[redacted] Zatin Gupta (42

INSTRUCTION:→ Types:

(1) Data Transfer Instructions
(2) Data Manipulation Instructions.
(3) Program Control Instructions.

(1) Data Transfer Instructions:-

- It move data from one place to another w/o changing data.

Common data transfer – Between –

(1) Memory & processor registers. ✓
(2) Processor registers & I/O ✓
(3) Between processor registers. ✓

Ex:

Name	Mnemonic	Description
Load	LD	Transfer from memory to processor Regist
Store	ST	Transfer from processor register to Mem
Move	MOV	Transfer from one register to other (Memory (1))
Exchange	XCH	Swap info. b/w two registers or a register & memory word
Input	IN	] Transfer data among registers & I/O terminals.
Output	OUT	
Push	PUSH	] Transfer data b/w processor register & Stack
Pop	POP	

(2) Data Manipulation Instructions:-

- Perform operations on data & can be of 3 Types as:-

(1) Arithmatic Instructions
(2) Logical & Bit manipulation instructions
(3) Shift Instructions.

(1) Arithmatic Instructions:

Name	Mnemonic	Name	Mnemonic
Increment	INC	Negate (2's comp)	NEG
Decrement	DEC		
Add	ADD		
Subtract	SUB		
Multiply	MUL		
Divide	DIV		
Add with Carry	ADDC		
Subtract with borrow	SUBB		

* ADD operation for different data type –

ADD I Add two binary integer numbers
ADD F Add two floating point numbers
ADD D Add two decimal numbers in BCD

(2) Logical & bit Manipulation Instruction:

Name	Mnemonic	
Clear	CLR	
Complement	COM	
AND	AND	$x \cdot 0 = 0$, $x \cdot 1 = x$
OR	OR	$x + 0 = x$, $x + 1 = 1$
Exclusive-OR	XOR	$x \oplus 1 = x'$, $x \oplus 0 = x$
clear carry	CLRC	
Set carry	SETC	
Complement carry	COMC	
Enable Interrupt	EI	
Disable Interrupt	DI	

3) Shift Instruction:

Name	Mnemonic
logical shift right	SHR
logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

B) Program Control Instruction:

Name	Mnemonic
Branch	BR
JUMP	JMP
SKip	SKP
Call	CALL
Return	RET
Compare (By subtraction)	CMP
Test (By ANDing)	TST

Conditional Branch Instructions —

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	Z=1
BNZ	" if not zero	Z=0
BC	" if carry	C=1
BNC	" if no carry	C=0
BP	" if plus	S=0
BM	" if minus	S=1
BV	" if overflow	V=1
BNV	" if no overflow	V=0

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

INSTRUCTION FORMAT

Manuals of system explain –
- Internal construction of CPU.
- Processor registers available.
- Logical capabilities of processor registers.
- H/w implemented instructions.
- Binary code format
- Precise definition of each instruction.

* It is a funcn of control unit to interpret each instruction code & provide necessary control functions needed to process the instruction.

Format: It is collection of bits of instruction as they appear in memory words or in control register.

* Bits of instruction are devided into groups called – fields.

Common fields in instruction formats are –

(1) Operation code: Specifies operation to be performed

(2) Address field: designates memory address or processor register.

(3) Mode field: Specifies way the Operands or effective address is determined.

Register Address :–

- Operands residing in processor registers – Specified with register Address.
- Operands residing in memory – Specified with Memory Address.

* Register Address is a binary number of K bits that define one of 2^K registers in CPU.

* No. of address fields in the instruction format of a computer depends on internal organization of its registers.

* Three types of CPU organizations –

(1) Single accumulator organization
(2) General register Organization
(3) Stack organization.

(42)

Ex of single accumulator register:—

— Instruction format in this type of computer uses one address field.

Ex: ADD X X: Address of operand.

It means AC ← AC + M[X].

AC: Accumulator.

M[X]: Memory word located at address X.

Exs of general register type organization:—

1. ADD R1, R2, R3 It means R1 ← R2 + R3.
   — It has 3 Register Address field.

2. ADD R1, R2 It means R1 ← R1 + R2
   — It has 2 Register Address field.

3. MOV R1, R2 It means R1 ← R2 (or R2 ← R1) depending on particular computer.
   — It need 2 Address field.

4. ADD R1, X It means R1 ← R1 + M[X]
   — Two Address field one for register R1 & other for memory address X.

Exs of Stack organization —

1. PUSH X, This will push word at address X to the top of stack.

2. ADD , consist No address field., This pop two top numbers from stack, adding the numbers & push the sum into stack.

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

Types of instruction format :→ Ex: X=(A+B) * (C+D).

(1) Three - Address Instruction :-

It can use each address field to specify either processor register or memory operand.

Register Transfer operation -

ADD R1,A,B	, R1 ← M[A] + M[B]	R1,R2 : Processor Registers
ADD R2,C,D	, R2 ← M[C] + M[D]	M[A] : Operand at memory address A
MUL X,R1,R2	, M[X] ← R1 * R2 .	

Adv: It results in short programs when evaluating arithmetic expressions.

Dis Adv: Binary coded instructions require too many bits to specify three addresses.

(2) Two Address Instruction :-

Most common in commercial computers.

Program -

MOV	R1, A	, R1 ← M[A]
ADD	R1, B	, R1 ← R1 + M[B]
MOV	R2, C	, R2 ← M[C]
ADD	R2, D	, R2 ← R2 + M[D]
MUL	R1, R2	, R1 ← R1 * R2
MOV	X, R1	, M[X] ← R1

(3) One Address Instruction :-

- It uses an implied accumulator (AC).

Program -

LOAD	A	AC ← M[A]
ADD	B	AC ← AC + M[B]
STORE	T	M[T] ← AC
LOAD	C	AC ← M[C]
ADD	D	AC ← AC + M[D]
MUL	T	AC ← AC * M[T]
STORE	X	M[X] ← AC

T: Temporary Memory location for storing intermediate result.

* All operation are done b/w AC register & memory operand.

(4) Zero – Address Instructions :-

- Stack organized computer does not use an address field for instructions ADD & MUL.

* PUSH & POP instructions need an address field to specify Operand that communicate with stack.

PUSH	A	TOS ← A
PUSH	B	TOS ← B
ADD		TOS ← (A+B)
PUSH	C	TOS ← C
PUSH	D	TOS ← D
ADD		TOS ← (C+D)
MUL		TOS ← (C+D) * (A+B)
POP	X	M[X] ← TOS

TOS : Top of Stack.

- It is said "zero-address" bcoz. of the absence of an address field in computational instructions.

* RISC Instructions :- (Reduced Instruction Set Computer) –

- It is restricted to the use of load & store instructions, when Communicating b/w memory & CPU.
- All other instructions are executed within registers of CPU w/o referring to memory.
- Program for RISC type CPU consist LOAD & STORE instructions that have one memory & one register address
- Computational type instructions have three addresses with specifying processor registers.

LOAD	R1, A	R1 ← M[A]
LOAD	R2, B	R2 ← M[B]
LOAD	R3, C	R3 ← M[C]
LOAD	R4, D	R4 ← M[D]
ADD	R1, R1, R2	R1 ← R1+R2
ADD	R3, R3, R4	R3 ← R3+R4
MUL	R1, R1, R3	R1 ← R1 * R3
STORE	X, R1	M[X] ← R1

} – Computational type instructions.

INSTRUCTION CYCLE

* Program is sequence of instruction.
* Program is executed in the computer by going through a cycle for each instruction.
  - Each instruction cycle is subdivided into a sequence of subcycles or phases.
  - Instruction cycle consists of following phases -
    (1) Fetch an instruction from memory
    (2) Decode the instruction.
    (3) Read the effective address from memory if instruction has an indirect address
    (4) Execute the instruction.
* After completion of step 4, control goes back to step 1 to fetch, decode, & execute next instruction. This process continues indefinitely unless a HALT instruction is encountered.

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

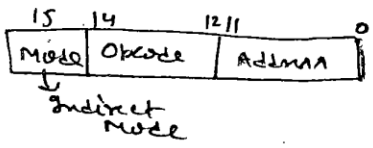
Fetch & Decode :-

Initially, PC is loaded with address of first instruction in the program.

SC: cleared to zero, providing a decoded timing signal T_0.

- After each clock pulse, SC is incremented by one, so that timing signals go through a sequence T_0, T_1, T_2 & so on.

Microoperations for fetch & Decode:



T_0: $AR \leftarrow PC$

T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$

T_2: $D_0, \ldots D_7 \leftarrow$ Decode IR(12-14), $AR \leftarrow IR(0-11)$, $I \leftarrow IR(15)$

At time T_0: Address from PC is transferred to AR.

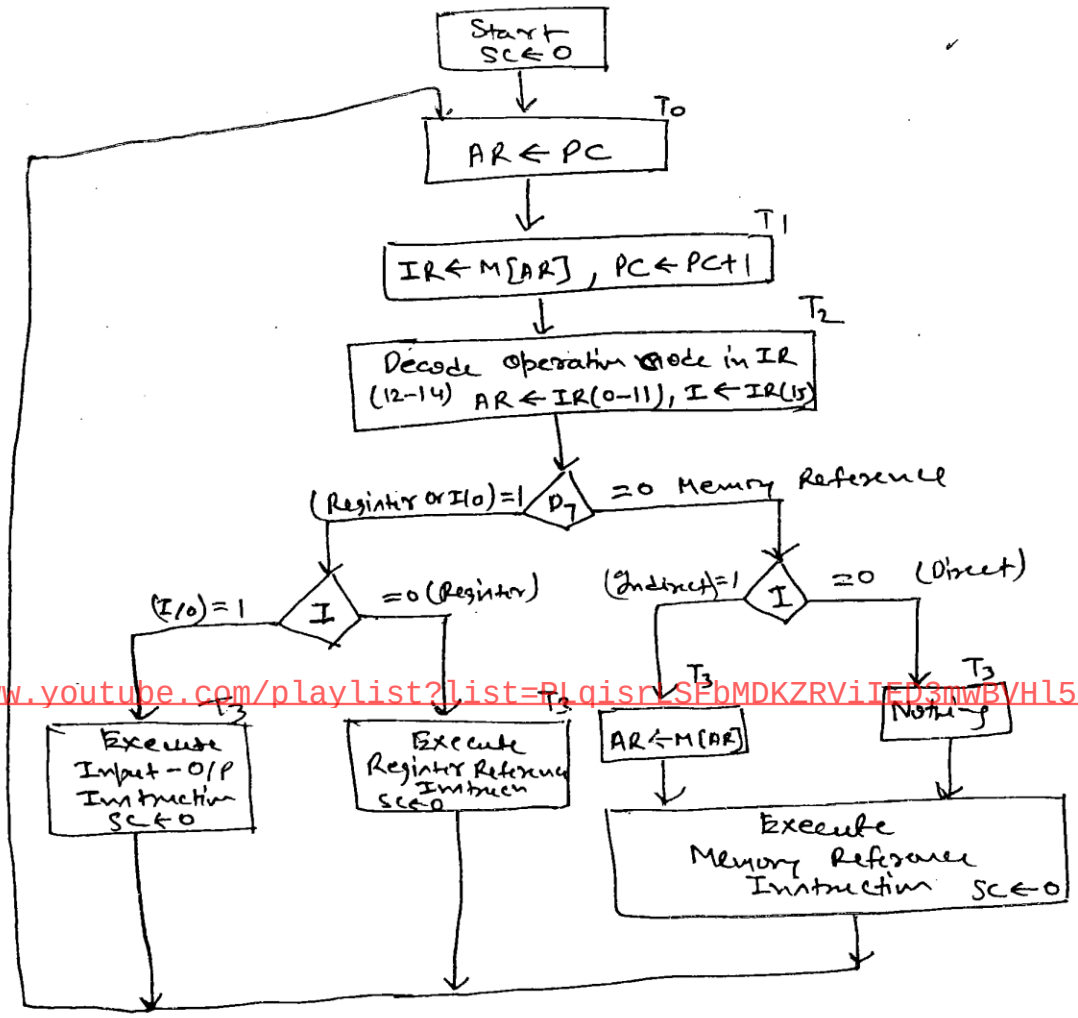
At time T_1: Instruction read from memory is then placed in IR, with clock transition associated with timing signal T_1. & PC is incremented by one for having address of next instruction in program.

At time T_2: - Operation code in IR is decoded.
- Indirect bit is transferred to flip flop I.
- Address part of instruction is transferred to AR.

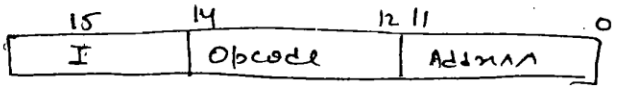
https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

* SC is incremented after each clock pulse to produce sequence T_0, T_1 & T_2.

Determine type of Instruction:- During cycle T_3, CU determine instructi

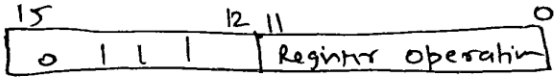


Flowchart for instruction cycle



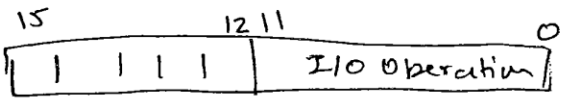
(Opcode = 000 through 110)

Memory Reference instruction



(Opcode = 111, I = 0)

Register Reference instruction



(Opcode = 111, I = 1)

I/O Instruction.

* Decoder O/P D_7 is equal to 1 if operation code is 111.
* If $D_7 = 1$, Instruction is either register reference or I/O Type.
* If $D_7 = 0$, Operation code must be one of the other seven values 000 through 110, specifying 'Memory reference instruction'.
* Control unit inspects the value of first bit of instruction, which is available in flip flop I.
* If $D_7 = 0$ & $I = 1$, we have memory reference instruction with an indirect address.
* It is necessary then to read effective address from memory.
* Microoperation for indirect address condition can be written as
  $AR \leftarrow M[AR]$

* These instruction types are subdivided into 4 separate parts -
* Selected operation is activated with clock transition associated with timing signal T_3, as -

  $D_7' I T_3$: $AR \leftarrow M[AR]$

  $D_7' I' T_3$: Nothing (B'cos effective address is already in AR)

  $D_7 I' T_3$: Execute a register reference instruction.

  $D_7 I T_3$: Execute an Input Output Instruction.

* After each clock cycle SC is incremented by 1 & after execution of instruction SC becomes 0.

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

Register Reference Instructions:

* It is recognized by control unit, when $D_7 = 1$ & $I = 0$.

* Bits from 0 to 11 i.e. IR(0-11), transferred to AR during time T_2.

* Control functions/Microoperations for register reference instructions are listed as -

$D_7 I' T_3 = r$ (Common to all register reference instructions)

$IR(i) = B_i$ [bit in IR(0-11) that specifies operations]

	r: SC ← 0	Clear SC
CLA	rB_{11}: AC ← 0	Clear AC
CLE	rB_{10}: E ← 0	Clear E
CMA	rB_9: AC ← $\overline{AC}$	Complement AC
CME	rB_8: E ← $\overline{E}$	Complement E
CIR	rB_7: AC ← shr AC, AC(15) ← E, E ← AC(0)	Circulate right
CIL	rB_6: AC ← shl AC, AC(0) ← E, E ← AC(15)	Circulate left
INC	rB_5: AC ← AC + 1	Increment AC
SPA	rB_4: if (AC(15) = 0) then (PC ← PC + 1)	Skip if positive
SNA	rB_3: if (AC(15) = 1) then (PC ← PC + 1)	Skip if Negative
SZA	rB_2: if (AC = 0) then (PC ← PC + 1)	Skip if AC zero
SZE	rB_1: if (E = 0) then (PC ← PC + 1)	Skip if E zero
HLT	rB_0: S ← 0 (S is a start-stop flip flop)	Halt computer

Execution of register-Reference instruction.

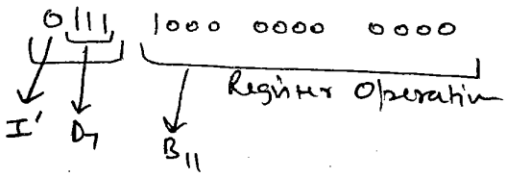
* These instructions are executed with clock transition associated with timing variable T_3.

* Control func^n for these instructions - $D_7 I' T_3$.

(In table 5.2)

Ex. CLA: Hexadecimal Code = 7800

Binary conversion



Control funcn: $D_7 I' T_3 B_{11} = r B_{11}$

* Skip Instruction: Skipping of instruction is achieved by incrementing PC once again.

* AC is +ve, when sign bit (AC(15)) = 0.
It is -ve, when sign bit (AC(15)) = 1.
AC is zero → (when all flip flops of register are zero).

* HLT: It clears start-stop flip flop S & stops the sequence counter from counting.

* To restore operation of computer, start-stop flip flop must be set.

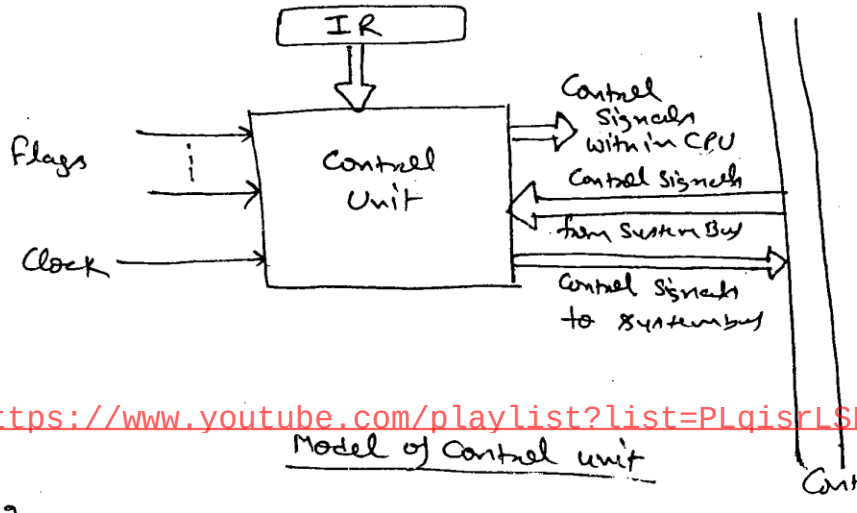
PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

HARDWIRED & MICROPROGRAMMED CONTROL

* Techniques for implementation of control unit are divided into two categories –
  (1) Hardwired Implementation
  (2) Microprogrammed Implementation.

Hardwired Implementation: It consist combinational circuit.



Model of Control unit

* In case of flags & control bus signals, each individual bit typically has same meaning.

Instruction Register: Control unit makes use of opcode & will perform different actions. (issue a different combination of control signals) for different instructions.

* To simplify control unit logic, there should be a unique logic input for each opcode.
* For this we will use decoder, it takes an encoded input & produces a single O/P.
* Each of 2^N different input pattern will activate a single O/P

Decoder with four input & Sixteen O/p –

I_1	I_2	I_3	I_4	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_{10}	O_{11}	O_{12}	O_{13}	O_{14}	O_{15}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

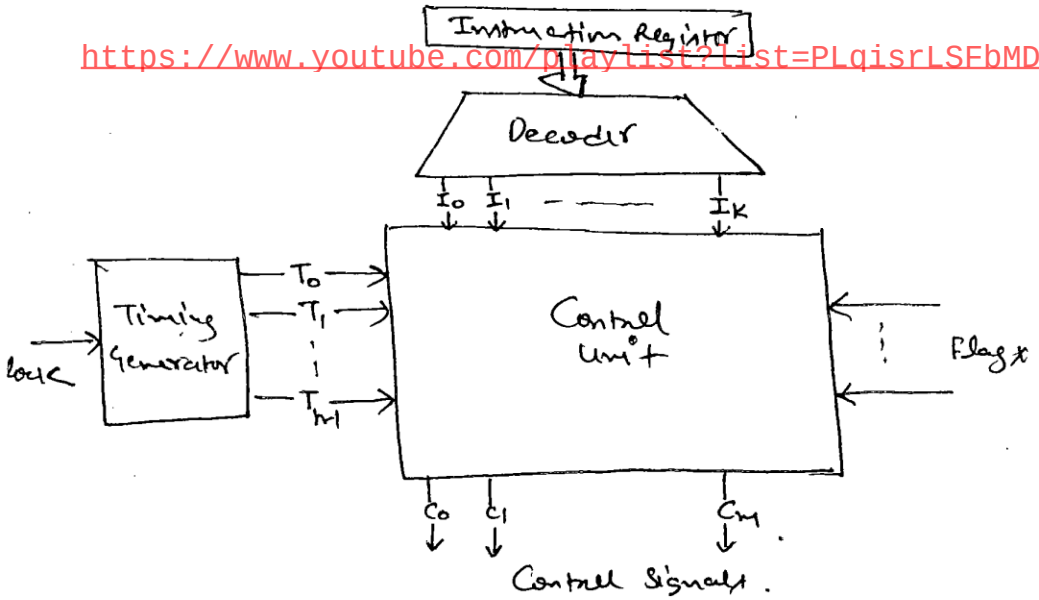
Clock: It issue repetitive sequences of pulses.
- Useful for measuring duration of micro operations.
- Period of clock pulses must be long enough to allow the propagation of signals along data paths & through CPU circuitry.

* Control unit emits different control signals at different time units with in a single instruction cycle.

So we will use a TC as input to control unit, with different control signal being used for $T_0, T_1, \ldots$ & so on.

- At the end of instructions cycle TC is re-initialized with T_0.

Control unit with decoded input -

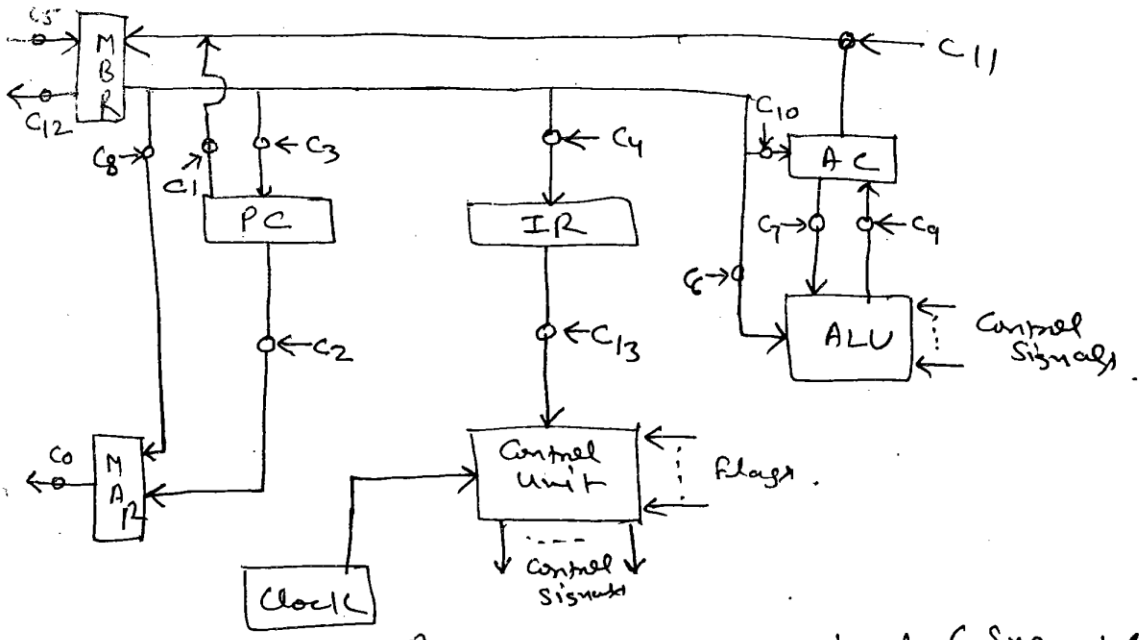


Control Unit logic -

* H/w implementation means to discuss internal logic of control unit, that produces O/P control signals as a function of its input signals.

* for each control signal, to derive a boolean expression of that signal as function of inputs.

Control Signal Example :-



Data path & control signals (Example)

Micro-operations & control signals :-

Micro operation		Active control Signals.
Fetch:	t_1: MAR ← PC	C_2
	t_2: MBR ← Memory PC ← (PC)+1	C_5, C_R
	t_3: IR ← (MBR)	C_4
Indirect:	t_1: MAR ← (IR(Address))	C_8
	t_2: MBR ← Memory	C_5, C_R
	t_3: IR(Address) ← (MBR(Address))	C_4
Interrupt:	t_1: MBR ← (PC)	C_1
	t_2: MAR ← Save-address PC ← Routine-Address	
	t_3: Memory ← (MBR)	C_{12}, C_W.

C_R: Read control signal to system bus

C_W: Write control signal to system bus.

- Consider control signal C_5 → This signal causes data to be read from external data bus into MBR.

Let us define two new control signals, P & Q, that have following interpretation-

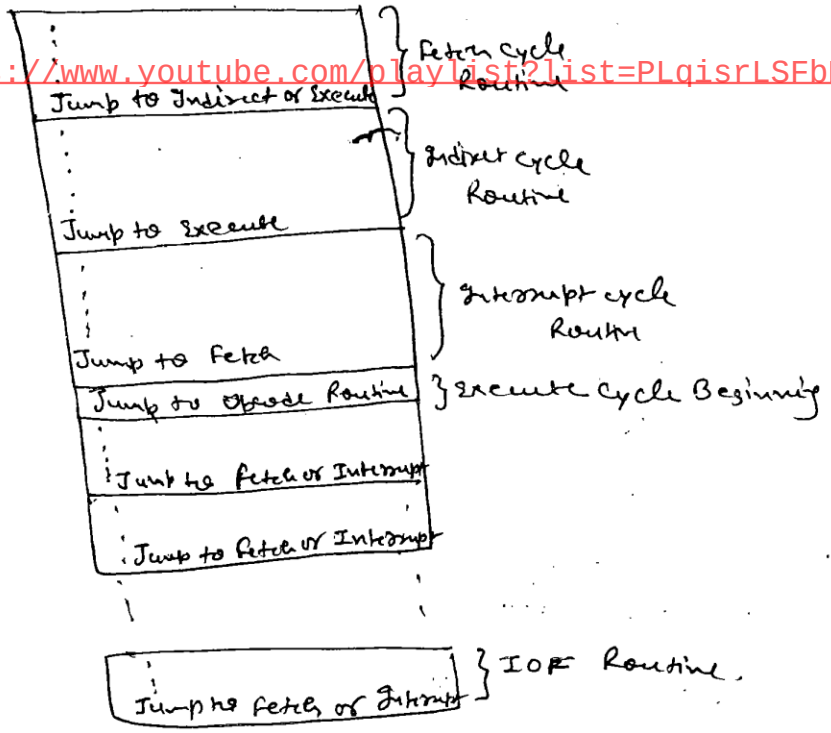
P	Q	
0	0	Fetch cycle
0	1	Indirect Cycle
1	0	Execute Cycle
1	1	Interrupt cycle

- Boolean expression for C_5:

$$C_5 = \bar{P}.\bar{Q}.T_2 + \bar{P}.Q.T_2$$

* Same process is repeated for every control signal generated by CPU. Result would be a set of boolean equations that define behavior of Control unit & hence of CPU.

Microprogrammed Control Unit

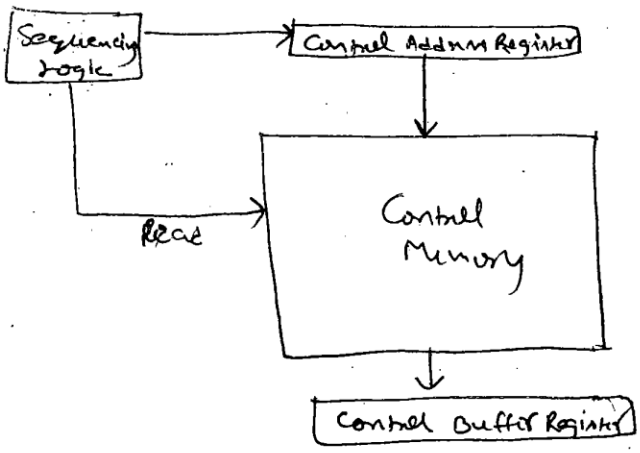


- fetch
- Indirect
- Execute
- Interrupt.

Organization of control memory.

(Behavior of control unit)

- Microinstruction in each routine are to be executed sequentially.
- Each routine ends with a branch or jump instruction indicating where to go next.



Control unit Micro Architecture.

Control Memory: It stores "set of microinstructions".

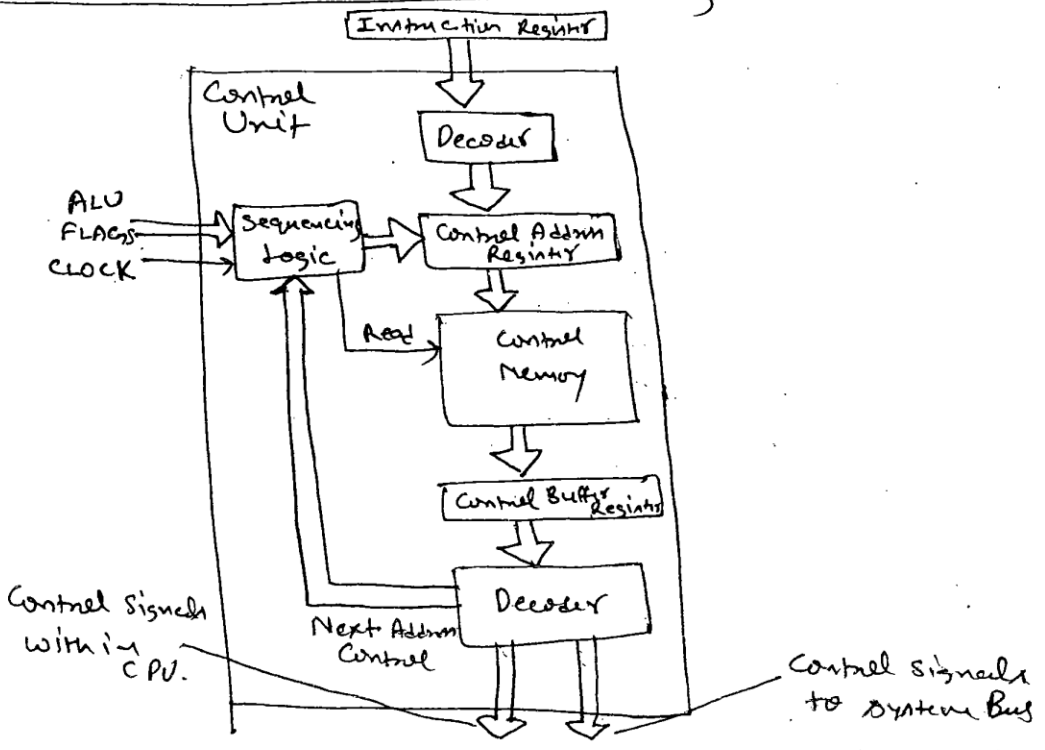
Control Address Register: It contains address of next micro instruction.

Control Buffer Register: when microinstruction is read from control memory, it is transferred to control buffer register.

* Reading a microinstruction from control memory is known as Executing that microinstruction.

Sequencing logic: It loads the control address register & issues a read command.

functioning of microprogrammed control unit :-



* Control Unit has – Inputs – IR, ALU Flags, clock
Outputs – Control Signals.

Control unit functions –

(1) To execute an instruction, Sequencing logic unit issues a READ command to the control memory.
(2) The word whose address is specified in control address register is read into control buffer register.
(3) Content of control buffer register generates control signals & next address information for sequencing logic unit.
(4) Sequencing logic unit loads a new address into control Address register based on next address information from control buffer register & ALU flags.

↳ During one clock pulse

* After completion of each microinstruction, sequencing logic unit loads a new address into control address register.

* Depending on the value of ALU flags & control buffer register, one of three decisions is made –

(1) Get the next instruction : – Add 1 to control Address register.
(2) Jump to new routine based on jump microinstruction – load the address field of control buffer register into control Address register.
(3) Jump to M/c instruction routine – load the control address register based on opcode in IR.

Mapper Decoder – translate opcode of IR to control memory address

Tower Decoder – It is used for vertical microinstructions., It translate code into individual control signals.

MICROINSTRUCTION SEQUENCING

Basic Tasks:- performed by microprogrammed control unit -

(1) Microinstruction Sequencing - Get next microinstruction from control memory.

(2) Microinstruction Execution - Generate control signals needed to execute the microinstructions.

Designing Considerations of microinstruction sequencing technique :-

- Size of microinstruction: Minimizing size of control memory, reduces cost of that component.
- Address Generation time: To reduce time for execution of microinstructions.

* During execution of program, Address of next microinstruction to be executed in one of categories -

(1) Determined by Instruction Register → Occurs only once per instruction cycle (After fetch)

(2) Next Sequential Address

(3) Branch → Necessary part of program

Sequencing Techniques :-

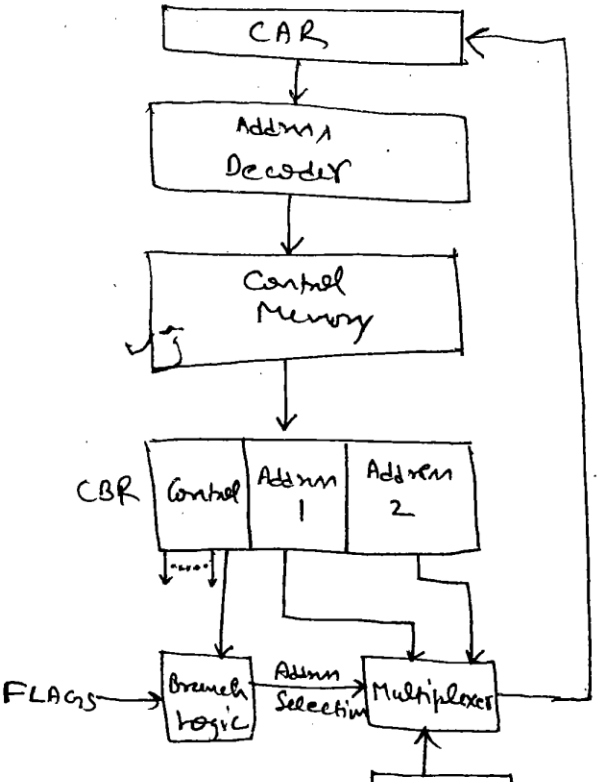
* For next instruction, control memory address is generated, by using -
  - Current microinstruction
  - Condition flags
  - Contents of instruction register

General categories of techniques :-

- Categories are based on format of address info. in microinstruction -

(1) Two Address field
(2) Single Address field
(3) Variable format

(1) Two Address field :-



Branch Control logic, Two Address field.

Multiplexer: It serves as a destination for both address fields plus instruction register.

* Based on address selection input, multiplexer transmits either the Opcode or one of the two addresses to Control Address register (CAR).
- CAR is subsequently decoded to produce next microinstruction address.

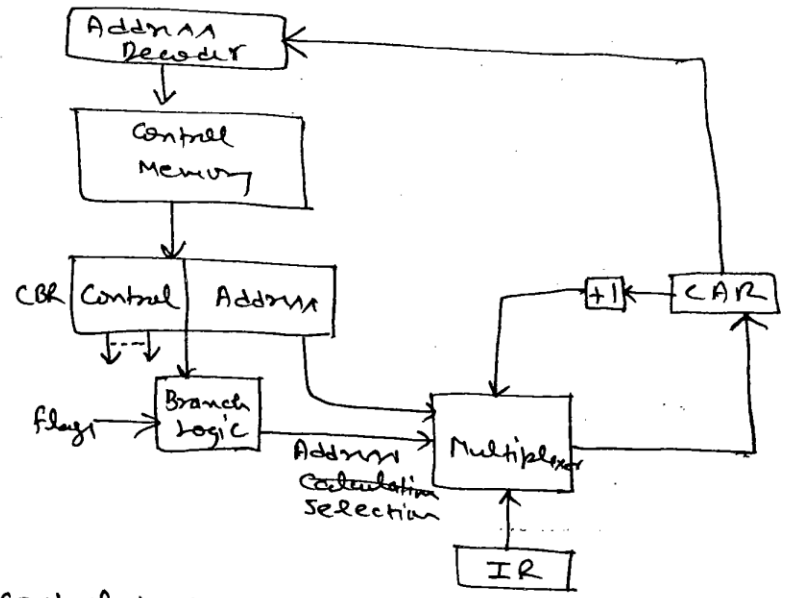
Branch logic: It provide address selection signals. Its input consists of Control unit flags plus bits from Control portion of microinstruction.

Disadv: It requires more bits in microinstruction than other approaches.

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

② Single Address field :-



Branch control logic, Single Address field.

* Options for next address are –

(1) Address field

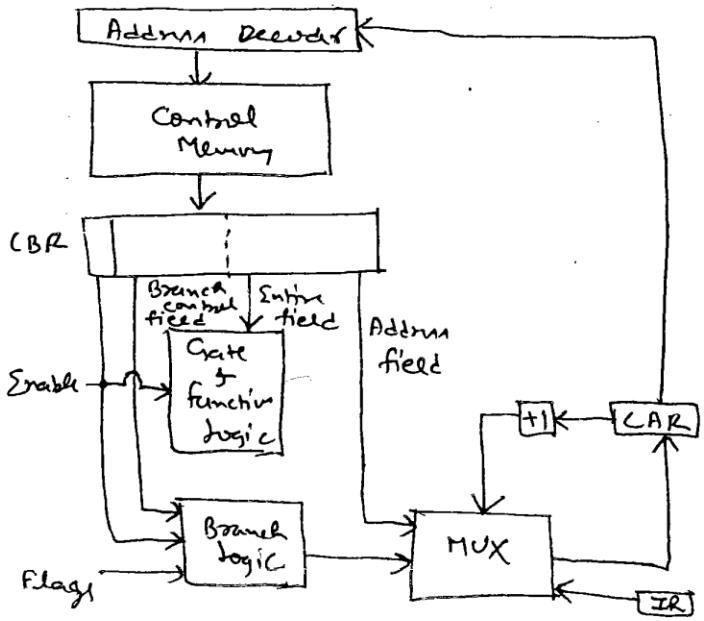
(2) Instruction register Code

(3) Next Sequential Address.

* Address selection signals determine which option is selected.

* Address field will often not be used. There is some inefficiency in microinstruction coding scheme.

(3) Variable format :-



* One bit designates which format is being used.

- In one format: Remaining bits are used to activate control signals.
- In other format: Some bits drive branch logic module & remaining bits provide address.
  - Either a conditional or unconditional branch is specified.
  - next address is either next sequential address or an address derived from instruction register.

Dis adv: One entire cycle is consumed with each branch microinstruction.

Address Generation:

* Techniques for address generation can be divided into two parts –
  (1) Explicit: Address is explicitly available in microinstruction.
  (2) Implicit: It requires additional logic to generate the address.

Explicit – (1) Two field
(2) Unconditional Branch
(3) Conditional Branch.

Implicit – (1) Mapping
(2) Addition
(3) Residual control

Two field Approach:

- Two alternative addresses are available with each microinstruction.
- Using either a single or two address field or variable format, various branch instruction can be implemented.

Conditional branch:

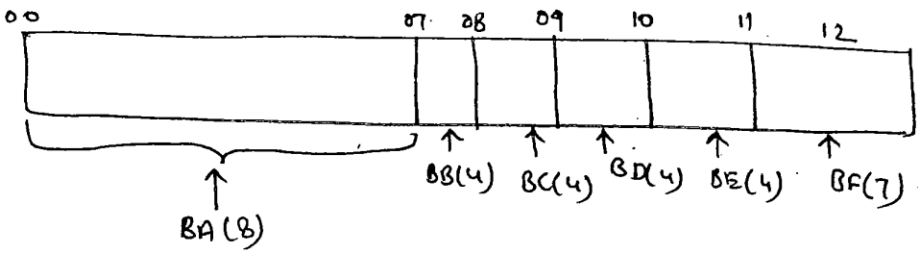
- Conditional branch instruction depends on following types of information –
  (1) ALU flags
  (2) Part of opcode or address mode fields of M/C instructions.
  (3) Parts of selected register such as sign bit
  (4) Status bit with in control unit.

Mapping:

- Opcode portion of M/C instruction must be mapped into a microinstruction address. (Occurs only once per instruction cycle).

Adding: - To add two portions of an address to form complete address.

Ex: IBM 3033 Control Address Register – 13 Bit long



BA(8): These 8 bits do not change from one microinstruction cycle to next.

- During execution, these 8 bits are copied directly from 8 bit field of microinstruction into Rightest order 8 bits of control Address register.

- 9+ define 32 microinstructions in control memory.

Remaining 5 bits: These are set to specify the specific address of microinstruction to be fetched next.

- Each of these bits are determined by 4 bit field (Except one is 7 bit field) in current microinstruction.

Ex: A bit in control address register might be set to 1 or 0 depending on whether a carry occurred on last ALU operation.

Residual Control: It involves use of microinstruction address that has previously been saved in temporary storage within control unit.

Techniques for generation of branch address:-

(1) Bit-ORing: In this branch address is determined by ORing particular bit or bits with current address of microinstruction

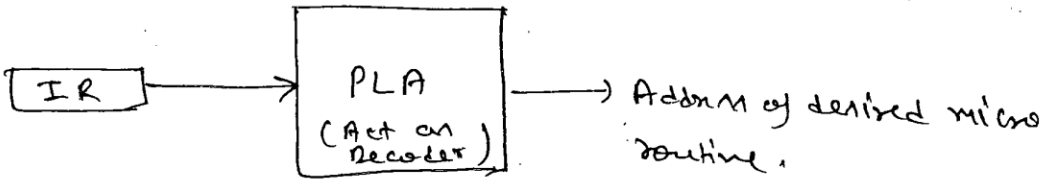
Ex: If current address is 170 & branch address is 172 then branch address can be generated by ORing 02 (Bit 1) with current address.

(2) Using condition variables: In this condition variables are used to modify the contents of Control Address Register directly, thus eliminate whole or in part the need for branch addresses in microinstructions.

Ex: SKIP_ON_CARRY instruction.

(3) Wide Branch Addressing:

- If no. of branches increases, it becomes more difficult to generate branch address.
- In this, PLA (Programmable logic Array) can be used to generate required branch addresses.
- This way of generating branch address, called wide branch addressing.



- Opcode of M/C instruction is translated into starting address of corresponding micro routine.
- Opcode bits of IR connects as input to PLA. O/P of PLA is address of desired micro routine.

Microinstructions with next Address field

Branch instructions :-

- It wastes one cycle.
- Many microinstructions having common functions to implement. So to reduce no. of cycles consumed by branch instruction, another strategy is to place next address field with each microinstruction.

Fields of Minstruction with next address field :-

- First time when microprogram execution starts, next address is as per Instruction Decoder O/P, which reflects Instruction Register, condition flags & external inputs during an instruction.
- Next address field directs it to next microinstruction.

⇒ It includes an address field as a part of every microinstruction to indicate location of next microinstruction to be fetched

Adv: - Separate branch & instructions are virtually eliminated.
- Few limitations in assigning addresses to micro instructions.

Disadv: Additional Bits for address field. (Around 16).

Microinstruction: An instruction that control data flow & instruction execution sequencing in a processor at a more fundamental level than M/C instruction.

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

Micro-Instruction format

Microinstruction: It describe set of micro operations occurring at one time & is known as micro instructions.

Microprogram: Sequence of instructions known as microprogram or firmware.

- With the help of each micro operation, control unit is allowed to do is to generate set of control signals.

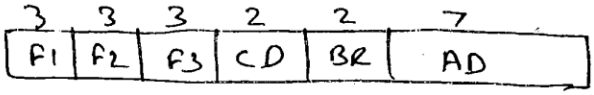
Control word: Set of bits, in which each bit represents one control line.

- Each micro operation is represented by different patterns of 1's & 0's in control word.
- Every micro instruction have control word & stored in control memory.
- Each control word have unique address.

It can be of two types -

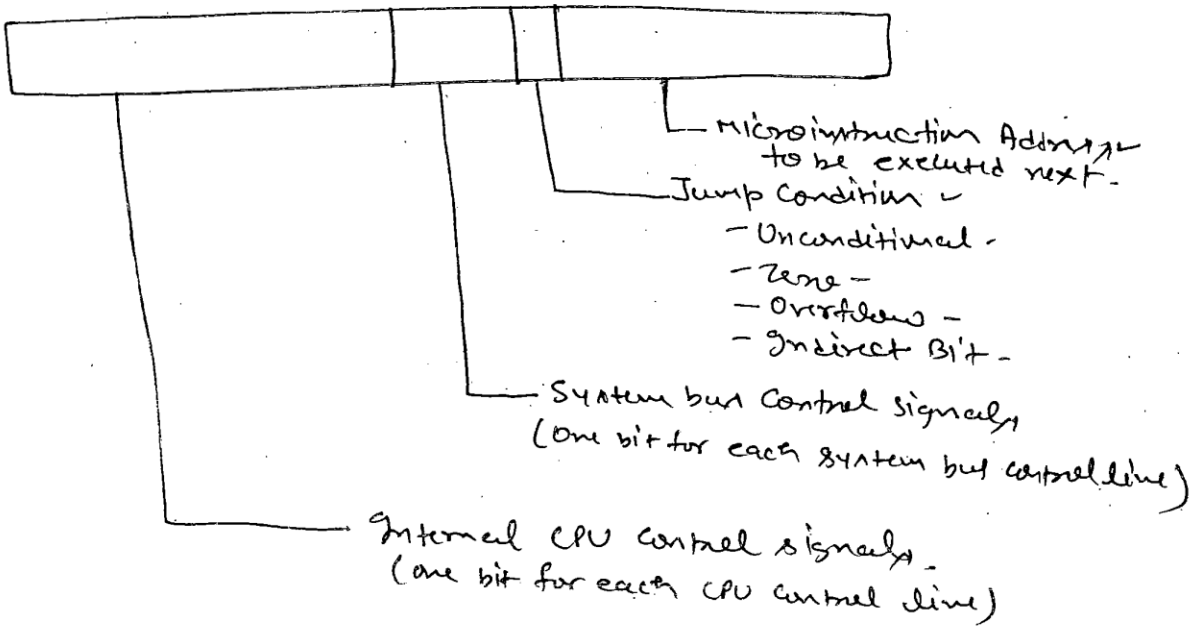
(1) Horizontal micro instruction.

(2) Vertical micro instruction.



F1, F2, F3 : microoperation field
CD: condition for Branching
BR: Branch field
AD: Address field.

(1) Horizontal Micro Instruction :→



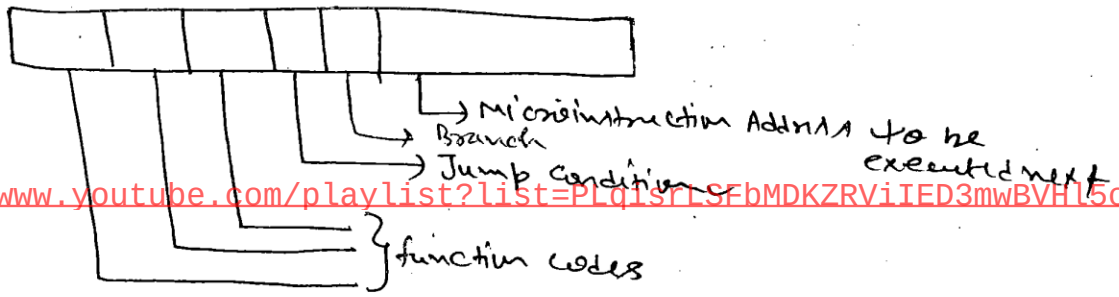
Such microinstruction is interpreted as –

(1) To execute this microinstruction, turn on all the control lines indicated by a 1 bit; leave off all control lines indicated by a 0 bit.

(2) If condition indicated by condition bit is false, execute the next microinstruction in sequence.

(3) If condition indicated by condition bit is true, next microinstruction to be executed is indicated in address field.

(2) Vertical Microinstruction :-



Description of microinstruction format :-

F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLR AC
011	AC ← AC + 1	INC AC
100	AC ← DR	DR to AC (DRTAC)
101	AR ← DR(0-10)	DR to AR (DRTAR)
110	AR ← PC	PC to AR (PCTAR)
111	M[AR] ← DR	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC − DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC′	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional Branch
01	DR(15)	I	Indirect Address bit
10	AC(15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

BR	Symbol	function
00	JMP	CAR ← AD if condition = 1 CAR ← CAR+1 if condition is zero.
01	CALL	CAR ← AD, SBR ← CAR+1 if condition = 1 CAR ← CAR+1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

Ex: Sequence of 4 operations in fetch cycle –

AR ← PC

DR ← M[AR], PC ← PC+1

AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

Symbolic program for this –

```
FETCH:  PCTAR          U   JMP   NEXT
        READ, INCPC    U   JMP   NEXT
        DRTAR          U   MAP
```

Binary Equivalent translator by assembler –

Binary Address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

PREFETCHING MICROINSTRUCTIONS

* It is a technique used in microprocessors to speed up the execution of a program by reducing wait states.

* Why do we require prefetching?

Modern microprocessors are much faster than memory, when program is kept. It means, program's instructions can not be read fast enough to keep microprocessor busy. Adding a cache can provide faster access to needed instructions.

* It occurs when a processor requests an instruction from memory before it is actually needed. Once the instruction come back from memory, it is placed in cache. When an instruction is actually needed, instruction can be accessed more quickly from the cache.

* Microprogrammed control leads to slower operating speed, because of time it taken to fetch microinstructions from Control memory.

* To achieve faster operation, next microinstruction can be prefetched while the current one is being executed.

– Execution time can be overlapped with the fetch time.

Difficulties:–

(1) Status flag & results of current μinstruction that is being executed are necessary to determine address of next micro-instruction.

(2) Straight forward prefetching may fetch a wrong instruction.

(3) Fetch must be repeated.

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

EMULATION:-

- A programmed control offers flexibility to add new instructions to instruction set of processor.
- Add to instruction set of a given computer M_1 an entirely new set of instructions that is in fact the instruction set of a different computer M_2.
- Programs written in M/c language of M_2 can be run on M_1, then we say M_1 emulates M_2.
- Emulation allows transition to new computer systems with minimal disruption.

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

Pipelining

To improve the performance of a CPU we have two options:

1) Improve the hardware by introducing faster circuits.
2) Arrange the hardware such that more than one operation can be performed at the same time.

Since, there is a limit on the speed of hardware and the cost of faster circuits is quite high, we have to adopt the 2nd option.

Pipelining: Pipelining is a process of arrangement of hardware elements of the CPU such that its overall performance is increased.

Simultaneous execution of more than one instruction takes place in a pipelined processor.

Let us see a real-life example that works on the concept of pipelined operation.

Consider a water bottle packaging plant. Let there be 3 stages that a bottle should pass through, Inserting the bottle(**I**), Filling water in the bottle(**F**), and Sealing the bottle(**S**).

Let us consider these stages as stage 1, stage 2 and stage 3 respectively. Let each stage take 1 minute to complete its operation.

Now, in a non-pipelined operation, a bottle is first inserted in the plant, after 1 minute it is moved to stage 2 where water is filled.

Now, in stage 1 nothing is happening. Similarly, when the bottle moves to stage 3, both stage 1 and stage 2 are idle.

But in pipelined operation, when the bottle is in stage 2, another bottle can be loaded at stage 1. Similarly, when the bottle is in stage 3, there can be one bottle each in stage 1 and stage 2. So, after each minute, we get a new bottle at the end of stage 3.

Hence, the average time taken to manufacture 1 bottle is:

Without pipelining = 9/3 minutes = 3m

```
I F S | | | | |
| | | I F S | | |
| | | | | | I F S (9 minutes)
```

With pipelining = 5/3 minutes = 1.67m

```
I F S | |
| I F S |
| | I F S (5 minutes)
```

Thus, pipelined operation increases the efficiency of a system.

Mr. Zatin Gupta (Assistant Professor)
Department of Computer Science,
KIET Group of Institutions, Ghaziabad, UP
Zatin.gupta@kiet.edu, Zatin.gupta2000@gmail.com

Design of a basic pipeline

In a pipelined processor, a pipeline has two ends, the input end and the output end. Between these ends, there are multiple stages/segments such that output of one stage is connected to input of next stage and each stage performs a specific operation.

Interface registers are used to hold the intermediate output between two stages. These interface registers are also called latch or buffer.

All the stages in the pipeline along with the interface registers are controlled by a common clock.

Execution in a pipelined processor

Execution sequence of instructions in a pipelined processor can be visualized using a space-time diagram.

For example, consider a processor having 4 stages and let there be 2 instructions to be executed.

We can visualize the execution sequence through the following space-time diagrams:

Non overlapped execution: (NON-Pipelined)

STAGE / CYCLE	1	2	3	4	5	6	7	8
S1	I_1				I_2			
S2		I_1				I_2		
S3			I_1				I_2	
S4				I_1				I_2

Total time = 8 Cycle

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

Mr. Zatin Gupta (Assistant Professor)
Department of Computer Science,
KIET Group of Institutions, Ghaziabad, UP
Zatin.gupta@kiet.edu, Zatin.gupta2000@gmail.com

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

Overlapped execution: (Pipelined)

STAGE / CYCLE	1	2	3	4	5
S1	I_1	I_2			
S2		I_1	I_2		
S3			I_1	I_2	
S4				I_1	I_2

Total time = 5 Cycle

Pipeline Stages: RISC processor has 5 stage instruction pipelines to execute all the instructions in the RISC instruction set. Following are the 5 stages of RISC pipeline with their respective operations:

Stage 1 (Instruction Fetch): In this stage the CPU reads instructions from the address in the memory whose value is present in the program counter.

Stage 2 (Instruction Decode): In this stage, instruction is decoded and the register file is accessed to get the values from the registers used in the instruction.

Stage 3 (Instruction Execute): In this stage, ALU operations are performed.

Stage 4 (Memory Access): In this stage, memory operands are read and written from/to the memory that is present in the instruction.

Stage 5 (Write Back): In this stage, computed/fetched value is written back to the register present in the instructions.

Instruction Pipeline: In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle.

This type of technique is used to increase the throughput of the computer system.

An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline.

Mr. Zatin Gupta (Assistant Professor)
Department of Computer Science,
KIET Group of Institutions, Ghaziabad, UP
Zatin.gupta@kiet.edu, Zatin.gupta2000@gmail.com

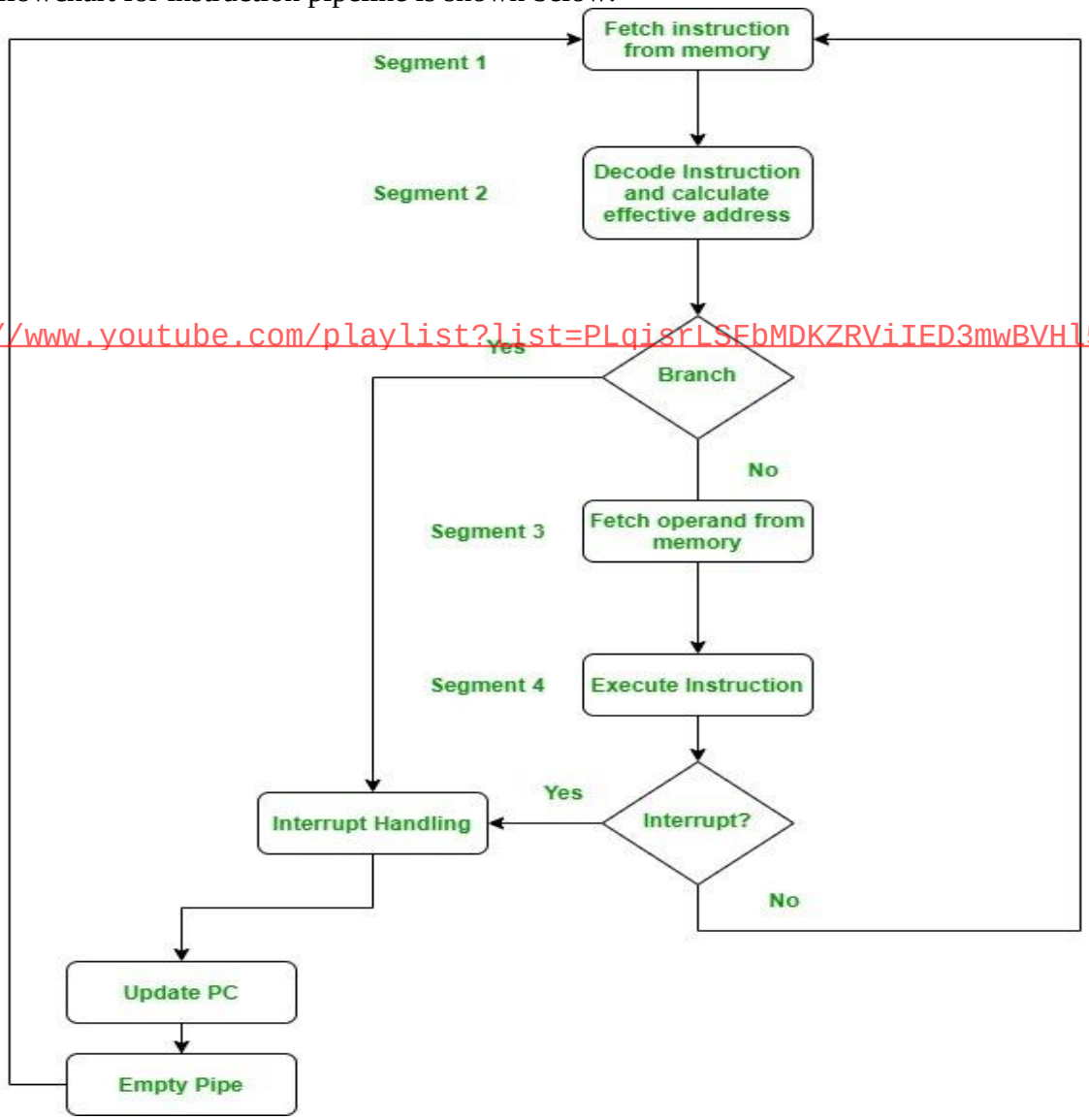
Thus, we can execute multiple instructions simultaneously.

The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

In the most general case computer needs to process each instruction in following sequence of steps:

1. Fetch the instruction from memory (FI)
2. Decode the instruction (DA)
3. Calculate the effective address
4. Fetch the operands from memory (FO)
5. Execute the instruction (EX)
6. Store the result in the proper place

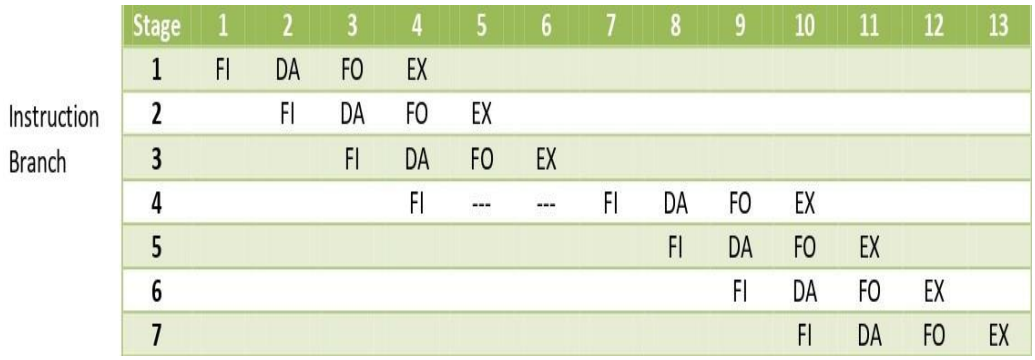
The flowchart for instruction pipeline is shown below.



Mr. Zatin Gupta (Assistant Professor)
Department of Computer Science,
KIET Group of Institutions, Ghaziabad, UP
Zatin.gupta@kiet.edu, Zatin.gupta2000@gmail.com

https://www.youtube.com/playlist?list=PLqisrLSFbMDKZRViIED3mwBVHl5c4LG-L

Let us see an example of instruction pipeline.



Here the instruction is fetched on first clock cycle in segment 1.

Now it is decoded in next clock cycle, then operands are fetched and finally the instruction is executed.

We can see that here the fetch and decode phase overlap due to pipelining.

By the time the first instruction is being decoded, next instruction is fetched by the pipeline. In case of third instruction we see that it is a branched instruction.

Here when it is being decoded 4th instruction is fetched simultaneously.

But as it is a branched instruction it may point to some other instruction when it is decoded.

Thus, fourth instruction is kept on hold until the branched instruction is executed.

When it gets executed then the fourth instruction is copied back and the other phases continue as usual.

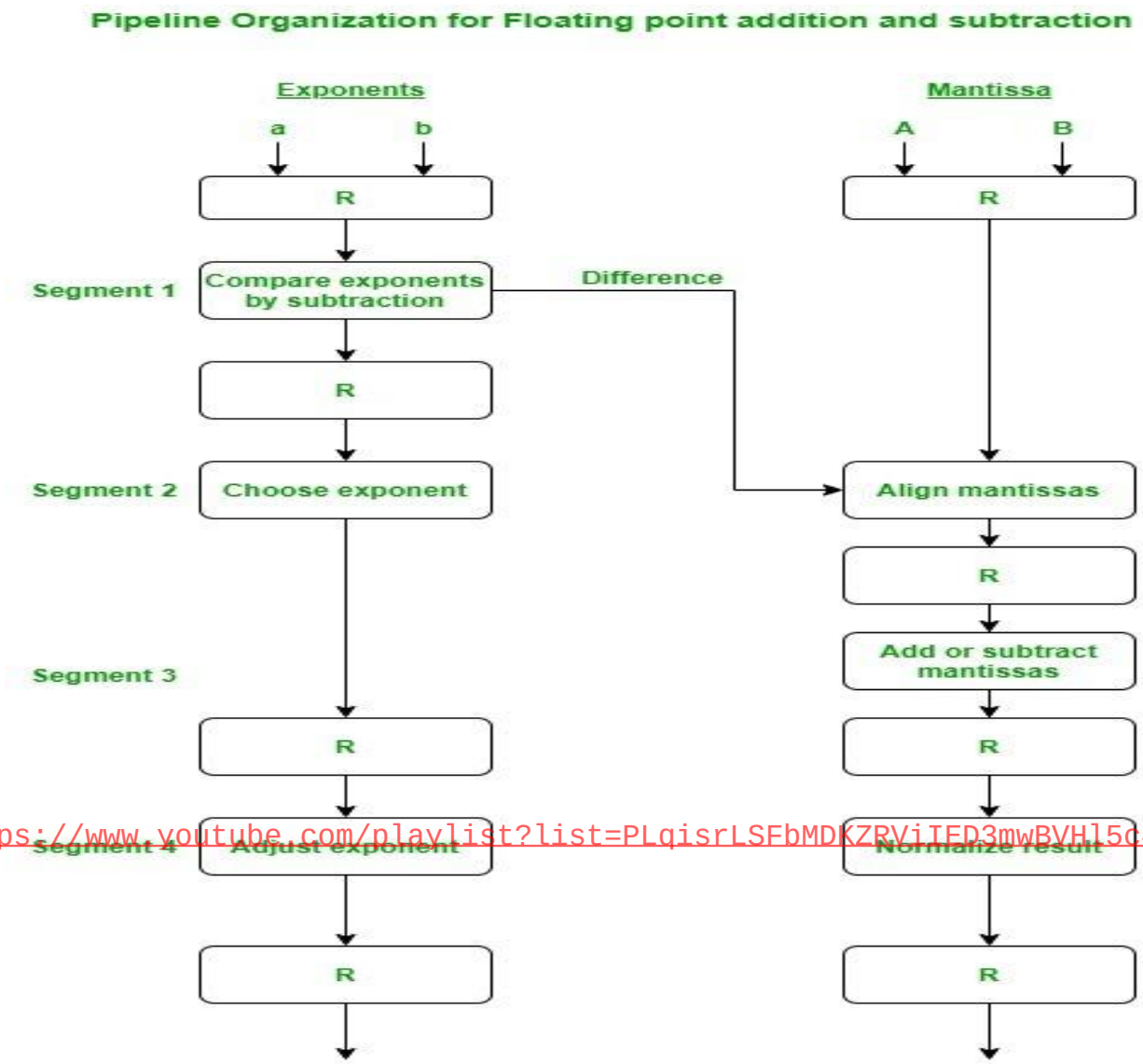
Arithmetic Pipeline: An arithmetic pipeline divides an arithmetic problem into various sub problems for execution in various pipeline segments.

It is used for floating point operations, multiplication and various other computations.

The process or flowchart arithmetic pipeline for floating point addition is shown in the diagram.

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

Mr. Zatin Gupta (Assistant Professor)
Department of Computer Science,
KIET Group of Institutions, Ghaziabad, UP
Zatin.gupta@kiet.edu, Zatin.gupta2000@gmail.com



Floating point addition using arithmetic pipeline:

The following sub operations are performed in this case:

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalise the result

First of all, the two exponents are compared and the larger of two exponents is chosen as the result exponent.

The difference in the exponents then decides how many times we must shift the smaller exponent to the right. Then after shifting of exponent, both the mantissas get aligned.

Mr. Zatin Gupta (Assistant Professor)
Department of Computer Science,
KIET Group of Institutions, Ghaziabad, UP
Zatin.gupta@kiet.edu, Zatin.gupta2000@gmail.com

Finally, the addition of both numbers take place followed by normalisation of the result in the last segment.

Example:
Let us consider two numbers,

X=0.3214*10^3 and Y=0.4500*10^2

Explanation:

First of all, the two exponents are subtracted to give 3-2=1.

Thus 3 becomes the exponent of result and the smaller exponent is shifted 1 time to the right to give

Y=0.0450*10^3

Finally, the two numbers are added to produce

Z=0.3664*10^3
As the result is already normalized the result remains the same.

PREPARED BY: MR. PIYUSH GUPTA, MR. ZATIN GUPTA

Mr. Zatin Gupta (Assistant Professor)
Department of Computer Science,
KIET Group of Institutions, Ghaziabad, UP
Zatin.gupta@kiet.edu, Zatin.gupta2000@gmail.com