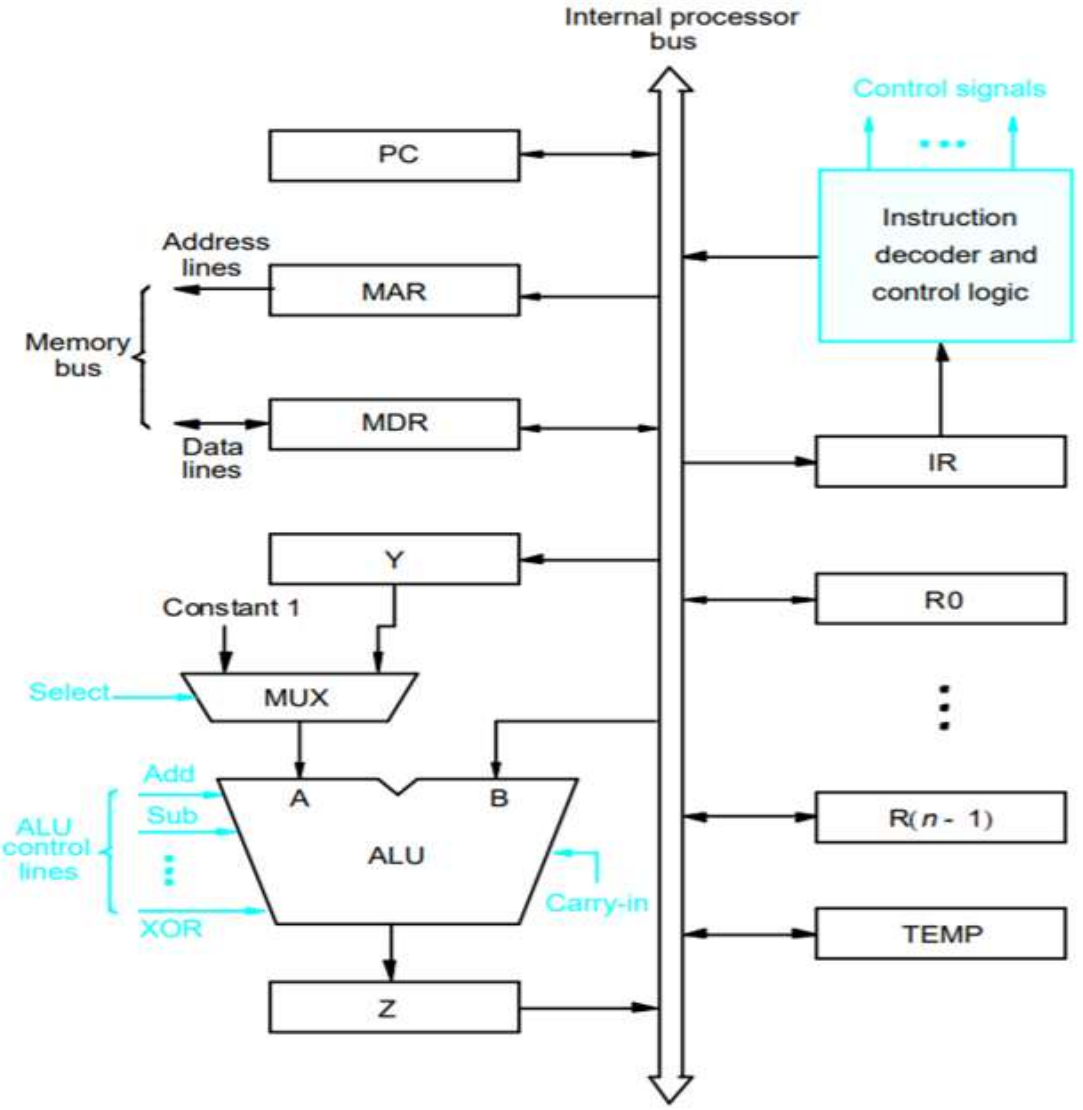
- ADD (R3) R1
  - Fetch the instruction from memory-IR
  - Fetch the operand from memory
  - Perform Addition operation
  - Load the result into R1

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

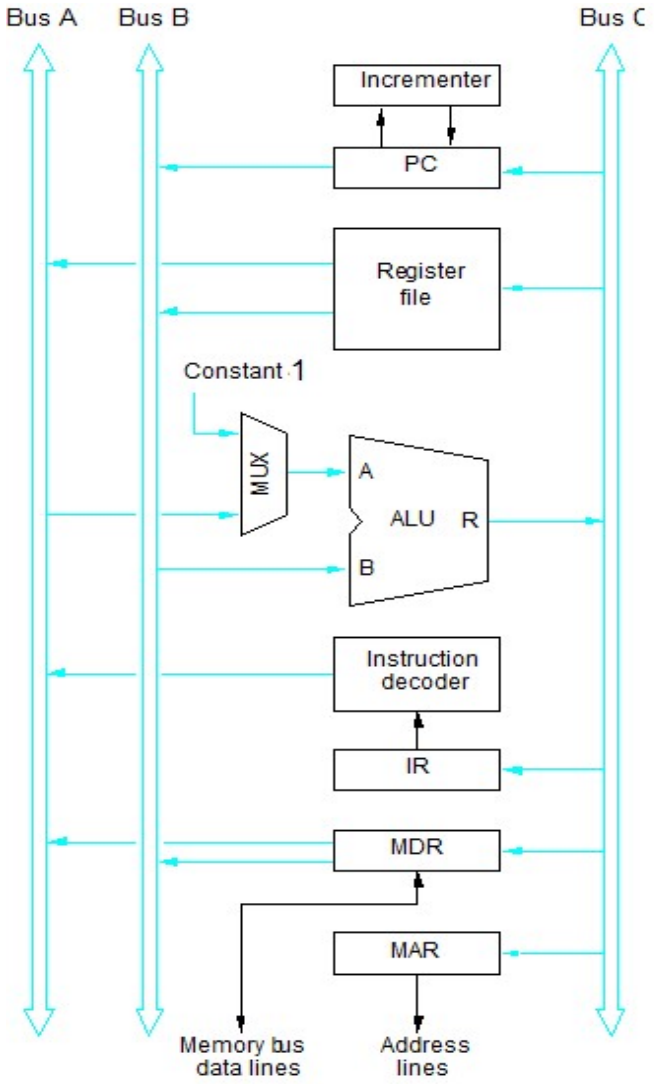
STEP	ACTION
1	PCout, MARin, Read, Select Constant 1, Add, Zin
2	Zout, PCin, Yin ,WMFC
3	MDRout, IRin
4	Offset-field-of-IRout, Add, Zin
5	Zout, PCin, END



Multiple Bus Organisation

Add R4, R5, R6

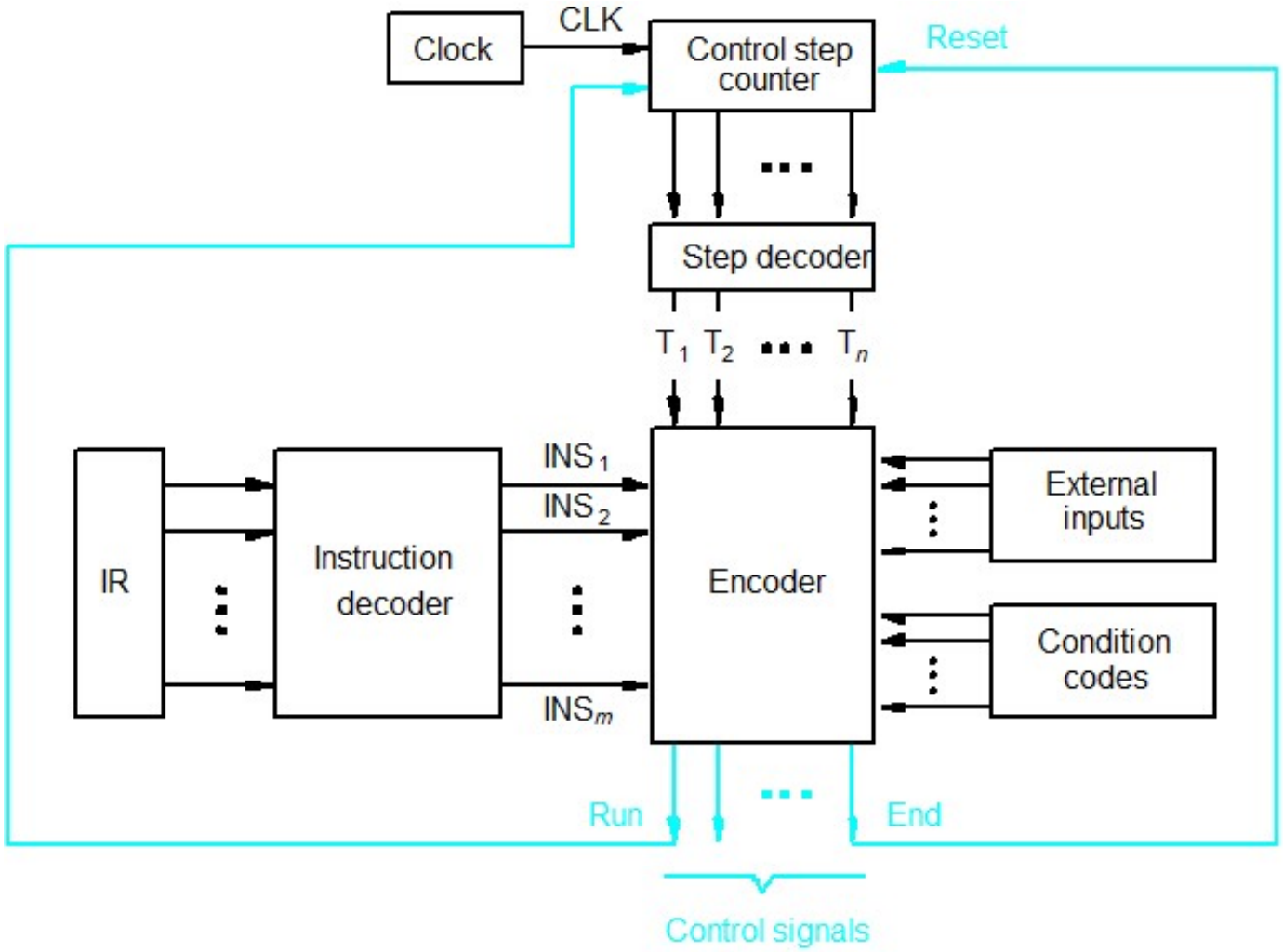
Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End



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 Control

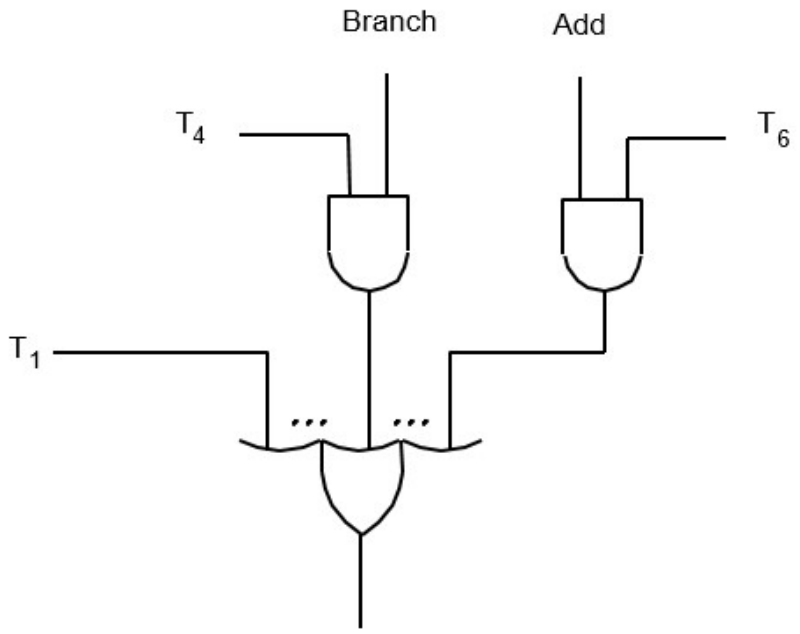


Hardwired Control

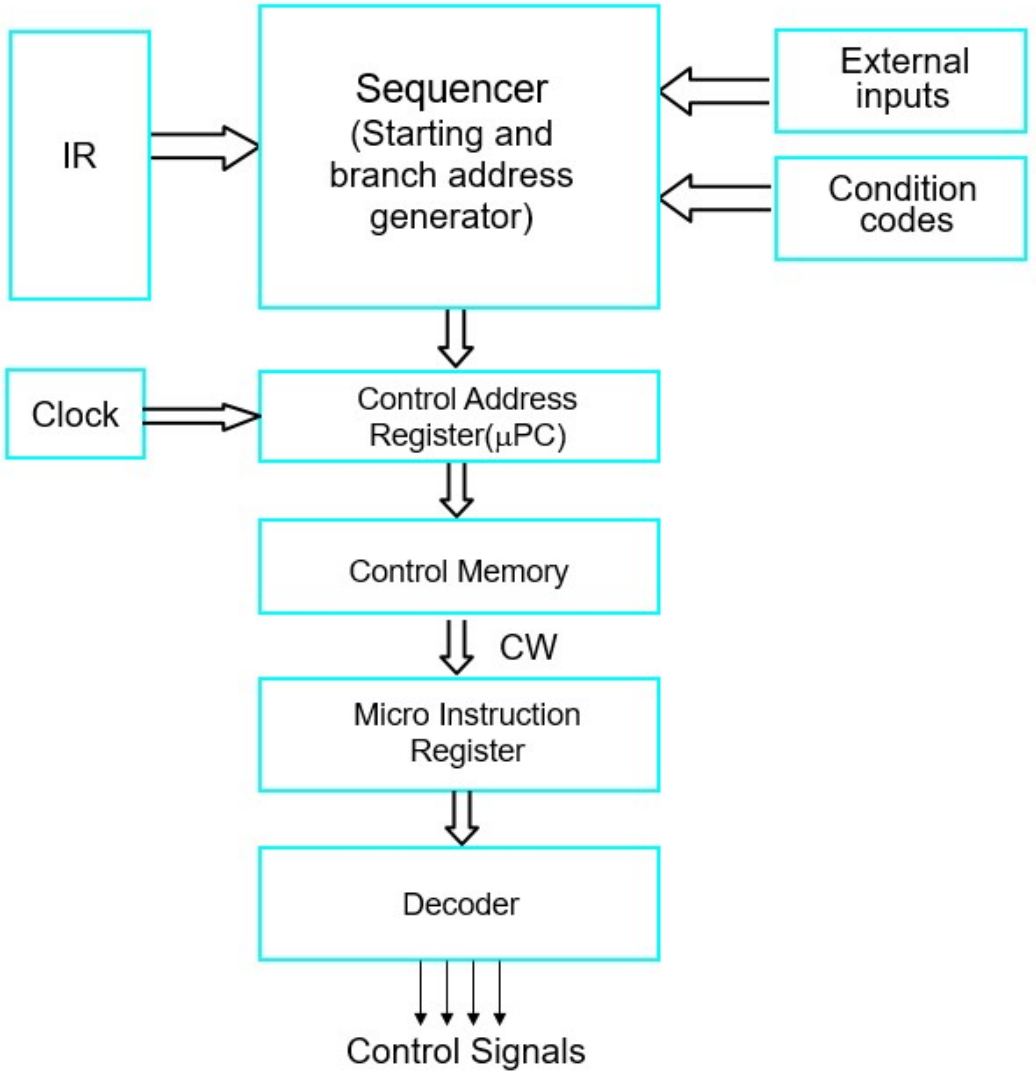
- The control unit **uses fixed logic circuits to interpret instructions** and **generate control signals** for them
- The **fixed logic circuits use contents** of the **control step counter**, **contents of the instruction register**, **contents of conditional code flag** and the **external input**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

Hardwired Control Signal Example

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



Micro-programmed Control



Micro-programmed Control

- **CONTROL MEMORY**
  - The micro routines for all the instructions in the instruction set of a computer are stored in a special memory called the Control Memory/Store
- **CONTROL WORD**
  - It is a word whose individual bits represent the various control signals.
- **MICRO-ROUTINE**
  - A sequence of control words corresponding to the control sequence of a machine instruction constitutes the micro-routine for that instruction.
- **MICRO-INSTRUCTION**
  - The individual control words in the micro-routine are referred to as micro-instruction

Micro-programmed Control

Micro - instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	..
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

Difference between Hardwired and Micro-programmed Control

ATTRIBUTE	HARDWIRED CONTROL UNIT	MICRO-PROGRAMMED CONTROL UNIT
SPEED	FAST	SLOW
COST OF IMPLEMENTATION	MORE	CHEAPER
FLEXIBILITY	NOT FLEXIBLE, DIFFICULT TO MODIFY FOR NEW INSTRUCTION	FLEXIBLE, NEW INSTRUCTIONS CAN BE ADDED
ABILITY TO HANDLE COMPLEX INSTRUCTION	DIFFICULT	EASIER
INSTRUCTION SET SIZE	SMALL	LARGE
CONTROL MEMORY	ABSENT	PRESENT

Micro Instruction

- A **straightforward way** to **structure microinstructions** is to **assign one bit position to each control signal**.
- However, **this is very inefficient**.
- The **length can be reduced**: most signals are not needed simultaneously, and many signals are mutually exclusive.
- All **mutually exclusive signals are placed in the same group in binary coding**.

Partial Format for Micro Instruction

Microinstruction

F1	F2	F3	F4	F5	F6	F7	F8	...

F1 (4 bits)	F2 (3 bits)	F3 (3 bits)	F4 (4 bits)	F5 (2 bits)
0000: No transfer	000: No transfer	000: No transfer	0000: Add	00: No action
0001: PC_{out}	001: PC_{in}	001: MAR_{in}	0001: Sub	01: Read
0010: MDR_{out}	010: IR_{in}	010: MDR_{in}	⋮	10: Write
0011: Z_{out}	011: Z_{in}	011: $TEMP_{in}$	1111: XOR	
0100: $R0_{out}$	100: $R0_{in}$	100: Y_{in}	16 ALU functions	
0101: $R1_{out}$	101: $R1_{in}$			
0110: $R2_{out}$	110: $R2_{in}$			
0111: $R3_{out}$	111: $R3_{in}$			
1010: $TEMP_{out}$				
1011: $Offset_{out}$				

F6 (1 bit)	F7 (1 bit)	F8 (1 bit)
0: SelectY	0: No action	0: Continue
1: Select4	1: WMFC	1: End

Micro Instruction types

- The grouping of control signals can be done in one of the two ways
  - **Vertical Micro-programming**
    - Each micro-instruction specifies single (or few) micro-operations to be performed.
  - **Horizontal Micro-programming**
    - Each micro-instruction specifies many different micro-operations to be performed in parallel.

Difference between Horizontal & Vertical Microprogramming

HORIZONTAL MICRO-PROGRAMMING	VERTICAL MICRO-PROGRAMMING
Long formats	Short formats
Ability to express a high degree of parallelism	Limited ability to express parallel micro-operations
Little encoding of the control information	Considerable encoding of the control information
Useful when higher operating speed is required	Slower operating speed

Addressing Modes

- **Specifies a rule for interpreting or modifying the address field** of the instruction (before the operand is actually referenced)

- Variety of addressing modes
  - to **give programming flexibility** to the user
  - to **use** the **bits in the address field** of the instruction **efficiently**

Types of Addressing Modes

- Implied Addressing Mode
- Immediate Addressing Mode
- Register Addressing Mode
- Register Indirect Addressing Mode
- Autoincrement/ Autodecrement Addressing Mode
- Direct Addressing Mode
- Indirect Addressing Mode
- Relative Addressing Mode
  - PC Relative Addressing Mode
  - Indexed Addressing Mode
  - Base Register Addressing Mode

Implied Addressing Mode

- **Address** of the **operands** are **specified implicitly in** the **definition** of the instruction
  - **No need** to **specify address** in the instruction
  - EA = AC, or EA = Stack[SP]

  Examples : CLA, CME, INP

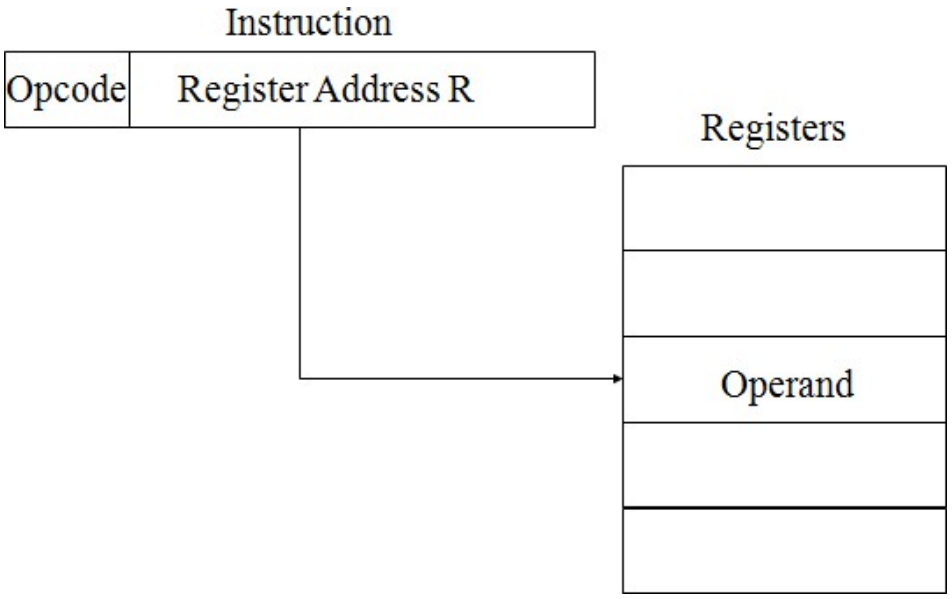
Immediate Addressing Mode

- **Instead of specifying the address of the operand, operand itself is specified**
  - No need to specify address in the instruction
  - However, **operand itself needs to be specified**
  - Sometimes, **require more bits than the address**
  - Fast to acquire an operand

Example : ADD 5

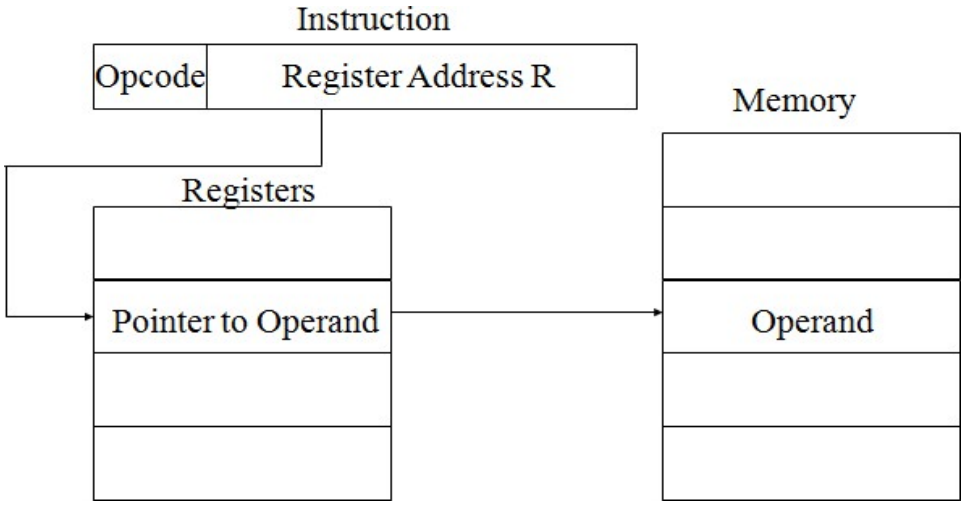
Register Addressing Mode

- **Address** specified in the instruction **is the register address**
  - **Designated operand need to be in a register**
  - Shorter address than the memory address
  - Saving address field in the instruction
  - Faster to acquire an operand than the memory addressing
    - EA = IR(R) (IR(R): Register field of IR)



Register Indirect Addressing Mode

- Instruction specifies a register which contains the memory address of the operand
  - Saving instruction bits since register address is shorter than the memory address
  - Slower to acquire an operand than both the register addressing or memory addressing
  - EA = [IR(R)] ([x]: Content of x)



Autoincrement/Autodecrement Addressing Mode

- When the address in the register is used to access memory, the value in the register is incremented or decremented by 1 automatically

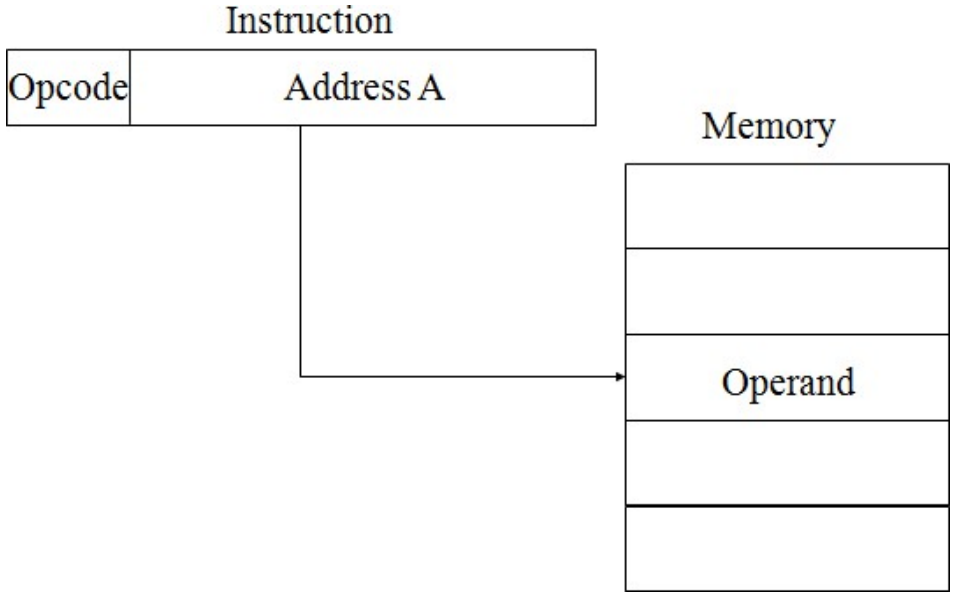
Example:

- **Autoincrement:** Load R1, (R2)+ is interpreted as R1 ← [[R2]] followed by R2 ← [R2] + d where d is step size(step size is dependent on size of operand).

- **Autodecrement:** Load R1, -(R2) is interpreted as R2 ← [R2] – d followed by R1 ← [[R2]] where d is step size size(step size is dependent on size of operand)

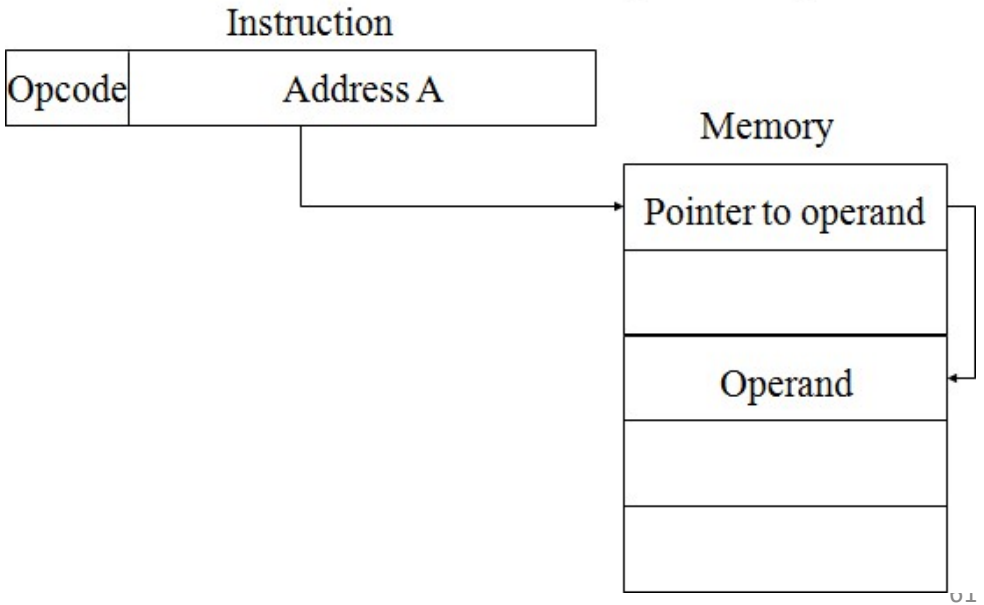
Direct Addressing Mode

- Instruction specifies the memory address which can be used directly to access the memory
  - Faster than the other memory addressing modes
  - Too many bits are needed to specify the address for a large physical memory space
  - EA = IR(addr) (IR(addr): address field of IR)



Indirect Addressing Mode

- The address field of an instruction specifies the address of a memory location that contains the address of the operand
  - When the abbreviated address is used large physical memory can be addressed with a relatively small number of bits
  - Slow to acquire an operand because of an additional memory access
  - EA = M[IR(address)]



Relative Addressing Mode

- The **address field** of an instruction **specifies the part of the address** (abbreviated address) which can be used **along with a designated register** to **calculate the effective address of the operand**
  - Address field of the instruction is short
  - Large physical memory can be accessed with a small number of address bits
  - EA = f(IR(address), R), R is sometimes implied

- 3 different Relative Addressing Modes depending on R;
  - * **PC Relative Addressing Mode** (R = PC)
    - EA = PC + IR(address)
  - * **Indexed Addressing Mode** (R = XR, where XR: Index Register)
    - EA = XR + IR(address)
  - * **Base Register Addressing Mode** (R = BAR, where BAR: Base Address Register)
    - EA = BAR + IR(address)

Addressing Modes Example

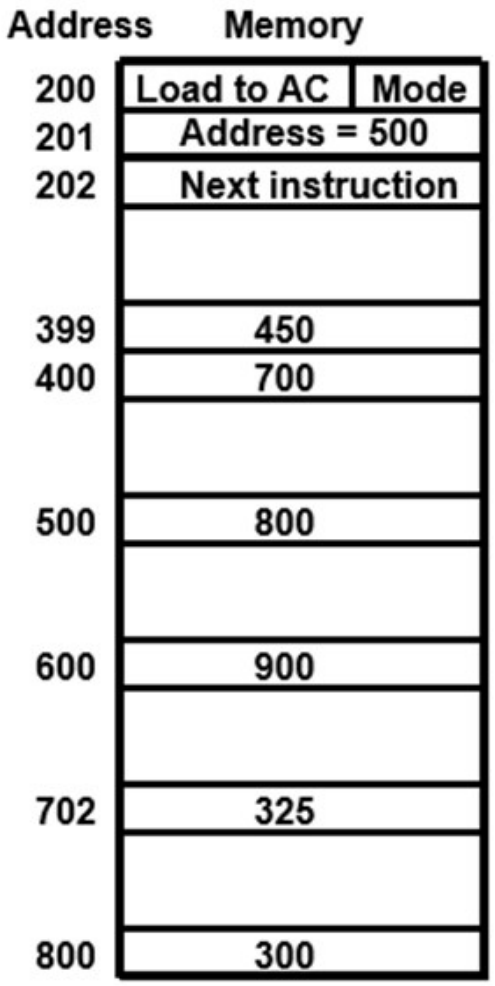
PC = 200

R1 = 400

XR = 100

AC

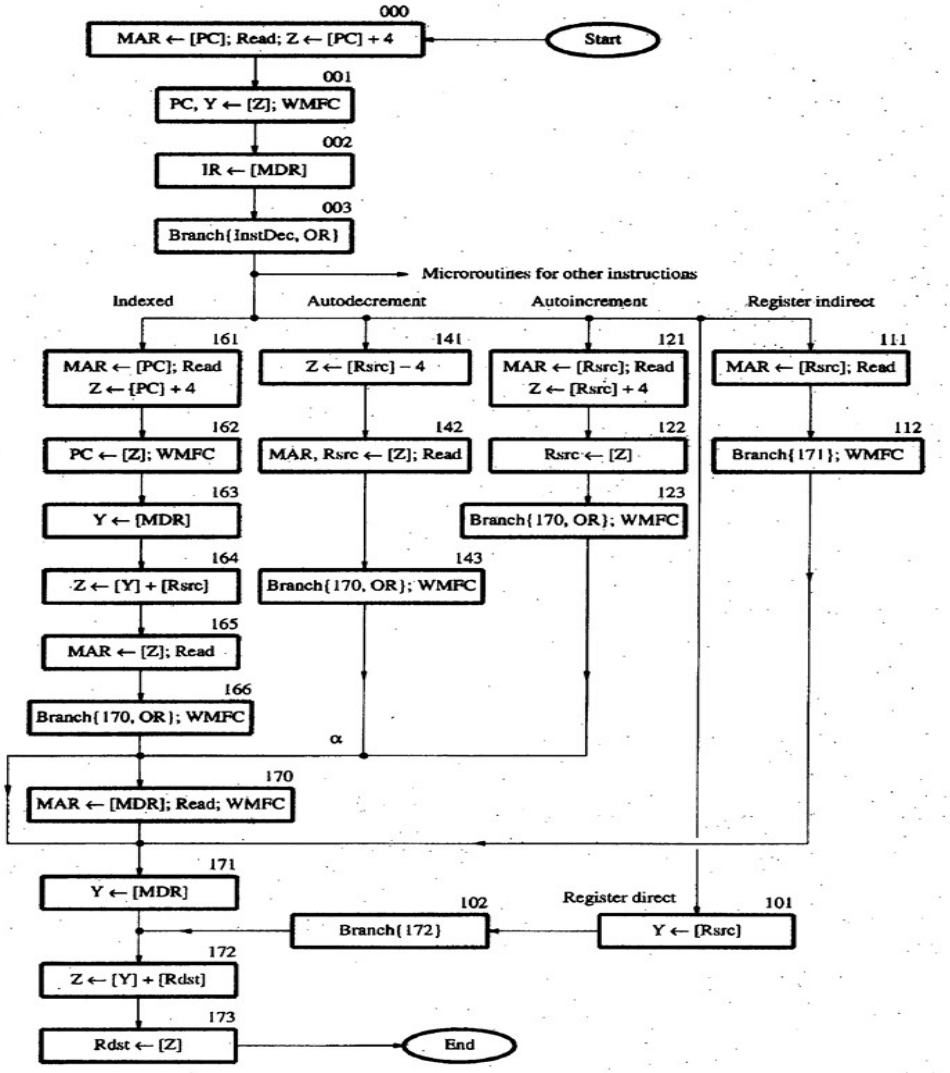
Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address	800	/* AC ← ((500)) */	300
Relative address	702	/* AC ← (PC+500) */	325
Indexed address	600	/* AC ← (XR+500) */	900
Register	-	/* AC ← R1 */	400
Register indirect	400	/* AC ← (R1) */	700
Autoincrement	400	/* AC ← (R1)+ */	700
Autodecrement	399	/* AC ← -(R1)+ */	450



Micro-Programmed Sequencing

- **If all micro-programs** require only **straightforward sequential execution of microinstructions** except for branches, **letting a µPC governs the sequencing would be efficient**.
- However, two disadvantages:
  - Having a **separate micro-routine for each machine instruction** results in **a large total number of microinstructions** and a **large control store**.
  - **Longer execution time** because it takes more time to carry out the required branches.
- Example: **Add src, Rdst**
- Four addressing modes: register, autoincrement, autodecrement, and indexed (with indirect forms).

Micro-Program for Add src Rdst



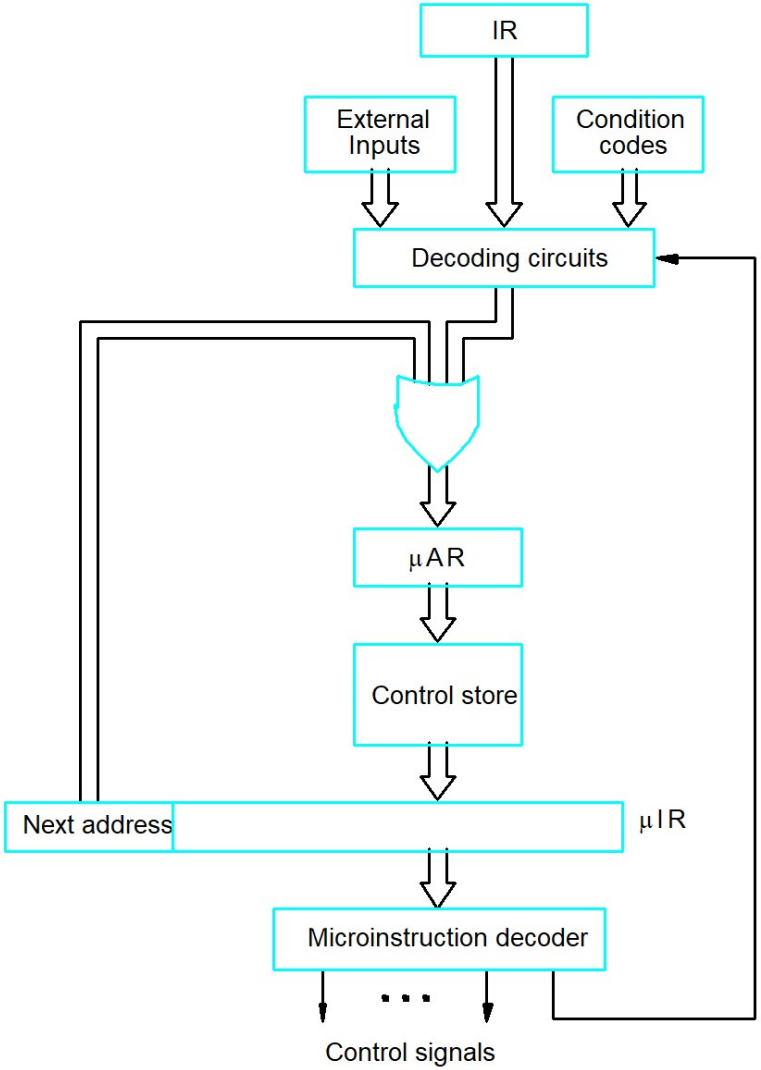
Wide Branch Addressing

- **Generating branch addresses** becomes more **difficult as the number of branches increases**.
- **Programmable Logic Array can be used** to generate the required branch address.
- The **simple and inexpensive way of generating branch addresses** is known as wide branch addressing
- This is **achieved by connecting the opcode bits** of the instruction register **as inputs to the PLA** which acts as decoder.
- The **output of the PLA is the address of the desired micro-routine**.

Micro Instruction with Next Address Field

- The **micro-program requires several branch microinstructions**, which perform **no useful operation in the datapath**.
- A **powerful alternative approach** is to **include an address field as a part of every microinstruction** to indicate the location of the next microinstruction to be fetched.
- **Pros: separate branch microinstructions are virtually eliminated**; few limitations in assigning addresses to microinstructions.
- **Cons**: **additional bits for the address field** (around 1/6).

Micro Instruction with Next Address Field



Micro Instruction with Next Address Field

Microinstruction

F0	F1	F2	F3

F0 (8 bits)	F1 (3 bits)	F2 (3 bits)	F3 (3 bits)
Address of next microinstruction	000: No transfer 001: PC_{out} 010: MDR_{out} 011: Z_{out} 100: $Rsrc_{out}$ 101: $Rdst_{out}$ 110: $TEMP_{out}$	000: No transfer 001: PC_{in} 010: IR_{in} 011: Z_{in} 100: $Rsrc_{in}$ 101: $Rdst_{in}$	000: No transfer 001: MAR_{in} 010: MDR_{in} 011: $TEMP_{in}$ 100: Y_{in}

F4	F5	F6	F7

F4 (4 bits)	F5 (2 bits)	F6 (1 bit)	F7 (1 bit)
0000: Add 0001: Sub ⋮ 1111: XOR	00: No action 01: Read 10: Write	0: SelectY 1: Select4	0: No action 1: WMFC

F8	F9	F10

F8 (1 bit)	F9 (1 bit)	F10 (1 bit)
0: NextAdrs 1: InstDec	0: No action 1: OR_{mode}	0: No action 1: OR_{indsrc}

Implementation of Micro-routine using next Micro-instruction address field

Octal address	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
0 0 0	0 0 0 0 0 0 0 1	0 0 1	0 1 1	0 0 1	0 0 0 0	0 1	1	0	0	0	0
0 0 1	0 0 0 0 0 0 1 0	0 1 1	0 0 1	1 0 0	0 0 0 0	0 0	0	1	0	0	0
0 0 2	0 0 0 0 0 0 1 1	0 1 0	0 1 0	0 0 0	0 0 0 0	0 0	0	0	0	0	0
0 0 3	0 0 0 0 0 0 0 0	0 0 0	0 0 0	0 0 0	0 0 0 0	0 0	0	0	1	1	0
1 2 1	0 1 0 1 0 0 1 0	1 0 0	0 1 1	0 0 1	0 0 0 0	0 1	1	0	0	0	0
1 2 2	0 1 1 1 1 0 0 0	0 1 1	1 0 0	0 0 0	0 0 0 0	0 0	0	1	0	0	1
1 7 0	0 1 1 1 1 0 0 1	0 1 0	0 0 0	0 0 1	0 0 0 0	0 1	0	1	0	0	0
1 7 1	0 1 1 1 1 0 1 0	0 1 0	0 0 0	1 0 0	0 0 0 0	0 0	0	0	0	0	0
1 7 2	0 1 1 1 1 0 1 1	1 0 1	0 1 1	0 0 0	0 0 0 0	0 0	0	0	0	0	0
1 7 3	0 0 0 0 0 0 0 0	0 1 1	1 0 1	0 0 0	0 0 0 0	0 0	0	0	0	0	0

Prefetching Microinstructions

- The **disadvantage of micro-programmed control** is that it results **slower operating speed** because of the **time it takes to fetch micro-instructions from** the **control memory**.
- This **problem** can be **solved by prefetching the next microinstruction** while the current one is being executed.
- In this technique, **the execution time** can be **overlapped with the fetch time**.

Complex Instruction Set Computer(CISC)

- A computer system with **large number of instructions.**
- **Large number of addressing modes.**
- It uses **Variable length instruction format.**
- It **has some instructions** that **perform specialized task** and are **used infrequently.**
- It has **instructions that manipulate operands in memory.**
- Some examples are **IBM 370/168, Intel 80386, Intel 80286, Sun-3/75.**

Reduced Instruction Set Computer(RISC)

- It has **relatively few instructions.**
- In comparison to CISC, it has **few addressing modes.**
- The **memory access is limited to load and store** instructions.
- **All** the **operations** are **done within the registers** of the CPU.
- **Single cycle instruction execution**.
- **Fixed length** easily decoded instruction format.
- **Hardwired control unit** rather than micro-programmed control unit.
- Some examples are **MIPS R2000, SUN SPARC, Intel i860, Motorola 8800.**

Examples of CISC and RISC

	CISC examples		RISC examples	
	IBM 370/168	**VAX 11/780**	**88000**	**R4000**
Year developed	1973	1978	1988	1991
The number of instruction	208	303	51	94
Instruction size (bytes)	2 - 6	2 - 57	4	4
Addressing modes	4	22	3	1
Cache size (KB)	64	64	16	128