UNIT- 3
CONTROL UNIT

Instruction format :-

What is Instruction ?

Program :- A sequence of instructions

```
main()
{
 . .
 . .
c=a+b
}
```

} It contain list of instructions.

Instruction :- An Instruction is a command given to the complete to Perform computer to perform a specific operation.

⇒ An Instruction is a group of bits (binary code) that instructions the computer to Perform a specific operation.

⇒ The Purpose of an Instruction is to specify both operation to be Performed by CPU and the set of operands (or data).

⇒ operation included add, sub, DIV, mul shift etc. operations are Performed on data (operands).

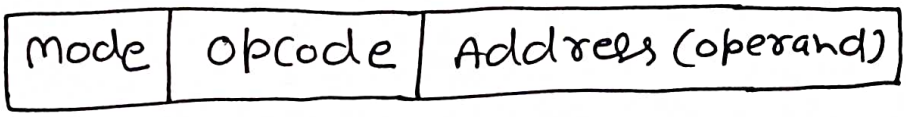
M·S

⇒ operands: included the input data and the results that are Produced.

What is Instruction format ?

⇒ An instruction format is a binary format which specifies a computer instruction.

⇒ The bits of the instructions are divided into groups called field.



(Instruction format (with mode field))

The most common field in instruction format are.

① opcode (operation code) field specifies the operation to be Performed such as add, subtract, multiply, Shift, comple ment etc.

② Address: An Address field specifies a memory address or Processor register where operand is Stored.

③ mode: A mode field specifies the way to operand or the effective address of the operand is determined (or located).

M.S

CPU organization and Types of Instructions:-

⇒ Computers may have instructions of several different length containing varying number of addresses (address field may be 3, 2, 1, or 0).

The number of address field in the instruction format of a computer depends on the internal organization of its registers.

⇒ Most computers fall into one of three types of CPU organization.

① Single Accumulator organization.
② General Register organization.
③ Stack organization.

Types of Instructions format / Types of address instructions:-

① Three - address Instruction format.
② Two - address Instruction format
③ one - address Instruction format
④ zero - address Instruction format.

M·S

Three and two Instruction format — general register-organization

one-address-instruction format — Single Accumulator organization.

zero-address-Instruction format — Stack Based organization

Classification of Instructions Based on CPU organization!-

① Single Accumulator organization!-

⇒ All operations are performed on an implied accumulator (AC) register.

⇒ With one operand implicitly in the accumulator register minimizes the internal complexity of a machine and allow for short instructions.

⇒ The instruction format uses only one address field (i.e one address instruction format).

② General Register organization!-

⇒ It uses of general purpose registers (R0, R1, R2 - - -) one of the most widely accepted models for machine architecture today.

M·S

⇒ Specifies all operands explicitly.

⇒ The instruction format needs 2 or 3 address fields according to the operation (i.e two or 3 - address format)

ex ADD R_1, R_2, R_3 // $R_1 \leftarrow R_1 + R_2$

this can be reduced from three or two address if the destination register is the same as one of the source registers.

ADD R_1 R_2 // $R_1 \leftarrow R_1 + R_2$

MOV R_1, R_2 // $R_1 \leftarrow R_2$

ADD R_1 X // $R_1 \leftarrow R_1 + M[X]$

Each Address field may specify a Processor register or a memory operand.

Stack organization :-

⇒ The operands are put into the stack and the operations are carried out on the top of the Stack (TOS). The operands are implicity specified on (TOS)

M·S

⇒ It does not use an address field for the instructions Like ADD, MUL, SUB DIV etc. (i.e zero - address instruction format)

⇒ It does not use an address fied for

⇒ PUSH and POP instructions are used to communicate with Stack which require an address field.

Note:- No Address field is required.

Ex

```
PUSH  A
PUSH  B
ADD
POP   C
```

Here ADD consists of any opcode and no address field. It has the effect of popping the top two numbers from the Stack thus all the operands are implied to be in the Stack.

Types of instruction format / Types of Address instructions:-

① Three - Address -Instruction format
② two - address- Instruction format
③ zero- address- Instruction format

M·S

Three-address Instructions:-

opcode	Destination Address	Source Address 1	Source Address-2

⇒ Three-Address instruction format has three address field.

⇒ one Address field is used for destination and two address fields for source.

⇒ Each Address field may specify a Processor register a memory operand.

⇒ It is also called general register organizations.

Ex ADD R_1, R_2, R_3 // $R_1 \leftarrow R_2 + R_3$

ADD R_1, A, B // $R_1 \leftarrow M[A] + M[B]$

Assembly languages notation (ALN)

Register transfer notation (RTN).

Two-address instructions:-

opcode	Destination Address	Source Address

⇒ two address instructions are most common in commercial computers.

M.S

⇒ It has two address field.

⇒ Each address field may specify a Processor register or a memory operand.

Ex

		D	S	
	ADD	R_1,	R_2	// $R_1 \leftarrow R_1 + R_2$
	MOV	R_1	R_2	// R

One Address Instructions :-

opcode	Operand Address S/D

⇒ It has only one Address field the operand specified in the instruction may be source or destination. depending on instruction. ~~on~~ instruction.

⇒ one Address instructions use a implied Accumulator (AC) register for all data manipulation.

⇒ implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.

⇒ All operations done between accumulator registers and memory operand.

M·S

⇒ It is also called single accumulator organization.

Ex ADD X // AC ← AC + M[X]

Ex Evaluate C = A + B (using one-Address - format)

So LOAD A // AC ← M[A]

ADD B // AC ← AC + M[B]

STORE C // M[C] ← AC

LOAD and STORE Instruction :- used for transfers to and from memory to AC register.

LOAD Instruction :- operand (like A) specifies the source operand and the destination Location is AC register.

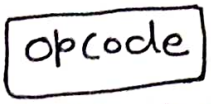
Ex LOAD A // AC ← M[A]

STORE Instruction :- operand like (C) specifies the destination Location and the source is AC register.

e.g STORE C // M[C] ← AC

M·S

Zero Instruction - format :-



⇒ In zero Address instructions Stack is used. Also called Stack based organization.

⇒ A Stack based computer does not use an address field in the instructions. (like ADD, MUL).

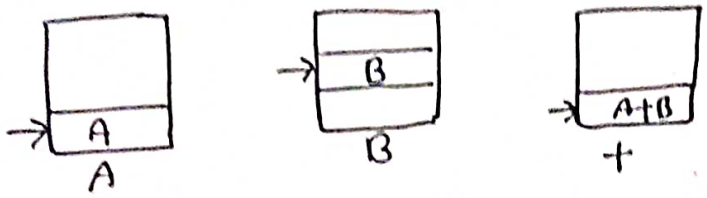
⇒ PUSH and POP instructions, However needs an Address field to specify the operand that communicate with Stack.

⇒ To evaluate a arithmetic expression first it is converted to Reverse Polish notation i.e Postfix notation.

⇒ The name "zero-address" is given to this type of computer of the absence of a address field an instruction.

Ex C = A + B

```
PUSH   A      // TOS ← A
PUSH   B      // TOS ← B
ADD           // TOS ← (A+B)
POP    C      // M[C] ← TOS
```



NOTE :-

① Always use minimum number of instruction (So that the Program length can be minimized)

② Always use minimum number of Register (Registers can be reused)

Eg $X = x + y * z - t / u$

So convert the expression into Postfix

$x + y * z - t / u$

$\Rightarrow x + yz* - t/u$ (yz* = T)

$\Rightarrow x + T - t/u$ (t/u = Y)

$\Rightarrow x + T - Y$

$\Rightarrow xT - Y$ (xT = Z)

$\Rightarrow Z - Y$

$\Rightarrow ZY -$

put the value Z and Y and T

M·S

x T + y —

xyz * + t u / — post-fix

X = xyz * + t u / —

zero Address instruction

		top of Stack(TOS)
PUSH	x	TOS – x
PUSH	y	TOS – y
PUSH	z	TOS – z
MUL		TOS = (y*z)
ADD		TOS = x+(y*z) = M
PUSH	t	TOS = M
PUSH	u	TOS – T
DIV		TOS – u
SUB		TOS T/u
POP	X	

NOTE!

(1) Infix (2) Prefix (3) POSTfix

A + B + A B A B +

(1) Precedence (2) Associativity

* / ( ) A + B – C →

M·S

Q2 $X = (A+B) * (C+D)$ using zero Address instruction format.

Sol convert the expression into postfix

$$\boxed{X = AB + CD + *} \text{ postfix}$$

zero Address instruction format

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) = M
POP X	//M[X] ← TOS

Q3 $X = P + Q/R - S * T$ (using zero instruction format)

Sol convert the expression into postfix

$X = P + QR/ - ST*$

$X = PQR/+ST*-$ post-fix

zero instruction format

```
PUSH   P        // TOS ← P
PUSH   Q        // TOS ← Q
PUSH   R        // TOS ← R
DIV             // TOS ← Q/R
ADD             // TOS ← P+Q/R
PUSH   S        // TOS ← S
PUSH   T        // TOS ← T
MUL             // TOS ← S*T
SUB             // TOS ← P+Q/R-S*T
POP    X        // M[X] ← TOS
```

Ex one-Address instruction format :-

$R = ((x + (y * z)) - (t / u))$

Sol Using one-instruction format

LOAD	y	// AC ← y
MUL	z	// AC ← AC * z ↑ y * z
ADD	x	// AC ← AC + x AC ← y * z + x
STORE	R_1	// R_1 ← x + (y * z)
LOAD	t	// AC ← t
DIV	u	// AC ← t/u
STORE	R_2	// R_2 ← t/u
LOAD	R_1	// AC ← R_1 AC ← x + (y * z)
SUB	R_2	// AC ← R_1 − R_2 AC ← x + (y * z) − t/u
STORE	R	// M[R] ← AC

Ex 2 Evaluate C = A + B (using one Address Instruction format)

Sol

C = A + B

```
LOAD    A     //  AC ← M[A]
ADD     B     //  AC ← AC + M[B]
STORE   C     //  M[C] ← AC
```

Ex3 $X = \dfrac{A + B * C}{D - E * F + G * H}$

using one-Address Instruction format.

Sol

$X = \dfrac{A + B * C}{D - E * F + G * H}$

```
LOAD    E     AC ← M[E]
MUL     F     AC ← AC * M[F]
              (AC ← E * F)
STORE   T     M[T] ← E * F   (Temporary Store value in Register)
LOAD    D     AC ← M[D]
```

Note:- Always use minimum number of instructions and minimum number of distinct registers.

example - two - instructions format

Q1 $$x = \frac{(A + B * C)}{(D - E * F + G * H)}$$

(using two - instruction format).

Sol $$x = \frac{(A + B * C)}{\underbrace{CD}_{R_2} - \underbrace{E * F}_{R_3} + \underbrace{(G * H)}_{R_3}} \rightarrow R_1$$

R_2 ... R_3 ... R_2

MOV	R_1,	B	$R_1 \leftarrow M[B]$
MUL	R_1,	C	$R_1 \leftarrow R_1 * M[C]$
ADD	R_1,	A	$R_1 \leftarrow R_1 + M[A]$
mov	R_2	D	$R_2 \leftarrow M[D]$
mov	R_3	E	$R_3 \leftarrow M[E]$
MUL	R_3,	F	$R_3 \leftarrow R_3 * M[F]$
SUB	R_2, R_3		$R_2 \leftarrow R_2 - R_3$
MOV	R_3, G		$R_3 \leftarrow M[G]$

M-S

R = ((X + (Y * Z)) − (D/4))

SUB	T	AC ← D − E * F
STORE	T	M[T] ← AC
LOAD	G	AC ← M[G]
MUL	H	AC ← AC * M[H] (AC ← G * H)
ADD	T	AC ← AC + M[T]
STORE	T	M[T] ← D − E * F + G * H
LOAD	B	AC ← M[B]
MUL	C	AC ← AC * M[C]
ADD	A	AC ← AC + M[A]
DIV	T	AC ← AC / M[T]
STORE	X	M[X] ← AC

Ex-3 X = (A+B) * (T+Q)

Ex-4 $Y = \dfrac{(A - B + C * (D * E - F))}{G + H * I}$ two address instruction format.

Do yourself

M.S

Ex2

$$X = \frac{A + B * C}{D - E * F + G * H}$$

By using three Address instruction format:

Sol

$$X = \frac{A + B * C}{D - E * F + G * H}$$

MUL	$R_1, B, C,$	$R_1 \leftarrow M[B] * M[C]$
ADD	$R_1, R_1 A$	$R_1 \leftarrow R_1 + M[A]$
MUL	R_2, E, F	$R_2 \leftarrow M[E] * M[F]$
SUB	R_2, D, R_2	$R_2 \leftarrow M[D] - R_2$
MUL	R_3, G, H	$R_3 \leftarrow M[G] * M[H]$
ADD	$R_2, R_2, R_3.$	$R_2 \leftarrow R_2 + R_3$
DIV	$X \; R_1, R_2$	$M[X] \leftarrow \frac{R_1}{R_2}$

Ex-3 two - Address instruction format.

$$X = \frac{A + B * C}{D - E * F + G * H}$$

by using two - address Instruction format

M.S

$$X = \frac{A + B * C}{D - E * F + G * H}$$

MOV	R_1, B	$R_1 \leftarrow M[B]$
MUL	R_1, C	$R_1 \leftarrow R_1 * M[C]$
ADD	R_1, A	$R_1 \leftarrow R_1 + M[A]$
MOV	R_2, D	$R_2 \leftarrow M[D]$
MOV	R_3, E	$R_3 \leftarrow M[E]$
MUL	R_3, F	$R_3 \leftarrow R_3 * M[F]$
SUB	R_2, R_3	$R_2 \leftarrow R_2 - R_3$
MOV	R_3, G	$R_3 \leftarrow M[G]$
MUL	R_3, H	$R_3 \leftarrow R_3 * M[H]$
ADD	R_2, R_3	$R_2 \leftarrow R_2 + R_3$
DIV	R_1, R_2	$R_1 \leftarrow R_1 / R_2$
MOV	X R_1	$M[X] \leftarrow R_1$

M.S

~~MUL~~ ADD	R_1, A	$R_1 \leftarrow R_1 + M[A]$
MOV	R_2, D	$R_2 \leftarrow M[D]$
DIV	R_2 E	$R_2 \leftarrow D/E$
SUB	R_1, R_2	$R_1 \leftarrow R_1 - R_2$
MOV	X, R_1	$M[X] \leftarrow R_1$

Ex-4 $x = \dfrac{A - B + C * (D * E - F)}{G + H * I}$

write code two-address instruction format. ~~Three~~

Do yourself

Three-Address instruction format examples

Ex-1 $A + B * C - D/E$

MUL	R_1	B, C	$R_1 \leftarrow B * C$
ADD	R_1	R_1 A	$R_1 \leftarrow R_1 + A$
DIV	R_2	D, E	$R_2 \leftarrow D/E$
SUB	X	R_1, R_2,	$X \leftarrow R_1 - R_2$ (Result)

M.S

MUL	R_3, H	$R_3 \leftarrow R_3 * M[H]$
ADD	R_2, R_3	$R_2 \leftarrow R_2 + R_3$
DIV	R_1, R_2	$R_1 \leftarrow R_1 / R_2$
MOV	X, R_1	$M[X] \leftarrow R_1$

Ex2 $X = (A+B) * (C+D)$ using two Address instruction format.

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

MOV	R_1, A	$R_1 \leftarrow M[A]$
ADD	R_1, B	$R_1 \leftarrow R_1 + M[B]$
MOV	R_2, C	$R_2 \leftarrow M[C]$
ADD	R_2, D	$R_2 \leftarrow R_2 + M[D]$
MUL	R_1, R_2	$R_1 \leftarrow R_1 * R_2$
MOV	X, R_1	$M[X] \leftarrow R_1$

Ex-3 $A + B * C - D/E$ (by using two instruction format?

Sol $X = A + B * C - D/E$

MOV	R_1, B	$R_1 \leftarrow B$
MUL	R_1, C	$R_1 \leftarrow R_1 * M[C]$

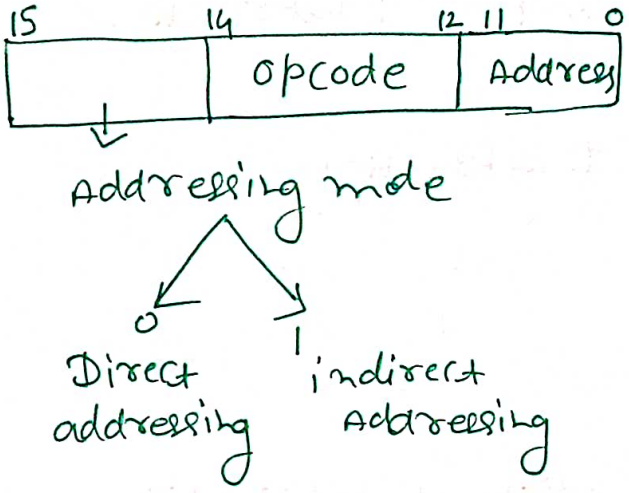
M.S

Computer Instruction:-

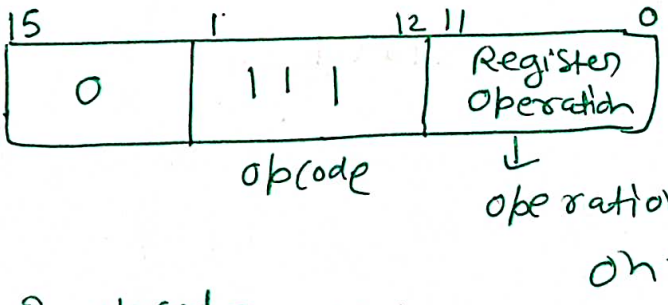
⇒ Memory reference
⇒ Register reference
⇒ I/O instruction

All three are 16-bits

Memory reference:-

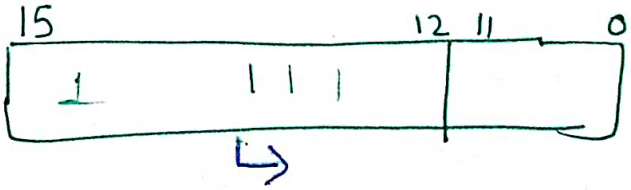


Register reference:-



Register Reference { opcode = 1 1 1
Addmode = 0

I/O instruction:-



Add. mode = 1

opcode = 111

→ I/O instruction

Types of Instructions:-

⇒ Data transfer Instructions

⇒ Data manipulation Instructions

⇒ Program control Instructions

Data Transfer Instructions:-

Transfer data from on location to another without changing the binary information content.

Data manipulation Instructions:-

Perform Arithmetic Logic, Shift operation.

Program control Instruction:-

It Provides decision making capabilities & Change the Path taken by the Program when executed in the computer.

Data Transfer Instructions:-

Name	Mnemonic
Load	LD / LOAD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
output	OUT
Push	PUSH
Pop	POP

(2) Data manipulation Instruction :-

⇒ Arithmetic Instructions
⇒ Logical & bit manipulation
⇒ Shift instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Substract	SUB
multiply	MUL
Divide	DIV

Logical & Bit manipulation :-

Name	mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive OR	XOR

Shift Instruction:

Name	mnemonic
Logical Shift right	SHR
Logical Shift Left	SHL
Arithmetic Shift right	SHRA
Arithmetic Shift Left	SHLA
Rotate Left	ROL
Rotate right	ROR

(3) Program control Instruction:-

Name	mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET

MICRO OPERATIONS :- The operation executed on data stored in register is called micro-operations.

ex Shift, load, Clear, Count.

Types of micro operations :-

(1) Register transfer micro operation — transfer Binary informations.

(2) Arithmetic micro operation :- Perform arithmetic operation on numeric Data Stored in registers.

(3) Logical micro operation :- Perform arithmetic ~~operation on numeric data Stored in registers.~~ Perform bit manipulation operations on data stored in registers.

(4) Shift micro operation :- Perform shift operations on data stored in registers

INSTRUCTION CYCLES:-

Basic function of a computer is execution of Program.

* A Program is a set of instructions.

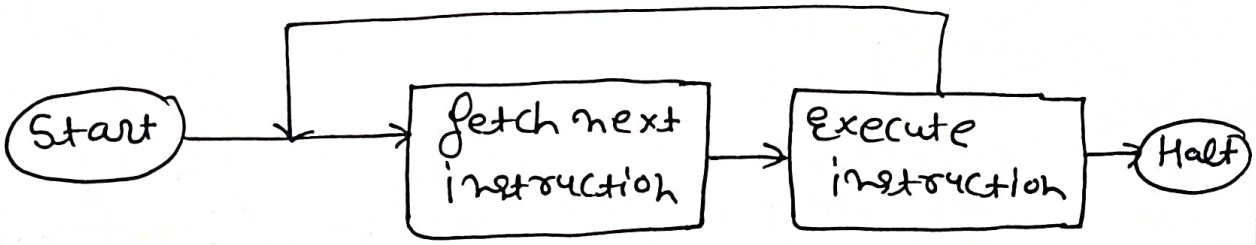
to Process an instructions:

(1) The Processor reads (fetches) an instruction from memory.

(2) The Processor executes the instruction.

(3) Execution may involve several operations.

Basic Instruction cycle;



⇒ Execution cycle for a Particular operation is dependent on the type of operation.

⇒ Reference to memory can be one/many.

⇒ Instead of reference of instruction can specify an I/O operation.

M·S

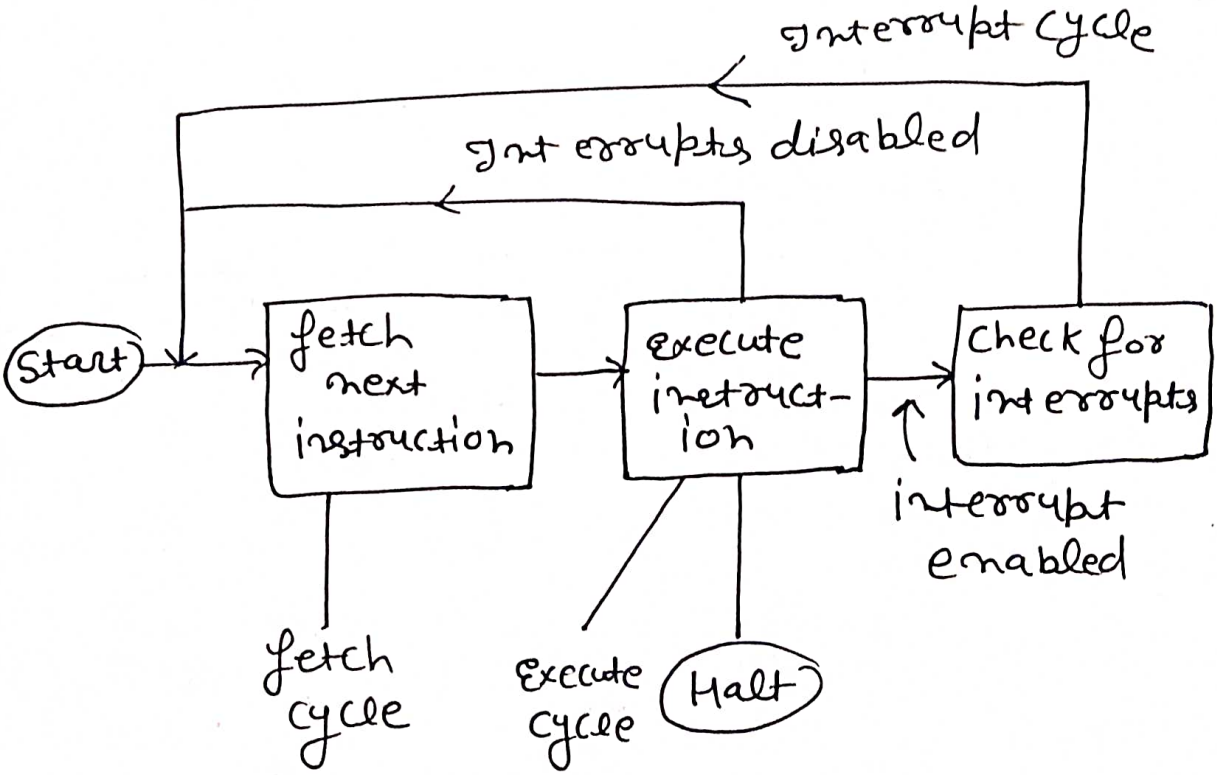
Interrupts and the instruction cycle:-

virtually, I/O devices may interrupt the normal Processing of Program.

On this account, execution is suspended control gets transferred.

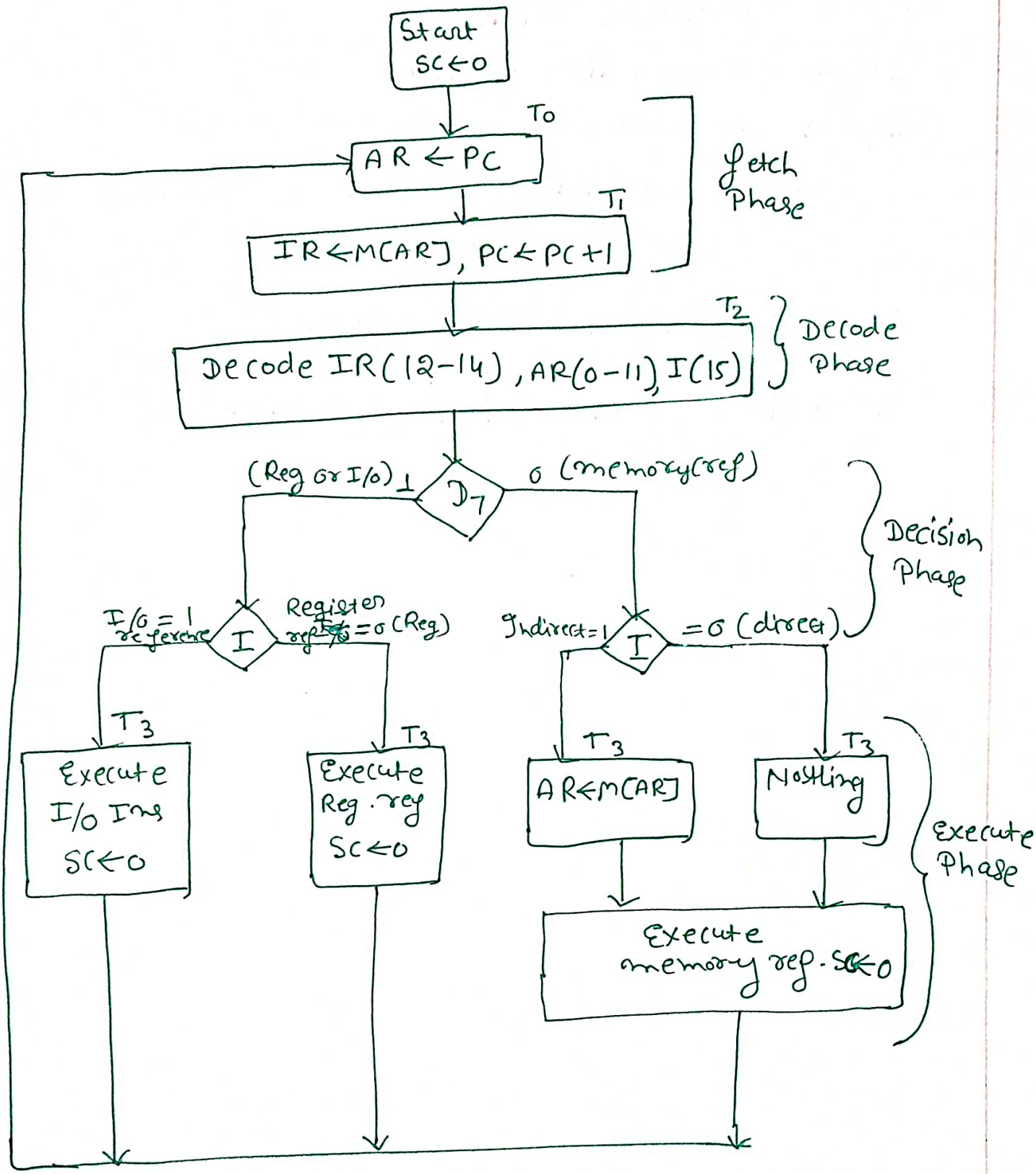
to serve this Purpose, an interrupt cycle is added to instruction cycle.

Instruction cycle with interrupt:-



M.S

Instruction cycle :-



AR → Address Register

PC → Program counter

IR → Instruction Register

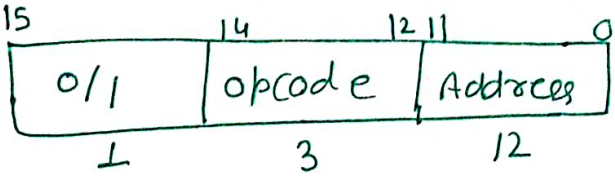
D_7 → Decoder output

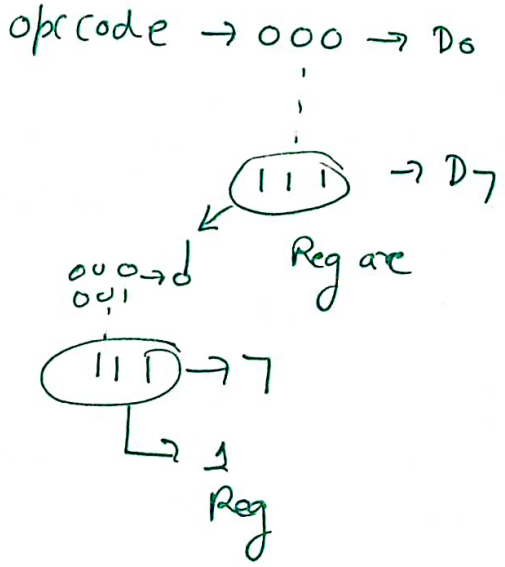
I → Instruction format

SC → sequence counter

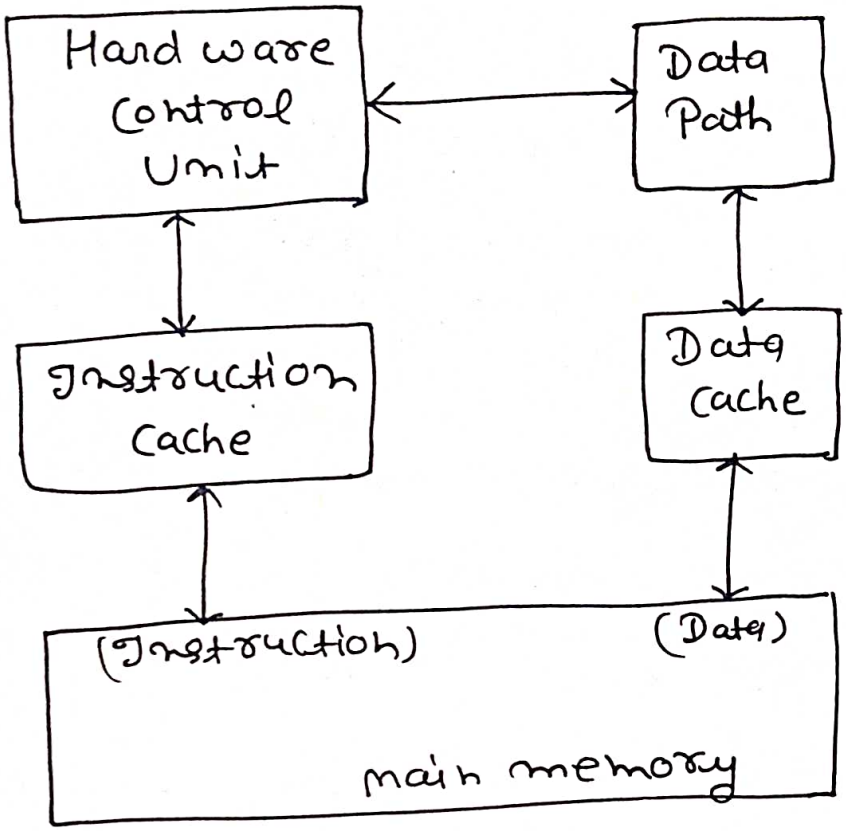
T_0, T_1, T_2, T_3 → Clock cycle

Decoding Instruction format





RISC (Reduced instruction set computer):-



(RISC ARCHITECTURE)

Characteristics of RISC Processor:-

⇒ Ability to execute one instruction per clock cycle.

⇒ efficient pipelining.

⇒ RISC Processor consists mostly register-to-register operations.

⇒ It consists of relatively few instructions (< 50)

M.S

⇒ It consists of relatively few Addressing modes.

⇒ It consists of fixed length Instruction format and easily decoded instruction format.

⇒ RISC processor contains Hardwired control unit rather than microprogrammed control unit.

⇒ very few Instruction refer memory.

⇒ memory access limitted to load and store Instructions.

⇒ Complexity is there in the compiler.

⇒ It consists of multiple register set.

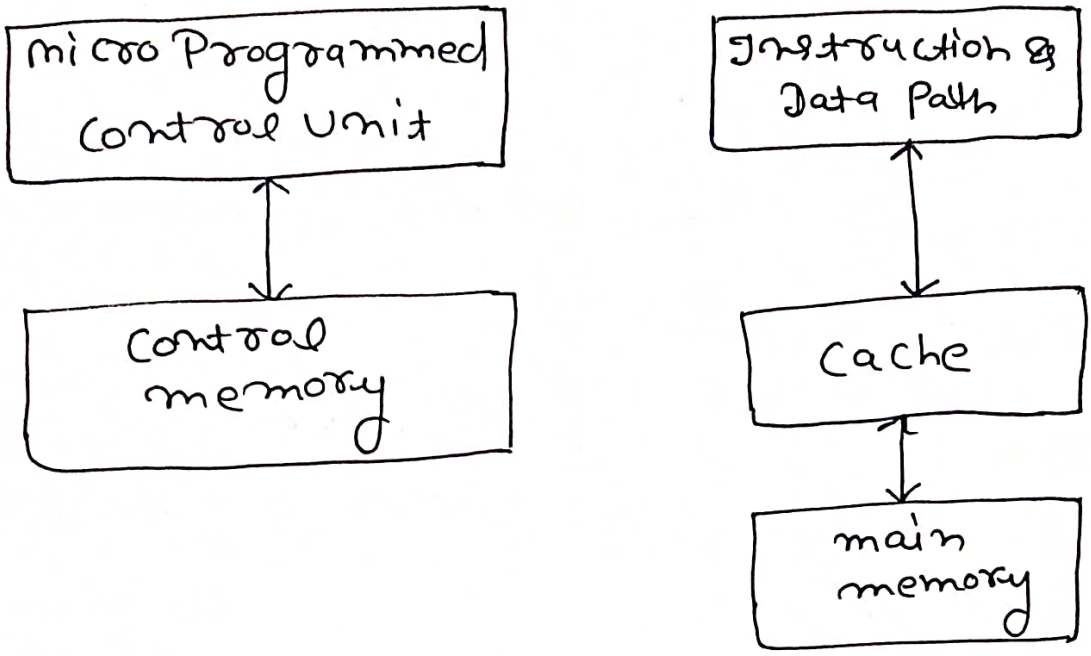
⇒ All operations done with in the registers of the CPU. Hence A relatively large number in the Processor.

⇒ Use of overlapped register windows to speed up Procedure call and return.

⇒ Compiler support for efficient translation of High-level language Programs to machine-level language Programs.

⇒ An Average CPI is less than 1.5 RISC Processor.

CISC (Complex instruction set compute):-



(CISC architecture)

Characterstics of CISC architecture:-

⇒ variable length instruction format used by CISC architecture.

⇒ Large number of Instructions. typically from 100 to 250 instructions.

⇒ A large variety of Addressing modes typically from 5 to 20 different Addressing modes.

M.S

⇒ Large number of Instructions, typically from 100 to 250 instructions.

⇒ A large variety of Addressing modes typically from 5 to 20 different Addressing modes.

⇒ Complex instructions take multiple cycles to execute. An average CPI is between 2 and 15.

⇒ most of the instructions may refers memory to manipulate operands in memory.

⇒ Complexity in the microProgram.

⇒ execution time is more.

⇒ instructions are larger than one word size.

⇒ It supports more data types.

⇒ single register set.

⇒ Instructions decoding Logic is complex.

⇒ CISC Architecture contains a microProgrammed control unit not a hardwired control unit.

M.S

Microprogrammed control unit:-

A control unit whose Binary control values are saved as words in memory is called a micro Programmed control unit.

Advantages:-

① micro-Program can be easily.
② flexible.
③ Better in terms of scalability than Hardwired.

Disadvantages:-

① Hardware cost is more because of control memory and its access circuitry
② Slower than hardwired.

Advantage and Disadvantage in Hardwired:-

Advantage:-

① faster Processing.
② simple to implement or configure.

Disadvantage:-

① It is not applicable to change the structure and Instruction set once It is developed.

② The design of the computer is complex.
③ The architecture and instructions set are not specified.
④ It is quick.
⑤ It has a Processor to create signals to be executed in the right sequence.
⑥ It operates through the need for flip-flops, chips, and sequential circuits.

Micro Programmed:-

① It is applicable to make modification by changing the microProgram saved in the control memory.
② The design of the computer is simplified.
③ The architecture and instruction set is specified.
④ It is moderate comparatively.
⑤ It facilitates the micro sequencer from which Instruction bits are decoded and executed.
⑥ It controls the sub-devices including ALU, registers, buses, instruction registers.

M·S

HARDWIRED CONTROL UNIT :-

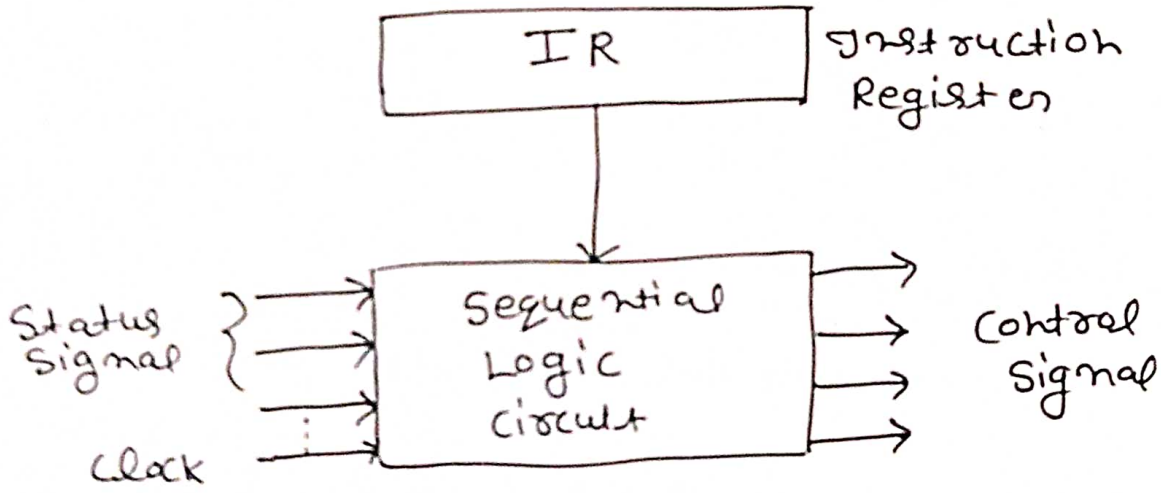
A control unit can be implemented Basically by two techniques.

① Hardwired Implementation

② micro-Programmed implementation.

⇒ Hardwired control unit is implemented as a sequential Logic circuit or a finite state machine. That generates a specific sequence of control signals.

Structure :-



It uses fixed logic. Interprets and then generates corresponding control signal factors for design.

⇒ Amount of hardware.
⇒ Speed of operation
⇒ Cost.

M.S

There are four techniques for design of Hardwired control unit.

(1) State-table method.

(2) minimizes Hardware.

(3) Constructs a State transition table.

(4) every generation of States has a set of control signals.

(2) Delay-element method:-

control signals follow a Proper sequence.

A specific time delay between two groups of control signals.

to ensure synchronisation, D-flip flops are controlled by a common clock signal.

(3) Sequence-Counter method:-

It uses Counter for timing Purposes.

(4) PLA method:- It uses Programmable logic array.

M·S

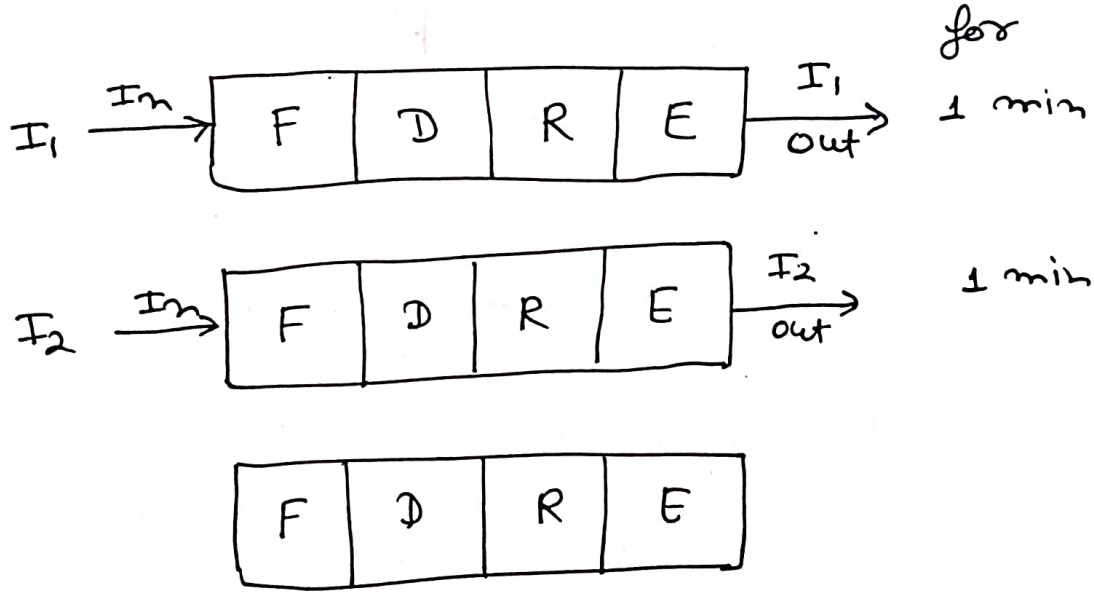
PIPELINE:-

Pipelining is the Process of arrangement of Hardware elements of CPU such that its overall Performance is increased.

Simultaneously execution of more than one Instruction takes place in pipelined Processor.

In pipelining multiple Instruction are overlapped in execution.

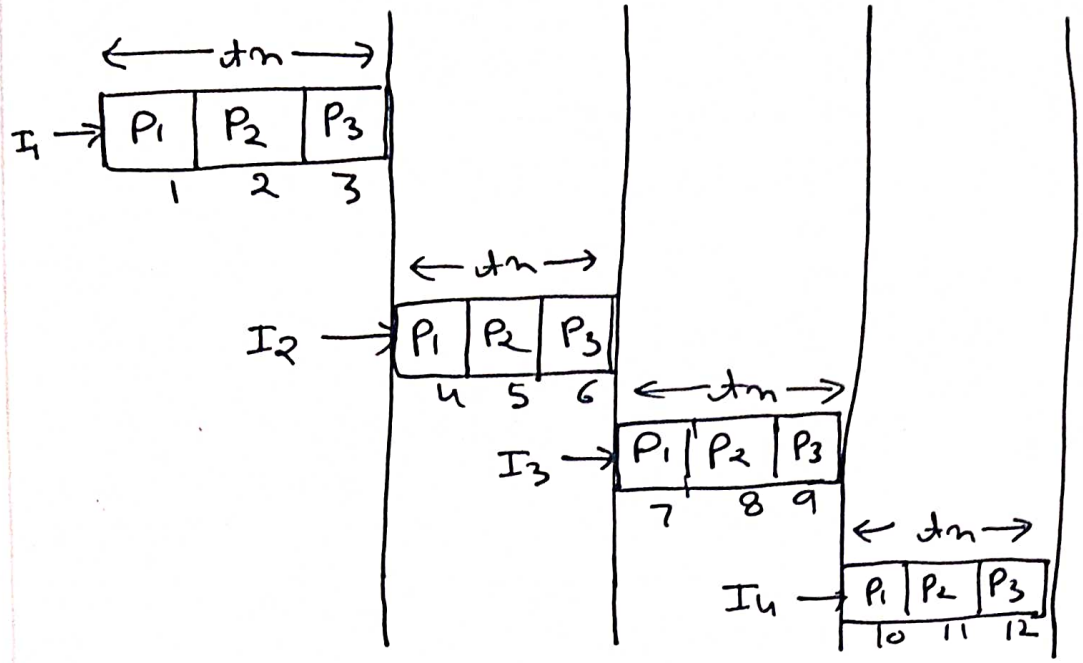
Sequential execution:-



NON Pipelining Representation:-

Phase = 3 Instructions = 4

M.S



total clock cycle = $n \times k$

$= 3 \times 4$

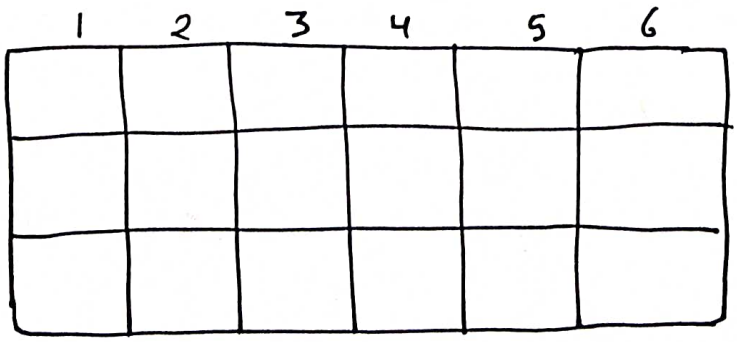
$= 12$

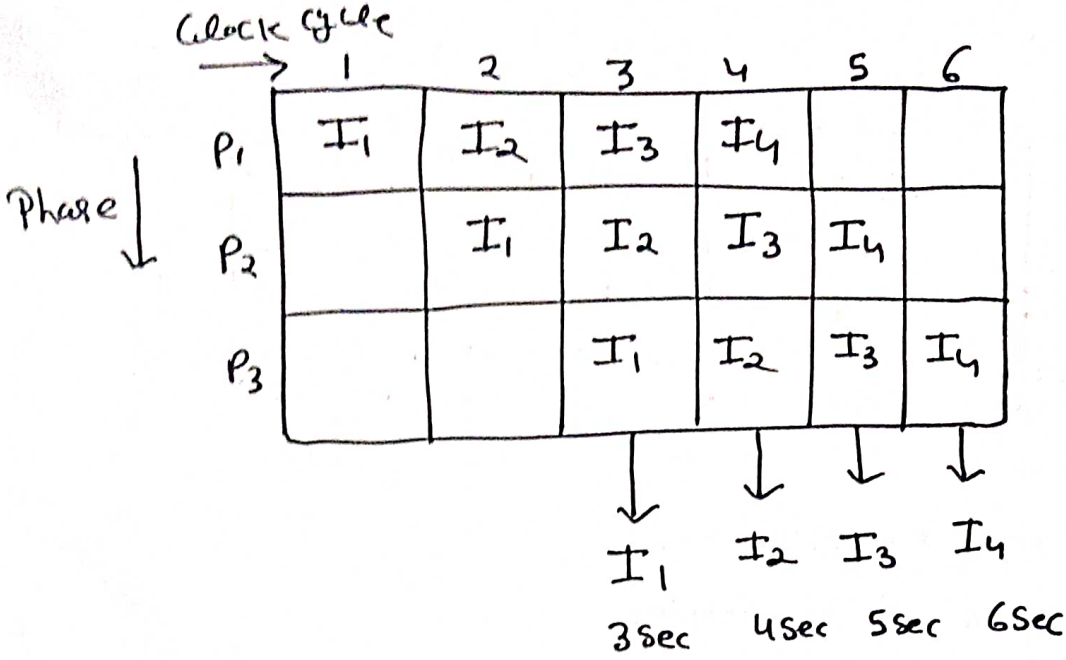
$= k \times n$ or $k \times n$.

where ⇒ k is number of Phase

⇒ n is number of Instruction.

Pipelining clock cycle :-





Space Time Diagram or Phase Time Diagram

IF
ID
EX
MEM
WB

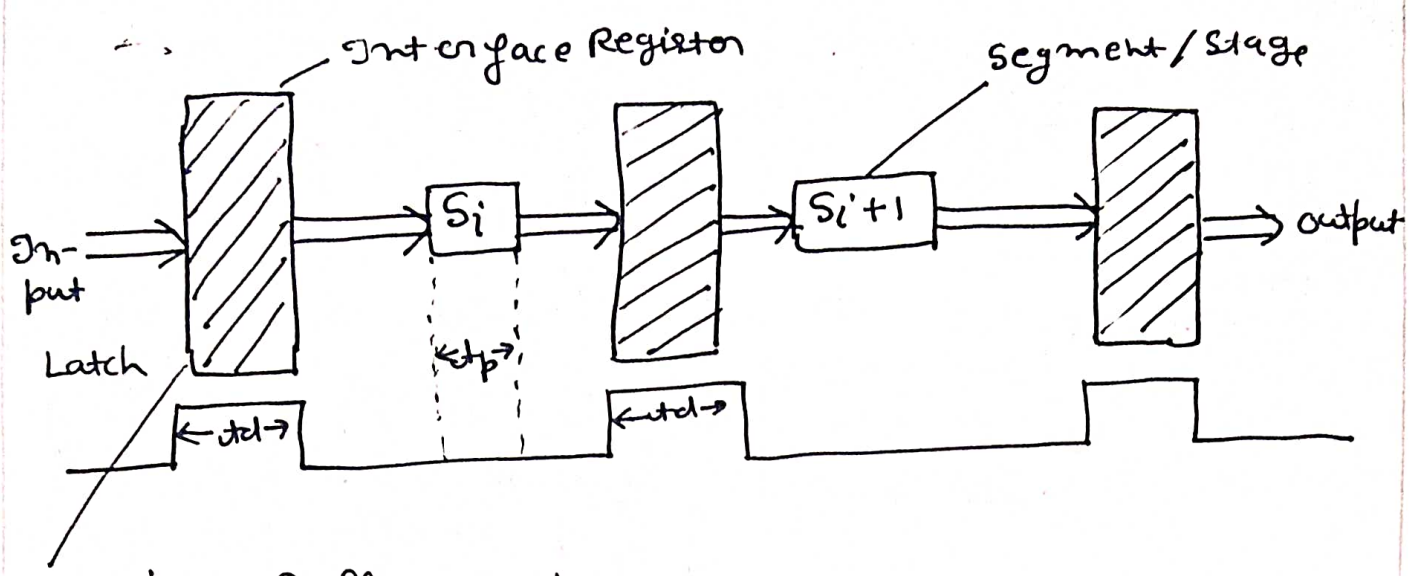
total clock cycle = $k+n-1$

$= 3+4-1$

$= 6$ sec

Factors in pipeline Architecture :-

1. Speed up Ratio — S_k
2. frequency — f
3. Throughput — H_k
4. efficiency — E_k



It is a Buffer / Register that holds instructions temporary.

t_n ⇒ time for complete execution of instruction (NP) non pipeline.

t_p ⇒ time taken in Phase.

t_d ⇒ delay time (time taken in buffer or / Registers)

$$\tau = t_p + t_d$$

time taken by one Phase = τ (tau)

① Speed up Ratio S_k ⇒

M·S

$$S_k = \frac{\text{time taken in Np}}{\text{time taken in P}}$$

$$= \frac{n \times t_n}{(k+n-1) \times \tau}$$

$n \geq k$

↳ multichgrates

$n = 1000$

$K = 3$

$\Rightarrow 1000 - 1 + 3$

$\Rightarrow 1002$

$\rightarrow K + (n-1) \leq n$

$$= \frac{\cancel{n} \times t_n}{\cancel{n} \times \tau}$$

$$\boxed{S_k = \frac{t_n}{\tau}}$$

② Frequency $= \frac{1}{\tau}$

$$\boxed{f = \frac{1}{\tau}}$$

③ Throughput $(H_k) \Rightarrow$

Number of task completed per unit time.

$$H_k = \frac{n}{(k+(n-1)*\tau}$$

M·S

$$= \frac{n}{(k+n-1)} \times \frac{1}{\tau}$$

$$H_k = \frac{n}{k+n-1} \times f$$

$$\boxed{H_k = \frac{n \times f}{k+n-1}}$$

(4) efficiency $E_k \Rightarrow$

$$E_k = \frac{S_k}{K}$$

$$E_k = \frac{n \times t_n}{(k+n-1) \times \tau}$$

$$\boxed{E_k = \frac{n \times k}{k+n-1}}$$

$\tau =$

$= \frac{n \times t_n}{}$

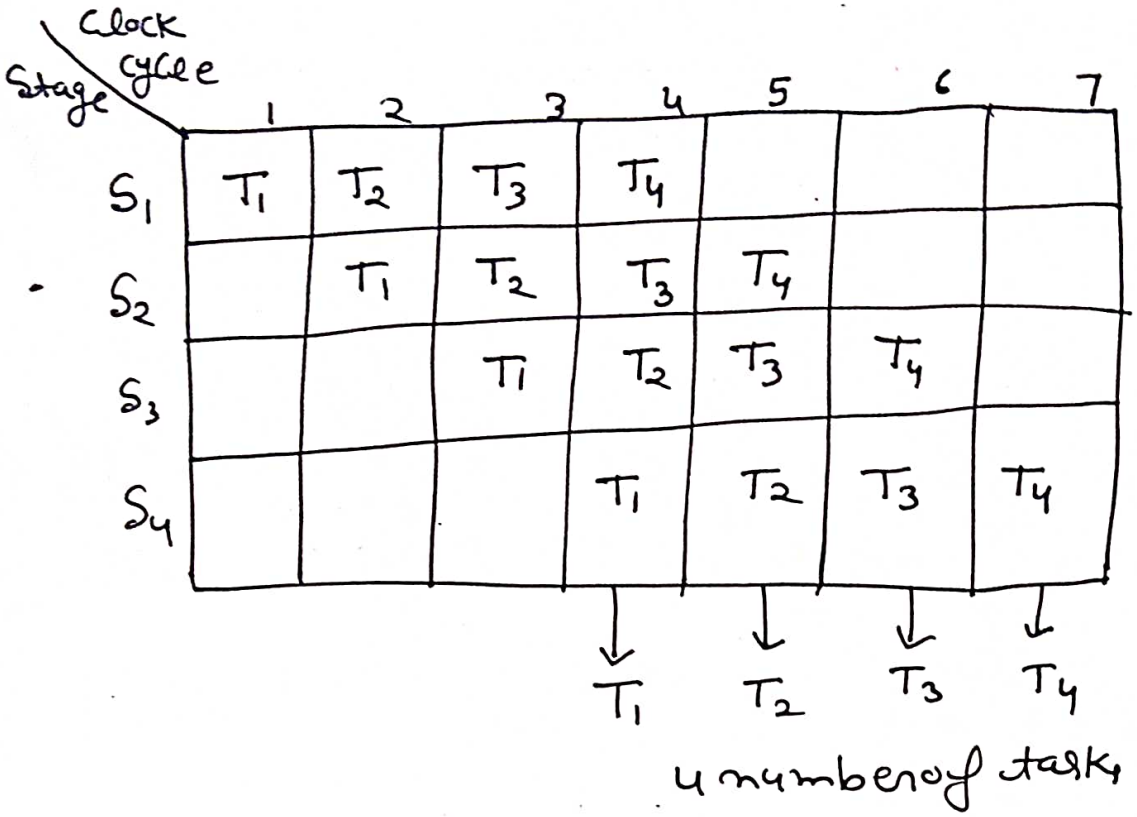
$S(k) = \frac{t_n}{\tau}$

$= \frac{\tau n}{\tau} \times K$

Pipeline Performance:-

n: Number of tasks to be Performed conventional machine (Non-pipelined)

t_n: clock cycle



4 clocks cycle pipeline is full then one task completed.

task = 4 Stage = 4

Linear Pipeline Processor:-

Linear Pipeline Processor is a cascade of Processing Stage which are linearly connected to perform a fixed function over a stream of data following from one end to another.

⇒ A Linear Pipeline Processor is constructed with a K Processing Stages.

M·S

T_1 : Time required to complete 1 task.

$T_1 = n \times t_n$

pipelined machine (k stages)

t_p : clock cycle (time to complete each sub operation)

t_k : Time required to complete 1 task.

$T_k = (k+n-1) \times t_p$

Speed up

S_k : speed up

$$S_k = \frac{n \times t_n}{(k+n-1)\, t_p}$$

$$\lim_{n \to \infty} S_k = \frac{t_n}{t_p} = (k \text{ if } t_n = k * t_p)$$

M.S

⇒ External input is supplied to the pipeline from stage S_i

⇒ The Processed result is passed from stage S_{i+1}. for all $i = 1, 2, 3, \ldots k-1$

⇒ The final result is produced from the last pipeline stage S_k.

$1\,Hz = 1\,cycle/seconds$

$1\,Sec = 1 \times 10^6\,\mu sec$

$$S_k = \frac{T_1}{T_2}$$

$$S_k = \frac{n \times t_n}{(k+n-1)\,t_p}$$ (NP) (P)

if n values increased,

maximum speed is = k

maximum speed up ratio $S_k = k$

M.S

Ex1 Determine the number of clock cycles that it takes to Process 200 tasks ing 6 segment (Phase) pipeline.

Sol $K = 6$ $n = 200$

Number of clock cycle $= K + n - 1$

$= 6 + 200 - 1$

$= 205$ clock cycle

Af

Ex-2 A non-pipeline system takes 50 ns to Process a task. The same task can be Processed in a 6-segment Pipeline with a clock cycle of 10ns. Determine the speed-up ratio of the Pipeline for 100 tasks. What is the maximum speed-up that can be achieved

Sol In Non-pipeline

$t_n = 50$ ns

In pipelining

$t_p = 10$ns $K = 6$

$n = 100$

Speed-up Ratio $(S_K) = \frac{n t_n}{(k+n-1)\, t_p}$

$= \frac{100 \times 50}{(6+100-1) \times 10}$

$= \frac{100 \times 50}{105 \times 10}$

$= \frac{100}{27}$

$$S_K = 4.76$$

When maximum speed up (S) can be achieved when no of task increases is

$S = k$ as $n = \infty$

So the maximum speed-up will be 6 i.e

$$S = 6$$

M.S

Ex-3 Consider the execution of Program 15000 instructions by a linear (one like in execute) Pipeline Processor with a clock rate of 25 MHz (megaHertz). Assume that the instruction pipeline has 5-stages and that 1 (one) Instruction is issued per clock cycle. The penalties due to branch Instructions and out-of-sequence execution are ignored.

(a) Calculate the speed-up factor in using this pipeline.

(b) Calculate the efficiency and throughput.

Sol given

Number of Instruction

$n = 15000$

frequency of the clock

$f = 25\,MHz$

Number of Pipeline Storage

$K = 5$

M.S

Instruction is issued per clock cycle

$1\,Hz = 1$ Per Sec

$f = \frac{1}{\tau}$ (Clock Period)

① Speed-up S_k

$$S_k = \frac{T_1(NP)}{T_2(P)}$$

$$\boxed{S_k = \frac{nk\tau}{(k+n-1)\tau}}$$

$$S_k = \frac{nk}{k+n-1}$$

$$S_k = \frac{15000 \times 5}{5+15000-1}$$

$$S_k = \frac{75000}{15004}$$

$$\boxed{S_k = 4.999}$$

② effeciency E_k

$$E_k = \frac{S_k}{k} = \frac{n}{k+(n-1)}$$

M.S

$$E_k = \frac{4.999}{5}$$

$$\boxed{E_k = 0.999}$$

③ Throughput H_k

$$H_k = \frac{n}{(k+n-1)\tau}$$

$$H_k = \frac{nf}{k+n-1}$$

$$= \frac{nf}{k+n-1}$$

$$H_k = \frac{E_k}{\tau}$$

$$= \frac{n}{(k+n-1)\tau} = \frac{nf}{k+n-1}$$

$$= \frac{15000 \times 25}{5+(15000-1)}$$

$$= \frac{15000 \times 25}{15004}$$

$$= \frac{375000}{15004}$$

$$\boxed{H_k = 24.99 \text{ MIPS}}$$

(million instruction per/sec)

M·S

Ex-4 Consider a non-pipelined Processor (P_1) with a clock rate of 25 MHz and average cycles per instruction (CPI) of 4. The same Processor (P_2) with 5 Stages but due to the internal pipeline delay, the clock speed is reduced to 20 MHz. Assume there are no Stalls in the Pipeline.

(a) if a Program of 100 instructions is to be executed on both Processor what is Speed-up of P_2 compared to that P_1?

(b) find (MIPS) (Throughput find out) rate of P_1 as well P_2 during execution of the Program.

Sol given

$1 \text{ meg Hz} = 10^6 \text{ Hz}$

CPI = 4 ← Clock Per Instruction

$f_1 = 25 \text{ MHz}$

$= 25 \times 10^6 \text{ Hz}$

$n = 100$

$K = 5$

total time for non-pipeline Processor

M·S

T_1

$$T_1 = \frac{n \times CPI}{f_1}$$

$$= \frac{100 \times 4}{25 \times 10^6 Hz}$$

$$= \frac{400}{25 \times 10^6 cycle/sec}$$

$$= \frac{400 \text{ sec}}{25 \times 10^6 cycle}$$

$$= \frac{400 \times 10^6 \mu sec}{25 \times 10^6 cycle}$$

$$= 16 \mu sec$$

No of instruction

$\frac{\text{Ins} \times \text{cy}}{f}$

1 Hz = 1 cycle

1 sec = 1×10^6 usec

total time in pipeline Processor T_2

$$T_2 = (K + n - 1)\tau$$

$$T_2 = \frac{K + n - 1}{f_2}$$

$$= \frac{5 + 100 - 1}{20 \times 10^6 Hz}$$

$\tau = \frac{1}{f}$

1 Hz = 1 cycle/sec

1 sec = 1×10^6 usec

M.S

Ex-5 The time delay of 4 stages are $\tau_1 = 60ns$, $\tau_2 = 70ns$, $\tau_3 = 90ns$ and $\tau_4 = 80ns$ respectively and the interface latch has a delay of 10ns calculate

(a) Clock Period

(b) frequency

(c) speed-up

Sol Latch is the High-speed-register

(a) Clock Period

$$\tau = \max_{i=1}^{K} \{\tau_i\} + d$$

$$\tau = \tau_m + d$$

$$\tau = 90ns + 10ns$$

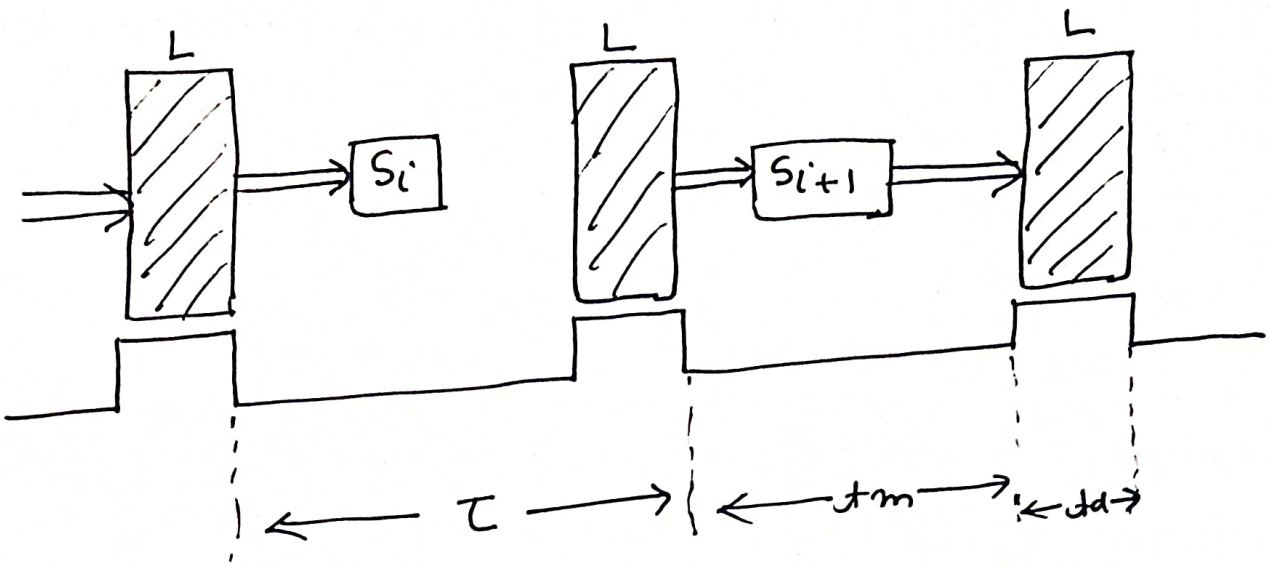
$$\tau = 100ns$$

d is latch delay time

where d = latch delay

τ_m = maximum stage delay

M·S



② frequency

$$f = \frac{1}{\tau}$$

$$= \frac{1}{100}\ ns$$

$$= \frac{1}{100 \times 10^{-9}\ sec}$$

$$= \frac{1}{10^{-7}\ s}$$

$$\boxed{f = 10^{7}\ Hz}$$

frequency Hz or MHz

$1\ ns = 1 \times 10^{-9}$

1 CPS = 1 Hz

cycle per sec

$1\ MHz = 1 \times 10^{6}\ Hz$

③ speed up = $\frac{\text{non pipeline cycle time}}{\text{pipeline cycle Time}}$

$$= \frac{\tau_1 + \tau_2 + \tau_3 + \tau_4}{\tau}$$

M·S

$$= \frac{60 + 50 + 90 + 80}{100}$$

$$S_K = 2.8 \text{ ns}$$

Ex-6 Consider a 4- segment floating point adder Pipeline. The time delay of the 4- segment in the Pipeline are $\tau_1 = 50$ns, $\tau_2 = 30$ns $\tau_3 = 95$ns and $\tau_4 = 45$ the interface latch is 5ns.

(a) How Long would it take to add 100 pair of numbers in the Pipeline.

(b) How can we reduce the total time one Half of the time calculated in Part (a).

Sol

(a) Clock Period

All Stage maximum td

$$\tau = \max_{i=1}^{k} \{\tau_i\} + d$$

$$\tau = t_m + d$$

$$\tau = 95 + 5$$

$$\tau = 100 \text{ ns}$$

M·S

for $n = 100$ $k = 4$ $\tau = 100ns$

time to add 100 number in pipeline

$$T_k = (k+n-1)\tau$$

$$= (4 + 100 - 1) \times 100$$

$$= (4 + 99) \times 100$$

$$= 10300ns$$

$$= 10.3\,\mu s$$

(b) to reduced total time period divided Stage 3 into 2 Stage.

$T_3 = 95ns$ → $T_{3_1} = 50ns$

$T_3 = 95ns$ → $T_{3_2} = 45ns$

So now total no of stages are 5 such as

$T_1 = 50ns$ $T_2 = 30ns$ $T_{3_1} = 50ns$

$T_{3_2} = 45ns$ and $T_4 = 45ns$

Clock Period $(\tau) = T_m + d$

$= 50ns + 5ns$

$= 55ns$

M·S

$$= \frac{100 \times 20 \text{ MHz}}{(5 + 100 - 1) \text{ sec}}$$

$$= \frac{100 \times 20 \times 10^6}{(5 + 99) \text{ sec}}$$

$$= \frac{2000}{104} \text{ MIPS}$$

$$= 19.23 \text{ MIPS}$$

(million of instructions per second).

Through put of Non pipelined Processor P_2

$$WP_2 = \frac{1}{CPI \times T}$$

cycle per sec $T = \frac{1}{f}$

$$= \frac{1 \times f}{CPI}$$

$$= \frac{1 \times 25 \times 10^6}{4}$$

$$= 6.25 \times 10^6 \text{ Instruction/sec}$$

$$= 6.25 \text{ MIPS}$$ Ans

M.S

$$= \frac{5 + 99}{20 \times 10^6 \text{ Hz}}$$

$$= \frac{104}{20 \times 10^6 \text{ cycles/sec}}$$

$$= \frac{104 \times 10^6 \text{ usec}}{20 \times 10^6 \text{ cycles}}$$

$$= 5.2 \text{ usec}$$

$$\text{Speed up factor} = \frac{T_1}{T_2}$$

$$= \frac{16 \times 10}{5.2}$$

$$= 3.08$$

$$\boxed{S_k = 3.08}$$

(2) Through put or (ω)

H_k = No of Instruction executed/sec

throughput for pipelined Processor P_1

$$\omega_{P_1} = \frac{nf}{k + n - 1}$$

M·S

total time $(T_k) = (k+n-1)\tau$

$= (5+104-1) \times 55$

$= 5720\ ns$

$= 5.720\ \mu s$

Result