COMPUTER ORGANIZATION

(5)

Computer Organization.

8086 → µp ✓.
Microcontroller 8051 ✓
CO Theory — Morris Manno / William Stallings. ↑ H/w issues. ↑ S/w issues.
[Hennesy].
→ Advanced Computer Architecture - Multiprocessor Env.
→ Different tech to improve Computer performance.

High level abstraction

A.K Ray ⇒ Gate CO
Ayala → H/w Architecture implemented in 8051.
Von Neumann →

3rd edn

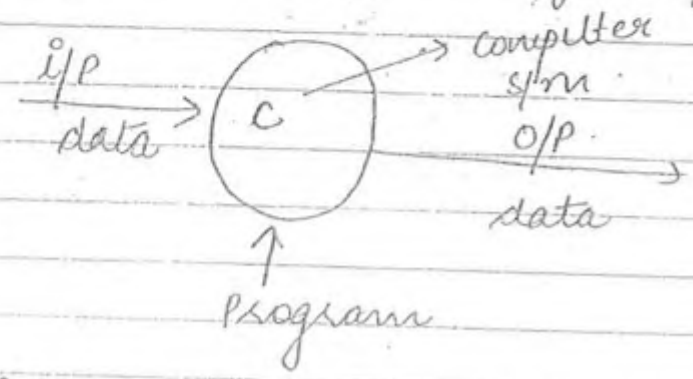
[COMPUTER ARCH & quantative approach :- Hennesy] ✓

→ Three Kinds of memory:
↳ long term memory ROM
↳ short term " SRAM → flip flop memory.
↳ DRAM → middle term memory.
↳ When the power is off, data is present until capacitor gets discharged

Computer

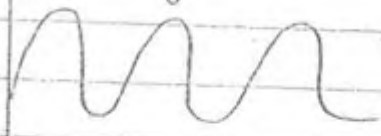
Defn → computer s/m → [It converts one form of data into another form] under the control of program.
(→ inventor)

Objective of computer s/m →
Execution of a program

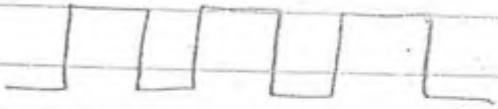


* Analog signals means continous wave
* Digital signals means discrete wave.

Analog



Digital



→ Signals understand only 2 values!-
0 & 1

→ Computer Understands only digital signals.
It understands sequence of 0's & 1's!

→ The language understood by computer is known as Binary language.

Date:
Page No.:

→ The digital signals consists of 2 values or 2 states 1's and 0's.

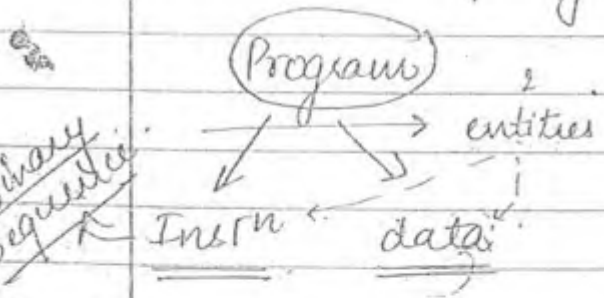
(Base).

→ The no s/m whose radix is 2 called Binary no s/m so ultimately computer understands. Binary language.

CPU understand only Binary language. As in our computer, ASCII is a translator to convert one form to another such as Keyboard & CPU.

→ Program → Sequence of Instructions along with data.

a+b → Identifying the Instⁿ is not meaningful operation. Processing is main key for getting meaningful data.



Defⁿ① Instruction is a binary sequence/pattern (1's and 0's pattern) 101111 ✓ which is designed inside the processor) to perform some task.

✓ have their own task.

→ Data → Binary sequence associated with a value

→ While executing the instⁿ, how the processor come to know about task

→ Program stored in memory → address.

→ How our processor recognize the operations associated with respect to binary pattern. It is done by Instⁿ format.

→ Instructⁿ format → layout of Instruction
↳ Depends upon size of the Instⁿ
↳ Internal structure of Instruction.

Before giving the layout of Instⁿ, we need to give the size of Instⁿ / length of Instⁿ.

Instⁿ format | opcode | Address of operand |

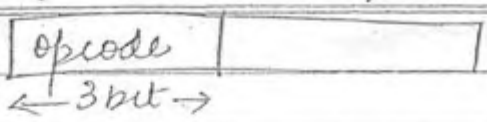
opcode → Type of Task

→ How many operatⁿ introduced? (inputs).

Encoding If there are 2^n combinations. These are defined by n bits (output).

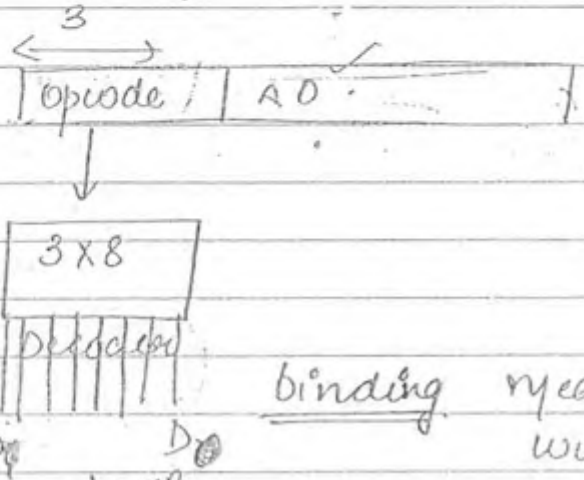
Decoding If there are n bits (I/P). Then it is defined by 2^n bits (O/P).

Instⁿ (2) Instructions are encoded binary pattern which is associated with unique operation.



Q. How the length of opcode is restricted?

A. The length of the opcode field depend on based no of operations supported by processor.



binding means meaning associated with symbol.

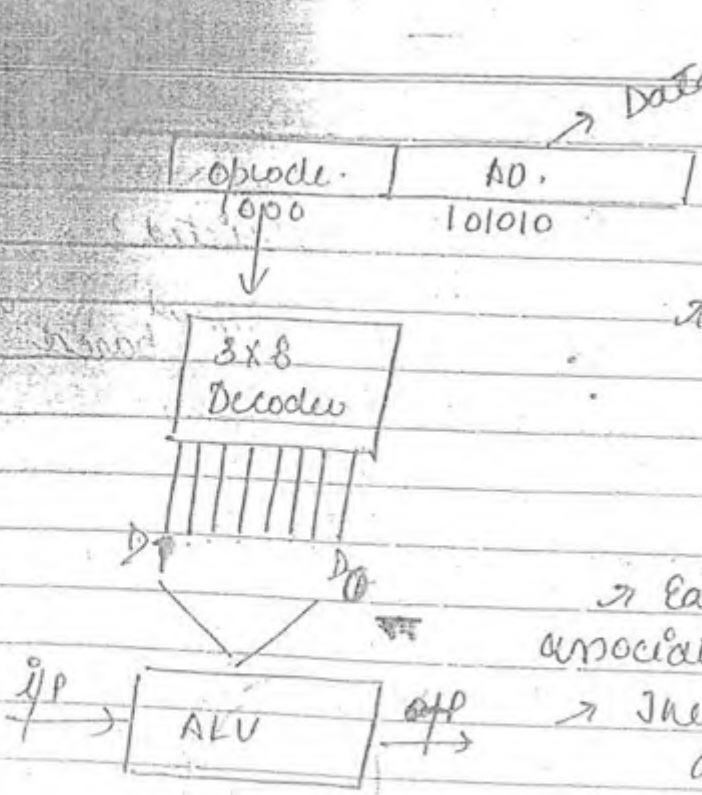
association b/w signal & opearn.

Signal	opearn
D0	+
D1	−
D2	*

at the time of designing, the designer responsibility is

1.
2.
3. Introducing
4. Introducing binding.

After introducing the operations, the operation is enabled by ALU. But not only enabling the operation is the criteria. We need to process the data i.e input the data



→ Whatever instn is fetched from memory it is fetched onto IR.

→ Each and every signal associated with one operation.

→ There are no 2 decoded signals generated w.r.t 1 operation.

→ After Decoding signal is generated, then Binding took place.

→ Do associated with + operation. That + operation is enabled in ALU. Then there is a need of data. That data is given to instn circuit.

→ Address of operand :→ gives the info about data.

→ Read the data i/p from AD and give it to ALU and then o/p is generated.

Date :
Page No. :

Performance Evaluation of Computer S/m.

Performance :-

32 bit processor 64 bit processor, There is a variation in performance.

mips processor → Computable element.

Eg. $C = a + b$.

Computable element, we are computing the result of c after performing addⁿ on a and b.

Performance is now computable element ie it is dependable element

duration of starting & ending of prog.
↑
latency of a program

Single S/m performance :->

It is depending on execution time, elapsed time or response time

Execution time :-> means latency of a program execution

single S/m	X m/c.	Y m/c → single S/m
Task	T_1	T_1
Execut^n Time	5 nsec	10 nsec

When the execut^n time is less, performance is high

*. $P \propto \frac{1}{ET}$

If Execution time is Increases, performance decreases.
" " decreases, " increases.

Speedup ⇒ $\frac{\text{performance of } x}{\text{performance of } y} = \frac{\frac{1}{ET_x}}{\frac{1}{ET_y}}$

Quantative approach of computer system

$$\text{Speedup} \Rightarrow \frac{ET_y}{ET_x}$$

In previous eg ⇒ Speedup = $\frac{10}{5}$ = 2.

x is running 2 times faster than y.

Speedup is used to compare 2 entities and choose the best one.

Performance of a Server S/m.

This performance is depending upon throughput of a S/m.

Throughput = no of tasks executed in a unit of time.

Server (not a single s/m)

x y performance depending upon throughput directly.

10hrs 10hrs.

[P ∝ TP]

(10 10)

unique

$$\text{Speedup} = \frac{P \text{ of } x}{P \text{ of } Y} = \frac{TP \text{ of } x}{TP \text{ of } Y}$$

Eg Speedup = $\frac{10}{5} = 2$ ⇒ x Server runs 2 times faster as compared to y server.

→ With respect to CO, we need to concentrate on single s/m environment.

→ How to dec execution time, is a objective of s/m designer.

Amdhal's Law:

This law focus on Improvement gain by making use of common case fast.

Design:- High performance processor:

(P) → Price 100$.

Within the price limit, we need to accomadate the performance. Then we call it as Efficient designer.

While analyzing s/m we divide into 2 parts

Frequently used Parts

Rare parts

Eg: Television remote control
Introducing 100 keys on remote is not efficient design.

Eg: 90% are floating pt } They are utilized
10% are integer } by processor.

frequently used — Rare Operations.

common case

Points:
Analyzing s/m before taking modifications into frequent case and rare case.

Take the enhancements only on frequent case.
It improves the performance within the cost and price limits by making using common case fast criteria.

2 ways

Speed up = ?

We need to calculate the speed up or performance gain. How much perf is gained with enhancement.

Benchmark program.

Speed up = Performance of (task) which are running on the s/m with Enhancement / perf of task which are running on the s/m w/o Enhancement.

$$\Rightarrow \frac{1}{\text{Execution Time System Enhancement}}$$

$$\frac{1}{\text{ET s/m w/o Enhancement}}$$

Benchmark prg → program which runs on specific type of instrn.

Date:
Page No.:

Speed up = $\frac{\text{ET s/m w/o Enhancement}}{\text{ET s/m with enhancement}}$

* Speedup ⇒ $\frac{\text{ET old s/m}}{\text{ET new.}}$

This eqn depending on 2 factors :-

① The first factor is fraction enhancement

ie How much fractn of original m/c undergoes enhacement. Only common case undergoes enhacement ie fractn

Eg fractn enhacement is 90% floatingpt.

② Speedup Enhancement : How much faster the task has run, if the enhacement portion also included.

Speed up = $\frac{\text{ET old (1)}}{\text{ET new.}}$ ↷ map probability.

ETnew = Execution time of unenhanced portion + Execution time of enhanced portion.

Eg ET old = 1

ET new ≤ ET old

ET new ≤ 1

$\frac{\text{Etold (1)}}{\text{Etnew}}$

Etold = 1

Etnew ≤ Eold

→ Fract^n enhancement = f. $S_{overall} = \frac{ET_{old}}{ET_{new}}$.

→ Speedup enhancement = S.

Calculate ET_{new}.

ET_{new} = ET of Unenhancement + ET of enhancement

$\Rightarrow ET_{old}(1-F) + \frac{ET_{old} F}{S}$.

$\Rightarrow ET_{old}\left[(1-F) + \frac{f}{S}\right]$ ✓.

$S = \frac{ET_{old} f}{ET_{new} f}$.

ET_{new} Fract^n = $\frac{ET_{old}$ frac$^n}{S}$

overall $\Rightarrow$ ET_{old}

$ET_{old}\left(1-F\right) + \frac{F}{S}$

$\Rightarrow \frac{1}{(1-F) + \frac{F}{S}} = \left[(1-F) + \frac{F}{S}\right]^{-1}$

Single inv. enhancement

$S_{overall} = \left[(1-F) + \frac{F}{S}\right]^{-1}$

↓

Total S/w Speedup after enhancement

Raw

$ET_{old}((1-F)$

$+ ET_{old} F$

(S)

$ET_{old}(1-F) + \frac{ET_{old} F}{S}$

$S_2 \frac{ET_{old} F}{ET_{new} F}$

$= \frac{1}{S}$

$+ f_1 + f_2$

If multiple enhancement takes place.

$S_{overall} = \left((1 - \sum_i F) + \sum_i \frac{F_i}{S_i}\right]^{-1}$

Where $i = 1, 2, 3 - - -$

If 2 enhancement takes place:-

$S_{overall} = \left[1 - (F_1 + F_2)\right] + \frac{F_1}{S_1} + \frac{F_2}{S_2}\Big)^{-1}$

Date:
Page No.:

Numerical:

Ques Consider Hypothetical s/m used in mathematical model simulation. Suppose floating point instructns improved to run at the rate of 2x. But only 10% of instructions are floating point. What is the overall speedup? Examine is it successful improvement or not? & take corrective decision.

10% instns are floating point ✓.

$S = 2$

10% FP

90% I

$$\text{Soverall} = \frac{1}{(1-F)+\frac{F}{S}} = \frac{1}{\left(1-\frac{10}{100}\right)+\frac{\frac{10}{100}}{2}} = 1$$

System (mathematical model simulation)

floating → 10% (improved) ⇒ 2x ⇓ $S = 2$, $F = 10\%$

int → 90%

$$\Rightarrow \frac{1}{(1-0.1)+\frac{0.1}{2}}$$

$$\Rightarrow \frac{1}{0.9+0.05}$$

$$\Rightarrow \frac{1}{0.95}$$

$$\Rightarrow 1.05$$

Soverall

0.05 / 0.9 / 0.95 ; 1.0 / 0.1 / 0.9

$\frac{1}{0.95}$; $\frac{1}{10\times 2}$; 2 | 10 ; 0.5 ; 0.05

↳ It is not correct enhancement

Here rare case goes under enhancement only fraction of speedup is improved, perf gain is very less.

In the above case rare event ~~X~~ undergoes enhancement so the performance gain is very less.

To take the corrective measures, integer unit undergoes enhancement instead of floating point.

Under this condn, the speed up is equivalent to :→ (Integer)

$f = 0.9$, $s = 2$.

$$\Rightarrow \frac{1}{(1-0.9) + \frac{0.9}{2}}$$

$$\Rightarrow \frac{1}{0.1 + 0.45} \Rightarrow 1.81$$

1.0 / 0.9 / 0.1 / 0.45 / 0.9

Speed up increases, if we improve frequent operations.

Ques (2) Consider a memory sys, suppose a cache memory is 10 times faster than main memory and that can be used in 70% of the time. How much speedup we gain.

Memory

Cache memory — 70% — ⇒ $f = 70\% = 0.7$

main memory — 30% — 10 times — $f = 10\% = 0.1$, $s = 10$

⇒ $s = 10$ times

37 / 100 / 74 / 26

$$S_{overall} = \frac{1}{(1-f) + \frac{f}{s}} = \frac{1}{(1 - \frac{70}{100}) + \frac{70}{100}}$$

$$\Rightarrow \frac{1}{(1-0.7) + 0.07} = \frac{1}{0.37} = \frac{10}{0.37} = 2.71$$

1.0 / 0.7 / 0.3 / 0.07 / 0.3 / 0.37

Date :
Page No. :

Ques → 3 enhancements with following specifications are proposed for new architecture if it is $S_1 = 30$, $S_2 = 20$, $S_3 = 15$. If enhancements 1 and 2 are each usable for 25% of the time what fraction of the time enhancement 3 will be used to achieve overall speedup of 10.

3 enhancements

$$S_{overall} = \left[1 - (F_1 + F_2 + F_3) + \frac{F_1}{S_1} + \frac{F_2}{S_2} + \frac{F_3}{S_3}\right]^{-1}$$

$$10 \Rightarrow \left[1 + \left[\frac{25}{100} + \frac{25}{100} + F_3\right] + \frac{\frac{25}{100}}{30} + \frac{\frac{25}{100}}{20} + \frac{F_3}{S_3}\right]^{-1}$$

$$\Rightarrow \left[1 + \left[0.25 + 0.25 + F_3\right] + \frac{25}{30} + \frac{25}{20} + \frac{F_3}{15}\right]^{-1}$$

$$10 \Rightarrow \left[1 + \left[0.50 + F_3\right] + 0.8 + 1.25 + \frac{1}{15}F_3\right]^{-1}$$

$$10 \Rightarrow \left[1 + \left[0.50 + F_3\right] + 0.8 + 1.25 + 0.06F_3\right]^{-1}$$

$$1 + \left[0.50 + F_3\right] + 2.05 +$$

$$10 \Rightarrow \left[3.55 + 1.06F_3\right]^{-1}$$

$$\frac{1}{10} \Rightarrow 3.55 + 1.06F_3$$

$$0.1 = 3.55 + 1.06F_3$$

$$0.1 - 3.55 = 1.06F_3$$

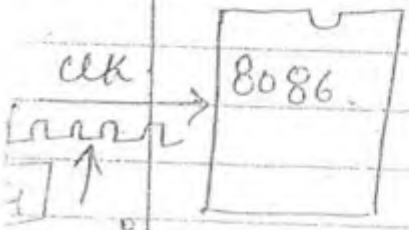
$$F_3 \Rightarrow 0.45$$

$$\Rightarrow 45\%$$

CPU Execution time Calculation :-

→ Each and every processor is connected with common clock which runs at constant speed.

→ $T_1 = 1ns$ $T_2 = 2ns$ } Its not constant running clock.



↳ Clock pin is also known as I/P pin some info is carried over to processor

Clock Generator :- Clock pin is connected with C.G.

The O/P of C.G are time states.

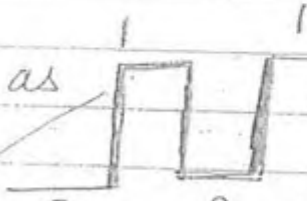
Clock generator

→ The entire 8086 is controlled by clock. Clock generator must run at constant speed. Then only time states will be having unique value.

Eg → For every operation, 4 ts are involved.

$t_1 t_2 t_3 t_4$ must be common ie 1ns

Time state is defined as



to rising edge and 010 ie falling edge to falling edge

rising edge.

Date:
Page No.:

Edge triggering signals → When the signal is activated based on time state ie 101 ✓

Level triggering signals

Active High →
Active low →

The signals which are activated based on these transitions called as edge triggered signals.

* Clock / tick / clock tick / cycle / clock cycle / clock period → 101 / 010 ✓.

* Based on clock we are calculating execution time reqd by a computer

CPU Execution time = program Execution time
= no of seconds per program.

Exec. time ⇒ Seconds / Program

Program ⇒ no of inst^n p. → To execute one inst^n → how many clock cycles reqd

CPU Time ⇒ Seconds / Prog ⇒ Inst^n / Prog × Cycles / Inst^n × Seconds / cycles.

CPU Execution time = no of clock cycles reqd × clock cycle Time [time is given].

Or.

time is not given, frequency is given.

CPU Time ⇒ $\frac{\text{no of clock cycle reqd} \times \text{clock cycle time}}{\text{rate}}$.

If we know the no of clock cycles, instn count, we can calculate average no of clock cycles per instruction.

CPI ⇒ $\frac{\text{no of clock cycle reqd}}{\text{Instruction count}}$

Under that condn no of clock cycles reqd

⇒ CPI × IC

Substitute no of clock cycles reqd for the program value * cpu time

CPU Time ⇒ CPI * IC × clock cycle time

⇒ $\frac{\text{CPI} \times \text{IC}}{\text{rate}}$.

Each and every instn will be having their own clock cycle.

CPU time = $\sum_i CPI_i * IC_i * cc\,time$

Or

$= \sum_i \frac{CPI_i * IC_i}{rate}$

Date:
Page No.:

i → Data transfer instⁿ, Data manipulation Transfer of control.

Instruct^n set

A complete processor supports with 3 categories of Instⁿ ie

① Data transfer Instⁿ
② Data manipulation Instⁿ.
③ Transfer of control Instⁿ.

} more convention Instⁿ

Data transfer Instⁿ :→ while execution of these instructions, data is transferred from source to Destination w/o change.

we need to identify source of addresses, How many places the data is present.

The data is present in 2 places :→

① processor register
② Memory.

} → we need addressing modes. It shows where the data is present to operate Instⁿs.

→ multiple locations.

→ Source is either Register or memory.
Destination
If its read opratⁿ → Destⁿ → register
If its write opration → Destⁿ → memory.

To execute these instructions register & memory acts as a source and Destination based on operation nature.

If operation is read, register is Destination and if it is write, ~~register~~ memory is Destination

rule

→ The constraint of data transfer instruction is source may or may not have storage. But destination is always have storage

immediate value

D S

Eg MOV #23, AX ✗

constant

Mnemonic indicates type of operation

Mnemonic followed by dest^n, dest^n & source.

→ constant does not have storage function ality

→ Register → storage functionality.

D S ✓

MOV AX, (#23) ✓

register

source may or may not have storage

immediate value

Dest^n storage

Data:
Page No.:

Data transfer Inst^n p are divided into 2 types :→

✓ ① Register to Register transfer operations (MOV)

✓ ② Memory to Register " "

✓ ③ Register to memory " "

① Mov → Data transfer Inst^n ~~MOV~~ R0 R2

② MOV R0 #2345 ✓ (reverse is not valid)
Dest^n need to have storage
Storage constant (Immediate Value).

③ Point :→ 8086 µp allows move Inst^n to perform data transfer operation from register to register & memory also.
Dest^n
MOV R0, 2345 ✓ (register) [read operat^n]
If any no is not followed by # it is treated as address.
(Address of memory).

MOV R0, 2345
address

MOV 2345, R0. ✓ [Dest^n memory] [write operat^n]
address
of memory (storage f^{n}ality).

Registers are present in m

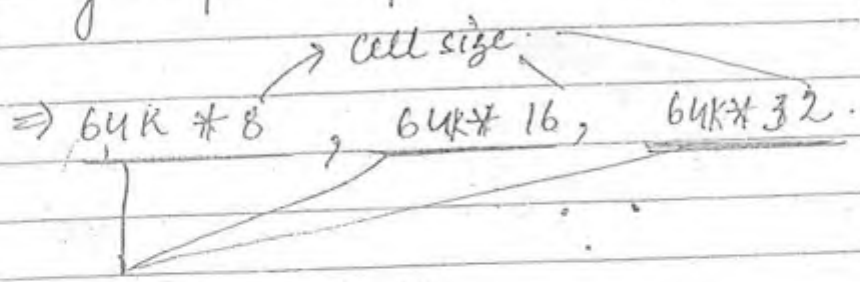
- Memory to memory transfer operation is not allowed There is no use.
- Memory refrence data transfer operations are :→
  1. read and write

memory is divided into equal parts it known as cell
Cell consists of unique no is known as address.

- Memory chip is divided into equal sized parts called cells.
- Each cell is identified with unique no called as address.
- Each cell understands 2 control signals.
  1. Read & write
- To acces any memory cell, send address along with control signal.

Date:
Page No.:

Memory chip is represented as following notation i.e



Total capacity of chip.

→ By using this capacity, we came to know about no of cells. It itself gives how many address lines are reqd to enable 1 cell. We enable to identify address space of that chip.

$2^{6} \times 2^{10}$
2^{16}

Total no of cells = $2^6 \times 2^{10} \Rightarrow 2^{16}$

no of address lines = 16.

Address space → min val to max val.

→ $(0000\ 0000\ 0000\ 0000)_2$ to $(1111\ 1111\ 1111\ 1111)_2$

→ Each and every memory chip will be having their own address space.

→ Memory addresses are represented in Hexadecimal no sys. because of easy recognization memory is also important.

0000 0000 0000 0000 → 0000 → Each digit is equivalent to 4 binary bits.

1111 1111 1111 1111 → FFFF

→ The 2nd parameter indicates cell size:-

Default cell size = 8 bits

64KB → is valid K = 16, K = 32

64KW ↑ it is only valid, if there is binding with 16, 32 & so on...

→ If memory address points 8 bit memory cells called as BYTE ADDRESSABLE

→ If memory address points the cell size based on the processor word length. Called as WORD ADDRESSABLE

Suppose 16 bit processor, 32 " "] word length = 16, 32.

→ Memory address interpretation methods.

Eg 8086 whose word length is 16 bits.

Instn.

MOV AX 2000

↓ ↓ 8 bits

register size = 16 bits

16 bit processor

Right Byte ↑	23	2000
2000 → 23 → RB	45	2001
2001 → 45 → RB		2002

ambiguity

AX = 2345 ✓

AX = 4523

Byte addressable memory.

↳ we identify through 2 types as follow

Date :
Page No. :

Memory Interpretation are of 2 types :→

① Little endian.
② Big endian.

① Little endian

| 3 | 2 | 1 | 0 | (address)
| 3 | 2 | 1 | 0 | bytes
(low to high).

low address consists of low byte
High " " " High "

8086 allows Ist method, accessed 2 locations.

low (2000) – 23 → LB.
High (2001) – 45 → HB

8 bit = 1 byte
16 bit = 1 word = 2 byte

LB HB.
AX = 23 | 45
AX = (HB) 45 | 23 (LB)

→ 32 bit processor we need to access 4 location.

| 23 | 2000.
| 45 | 2001
| 28 | 2002
| 64 | 2003

reg = 64 28 45 23. ✓

② Big endian method :-

| 4 | 2 | 1 | 0 |
| 1 | 2 | 3 | 4 |

low address → Higher Byte
High address → lower byte

The default Memory address Interpretation method LITTLE ENDIAN

28/12/2010

RISC processor

In Risc processor, memory to register, memory to register, Data transfer instructions are implemented with load and store Instr's respectively.

(Memory → register)

Load - read operatn. mem → reg (I/P)

Store → write Operatn. reg → mem (O/P).

After enabling the ALU, ALU is waiting for data it present in register. Data is transferred through Data transfer instr's.

Eg Load r0, [2000] → \contents.

memory content whose address in 2000 is transferred to r0.

Eg Store r1, [2000] register → memory

S D

Note → In the RISC, store instr^n is followed by source But all other instr^n is followed by Destination

* Except store instr^n all the instr^n's are followed by Destination

Date :
Page No. :

Data Manipulation Instⁿ :
While execution of these instructions, the data undergoes change. These are divided into 2 types.

① Bit manipulation Instrn's
② Byte manipulation Instrn's

Bit manipulation Instrⁿ : only one bit undergoes change.

① Instrⁿs : Set, Clear.

Set C → carry flag is equivalent to 1 while executing this Instⁿ.

Set b

Set b Psw, ② → Set bit → register

or

Set b r0, 2 → Position undergoes change
accepting as Set bit. I/P register

r0 = 2311
⇒ 00100011
7 0
00100111

Bit manipulation change

Set / Clr

② Clr b r0, ①

7 6 5 4 3 2 1 0
00100011
↓ Change.
00100001

✓ This Type of Instructions are useful in setting or resetting command word register.

✓ Command word register each bit is having their own meaning.

If bit = 1 → Set → Rotate priority is reflected
If bit = 0 → Reset → fixed priority is reflected.

(2.) Byte manipulation

These instructions are divided into 3 types

1. Arthimetic Instⁿ
2. Logical Instⁿ
3. Shift & Rotate Instⁿ

A. One form of I/P is accepted & other form of O/P is generated based upon Instⁿ

(1) Arthimetic Instⁿ : Arthimetic Instⁿ are used to perform mathematical models simulation

mathical formula / solⁿ.

These Instⁿs are divided into several types :→

1. ADD
   SUB
   MUL
   DIV
   INC
   DEC

(2.) Logical Instⁿ : To enable or dipable the internal components of the processor logical Instⁿs are used.

Level Triggered Signals are using logical Instⁿs.

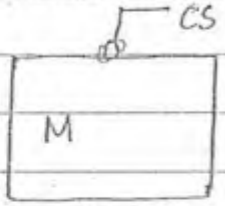
↓

Based on pin value, we are having 2 values i.e high / low. Some pin are activated with high or low.

Date:
Page No.:

select pin

Eg: If chip is in enable state, then memory is active.



Eg: Nand gates are communicating with chip select pin

* Logical operatns are used to enable / disable the internal components of processor ie memory.

If any one of the I/P of Nand gate is 0, the o/p is high.

* Level Triggered signals.

2 types.

Active High Signal

Eg: (ALE)
(Address Latch Enable).
(Pin name).
It is connected with latch.
Pin without bar → Active high.

Active low Signal

Pin name with bar
It indicates Active low

Eg: $\overline{CS}$

Logical Instns: AND, OR, NOT, XOR, CMP
compare

(3) Shift & Rotate Instn

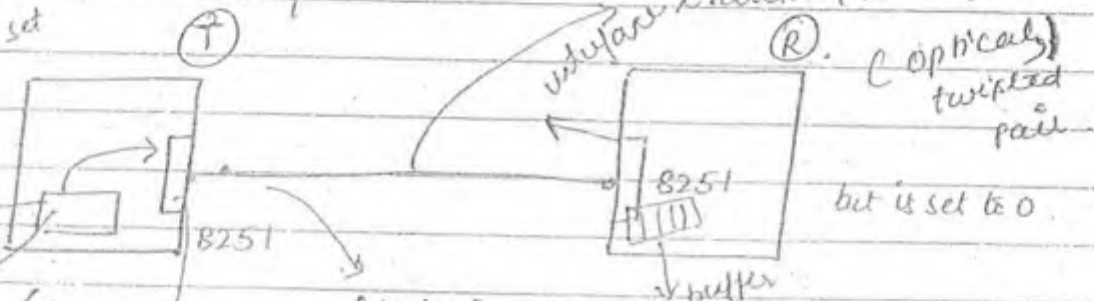
Shift Instn: This Instruction is used in serial data communication. Serial

Data communicat^n means bit by bit transmission. It is supported (suitable) when the distance is large between the nodes. (source node or dest^n node)

(2) 8251 USART (Universal synchronous Asynchronous Reciever & Transmitter).

→ It is an interface to perform Serial Data Communicat^n.

→ How it is executed?



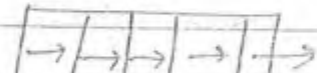
→ It consists internal buffer

Buffer is used to hold data.

Based on buffer status we can conclude the transmission is over or not?

Once the buffer is full, transmission takes place.

Shift register are used for serial data transmission.



but by bit transmission

When sender buffer gets empty, Stop transmission

one node ... bit in the command word register. ... is full, start the transmission of LSB of buffer is placed on ... register.

Shift operations are divided into 2 types :→

rest of the bits are shift towards right one bit position. Repeat the same operatn until buffer becomes Empty.

↓ Two Types.

Logical Shift

Arithmetic Shift

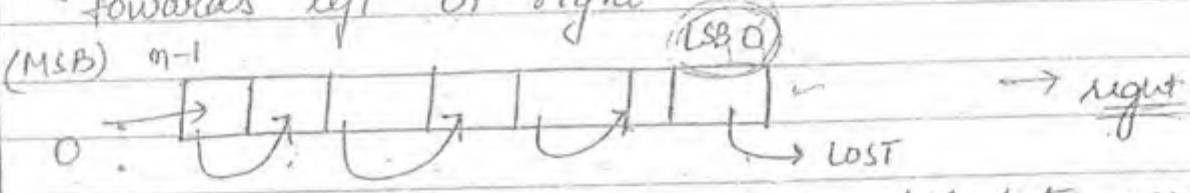
Logical shift right

Logical shift left

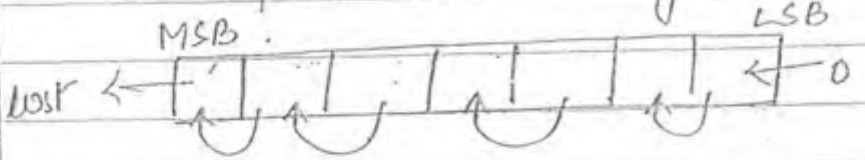
~~Logical~~ Arithmetic Shift left

Arithmetic shift right

→ Logical Shift operatn → Sign bit is shifted either towards left or right.



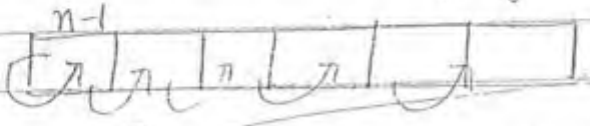
Logical shift right → LSB is lost, 0 is appended. All bits are moves towards right direction



Logical shift left

Serial data Commn → Logical shift operatn is used.

→ Arithmetic Shift operatn → preserve the sign bit, we are not changing the position.



No new 0 is added.

Before & After sign is same

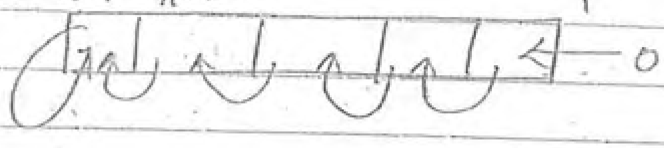
$R_0 =$ [0]1111011 → lost ⇒ ASR → [0]0111101

↓ it copies the value

(Shift operatn)
Common property of Shifting → loss of data
lost

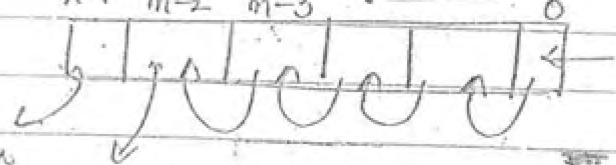
Shift left.

n-1 n-2



If its n bit register;

n-1 n-2 n-3 0



n-1 remain same

n-2 is lost. If n-2 is moved towards n-1 then sign bit is lost.

lost

γ_0 = 1 0 1 1 0 1 1 1 ← 0.

= 1 1 1 0 1 1 1 0 → Here we preserve sign

→ Arithmetic Shift operatns are used in Sign no's multiplicatn.

Rotate

Conclusion of shift operatn

*[After certain operatn, the original Data is lost] *

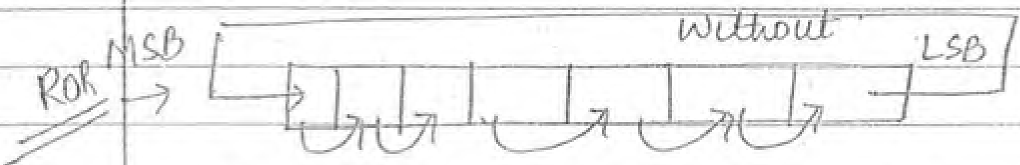
* Use / Significance :→ Rotate instrns are used to perform optimizations in the program.

Date :
Page No. :

Rotate Instr^ns are divided into 2 types

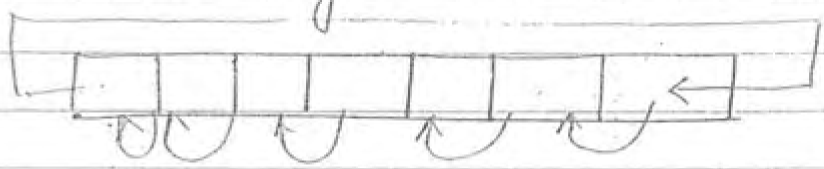
- Rotate with carry (ROL, RCR — carry)
  - Left
  - Right
- Rotate w/o carry (ROR, ROL — w/o carry)
  - Left
  - Right

→ Rotate w/o carry → ROR
→ ROL
↓
without



[LSB → MSB]
01101 ⇒ ro.
⇒ 10110

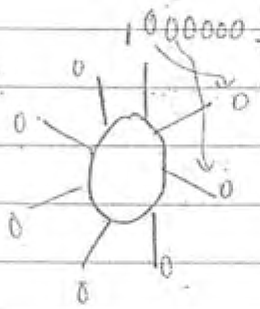
→ No new data entered onto register, only position's are changed.



[MSB → LSB]
r_c = 11011
10111

ROL

Motor



Stepper motor

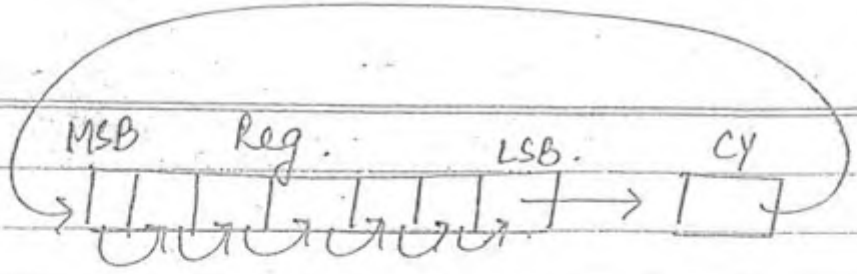
anticlockwise ⇒ rotate RoL
Clockwise → RoR

→ Stepper motor Implementation :→

(Book AK Ray)

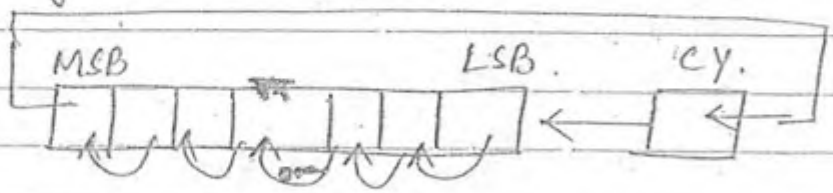
Rotate operation are meant for optimization. In stepper motor interface implementation with 8086, RoL and RoR instr^ns are used to perform clockwise & anticlockwise ~~moving~~ movements.

RCR



Rotate right along with carry. LSB is transferred to carry & carry moved to LSB. Rest of bits are moving towards right.

RCL



These type of instn's are used in

① to identify the no is +ve or -ve. In RCL. After MSB shifted to CY. If CY is set the no is -ve otherwise if it is reset then no is +ve.

RCL →

② to identify the no is even or odd.

RCR

LSB bit is always 1 if no is odd.
LSB bit is always 0 if no is even.

Mem

0	0	0 even
1	0	1
2	1	0 odd
3	1	1
4		
5		
6		
7		

Jmp 2004 — Jump to Target address

Date:
Page No.:

Jmp concept

Transfer of Control Instructions.

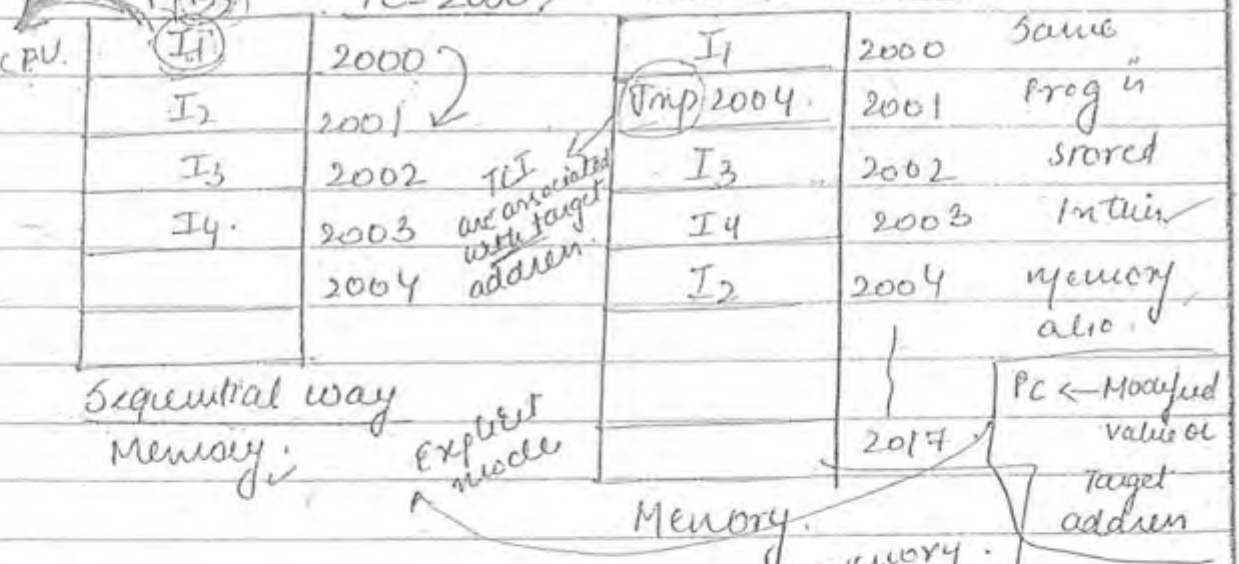
→ While execution of these instructions, the program execution sequence is changed. Every transfer of control Instⁿ is associated with target address.

↓ Program counter

Program execution

* → The program execution sequence is controlled by program counter register. Program counter points starting instⁿ address & immediately it is pointing to nxt instⁿ address

PC = 2000 / Random locations



Assume program consists of 4 Instns

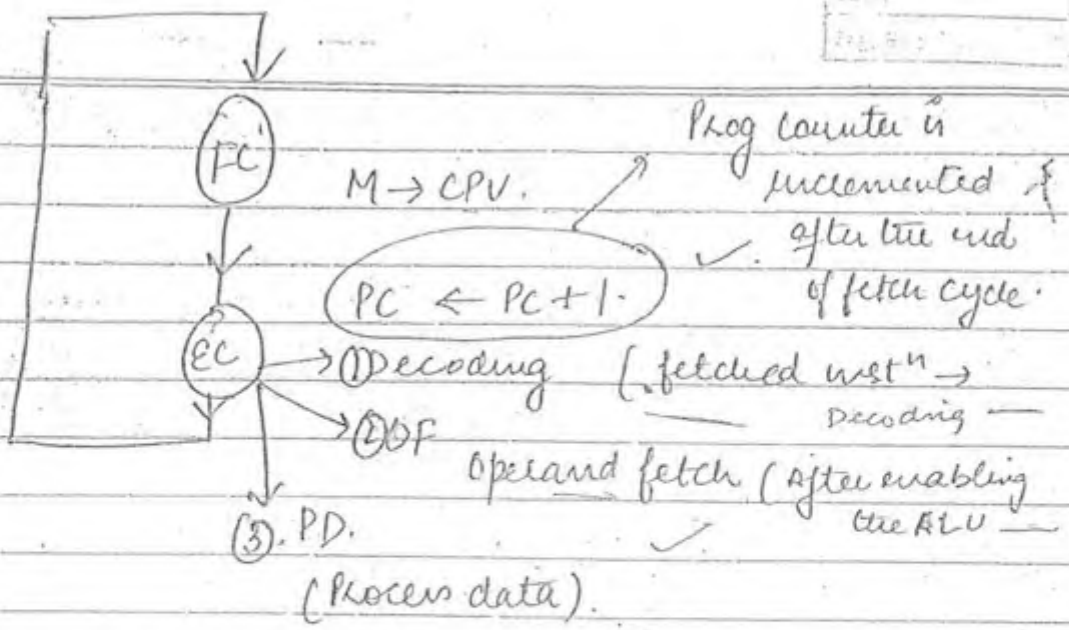
(PC+1) → Implied mode

Once the instⁿ I_1 is transferred to CPU → Fetch cycle. At the end of fetch cycle or Before executing the I_1, the program counter points to nxt address (updated)

IC → When Instⁿ are stored in random way / locatⁿs of memory.

FC, EC

Transfer of control Instⁿs are needed to load the successive address into Program counter.



Note Points:

① Program counter is operated under 2 modes:-

① Implicit mode ✓ * PC ← PC + Step size

② Explicit mode ✓ * PC ← modified value.

Step size → Memory-byte addressable } Memory location 2 byte

Inst^n is 16 bits

Here memory locat^n reqd to store inst^n is 2 byte

Here step size = 2 byte

Program counter is incremented with step size 1, 2, 3, 4 --- so on.

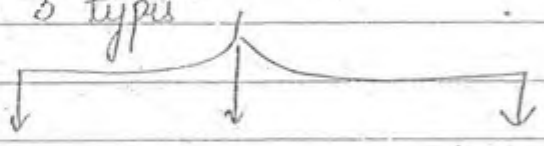
① → PC is operated under implicit mode where all the inst^ns are stored sequential memory locations

② PC is operated under explicit mode where the inst^ns are stored in random locat^ns of memory

Date:
Page No.:

→ Under the 2nd mode of operation, there is a need of transfer of control instructions to load the modified value into PC

→ Transfer of control operations are divided into 3 types



Branch Jump SKIP

These transfer of control operations again classified into

① Unconditional ✓
② Conditional ✓

→ Unconditional transfer of control while executing these instructions, program counter is loaded with target address w/o checking any condn.

Eg

Uncond^nal → (B) 2000 H PC = 2000.
JMP 3000 H PC = 3000.
SKIP 2050 H PC ⇒ 2050.

TA (next address instⁿ).

CALL 3500 H R ⇒ 3500

→ Conditional transfer of control instructions :-
While execution of these instructions, condition undergoes evaluation. If it evaluates to true target address is loaded into PC. If it is false, nxt instⁿ is executed

JNZ 2000. Jump non zero. PC ← 2000

Taking previous instn into account we can validate true or false.

Mov CX, 0010 H	Mov CX, 01h
Dec CX 22	Dec CX, 00h.
True	False

As long condn is true program counter is loaded with target address. otherwise nxt instn is executed.

In 8051 djnz R0, 2000h

↓

decrement Jump if non zero.

No need to take previous Instn Pnts accnt before loading target address.

Subprograms

↓

to Implement Subprograms, transfer of control instn are used.

① Call

② Return.

Repeatdly appeared instrn's in the main program collect as I programs and stored it in the separate memory space.

Sort() — 5 instn.

Date:
Page No.:

250 Instⁿ :- if program language is not supported by subprogram. then
→ LOC is increase. ← disadvantage
→ Memory space Inc.
→ Time inc.
→ Cost Inc.

Characteristics of Subprogram :→

(1) Single entry point } → call → main program
Single exit point } → return → Subprogram.

(2) Main program is suspended during the Subprogram execution.

(3) Control is send back to main program after completion of Subprogram.

→ To implement subprograms, there is a need of run time stack. to store the return addresses

1000	—	2000
	—	—
1003.	— → main program	—
1004	CALL 2000	—
1005.	—	return.
⋮	—	(Subprogram)
	—	

When there is call, there is return.

Call indicates entry point of subprogram.

Unconditional transfer of control instn.

(Main prog)
1000
1003
1004 Call 2000
PC = 1005.

2000 (Subprogram)
=
=
Single return (Exit).
1005 return address is stored onto runtime stack

single entry

* When call instn is invoked / executed, it internally invokes push operation i.e prog counter value is pushed onto top of stack. PC is loaded with target address.

TOS	1005

* When there is ret instn, call instn internally invokes pop operation

POP PC TOS ⇒ 1005

1005

2000
=
=
Ret 2006
2007 ← PC
1005

There is a need of ~~return~~ runtime stack which store return address. Then we call it as Subprogram.

Call	Ret
|	|
Push	Pop
|	|
Main	sub.

Date:
Page No.:

To handle the Interrupt :→

To implement Interrupt controller concept in the processor environment. There is a need of Subprogram concept. Each interrupt related Subprogram is present in the memory to handle different Interrupts. There is a need of transfer the control from main program to subprograms while maintaining the return addresses in the runtime stack.

* The transfer control operation is divided into 2 categories :→
  1. Direct transfer of control operatn.
  2. Indirect transfer of control operatn.

(Call) → Direct transfer of control operatn :- target address is associated with mnemonic.

(Return) → Indirect transfer of control operatn :-

TOC	Uncondnal	Condnal
Direct	Call, Jump, Branch, Skip	DNZ (loops)
Indirect	Return (ret)	—

Point: Implement one control instrs are enough to get the nxt instrn address.

Numericals :→

Q1. Ques ① Consider the unpipelined (non pipelined) processor. Assume that it has 1ns clock cycle and that it uses 4 cycles for ALU operations and branches, 5 cycles for memory operations. Assume that relative frequency of these operations are 40%, 20% and 40% respectively. What is the Instn execution rate?

CPU Time = ?

Clock cycle time = 1ns.

no of cycles = 4 cycles for ALU + branches
no of cycles for memory ⇒ 5

[CPU time = $\sum$ CPI$_i$ * IC$_i$ * cc time]

CPU time ⇒ (0.4 * 4 + 0.2 * 5 + 0.4 * 5) 1ns

⇒ 4.4 ns

Page no 90 Q4

Point The components of

Date :
Page No. :

Computer Architecture

→ Computer Arc is classified into 2 types based on memory ie is single memory architecture, multilevel memory architecture.

Single memory architecture : This architecture is also known as VON NEUMANN ARCHITECHURE

This architecture is designed based on **Stored program concept.**

↓

only one memory space is used to store prgm and data.

Performance $\propto \frac{1}{\text{Access Time}}$

Access Time = low.

* Von Neumann arch → If the memory is organized properly, access time is less.

→ Stored program concept means Instn & data both are in their same memory. After performing the operatn result is also stored into the same memory. To make it successful architecture use memory management policies to separate the Instn area from data area. So. Ultimately access time is decreases. This concept is implement in 8086 µp.

(16 bit Processor, Register size = 16 bit, 1MB)

8086 → word length

↓

16 bit processor

↳ ALU register size = 16 bit

Byte addressable

Physical memory = 1MB ⇒ 1M × 8 bit

⇒ 1M X 8.

⇒ 1024 X 1024

⇒ $2^{10} \times 2^{10}$

⇒ 2^{20}

⇒ No of address lines = 20.

address space ⇒ ~~20's to~~

20 0's to 20 1's.

⇒ 5 0's. i.e 00 000

5 1's. FF FFF

2^{20} → Instn / data } Both is present

Eg

Col1
100H

No policy is maintained while using HNO (random).

Col2
10H.

policy ⇒ row – column

1	R0
11	20

M. 63 HNO want to access.

100
62

M

AT = Search time + to search AT.

AT ⇒ high.

↓

Perf = dec.

AT ⇒ AT.

There is no search time.

AT = less

↓

Perf = inc

$AT \propto \frac{1}{Perf}$

Segmentation ⇒ To organize 2^{20} address space we need segmentation.

When 2 different entities are stored in same memory locatⁿ to access them efficiently. There is a need of memory mgt policies. 8086 designer used segmentation policy ie 1MB physical memory is divided into 16 logical segments. Each segment size is of 64 KB

1MB ⇒ 16 logical segments

↓

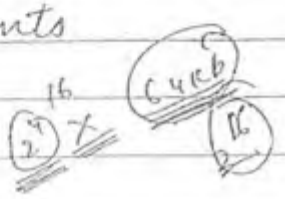
64 KB

⇒ 16 X 64 KB.

⇒ $2^4 \times 2^6 \times 2^{10}$ → 16 address line.

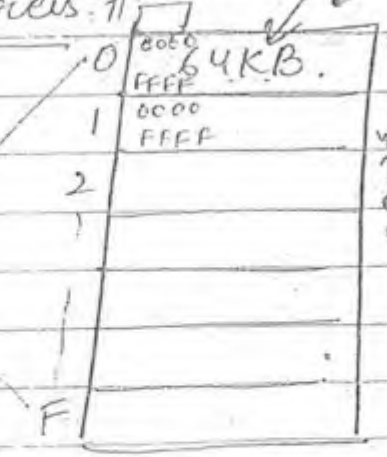
⇒ 2^{20}. (no new cell added no cell removed after segmentation)

[Same cell status is maintained).



Base address. Offset address

The outside address is known as Base address and inside address is offset address.



Two entities are stored in same memory To access each entity in efficient way we need Segmentation

8086 supports 4 segment registers called as CS (code segment register, data segment register, stack segment register, Extra segment register).

Segment registers are used to store the Base address of a segment where code is available.

(1) [CS = 0]

00000 → code
0FFFF.
↑ Base address.
↓ Instn

Code segment register is used to store the Base address of the segment where code is available.

(2) DS → is used to Base address of the segment where the data is available.

total address space.

[DS = 1] ⇒ 10000. → data
1FFFF

Point If Base address is not shared by different segment registers then it is called Non Overlapping memory organization

Date:
Page No.:

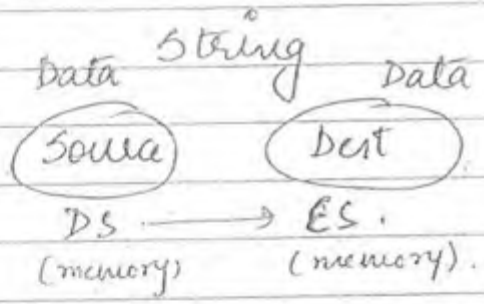
(3) Stack Segment → is a register used to store the base address of the segment where return address are available.

(4) Extra Segment register used to store the base address of the segment where the data is available

Ques Why 2 Data area is regd.

There is need of identify source address of data & Destn address of data, incase of string operatn [data xfer operatn], Extra segment is used.

↗ If there is no string operatn → No use of Extra segment



✓ with same register, 2 data won't possible

Point → In order to execute the string operatn, there is a need of extra segment used to point Destn address.

↗ All our pc's allows VON Neumann Arch. Due to this reason it run in RAM not in ROM Code memory is present in physical memory. (code Area & Data area is same).

Date: 29/Dec/2011
Page No.:

Pointer & Index register

→ The offset address of memory segments pointed by Pointer registers and index registers. Apart of ↓.

To point the offset address of the segment.

→ 8086 supports with 3 pointer registers and 2 Index registers

3 pointer registers →
- Instruction pointer register (IP)
- Base pointer register (BP)
- Stack pointer register (SP)

2 Index registers →
- Source Index Register (SI)
- Destination Index registers (DI)

Point's → Registers are referred by names or internal addresses.

→ To enable single memory location there is need of BA and offset address.

When there is a need of address of data we need BA & offset

Base	offset
CS	IP (Instⁿ)

→ Instruct^n pointer register.

Instruction pointer register holds the offset of an Instⁿ.

CS : IP

Binding: DS : BP/SI/DI

If in correct binding if operation is not string operation

Three possibility of holding address of data

(7). If there is no string operation no use of ES and

Date:
Page No.:

(5) If operation in String operat^n.

DS : SI ES : DI
Source Dest

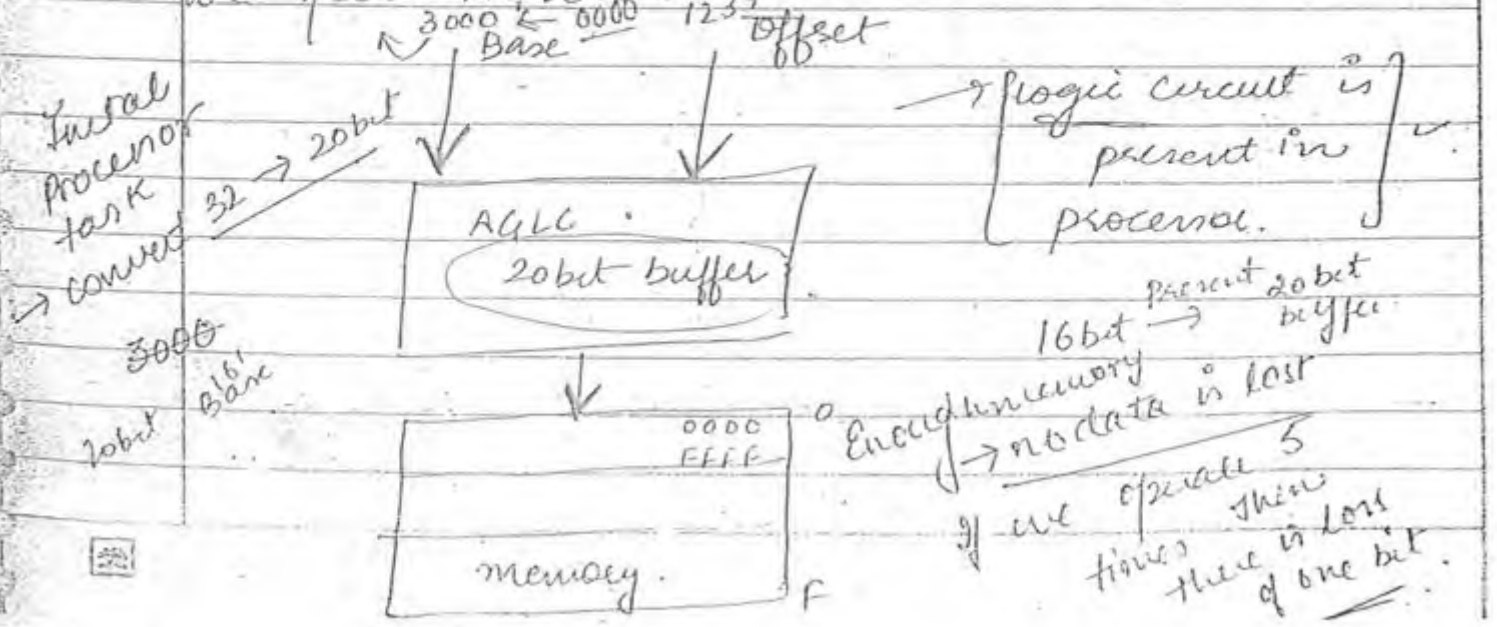
(1) a. Base pointer register, Source Index register, Dest^n Index register point the data offset. (If the operat^n is not a string operat^n)

(b) If Operation is a String Data Segment register is binded with SI to point Source address and Extra Segment is binded with DI to point the Dest^n address.

Case3 Stack Area : * Stack pointer register :

SS : SP → points / holds the offset of return addresses (Stack).

Suppose each register size is 16 bit. But internally it consists of Address generation logic circuit). As we are having a need of 20 bit address. But when we combine Base and offset ie 16 bit + 16 bit = 32 bit. To convert this 32 bit → 16 bit we need AGLC. → 4 zeros are added at LSB.



Doubt —

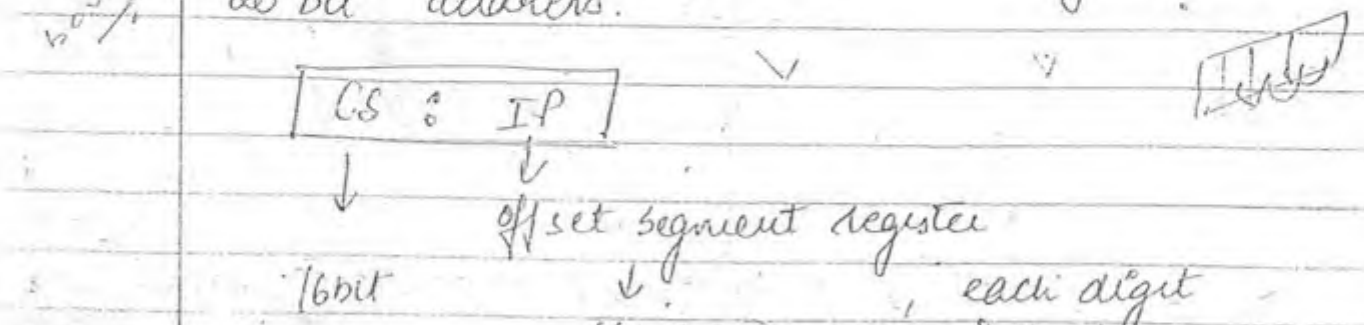
To read an Instruction or Data from the memory there is a need of Base and offset addresses to enable the memory cell.

These 2 addresses are pointed by Segment register and pointer registers. that means 32 bit address is given as I/P to the memory. Because we are having 8086 memory size is 1MB (2^{20}) There is need of Logic Circuit to convert 32 bit address into 20 bit address

That circuit is called as (AGLC) Address generation logic circuit.

It consists of 20 bit buffer It accepts the Base address as an I/P and performs Shift left operation upto 4 times

After that it accepts offset as a Input and performs OR operations with Intermediate result Then it generates 20 bit address.



AGLC

00110 0000 0000 0000 → Performs shift operatn 4 times

00110 0000 0000 0000 0000
0001 0010 0011 0100 ① OR operation is performed.

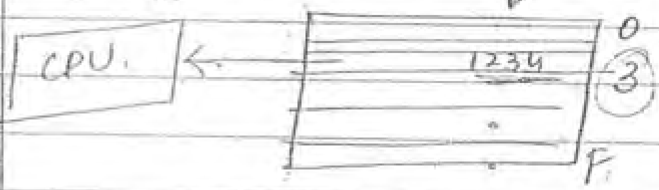
0011. 0001 0010 0011 0100 ② Logical shift operatn

(3) 1 2 3 4.

20bit intermediate regtt. → MSB of segment register respectle denotes Base address.

offset address BA 3|1234 offset adrs

20bit address is connected with memory. Memory is segmented in segmentation way.



→ BA is associated with shift register that will apply 4 shift operatn

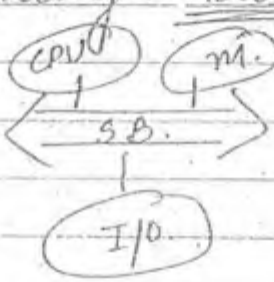
Multimemory Architecture

This architecture is also called as "HORVARD" architecture. It is also called LOAD & STORE architecture.

It consists of 2 memory chips. One is code memory & Data memory (2 memory are separately maintained in processor, one is holding instrn & other holds Data). No segmentation policy is reqd.

Need → Separation of Instrn space and data space.

Q. Howard architecture is implemented in 8051 µc (microcontroller).

Q.	Processor (8086)	Controller (8051)
(1)	µp is off chip	(1) µc is on chip processor.
computer I/O CPU memory	There is CPU, memory and I/O. They are communicated through Bus. 	There is no need of external communication. As I/O device, CPU, memory all present on the same board.

Ques: How the µc implement HORVARD Arch?

8051 → Two kind of memory chip

- RAM (data) → pointed by DPTR → Data memory
- ROM (Instⁿ's) → pointed by PC. → code memory

A. 8051 consists of 2 memory chips one is
(1) Read only memory
(2) Random access memory.

(1) ROM is pointed by Program Counter

(2) RAM is pointed by Data pointer Register

Date:
Page No.:

→ ROM is code memory whereas RAM is data memory

* User programs are not allowed to execute in BOST MC

→ In the embedded systems, BOST MC are ~~contr~~ used.

Embedded S/m's :→

→ Any device which includes programmable processor called as Embedded S/m. Any Microwave Oven (1) Accepts User I/P ✓ 23 ✓

(2) Decrements User I/P 22.

(3) JNZ L (Jump if non zero).

NZ	00
T	F

move to previous target address contain

Once the User I/P becomes zero it stops the process. (4) Stop. (Stop the heating process)

→ Data (23) is stored in RAM. As Power is off Data is lost due to RAM is Non volatile memory.

→ Code memory → ROM (code remain same, I/P data is different due to RAM).

→ VON neumann arch is suitable for PC's. and Harvard arch is suitable for... Embedded S/m's

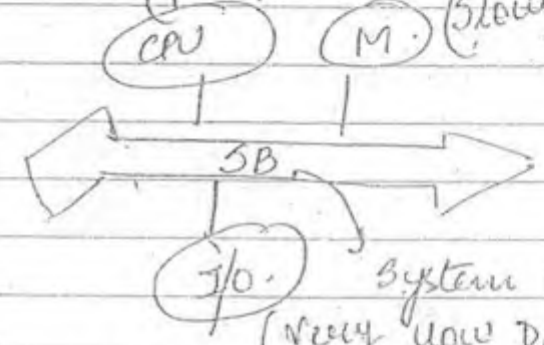
Components of Computer System

Computer sys consists of 3 components :-

1. CPU
2. Memory.
3. I/O Devices.

These 3 components are communicating each other using an efficient channel.

Efficient communicatn channel is called as sys Bus



Communicatn is reqd due to the reason that all of three acquiring different functionality such as.

1. CPU → processing ✓
2. Memory → storage of program.
3. Input/O/P → provide External communicatn
   System → user
   user → sys.

Date:
Page No.:

There is a need of communication b/w 3 components i.e SB [System Bus]

"mm" b/w fast & slow, there is Speed gap. To minimize the speed gap, we need Synchronization.

System Bus Key characteristics is

Shared transmission media i.e one transmission at a time

S/m bus control is in the hands of CPU because it is Fast device which is communicated with S/m bus.

S/m bus consists of 3 categories of lines.

① Address line ② Data line ③ Control line

① Address lines are used to carry the address to memory & I/O.

→ Address lines are Unidirectional CPU only initiates memory / I/O operation by sending the address to I/O or memory.

Based on no of address lines Total capacity of memory which is supported by the processor is determined.

Memory is controlled by address lines if we are having 20 address lines, we can have 1MB. Memory.

② Data lines are used to carry the data b/w CPU & memory & I/O Data lines are

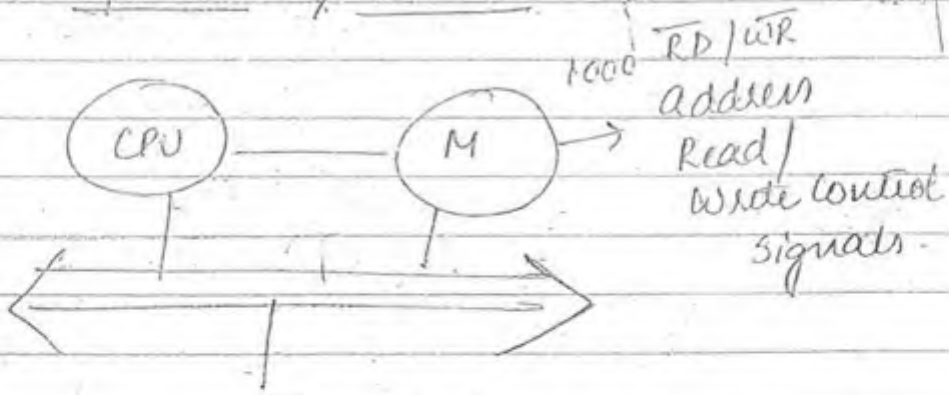
Bidirectional Based on the no of data line, the performance of processor depends.

Eg → 8 bit processor ⇒ 8 data lines.

(3) → Control lines are used to carry the control signals and timing signals.

→ Control signals are used to enable the operation and timing signals are used to provide synchronization.

→ Control lines are Bidirectional. It is used to carry read & write signals from CPU to memory or I/O devices and also carries interrupt signals from I/O devices to CPU



* Each and every I/O device is having address which is equivalent to port address.

I/O. → Port address = I/O address.

Assume (address = 1000)

RD / WR

Memory address space

≠ I/O address space

Date :
Page No. :

CPU —want communicate→ memory.

↳ need of comm" channel → [need of Address & CS]

AL → 1000
CL → $\overline{RD}$

] 1000 Address. $\overline{RD}$ – Control signal ✓] Memory.

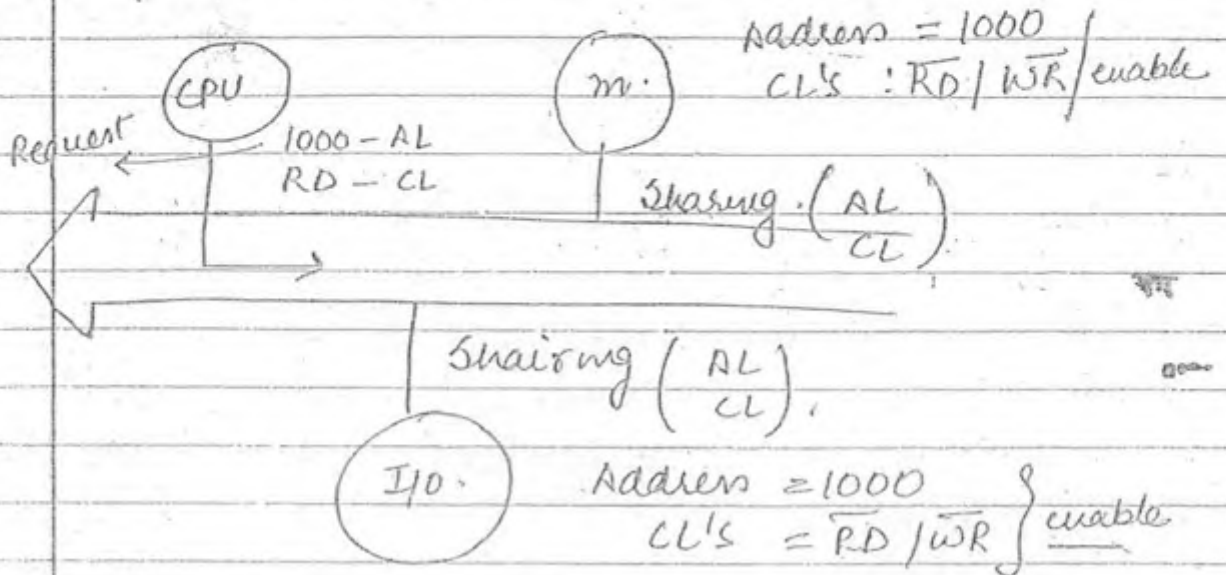
1000 – Address
$\overline{RD}$ – control signal] I/O

↓ Comparison takes place b/w memory & I/O. Same address is present in both. Thus is even both components are having same control signals. In this scenario we lead to unsuccessful operation. CPU time wasted / Performance decreases. To avoid this prob, we require to modify s/m bus.

When the CPU communicates with memory. It sends the signal requests ie address. along with the control signal is placed on the s/m bus (address lines & control lines). Assume that memory address is matching with I/O address. and control signals leads to same scenario. The characteristic of bus is shared media so memory and I/O devices access the address and control signal present on the s/m bus.

If the address and control signals are matching with memory and I/O. Two components are initiating the job at a time it is unsuccessful operation.

because of sys bus allows one transmission at a time.
The no of clock cycles are increases, the perf decreased.



"Single Requests" enables 2 devices.

Point :- To overcome this problem we can use 2 types of system bus organizations one is

① Isolated I/O.

=

② Memory mapped I/O.

Date:
Page No.:

Isolated I/O: Use the same bus but different control signals

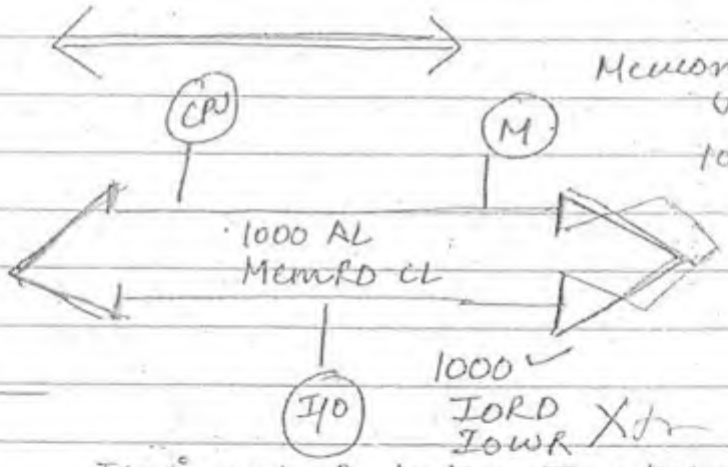
Memory related control signals.

Different CS

- Memory related control signals
  - MEMRD
  - MEMWR
- I/O
  - ~~MEMRD~~ IORD
  - ~~MEMWR~~ IOWR

Control signals are divided into 2 types

1. Memory related control signals ← MEMRD, MEMWR
2. I/O related control signals. ← IORD, IOWR



Memory is initiating the operation.

4 control signals are required

I/O is not initiating the operation

Address is matching but not control signals.

Isolated I/O mechanism is implemented in 8086 µp.

8086 consists of 3 pin

As the no of pins inc, complexity increases. µp has 3 pins instead of 4 pins.

① M/IO → Pin (Dual pin/unipin) (Active high / Active low)

↳ Set=1 ⇒ Active High enabled ⇒ memory enabled

Set=0 → I/O enabled (Active low pin)

Three Pins :→ X Both are enabled

(2) (1) M/IO ⇒ 1 1 → memory
RD ⇒ 0 0 1 → here memory read
WR ⇒ 1 0 0 operation is enabled

Mem read:
↳ we can't have - Read and write simultaneously

Memory mapped I/O

I/O
0 0
1 0
0 1 For I/O operat.
↳ Read (IORD)
↓ write (IOWR)

Points → The procesor which supports In and out ~~oper~~ Instructions That procesor Implements Isolated I/O concept

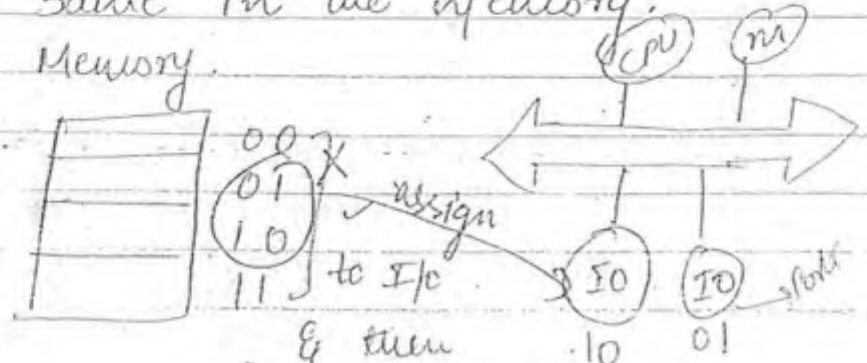
→ IN out related to I/O device.
↳ when there are executed, memory is unaffected.

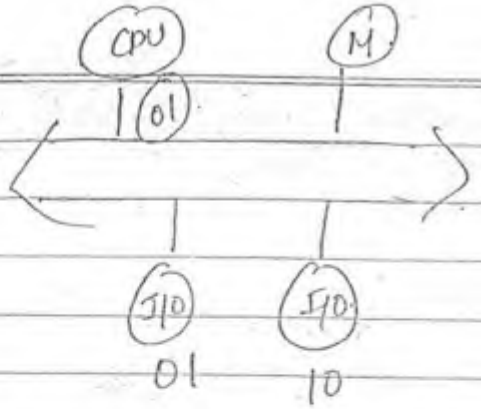
→ Isolated I/O is also known as I/O mapped I/O

→ It consists of single bus and some control signals.

Memory → Take some addresses from the pool of memory address space and assign those addresses to I/O devices and disable the same in the memory.

Memory.





I/O (01) is matched with CPU I/O (01) enabled but I/O

→ Even we have efficient memory space, we can't utilized efficiently bcoz of I/O devices are sharing the memory address space

→ 8051 we implements memory mapped concept

Central Processing Unit :

CPU consists of three components one is
1. register
2. ALU
3. control unit.

→ Register :- functionality is storage.

↳ Why it is not connected with sys bus?
As we have seen in previous scenario all memory are connected via sys bus. But register is not connected with sys bus.

CPU → execution of Prog → Prog → memory
(communication)
To get an ans" → we need sys bus

After the enabling the ~~data~~ ALU, we are accessing the data through memory. Access is High

To reduce the Acces time, There is need of register inside CPU

Design constraint

Ques Why registers are present in CPU ?

Ans (1) When the CPU executes the programs, always, communicatn takes place b/w memory and CPU.

(2) If the communication takes place, b/w the fast component and slow component. There is a need of synchronization to minimize the speed gap.

(3) under that condition, memory hierarchy is the synchronization mechanism used b/w CPU and memory.

Memory Hierarchy

Gathering all supported memories and arrange them on some priority basis.

Objective :→ (1) Synchronization.

(2) Balancing the Bandwidth. b/w CPU & memory.

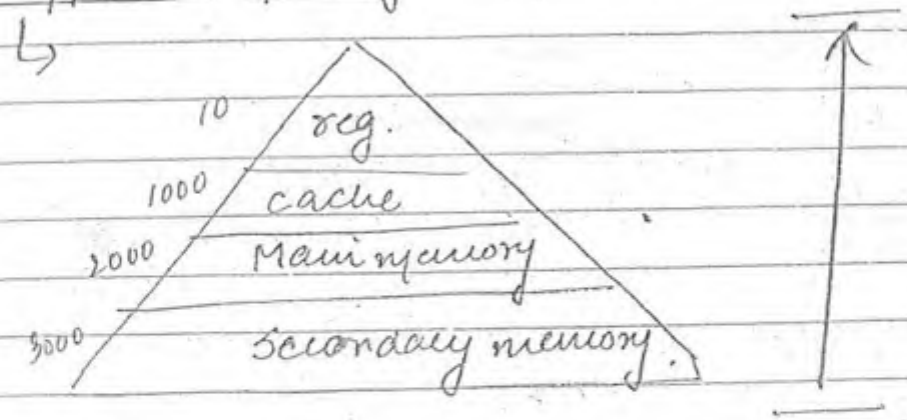
↓

Data transfer rate.

no of bits / sec

Date:
Page No.:

System Supported memory :-



Follow Bottom to top approach of memory Hierarchy.

① Capacity Decreases.

② Access time Decreases → Perf ∝ 1/Acc time.

→ Performance Increases.

→ cost per bit Increases

→ Instead of accessing data from memory, imp (valuable) data is stored in registers

Inst in I fetch on memory

→ Registers are used to store frequently used data in it.

→ Registers are expensive than cache due to security.

→ There is no processor w/o register. The mandatory register set is reqd for the functionality of S/m

Registers are used to store the data which is referenced by CPU frequently.

→ Each and every processor consists of a minimum set of registers

Mandatory register is as follows :-

① Program Counter

↓

Mandatory register

→ PC points starting instn address. and immediately it points to the next instr^{n} address.

→ ② Register ;)

③ Instruction register :

It holds the currently fetched instruction for the purpose of DECODE

register

because IR is predefined by Instruction format.

Inst^{n} address

PC → M[ ]

↓

| opcode | AD |

IR

Info about Data (Data is placed)

Decoder

↓ D·S

ALU

If there is no IR, no one can inform about opcode to this register because there is need of IR

Date:
Page No.:

ADDRESSING MODES

→ data related addressing modes.
→ Instⁿ related addr modes

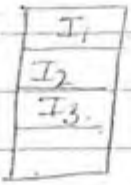
→ Shows the way where the required (object) is present

(object) → Instruction, Data

Addressing Modes are classified in 2 types.

(1) Sequential control flow Addressing modes (PC ← PC+1) (Implied mode)
(Data related addressing mode)

(2) Transfer of Control flow addressing modes



PC itself take care about adders.

Emphasis → How to get reqd data not on how to fetch Instⁿ

(1) Sequential control flow → concentrates on data not on Instⁿ.

→ Because the process of execution of Sequential Instructions called as Straight line Screening (no need of zig zaging, only straight path).

(2) Transfer of Control flow addressing modes.

→ focus on Instructions because of executing the program from random locations

Types of Sequential Control flow.

Addressing modes

Q These type of Instⁿs are divided into subtypes

(1) Register based Addressing modes.

(2) Memory Based Addressing modes.

(1) REGISTER BASED A.M :- only one Addressing mode is implemented is called as Register Addressing mode.

gives.

It (shows the) register name where the data is present.

Eg ① MOV AX, BX → Data is present in reg with name (BX)

register (D) register (S). EA = BX ✓

Eg ② MOV r_0, r_1 EA = r_1 ✓

D S

EA ⇒ register name.

Effective address :- The actual address where

Date :
Page No. :

1 Arithmetic computation required, 1 register reference required, 1 memory reference is req'd to get the data from the memory.

* Autodecrementing & Autoincrement Indexed Addressing Mode

Autodec / Auto inc] displacement is always constant ie 1

MOV r0, (r2)+ → Post inc (Auto)

Auto-increment ✓

EA = 0 + [r2]

EA = [r2] + 1

MOV r0, (r2)+ Post inc

ER2 0+[r2].

Preincrement / Predec :

MOV r0, +(r2)

EA = 1 + [r2] → First displacement is added, then we refer data.

→ This addressing modes are implement the loops ie 2 types :

(1) preincrement & Post dec

(2) predecrement Post inc

Some of the processors also introduced extra addressing modes called as Scaled addressing modes.

8086 allow scaled addressing

Scaled addressing modes means perform some arithmetic computations b/w the registers to calculate the effective address.

Eg

MOV AX, [SI] [DI]

EA = [SI] + [DI]

MOV AX, [BP] [DI]. Scaled means we need to perform arithmetic operation to calculate EA

EA = [BP] [DI]. displacement address.

MOV AX, [BP] [DI] 600

↓ Base index ↓ Relative

Eg MOV AX, [BP] [DI] 600 → displacement

↑ Base addres → Indexed addres

Effective addr ⇒ Base + Indexed + displacement

2 times

⇒ only scaled Addressing Mode's is supported by 8086 up: NO other processor supports.

⇒ Transfer of control flow Addressing modes.

↓ PC relative addressing mode.

⇒ is used in transfer of control flow addressing modes. In this addressing modes. the address of the successive instruction is transferred into program counter.

Eg JMP 2000 prog counter is automatically loaded with 2000,

PC ← 2000

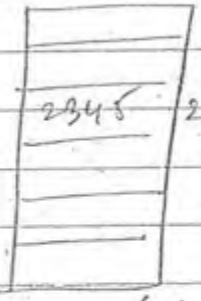
Date:
Page No.:

Effective address = Contents of reg. ⇒ [reg]

Eg reg = 2000

mov R0, (reg)

↑ Indicate Indirect Addressing mode (Memory)



mov R0, (reg) ⇒ 2 Sq Bracket ↓ Indicate Contents

mov R0, [[reg]]

[reg] → 2000

[[reg]] → R0 → 2345

↓ Content of register = EA

Auto → access the memory to get data).

1 reg reference to get the EA.

1 memory reference to get the data.

(4) Memory indirect Addressing mode :→

The effective address is present in the address of operand field.

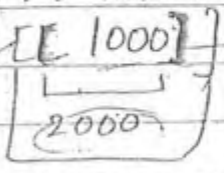
Eg MOV R0, (1000) → memory address where our effective address is present

↓ get EA

MOV R0, [[1000]]

[[1000]] again accessing to get data

EA was 2000



→ Contents of EA

R0 ← 2345

Note 1 Memory reference is reqd to read the effective address. and another memory reference is reqd to read the data.

Total memory ref = 2

→ Register Indirect is better than Mem

30/Dec/2010

Indexed Addressing

This addressing mode is used to implement the ARRAYS.

In this mode one register act as indexed register is denoted by (Ri)

Effective address = constant (+) [RL]

- constant → Displacement
- [RL] → Base address of an Array

Eg MOV R0, 2(R2) $R_2 = 1000$

2 Sq brackets.

constant

Memory reference

⇒ [2 + 1000]

EA = 2 + 1000 = 1002

Data = [EA] = [1002]

	1000
	1001
2345	1002

$R_2 = 1000$

How these addressing modes are implemented?

Implementation of addressing modes

Addressing modes are implemented in 2 ways:-

① one is implicit mode

② Explicit mode.

Implicit Mode Symbols are binded with mnemonics

Eg # ⇒ Anything followed by # is constant

It is only suitable when there are limited no of addressing modes are supported.

Otherwise Explicit mode is required if there are large no of addr modes.

MOV r0, # 2345
(# 2345 = data)

MOV r0, r1 — binded with regname. The processor automatically identify that data is present in (r1)

direct

MOV r0, 2000 (no # symbol, 2000 is memory address).

(data is present in memory.
• read the data & move to r0).

Indirect

(@) → indirect, min 2 reg are required.

1 reg → regname.
1 reg → other than value access the memory.

previous ref

MOV r0 (@) r1

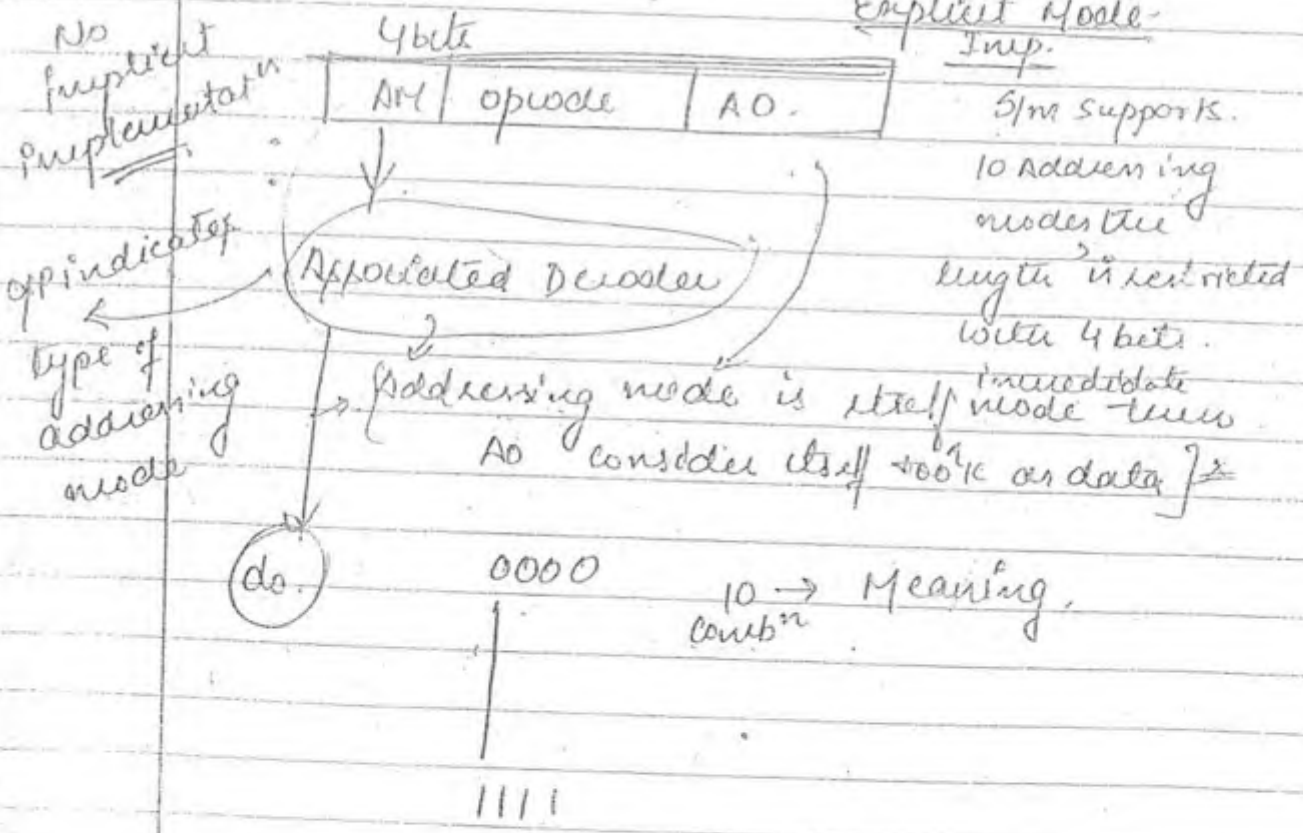
2 memory ref

MOV r0, (@) 2000

If the s/m supports with min set of addressing modes, Implicit implementation is enough to decode the type of addressing mode

So, instr^{n} format consists of only 2 fields one is opcode and other is addres of operand (No need of decoders to decode the type of Addressing mode).

Explicit Mode: There is a need of extra field need in the instr^{n} format to enable the type of Addressing mode.



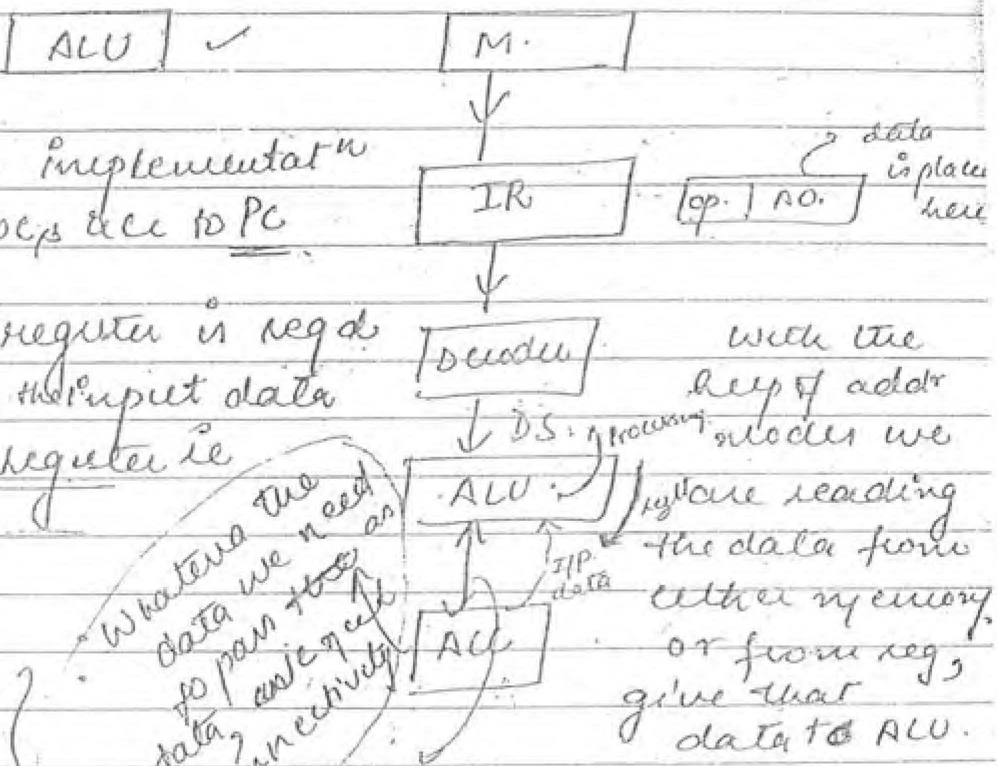
Here mapping of addressing modes are there so we can say here explicit mode implementation need

2 → IR
3 → ACC

Date :-
Page No. :

(3) Accumulator register

Accumulator is used to store the input data and o/p data of ALU



Here this implementatⁿ undergoes ACC to PC.

ACC: One register is reqd to pass the input data to ALU we need register ie ACC

With the help of addr modes we are reading the data from either memory or from reg, give that data to ALU.

Whatever the data we need to pass to ALU as I/P data, ACC is acquired the connectivity

Default register ie Acc is always required to store intermediate results.

One direction is providing the I/P data & other directⁿ is providing the result

register which has directly communication with ALU is known as Acc

MOV ACC, 40.

Date :
Page No.

① Memory Address Register / Memory Buffer register

↓ MAR   ↓ MBR

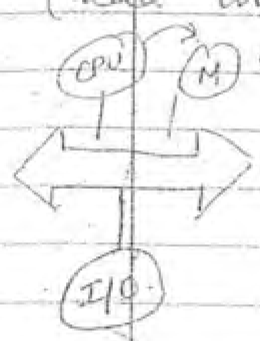
These registers carries address and Data. resp.

→ (MAR) → memory adr register is connected with address lines of S/m bus.

→ (MBR) → memory buffer Register are connected with Data line of S/m bus.

total seq of Prog Exec

Draw the sequence Diagram for read operatⁿ (to read an instⁿ from memory).



CPU want to communicate with Memory & I/O. There is a need of Partially transfer the address into MAR being it is connected with S/m address bus.

[PC] → MAR

$\overline{RD}$ → cl's of SB.

Memory mapped I/o (same add & same signals).

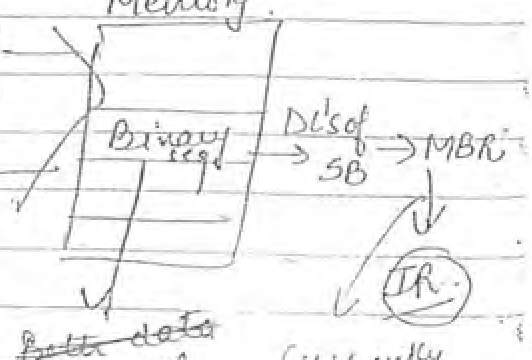
Memory is enabled to all PC

(instⁿ address) PC value transfered to MAR

[PC] → MAR → Addʳess lines of SB. (instⁿ address)

$\overline{RD}$ → Control lines of SB

Memory



Request multi include ~~first~~ address & control signal

Memory is enabled acc to PC

PC used to access Instⁿ address.

~~Both data~~ & Instⁿ

Currently ~~fetch~~ fetched instⁿ transfer to IR.

Date:
Page No.:

PC ← PC + Step size

When the source is PC, the destn is always IR

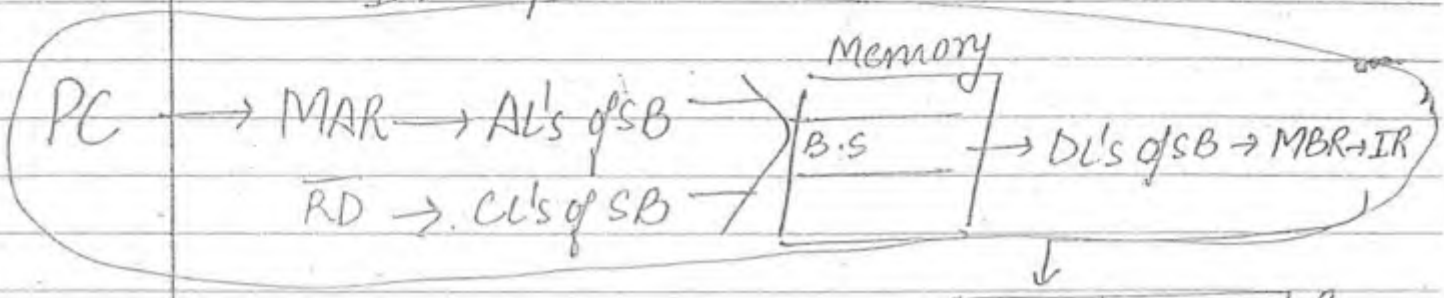
This process is called as fetch cycle

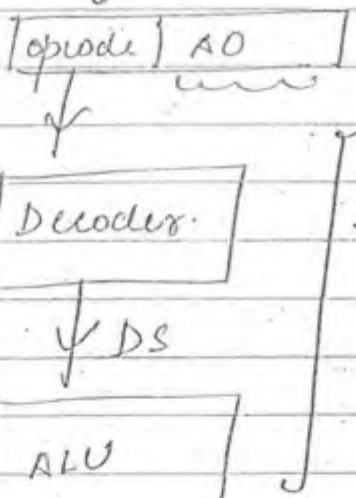
Instn fetch (memory to CPU)

The process of transferring memory content to CPU based on program counter called as Instruction fetch.

~~Destn~~ is a master, it itself controls memory. It initiates operation

· Instn fetch





After Instn fetch, decoding takes place.

The process of enabling the ALU gates is called as Decoding

If the data is present in the register, IR[AO] IR reg, AO field (Intel) IR[AO] field gives the register name where the data is present so the sequence of operation is

reg ⇒ IR[AO] → registername → ACC

Access register

Content transferred to ACC

84)

If the data is their in the memory different addressing modes are effected. Under that condition first assume that immediate mode is applicable. So sequence is

Immediate ⇒ IR[AO] → ACC Dest^n

Source = IR[AO] Dest^n ⇒ ACC

AO is itself data

(3) Other than Immediate addressing mode we need to calculate the effective address.

AO itself is not data then it is memory address.

Mem ⇒ IR[AO] ⇒ MAR → AL's → Decoder → DL's → MBR
RD → CL's

Source is address of operand field (Memory content itself is data)

→ Process of transferring the data from the IR[AO] into acc is called as

OPERAND FETCH 30/Dec/2010

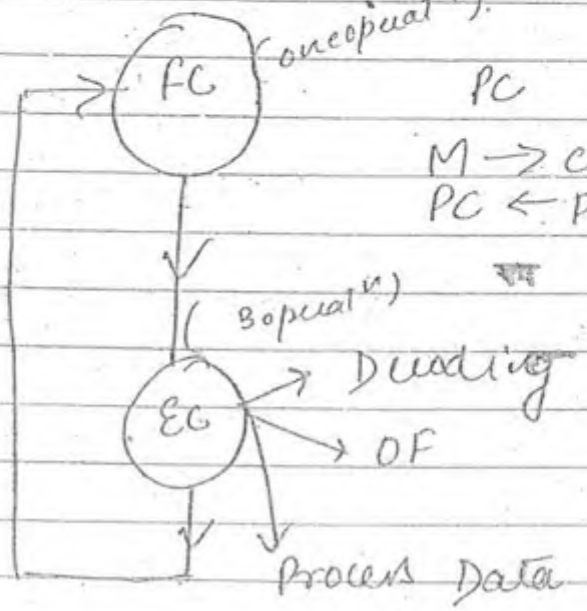
→ The process of transferring the data into ALU and converted one form to another form of data based on ALU operat^n called as Process data

Memory to CPU] Fetch cycle
Inside the CPU] Execution cycle

Date:
Page No.:

The Instr^{n} execution sequence is represented by Instr^{n} cycle. It consists of 2 subcycles one is fetch cycle & other is Execution cycle that is shown below

Instruction cycle



PC
M → CPU
PC ← PC + Step Size

EC takes more time as compared to FC

↓ once Instr^{n} execution is completed, processor is ready to execute next Instr^{n}.

If Source is PC → Dest^{n} (→ IR)
If Source is IR → Dest^{n} (ACC)

→ There is always delay due to more time took by EC so we can't say its not successful operation.

30/12/10

CPU Organization

CPU org is classified 3 types based on no of address fields present in the Instruction. ie ① Stack Organization
② Accumulator based m/c
③ general register orgn

Stack Org: It allows zero address instn it only supports with push and pop operation to transfer the data. It consists of only opcode field (no address field)

Format [opcode] Pop 2 times → operand1, operand2

ADD

Result Pushed stack.

Accumulator Based m/c

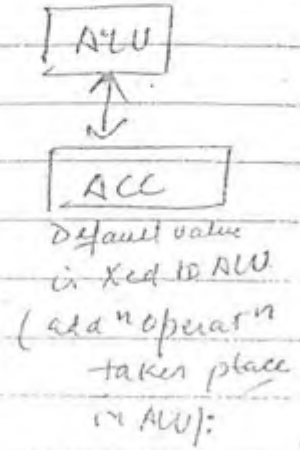
The organization allows 1 address instn The default register is accumulator.

format : [opcode | A0.]

ADD [2000]

Memory content of 2000

ACC ← ACC + [2000]



Date :
Page No. :

(3) General register organization :

This org allows 2 address & 3 address instr^ns

⇒ 2 address instr^n consists of 2 address fields in the instruction

After operat^n original data is nowhere present due to overwrite

| Opcode | Addr 1 | Addr 2. |

→ operand address / Dest^n addr.

→ Ops address

2 address field

D.

Add (r0) r1

r0 2000
r1 2000

r0 → r0 + r1.

r0 2) 3000.

Add r_0, r_1

$r_0 \leftarrow r_0 + r_1$

acts as source.

After performing operat^n r_0 act as dest^n

(Original data is nowhere present, In other words I/P data is modified with O/P data)

⇒ 3 address Instr^n format → To overcome the prob^m of 2 address Instr^n, where the no of registers are more, we can use 3 address Instr^ns.

Format :

| Opcode | Address 1 | Add 2 | Add 3 |

↓ Dest^n addr

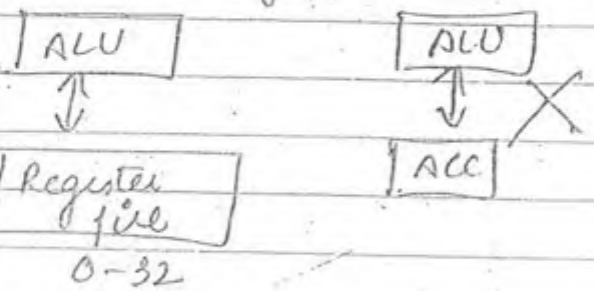
↓ ↓ Operand 1 & operand 2 address

Add r_0, r_1, r_2 r_1 r_2 added } No overwrite

Reduced Instr^{n} Set Computer (RISC) allows 3 addr^{s} Instn^{s}

bcoz the no of registers are more.

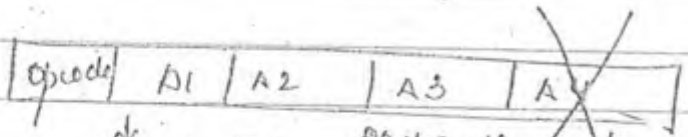
ALU connected with reg file (consists of 8 registers) primarily of 32 reg we can send data. But incase when we connect with ACC we need to send data thro ACC.



Add $\gamma_7, \gamma_{12}, \gamma_{13}$ (no need of maintaining one register as ACC).

Bcoz whole reg file is connected with ALU

4 Address Instn: → This Instn consists of 4 address field



There is no need of A4 as PC already point next instn addr.

Opcode: Oper^{n} addr
A1: Op1 addr
A2: Operand2 addr
A3: Next instn addr.
A4: next instn addr.

As long as PC is not pointing processor then there is need of 4 addr. But PC is mandatory register. So no need of 4 addr instn.

Consider the Hypothetical cpu which supports 1 address and 2 address instn formats. The 16 bit instns is placed in 128 word memory If there exists (2) 2 address instn How many 1 one address instns can be formulated.

16 bit instn

↓

128 word.

$= 2^7$

(128 word) 16 bit

| opcode | Add1 | Add2 |

2 address instn.

| opcode | Ad. |

1 address instn.

16 bit

| op | Add1 | Add2 |

2 | 7 | 7

2^7 cells → 1 word.

There are 2^7 cells, each cell contain 1 word.

Address size = 7

Address line = 7.

2 → 4 possible opcodes

00

01

10

11

2 comb^n are free

Among 4 comb^n, only 2 are defined.

Only one address is reqd

9	7
opcode	Addr.

⇒ no of free comb^n

$\Rightarrow 2 \times 2^7$

no of free comb^n | Address freed

If there is no free comb^n we don't move further.

One address Instn		2 address Instn			
opcode	Size		2	2	2
0000		00	op	A1	A2
0001		01		00	
		10		01	
		11		10	
1111				11	

opcode

⇒ Total possibilities 24.

Min possibility — Assume 3 possibility of 2 address.
(" of 1 address].

1 possibility of 2 address.]
3 possibility of 1 address]. Max possibility.

Case1 If 2 2address are already there, only 2 combⁿ are possible.

$\Rightarrow 2 \times 2^7 \Rightarrow 2^8$

→ Address size = 7bits

→ Instⁿ size = 16bits

Consider 2 address instⁿ ie

opcode (2)	Addr (7)	A2 (7)

(← 16 →)

Total = 2 address instⁿs are possible with the above format is 4.

ie
00 ✓
01 ✓ (Assume) already 2 instⁿs are upsted
10
11

The free combⁿs are 2. Use this combⁿs to define 1 address instⁿs.
ie

10 0000000
   7bits
10 1111111

11 0000000
11 1111111

2 free combⁿ $* 2^7$
⇒ 256

If there is no free combⁿ, then no alternative

Case1 If there exists 4 2 address instⁿs one address instⁿs are zero.

Case2 The min no of 1 address instⁿs supported by the above s/m is equivalent to {possibility X address.

1 Possibility X Address ie 2^7

$\Rightarrow 2^7$ instrns

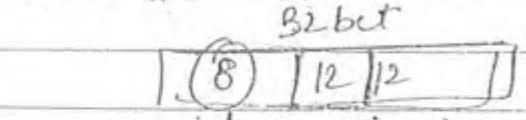
$\Rightarrow 128$

Case3 The max no of 1 address instⁿs supported by the above s/m is max possibility $\times 2^7$.

Max :→ $3 * 2^7 \rightarrow 128 \times 3$

$= 384$

Eg Consider a computer s/m that has 32 bit instⁿs and supports with 12 bit addresses If there are 250 2 address instⁿs How many 1 address instⁿ are supported.



$2^8 = 256$

256 combⁿs → 250 already defined.

Instⁿs $\Rightarrow 6 * (2^{12})$

Machine Instns & Addr Modes

(1) MVI → Move Immediate

(C)

```
        MVI   H, 02H  → Hexadecimal
        MVI   L, 05H
loop:   DCR   L   (4times)
First   JNZ   loop   Previous instn is nonzero
        DCR   H   move to next instn (255 times)
Second  JNZ   loop   only 1 time
```

H (02) ⇒ 8 bits register

L (05)

Hexadecimal ⇓ 8 binary bits

First loop executed = 4 times.

L = 05H ⇒ (4)

DCR = 01H.

JNZ = 3.

L = 0000 0000

(CY) 1 0000 0000

1111 1111 ⇓ 255

(2) position independent codes (subprogram)
⇓ We need to call subprog into main prog. We need JOC is reqd ie call instn.

Point ⇒ To Implement iterations, subprog there is need of JOC instn ie call

Based on call instn, PC value is modified.

Only one Addressing mode related to Instn. ie Relative instn mode

Every time PC is loaded with L1, same operation is loaded, infinite loop.

(3) Word 40 contains 60

(C) (b) whatever 30, the value in that is loaded X.

(d) load indirect 30

30 → 50 → 70 X.

EA

(c) load indirect 20.

20 → 40 → 60 ✓.

(4)

			Instⁿ size
1000–1007 Direct	MOV	R1, 5000	2 ✓
1008–1011 Indirect	MOV	R2 (R1).	1 → 4 locations ✓
1012–1015 Reg.	Add	R2 R3	1
1016–1023 MOV	MOV	6000, R2	2
1024–1027	(Halt)		1

Halt: simply enter into infinite loop. So enter into loop w/o any operation.

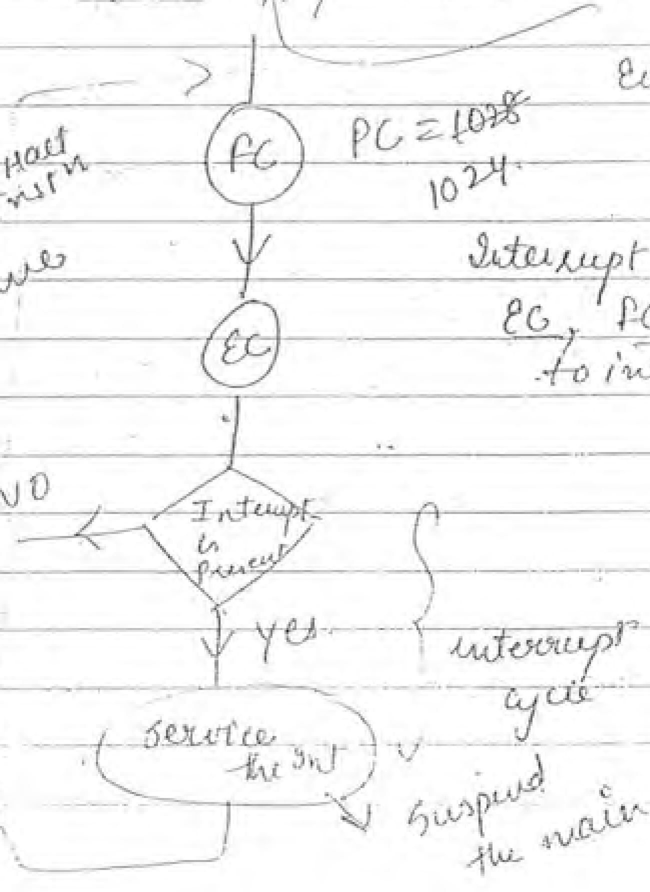
R1 5000 — 32 bits → 4 byte

→ Each word 32 bit, 32 bit require 4 memory locations.

To store one word, we need 4 locations

1 word = 32 bits memory is byte addressable.

⇒ CPU only respond to interrupts only after completion of current execution Instⁿ



After execution of (Halt instⁿ) for, PC is loaded with same addr. 1024

Every service routine ends with return instⁿ.

Interrupt may present during EC, FC but CPU respond to interrupts after execⁿ.

interrupt cycle

Suspend the main prog execution & store return address into stack

if interrupt is present

During the Add operation, return address will be 1016

Once add undergoes execution PC pt next instn address i.e 1016 is pushed onto stack.

(5) Register ← memory ⇒ 3 clock cycles.

Instn	F+D	execution time	Instn size (words)
I_1	4	3	2
I_2	2	3	1
I_3	2	1	1
I_4	4	3	2
I_5	2	0 ✓	1
	(14)	10	

(24) Ans.

Fetching and Decoding 2 clock cycles per word.

I_1 R1, 5000

I_2 R2 (R1) indirectly data is in memory.

I_3 Add.

I_4 Register → Memory

Exec cycle

F

D

OF X

PD X

Imp. This completes at the end of Decoding. No need of operand fetch & process data.

Date :
Page No. :

6 9mf

Add A[R0] , @B.

↓ indexed addressing mode

↓ indirect addr^n mode

Inst^n cycle → fetch, (Exec) → How many mem ref are there.

A and B → addresses present in memory.

| op. | A | B. |

ADD A[R0] , @ B; → mem addr.

Operand fetch

operand fetch

↳ 2 mem ref are reqd to read operand 1

2 mem ref are req. to read op 2.

EA = displacement + R0 (is in memory)

2 MR ⇒ one

~~to read~~

Add A[R0], @B
2 2

4 memory ref. to read data

4 + (2) → result writing ✓ (mem ref).

6

storing result into memory

Ques Consider a base m/c which supports with data transfer, data manipulation, transfer of control instructions which is running at 2 Gigahz speed. If the load instructions takes 3 clock cycles and store instruction takes 5 clock cycles. Manipulation inst^n takes 6 clock cycles and Branch inst^n takes 8 clock cycles. suppose the frequency of these inst^{n}s 30%, 20%, 30% & 20% respectively what is the speedup when the processor when the processor uses 1 ns. cycle time. where CPI is equal to 1.

Load inst^n = 3 clock cycles ⇒ 30%.
Store " = 5 clock cycles ⇒ 20%.
Manipulation inst^n = 6 clock cycles = 30%.
Branch Init^n = 8 clock cycles ⇒ 20%.

$$\text{CPU Time} = \frac{\text{no of clock cycle req'd} \times \text{clock cycle}}{\text{rate}}$$

$$\text{CPI} \Rightarrow \frac{\text{no of clock cycle req'd}}{\text{Inst}^n \text{ count}}$$

$$\text{CPU time} \Rightarrow 3 \times \frac{30}{100} + 5 \times \frac{20}{100} + 6 \times \frac{30}{100} + 8 \times \frac{20}{100}$$

$$\text{CPU time} = (0.9 + 1 + 1.8 + 1.6) \times 10^{-9} / 2 \times 10^9$$

$$\Rightarrow 5.3 \times 10^{-9}$$

$$\Rightarrow \frac{5.3}{2 \times 10^9}$$

$$\Rightarrow (5.3) * 0.5$$

$$\Rightarrow 2.65 \text{ ns}.$$

$$T = \frac{1}{2 \times 10^9}$$

$$T = 0.5 \times 10^{-9}$$

$$\Rightarrow 0.5 \text{ ns}.$$

Date :
Page No. :

CPI = 1.

CPU time of Ideal processor = CPI * Ic * CCtime
= 1 * 1 * 1ns.
⇒ 1ns.

Speed up ⇒ $\frac{\text{Execution time of Base m/c}}{\text{Execution time of ideal m/c}}$

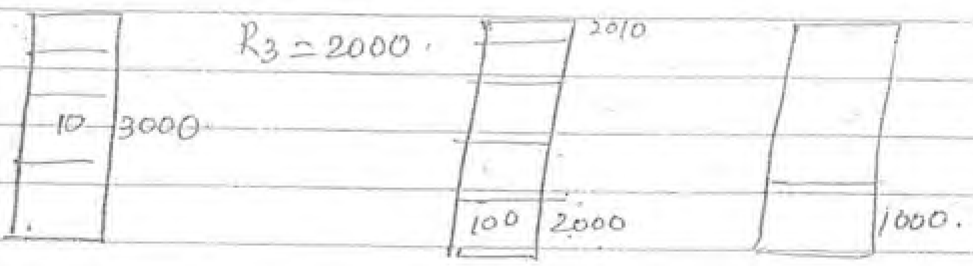
⇒ $\frac{2.65}{1}$ = 2.65 ✓.

✓. Ideal m/c is 2.65 faster than Base m/c.

Workbook – 91(Pg)

(9) (C)
a) Array → Indexed Addⁿ mode.
b). *A++ (Pointers) → Indirect.
(C) int temp = *x ⇒ Autoinc (loops).

(B)



M. Reference

(1) Direct ← 1000 – 1007 → MOV R1, 3000 1 → Direct → (to)

Register indirect ← 1008 – 1011 → MOV R2, (R3) → 10 — R1 = 3000. R2 = 10

Register ← 1012 – 1015 → Add R2, R1. { 0+10 > 20. — 10 + 10 = 20.

Register indirect A.M ← 1016 → 1019 → MOV (R3), R2. → 10 20 — 20 ✓

1020 → 1023 → INC R3. R3 ← R3 + 1. (R3) ← 2011 = 21.

(d) (1024 → 1027) → DEC R1 → entry — R1 ← R1 – 1 — R1 ← 10 – 1 = 9

R1 addr 1028 – 1035 BNZ (loop) ⇒ Target addr.

1036 – 1039 Halt ↳ TOC (uncondⁿal).

M(3000) = 1000 — R1 = 10 — R1 = 10

R3 = 2000

M[2000] = 100.

|

M[2010] = 100

loop is controlled by R1 value as R1 ⇒ 10

Instead of register if we are having M. Indirect AM then we will .... M.R as 41

(R1)

(9). (a) Reading data, modifying the data, updating the data at same locatⁿ

loop work for 10 times

- m[2000] = 100 ⇒ 110.
- m[2001] ⇒ 109
- m[2009] =

10 times

Dec R1 ✓

m[2010] = (100)

Loop is executed 10 times, as data is changed upto 2009. (2010)th locatⁿ → no operatⁿ is performed. After execution same value is reflected. i.e 100.

(10) (c) Byte addr. represent size of word i.e 4.

(1) M–R, R–M ⇒ 3 cycles.
(2) Add Instⁿ ⇒ 1 cycles.
Inc " = 1 cycle
Dec " = 1 cycle
(3) ~~Branch Instⁿ = ?~~

Fetch and Decode 2 clock cycles/word.

		Executⁿ	F+D
1 time (1)	MOV R1, 3000	⇒ 3 cycles	1×2 = 2
loop (2)	MOV R2, (R3)	→ 3 cycles×10	1×2 = 2×10
(3)	Add R2, R1	→ 1 cycles×10	1×2 = 2×10.
(4)	MOV (R3), R2	→ 3 cycles×10	1×2 = 2×10
(5)	Inc R3	→ 1 cycle×10	1×2 = 2×10
(6)	Dec R3	→ 1 cycle×10	1×2 = 2×10
(7)	Brnz	→ 90	2×2 = 4×10
(8)	Halt	→ 0	1×2 = 2 110

There is no operatⁿ. There is no need of operand fetch and process data.

Date:
Page No.:

(11) (c)

24bit Instrⁿ
⇒ 3 byte

300.

$I_1 \rightarrow$ 300 ← PC
302
I_2 = 303 ←
305
I_3 – 306 ←
308
I_4 – 309 ←
311.

Multiple of 3 → so we can say that 300×2 ⇒ 600

600 is multiple of 3

(12) (d)

16	1020
1	1001
18	1000

Rs ← 1 [Rs = 1]
Rd ← 1000 + 1 ⇒ 1001
Rd, 1000
1001, 1000 ⇒ Rd = 1001
~~18 + 1 = 19~~
Store 0 (RD), 20
1001, 20

(1) Imm.
(2) Index
(3) Direct
(4) Index

(1) mov Rs 1 — Immediate   Rs ← 1 → Rs = 1.
(2) load Rd, 1000(Rs); → Index   Rd ← 1000 + 1 = 1001.
(3) AddI Rd, 1000 → Direct   Rd = 1   Rd = 1000 + 1 = 1001
(4) Store I 0(Rd), 20 → Index.

EA: 0 + 1001   If I → is not addr, data
= 1001

(13) (d)

R1 → M[100]   Register → Memory
M[100] → R2   Mem → Register
M[100] → R3   Mem → Register

Mem is common factor

R1 → R2
R1 → R3.
R1 → M[100].

First R_1 → (transferred) R_2 (transferred) → R_3.
Register is used to speed up.
Register Time is less as compared to Mem access time. Instead of Transferring the data from Reg to Memory we will transfer data from Reg to Reg.

Date:
Page No.:

(14) (c) Self Relocating code → Subprogram.
Auto inc → add^n mode of data.

(15) (2) Privileged → m/c compulsory to handle the Interrupt.

(3) RFE instⁿ → After control send back to main prog, (2 interrupt handling is not possible.
Only one Interrupt is serviced part.

If interrupt is present, any service routine need to be executed. Then control move back to main prog.

(17) (d)

(18) μp = 4.5 μsec.
$I_1 = 25\ \mu sec$ H.P
$I_2 = 35\ \mu sec$
$I_3 = 20\ \mu sec$
$I_1 \rightarrow$ H.P
$I_3 \rightarrow$ L.P.
Range of time for I_3 ?

Response to Interrupt

✓ Response time → Time required to respond to that interrupt = 4.5 μsec

(3 Interrupt occur simultaneously)
⇒ After I_1, I_2, CPU respond to I_3 i.e

⇒ Min → I_3 Suppose one Interrupt I_3 is present

4.5 μsec.

Max. Range (I_3) = 20 + 4.5
↓ Response time
Service time

Min

(b) R_3 = Response time + Service time

$\Rightarrow 20 + 4.5$

$\Rightarrow 24.5$

Map $R_3 = 25 + 4.5 + 4.5 + 35 + 4.5 + 20$

$\Rightarrow 93.5$ μsec

(19) (d) Absolute add → address of operand field

Relative → arithmetic operation also.

↓ deals with data

(20) a

(21) c

Usages

Instruction Cycle

Instruction cycle consists of 2 subcycles, one is

(1) fetch cycle

(2) execution cycle

(1) Fetch cycle reads the instruction from the memory based on program counter address.

(2) At the end of this stage, program counter is incremented by step size. To point the next instruction address.

(3) In the execution cycle, three operations are carried out

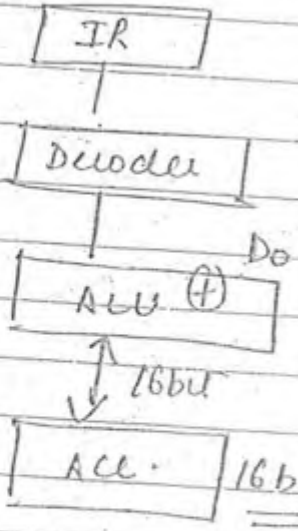
① Decoding → It enables the I/O gates.

② Operand fetch: It reads the data either from register or memory to Accumulator.

③ Process data: It converts one form of data into another form.

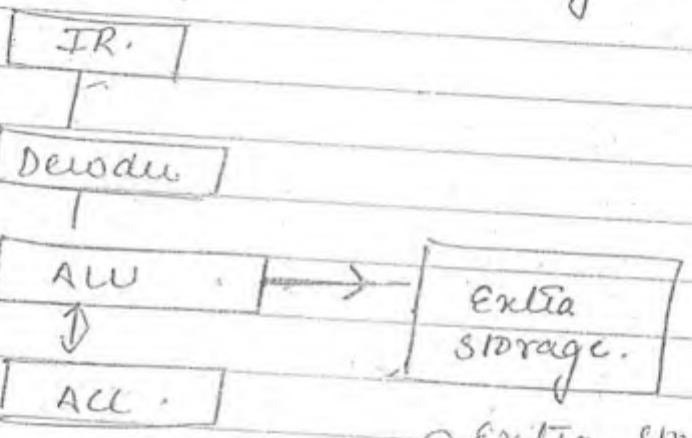
Date:
Page No.:

Suppose ACC is 16 bit



Do 16 + 16 →.
Result generated by ALU = 17 bit.
(17 bit data is lost).

So we can say, Extra storage is req'd.



When ALU generates a result which is greater than the size of accumulator then there is a loss of data.

Extra storage is changed acc. to ALU. ALU is not accepting any I/P data from ES.

To maintain correct result, generated by ALU, there is a need of extra storage which is communicated with ALU. In outward direction.

Extra storage is a flag register or program status word register.

Flag is flip flop (Bistable device), it can be set or reset based on the nature of ALU.

Date:
Page No.:

→ Flag register consists of set of flip flop. Individual flag is flip flop. which is set or reset based on the result status of ALU.

→ Flags are classified into 2 types
① Conditional flags.
② Control flags.

ALU

Conditional flags ⇒ These flags are associated with conditions. These are set or reset based on the status of condn.
All condn flags are directly connected with ALU.

Control flags ⇒ Based on the status of these flags, prog execution sequence is altered.
↳ programmer.

Conditional flags → types:

① CARRY FLAG.

Condn: It is extra bit out of MSB.

```
  10110
  10001
 /00111   → This operation is performed by ALU. After ALU, it checks the condn of extra storage.
```

① If condn is True, then set the carry flag.

CY = 1 (Set).

Date :
Page No. :

(2) If the cond^n is false :

CY
F = reset = 0. NC

→ In mathematical model simulat^n, this carry flag is effectively used

(2) parity flag → :

is used in the serial data communicat^n. The cond^n is

Odd no of 1's present in the result

In Arthimetic, parity flag is of no use

→ It is set to 1 (T=1) [P0 = 1]

F = 0 [PE = 1]

(3) → Auxillary flag carry flag is used in BCD arthimetic computations ie decimal arthimetic auxillary carry are used.

Decimal no are used in 2 ways.

① unpacked BCD.

② Packed BCD.

① Unpacked BCD ⇒ (1 digit = 8 bits → byte)

(8 bits) 0 — 0000 0000

(16 bits) 23 — 0000 0010 0000 0011

Date:
Page No.:

It requires more memory space to store the no's into the memory. To reduce the memory space packed BCD notation is used. In the packet BCD, (1 digit = 4 bits).

Nibble

1 digit = 4 bits ✓ 0 — 0000 ✓

23 — 0010 0011 ✓

→ By default we consider packed BCD if BCD is not mentioned as packed BCD or unpacked BCD.

Position

Condition: If there is an extra bit from 3rd bit to 4th bit.

→ lower nibble → higher nibble

89 → 1000 1001

+ 28 → 0010 1000

1011 0001

89 converted into packed BCD.

There is an extra bit from 3rd to 4th bit ie 1.

Higher nibble

Lower nibble

If the condn satisfies true then T = 1 = AC

False = 0 = NA No auxillary carry

(4) Zero flag:

Condn → Is the acc. value is zero. ✓

Zero flag: → If it is true, if the condn is zero.

it is false, if the condn is non zero.

eg: $x_0 = 12$, $x_1 = 12$

SUB x_0, x_1 → then condn is true → set

acc = 0 → Conclusion → 2 no's are equal

eg: compare operations

If both are zero, then set

If both are not zero, then reset

Date:
Page No.:

$r_1 = 8$

Sub r_0 r_1 — $r_0 = 12$

Case 2: ACC = 4 zero 4.

If the no is not equal then we can say whether the no is greater or less.

$r_0 = 12$
$r_1 = 8$
4

Zero flag = 0
CY = 0

If my answ → the operation is such that $r_1 < r_0$.

$r_1 = 23$
$r_0 = 12$

Case 3: Sub r0, r1 — $r_0 - r_1$

Zero flag = 0 → reset

CY = 1 → set

$\Rightarrow 12$
23 → Borrow is reqd.

Conclusion: Second parameter > 1st parameter

→ When executing Compare operation, it invokes Subtraction. If the 2 values are equal, Zero flag is set. If the 2nd value is less than 1st value, then zero flag is reset and carry flag is also reset.

→ If the 2nd operand is greater than 1st operand, then zero flag is reset & carry flag is set. It means the o/p of Compare operation reflects on 2 flags called as zero flag and carry flag.

Date:
Page No.:

```
  1 1 1
  1 0 1 1     4 bit register
  0 1 0 1     4 bit register
1/0 0 0 0     4 bit register
```

If value of acc is zero, then condn of zero flag is satisfied.

ALU. Acc content is zero, the result is zero.

Zero = Set = 1 = (Z) → loop is terminated. ✓

Zero = reset = 0 = NZ ✓ → loop is executed.

NZ → recognized by processor

Sign Flag :→

Condn Is the MSB Bit of Accumulator is 1. If it is 0, It means negative logic otherwise if it is zero, Zero then positive logic.

```
  1 0 1 0 1 1
  1 0 0 1 0 0
1/0 0 1 1 1 1
```

This part is reflected in acc

MSB.

→ Sign flag = F = reset = 0 = PL → Positive logic.

→ Sign flag = T = set = 1 = RL → Negative logic.

Overflow flag :→

→ Use: Overflow flag indicates the range of no's supported by the S/m.

If the max limit/min limit is exceeding, it sets the overflag to (1).

Date:
Page No.:

Condⁿ → Is there any carry into MSB & no carry out of MSB. or vice versa.

If any of the condⁿ satisfy ⇒ overflow flag is set.

Otherwise reset

```
  1 1  1 11      Carry into MSB.
  1 11 0 111     Carry out of MSB
  0 0 1 0 1 0 1
1/0 0 0 1 1 0 0   → reset
```

Overflow flag = Reset = 0 → NO overflow (NV)

= T = Set = 1 — OV (overflow)

Page No 112

(a) Only ALU takes the help of setting and resetting the flags.

(b)

```
  1 1   1   1
  0 1 0 0 1 1 0 1
  1 1 1 0 1 0 0 1
1/0 0 1 1 0 1 1 0
```

Status ⇒ CY = 1 ✓
Overflow = 0 ✓
Sign = 0 (reset) ✓

Carry

MSB ⇒ 1₀,0 ✓

Char	CY	NV	Sign	AC	P/E	NZ
Binary	1	0	0	1	0	0 ✓

Date:
Page No.:

Control flags:

↳ There are 3 control flags.

① Trap flag.
② Interrupt flag.
③ Direction flag.

① TRAP FLAG (Programmer): If Trap flag is equivalent to 1, Single step execution is activated.

Suppose prog contain 10 instⁿ, then one instⁿ is executed (Step the process)

Each enter button indicates the enable the trap. For the next instⁿ, we are pressing the enter button, or means we are setting trap.

1 ——— t=1 ↓ enter
——
——
——
10 ——

(tracing) → Single step execⁿ

t=0 exectⁿ starts from starting to ending until break pt..

(F10) ⇒ t=0 → all instⁿs are executed until Break pt.

define status of trap flag

(F8) = t=1 → Single Step execution

eg t=1 →
g=0 →

If trap flag is disabled, it means execute all instⁿs until Break pt

Date:
Page No.:

(2) Interrupt flag

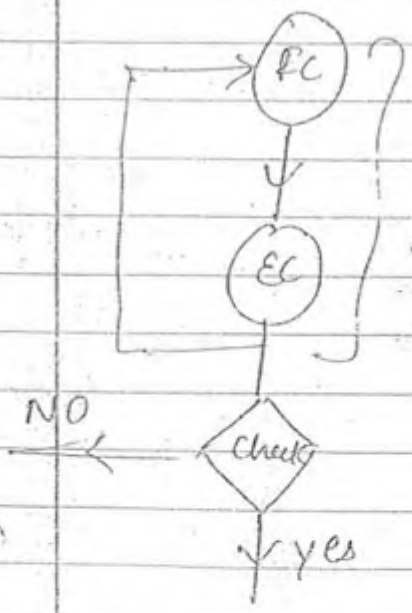
→ To enable or disable the interrupts this flag is used

Maskable → listening the states of Int flag.
Non maskable are not listening the Interrupts states of Int. flag.

Interrupt → 1 (Enable)
→ 0 (Disable)

P ← INTR
I1, I2, I3.

If =0 (No need of checking interrupts as the interrupt pin is already in disable state



If IF=0 we need to check up to this

If IF=1 =.

(3) Direction flag : This flag is used in string manipulations.

There is a need of transferring of bulk data then we use string operatn.

Date:
Page No.:

→ auto Decrement

Direct^n → 1

→ 0 → Auto Increment

Steps of String manipulation :

(1) Initialize source and Destination area
(2) Identify the length of the string.
(3) Identify starting address or ending addr of the string. which is passed as a input.
(4) Set or Reset the Direct^n flag.
(5) perform string move operat^n.

(1) DS : SI ES : DI

Source

	Offset
C	0000
S	0001
E	0002
	FFFF

String CSE

BA

DS : SI → SA

2000 : 0000

↳ offset

↓ BAS Segment

Starting addr

C	0000
S	
E	1000
	FFFF

ES : DI

3000 : 1000

(2) Identify the length of the string is 3.

CX = 3

Count register

(3) Start^n addr and ending addr.

Dec count
Inc offset

CLD ⇒ Clear Direct^n flag

Raj Monday
21/12/10 ☆ 2010

Date:
Page No.:

No need of Inc Source & Destn addrs. The Source & Destn addr are automatically Inc until count becomes 0.

→ CLD (All Instn Info are included in CLS).

→ MOVSB → Move Byte String from source to Destn.
Due to cld, we won't write MOV SB again. as count automatically perform dec operatn.

→ STD Set Direction flag → auto dec of Source & Destn addr.

* It is optimization because it is performed acc to Direction flag. The total execution is changed.

RISC Vs CISC

Architecture is divided into ~~RISC & CISC~~ two types based on the types of Instns. ie.

① fixed length Instns → RISC
② Variable length Instns. → CISC

Characteristics of RISC processor :→

① RISC supports Rich register set. more.

Page No.:

RISC processor consists of Overlapped Register Windows.
Each register window consists of 3 types of registers

① Input register
② local register
③ o/p register

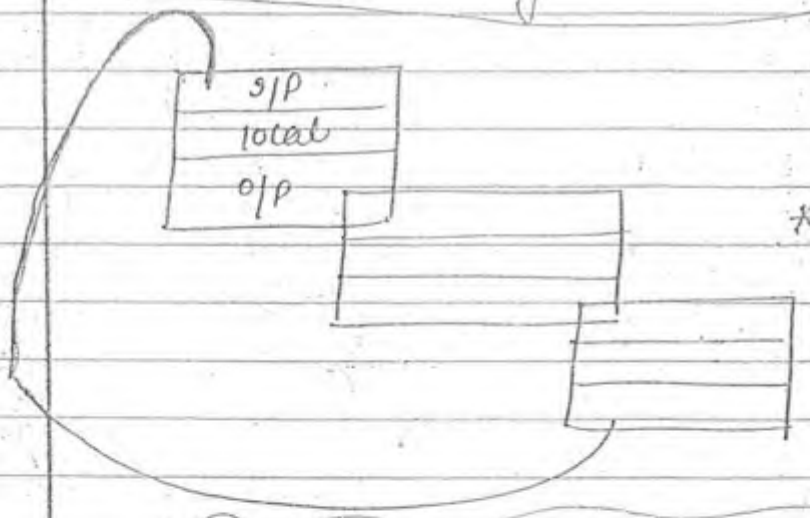
i/p reg.	C
local reg.	L
o/p reg.	C

RISC processor also supports with global registers (G)

Window size = L + 2C + G → global reg.
↓
No of local register

(N windows)

All global reg are accessible by reg window.



* o/p of Ist reg act as I/p of 2nd reg. and procedure.

* Register file = no of windows (W) [L + C] + G.
↑
Common register

Common register ⇒ 1 set.

Each window consists of one local + one common register (C).

More no of register ⇒ addressing no of modes will be decrease.

Perf inc

Date:
Page No.:

2nd characteristics

⇒ less no of addr modes. Because more no of registers. Most of time data is available in Register. So. addr modes is reduced.

Indexed or Register indirect are efficient mode to read the data from memory. So. most of the time we will use these addr modes.

3rd char ⇒ fixed length of Instn is supported.

4th char → one Instn per cycle (all the Instn size is same).

Average CPI of RISC = 1.

5th char ⇒ If all instn is completed within 1 cycle we can say Successful pipeline. Successful pipeline is implemented.

Eg of RISC ① → ARM → Advanced Risc m/c.

② → power pc

③ → Motrolo 68000

④ → Mips (Microprocessor w/o Interlocking pipeline stages)

(20) (a).

(23) (d)

12 reg windows = 12

global windows = 16

I/P = 8.

local = 16

O/P = 8.

Window size ⇒ 12.

Total no of reg = W(L+C) + G.

⇒ 12(16+8) + 16

⇒ 12(24) + 16

⇒ 304

Reg windows ⇒ 16+16+16

Regwindow = L+2C+G

⇒ 16+16+16 = 46

(24)

CISC	RISC
MOV AX, 05 — 1 clock	MOV AX, 00 — 1
MOV BX, 06 — 1 clock	MOV BX, 05 — 1
MUL AX, BX; 40.	MOV CX, 06 — 1
Multiply AX with BX	Start ADD AX, BX (2 inst. in 6 times 2*6=12)
42 clock cycle	loop Start; loop till CX=0. = (15)
	No mul operation, But then also we are performing mul
By default Immediate addr mode, the clock cycle is 1	1+1+1+12 = 15

Immediate addr mode (b)

Speed up = ET CISC / ET RISC = 42/15 = 2.8

Denominator run faster than Numerator

RISC code will run faster than CISC by 2.8

Change

Accessing

(28)

Operand Addr mode	Frequency %
Register	30.
Immediate	20
Direct.	22 → (Memory reference)
Memory Indirect	17 (2 memory ref)
Index	11 (3 memory ref)
	100 (1 → arithmetic)

No of cycles.

(Memory ref)
(1) Read from memory — 2.
(2) Index Arthimetic computations. — 1
(3) Operands are available in registers or within the Instn itself. — 0

Average operand fetch = ?

⇒ 30% of 0 + 20% of 0 + 22% of 2 + 17 of 4 + 11% of 3.

⇒ $\frac{30}{100} \times 0 + \frac{20}{100} \times 0 + \frac{22}{100} \times 2 + \frac{17}{100} \times 4 + \frac{11}{100} \times 3$

⇒ 1.45

(26) Change

What is the speedup in the Instn execution rate if the same or pipelined Adder a 0.4nsec overhead ↓ and consumed in setup and clock skew taken together.

AW = 4cycles ⇒ 45%
Branches = 3cycles = 15%
Memory oprn 5 cycles = 40%

~~T = 1ns ⇒ 1×10^-9~~
~~F = 1/T ⇒ 1/(1×10^-9)~~

execution time of pipelined ⇒ 1+0.4 ⇒ 1.4ns.

⇒ Execution time of unpipelined

28 (b).

(30) Relocation → pc relative
(d)

(31) (d)

(32)

(33) (C) → EA = {BX, DI, SI}

(34) b → EA = {BX, DI} 2000 ↑ displacement

Scaling Ind grup mode concept.

⇒ [BX] + [SI] + 2000 displacement

(34) (b) Need of 3 reg to store 3 values

g is no where took as I/P so we are replacing

Reg1	Reg2	Reg3
~~a~~	~~b~~	~~d~~
~~d~~	~~e~~ (old)	~~b~~
b	e (new)	d

If there is in I/P don't replace

(3) Registers

Date: 3/Jan/2011
Page No.:

Pipelining

→ High performanced architecture
↓ exhibits concurrent execution.

High performanced architecture performs concurrent execution.

→ Concurrency → more than one event (program) execution is called as concurrent execution.

event → program → Inst^n → Subprogram.

→ Concurrency implies to ① parallelism ② simultaneous ③ pipelining.

① Parallelism
② Simultaneous
③ pipelining

→ Parallelism: → Two or more events executed at same time period (min and max value both are present). (0-5 sec)

→ Simultaneous Two or more events executed at same time instance. (no range).

→ pipelining → Two or more events executed in overlapped time span.

Page No.:

Classification of High performanced Architecture.

According to Flynn's classification, architecture is divided into four types based on Instn stream and data stream.

(1) SISD (2) SIMD (3) MISD (4) MIMD

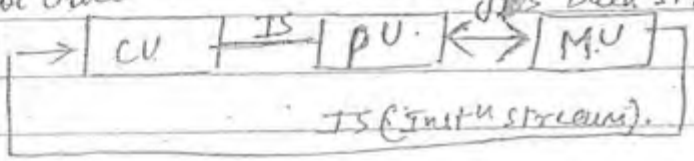
* Single Instn Single Data Stream

(1) SISD (only one Instn / data fetched at a time).

↳ Uniprocessor s/m * Single Port

The operations are.

Control Unit Processing Unit Memory Unit.

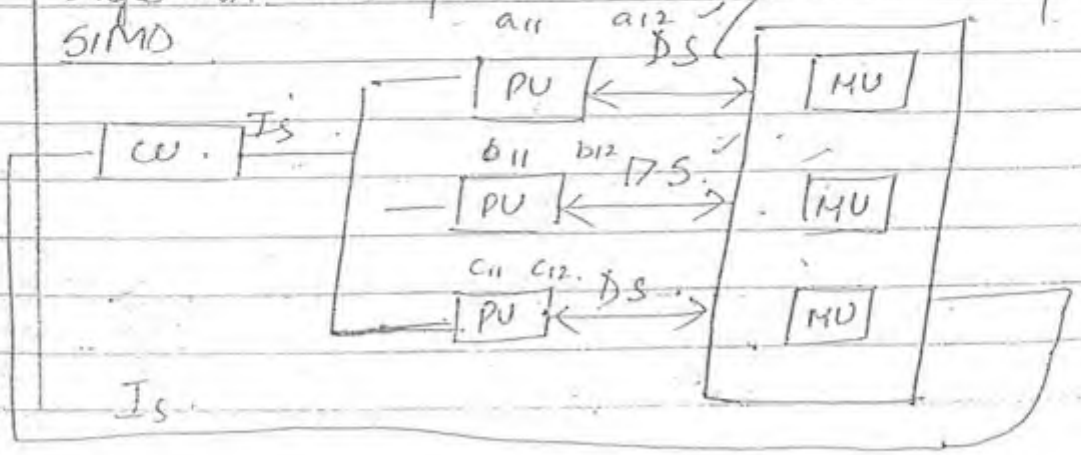


Each processor consists of one control unit. Uniprocessor where as if processor consists of more than 1 control unit it is multiprocessor. Only one memory accessing is possible.

Instn fetch → Unidirection

Memory fetch → Bidirection.

(2) SIMD Single Instn Multiple Data Stream. Multiport memory.



Single port only one element gets execution.

Date:
Page No.:

↗. If its multiport memory, w/o conflict data is transferred. multiple

↗. Same Instⁿ is operated on multiple data elements.

⇒. In matrix multiplication, data elements are different ie (A1, B1, C), operation on the data element is same.

↳. Two implementation of this architecture
(1) Array processors
(2) Associative Array processors.

⇒ The memory w.r.t Array processor is RAM chip.

↳ Associative Array processors → Associative memory chip.

(1) Array processor : MPP → Massively parallel processor.

(2) Associative Array processor

Stream PEPE

parallel element processor

Ensemble ?

(bit Serial associative) (bit slice associative).

Date:
Page No.:

If all the processors are homogeneous processing element (all are accessing same memory, performing same operations). So it means simultaneous execution is possible.
The way of accessing data is known as concurrency.

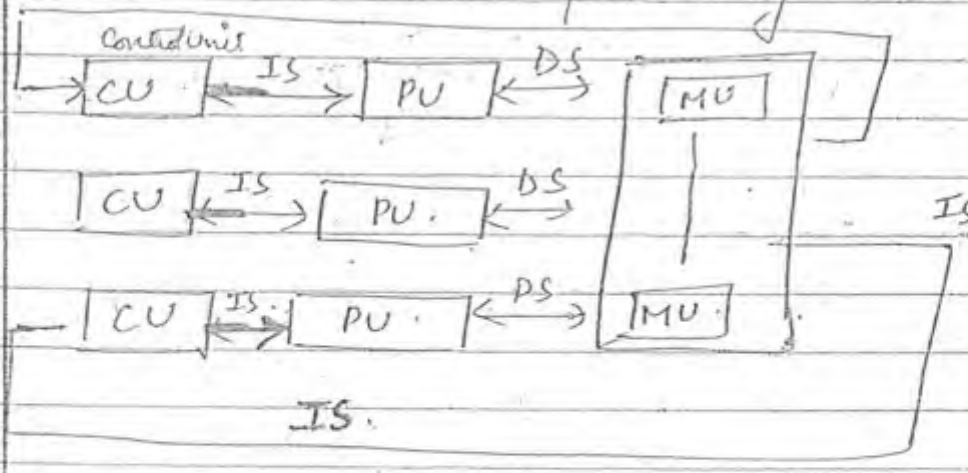
↗ SIMD exhibits (Spatial) concurrency.

MISD → (Multiple Instⁿ fetch)

↗ there is no implementation. Multiple processors are there, only one process is executed once. So it means its not economical.

MIMD → Multiprocessor architecture ↘.

Multiprocessor consists of control units, their associated processing units & memory.



↗ Multiple control units.
↗ Multiple port memory.
No simultaneous operations. Always homogeneous is not correct. Similar processor is not designed. Simultaneous execⁿ is not possible.

Date:
Page No.:

parallelism is achieved in pipeline asynchronous Multiprocessor. All processor need not to be simultaneous. Here we are achieving asynchronous concurrency.

→ Based on the march, SIMD is having concurrent execn, MIMD is having concurrent execn

Point → In the uniprocessor, stm. By the implementation of pipelining concept, concurrent executions are implement. Concurrent executions are exhibited.

Pipelining (overlapped executions)

principle The principle of pipelining is one stage o/p is connected with another stage I/p Input

Defn → Accepting one Input (new) at one end and before previously accepted Input appears as an output at the other end.

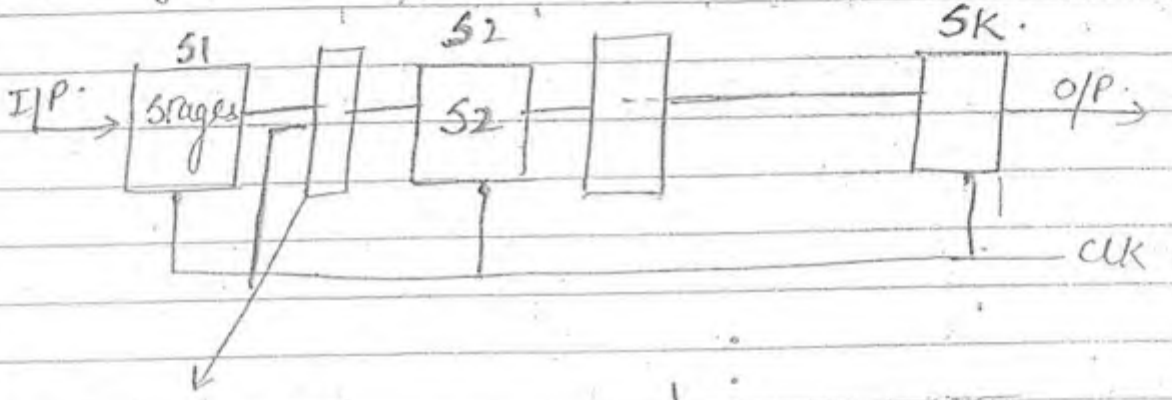
Old & new I/p both are there. Old execn is not yet completed, but new i/p is accepted. Both are possible for execn

Page No.:

The pipelining principle is accepting new I/P for every new clock cycle.

Concept of (Pipelining) → (decomposition) → Dividing the prob → Subproblem (Execute simultaneously)

It is continuous process - pipe consists of different stages such as.



Intermediate latches (to store result).
or
pipelined register

Pipelining consists of K stages, one stage O/P is connected to another stage I/P. All these stages are controlled by common clock.

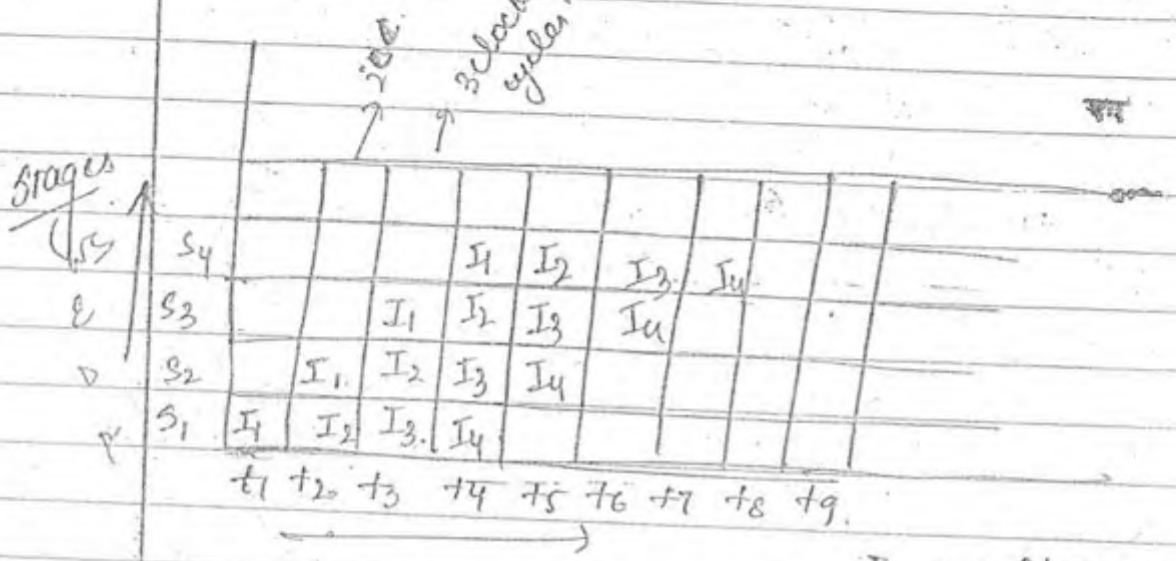
Intermediate result is stored in high speed latches. Also called as pipelined registers or buffers.

Page No.:

→ The instⁿ execution sequence in the pipeline is represented by space time diagram.

* Space diagram.

Program = 4 Instⁿs. (I_1, I_2, I_3, I_4).



x axis → time
y axis → stages.

I_4 ↓ → It appears as an o/p at one end then new I/P is accepted.

I_1 execution is completed till S_4.

Pipelining allow I/P's even if previous are not yet completed.

* Same clock cycle is used by Instⁿ 1 and Instⁿ 2 is overlapped Execⁿ. The first and last stage are non overlapped and rest of the stages are overlapped.

Page No.:

→ New I/P is accepted as pipelined I/P. For every clock cycle, new I/P must be entered in characteristics of pipeline.

→ After 4th clock cycle, no new I/P is accepted as all Instn are over.

→ Except 1 and 7 all are overlapped execution where overlapped execn is possible. It is known pipelining.

→ [Temporal Concurrency → Parallelisms]

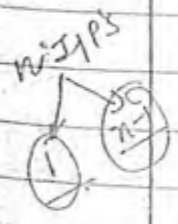
Same time can be utilized, same stage is utilized by multiple Instns.

→ Space diagram: indicates execn sequence of pipelining.

→ I_1 = 4 clock cycles (I_1 is present, Among 4 clock cycles, 3 clock cycles are overlapped with other Instn.

→ There are n inputs, k stages. If n inputs are divided into 1 and n-1.

n I/P's → 1 n-1.



→ n I/P's, k stages, 1, n-1, (K+n-1) cycles

(K+n-1) cycles to execute n instns of k stage.

Cycle time depend upon oscillator.

Date :
Page No. :

Derivation :-

→ When n tasks are executed with K staged pipelined the first task requires K cycles to complete the operation. The remaining (n-1) tasks emerge from the pipe at the rate of 1 task per clock cycle, so it requires (n-1) cycles to complete the operation. Therefore

total cycles :- → $K+n-1$

→ If clock cycle time is t_p, the total execution time is equivalent :

$(K+n-1)\,t_p$.

n tasks
K stage

→ Consider non pipelined processor that performs same operation and each tasks takes a time t_n to complete the operation. The total execution time is equivalent

$n t_n$

→ Speedup :- $\frac{\text{execution time of non pipelined}}{\text{execution time of pipelined}}$.

$\Rightarrow \frac{n t_n}{(K+n-1)\,t_p}$.

Page No.:

→ As the no of tasks increases ie consider an ideal pipelined processor where average CPI ≥ 1:

CPI ≥ 1 ✓.

Clock per Instⁿ ⇒ 1.

→ Under this condⁿ, execution time of pipelined

→ n * tp

Execⁿ time = n * tp.

Speedup = $\frac{n * tn}{n * tp}$.

→ Speedup ⇒ $\frac{tn}{tp}$

→ If we assume that the time takes to process a task is same in the pipelined and non pipelined processors.

Tn = K * tp.

→ ~~Its time req~~ ↳ pipelined execⁿ time.

↑ non pipelined execⁿ time

Under this condⁿ → $\frac{K * tp}{tp}$ = Speedup

K ≥ Speedup

→ The above formula states that if the cpu is working with 100% efficiency. The maximum speedup = pipelined depth

no of stages present in pipeline. → pipelined depth.

Problem :⇒

The Instn pipelined which has speedup factor (10) while operatn with 80% efficiency How many no of stages are present in that Instn pipeline

→ Speedup factor = 10. ✓

Efficiency = 80% (

No of stages = ?

Possible $S = K$ with 100%.

80% ——— 10

100% ——— $\frac{10}{80} \times 100 \Rightarrow \frac{10}{80 \times 10\%}$

$\Rightarrow 12.5 = 13$

Max speed up

① 13 Ans ✓

Ques

Consider 2 pipelined A and B where pipelining A having 8 stages of uniform delay of 2 nsec The pipelined B is having 5 stages with respect to stage delays 2nsec, 3nsec, 1nsec 2nsec and 2nsec. How much time is saved if 100 Instns are pipelined using A instead of B?

Ans 98 nsec

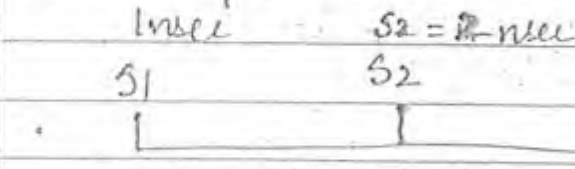
Suppose Buffer are also used → then

TP = Max of stage delay + max of buffer delay + setup time

→ Uniform delay pipelined → All stage delay are equal

If t = 1nsec → All stages must complete in 1nsec.

→ Non Uniform delay :- The stage operation itself takes more time.



Different time are associated with stages. It is known as non pipelined.

tp = tp

tp ≥ max stage delay.

→ CLK cycle is changed to max stage delay ie 3nsec. for every 3nsec, the O/P line is given as I/P to next line. It is known as non uniform stage.

→ Buffers are used as interface reg, Latches, If all buffers are controlled by clock, if buffer delay is zero then there is no time adjusted w.r.t clock.

Eg Buffer delay = 5nsec

→ tp = max stage delay + buffer delay.

Case Given all the possible cases, at the time of designing the pipeline, 1nsec is time to maintain. Designed pipeline is in operation, whatever assumption before

Date :
Page No. :

every stage inst^n

pipeline, once it is in operation small fract^n delay is possible to complete the stages.

Set up the clock ⇒ once before the pipelined designing is over. Before delivery we need to adjust the clock. The fract^n of time is changed ie set up time, skew time.

t_p = max stage delay + Buffer delay + Setup time.

Uniform ⇒ $\boxed{t_p = t_p}$

Non Uniform ⇒ $\boxed{t_p = \text{max stage delay}}$

Buffer delay ⇒ t_p = max stage delay + Buffer delay.

Setup time ⇒ t_p = max stage delay + Buffer delay + Set up time.

~~2 stage pipelining ⇒~~

Consider four stage inst^n pipeline, where different Inst^ns are spending different amt of time at different stages.

Page No.:

① How many clocks are reqd to complete them?

② What is the speed up?

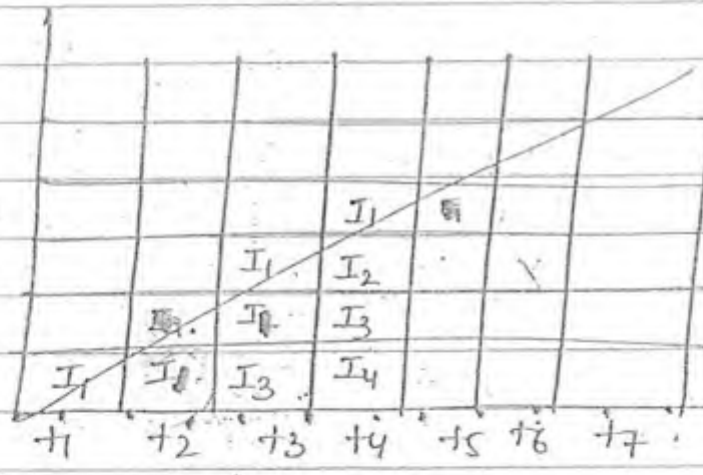
	S_1	S_2	S_3	S_4	
					add all
I_1	2	1	2	2	7
I_2	1	3	3	1	8
I_3	2	2	2	2	8 ⇒ 15 cycles
I_4	1	2	1	2	6
					29

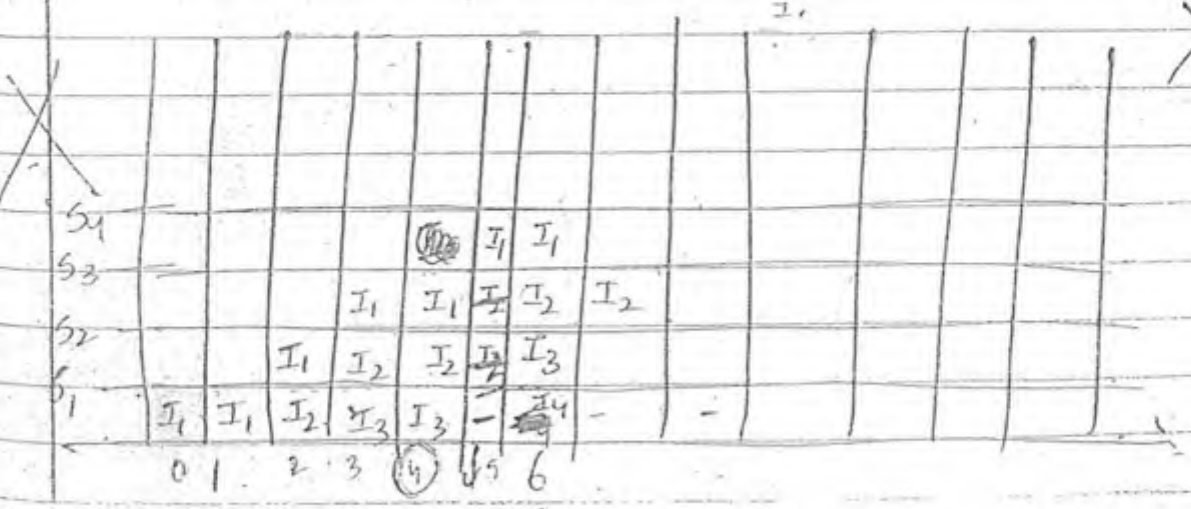
Speed up = Exec^n time of NP / Exec^n time of Pipeline

⇒ 29/15

≈ 2

Unprocessed pipeline





Waiting for space. S_1 busy with I_3.

Date :
Page No. :

Q1 Pipelined procs is having fetch decode execution and ~~write~~ write stages. FDEW

FDEW

If following instns are executed in the above pipeline what is the speed up if fetch decode write operation takes 1 clock which execution takes 3 clocks for multiplication and division and 1 clock for other instns.

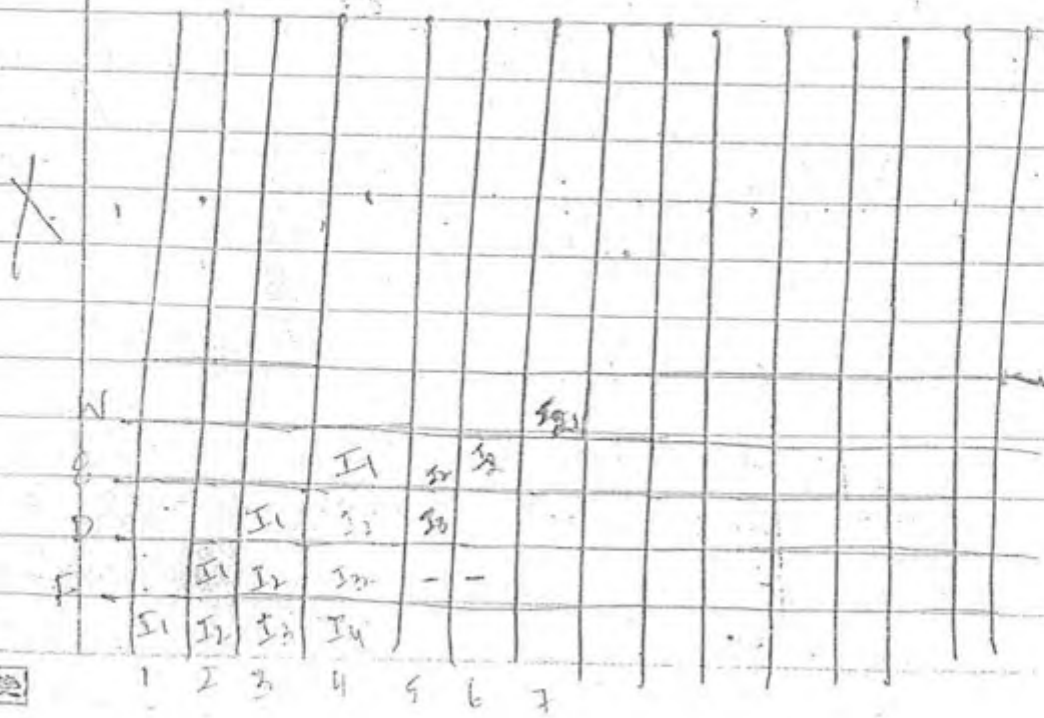
ADD R5, RD, R1 — no data dependency

MUL R7, R8, R9 → 3cc

SUB R3, R4, R6

DIV. R10, R11, R12 → 1 clock

STORE R13, X.



Date: 3/1/2011
Page No.
Monday

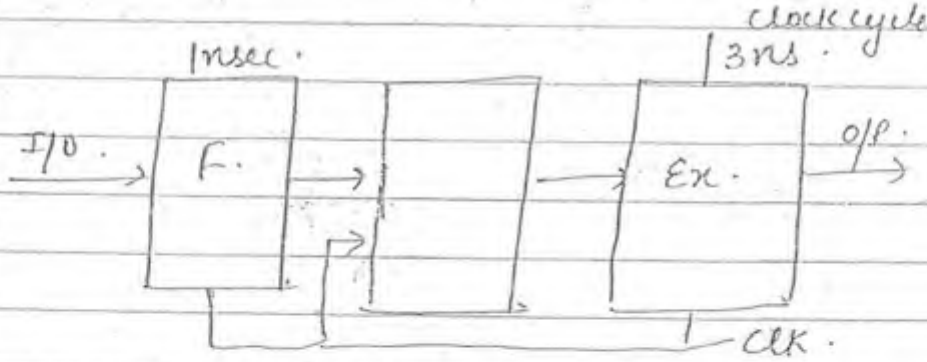
	F	D	E	W
Add	1	1	1	1
Mul	1	1	3	1
Sub	1	1	1	1
Div	1	1	3	1
Store	1	1	1	1

Clock cycles = 12 ✓.

Speedup ⇒ $\frac{\text{non pipeline}}{\text{pipeline time}} = \frac{24}{12} = 2$ ✓.

2 Stage pipelining:

→ The pipeline consists of 2 stages. one is fetch and execution. is not successful pipelining because the execution cycle takes more time than fetch cycle for decoding: and operand fetch & process data.



→ Always the delay is present at the execution stage, to make it successful two methods are used ① Decomposition of delayed stage

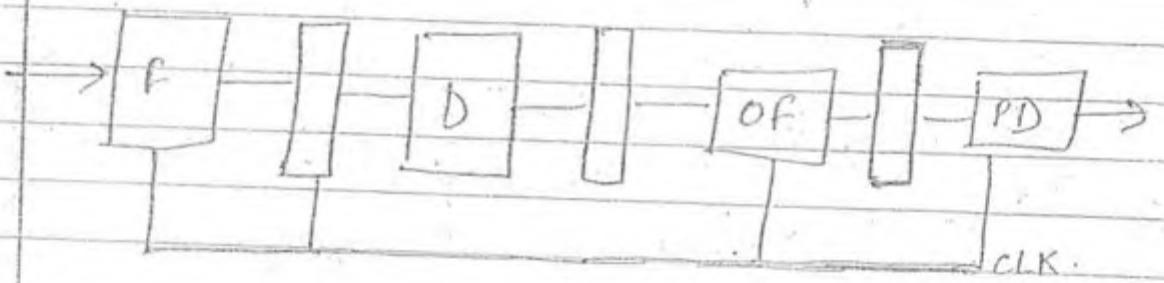
② Replication of delayed stage.

Date :
Page No. :

7 Decomposition states that divide the delay stage into substages until maintaining uniform delay.

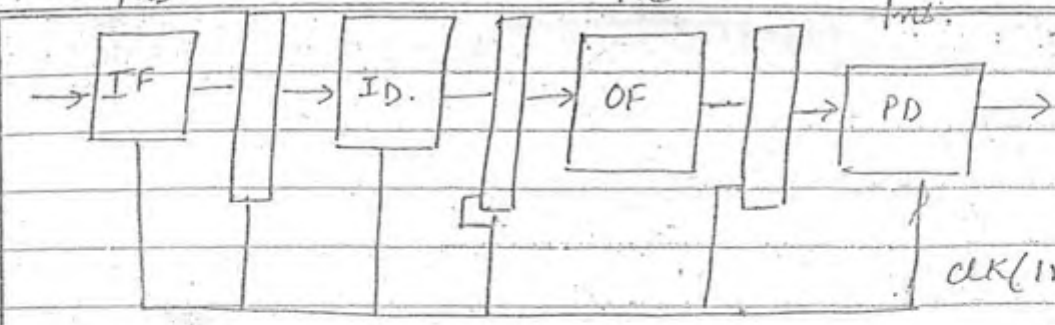
This is possible in instruction pipeline with substituting 3 stages in place of execution stage.

Now 2 stage pipeline becomes 4 stage pipeline
(1) Fetch (2) Decode
(2) Operand fetch.
(3) process data.



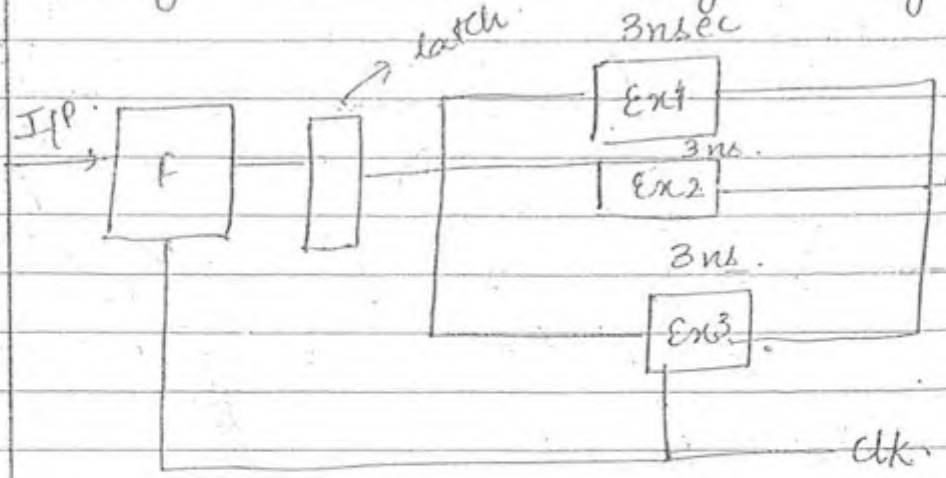
(1) 1st 2 I/P's are accepted; To accept 3rd, it accepts at 5th. So it is unsuccessful.

EX		I_1	I_1	I_1	I_2	I_2	I_2	I_3	I_3
IP	I_1	I_2	–	I_3					
	1	2	3	4	5	6	7	8	9.



PD				I_1	I_2	I_3	
OF			I_1	I_2	I_3		
ID		I_1	I_2	I_3			
IF	I_1	I_2	I_3				

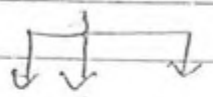
3) Decomposition is not possible use the 2nd method ie replication. Replication means arrange the duplicate stages in ~~along with~~ at the delayed stage.



Each stage require 3nsec.

Date :
Page No. :

Inst^{n} fetch Execution Stage



Clock cycle I_1

CC_2 I_2 I_1

CC_3 I_3 I_1 I_2.

CC_4 I_4 I_1 I_2 I_3.

New I/P is added to every clock cycle so we can say

CC_5 I_5 I_4 I_2 I_3

CC_6. I_6 I_4 I_5 I_3.

it is Successful pipelining.

Loop level Parallelism:

- One Iteration stages is not overlapped with other Iteration stages.

→ ambiguity

Loop level Parallelism. one Iteration Stages overlapped with other Iteration.

I_2, I_3, I_4 After completion of I_1 then start with (I_2)

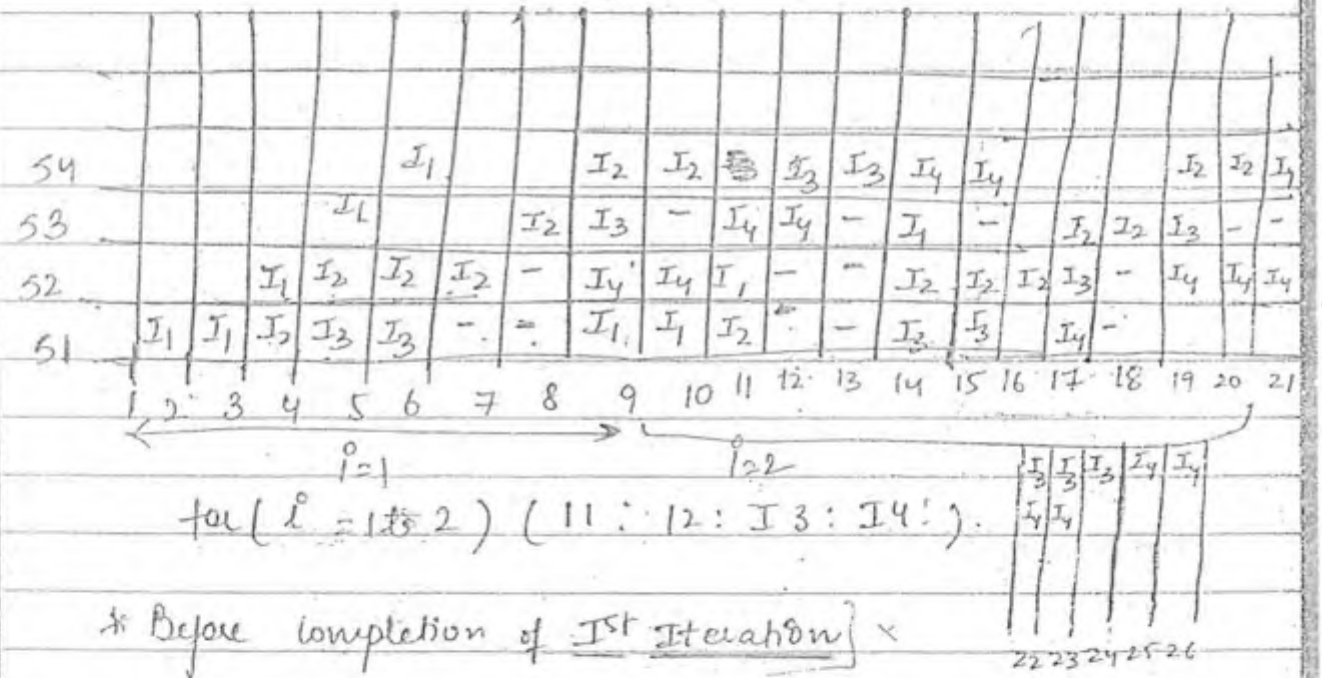
Q7

Reservation table

	S1	S2	S3	S4
I1	2	1	1	1
I2	1	3	2	2
I3	2	1	1	3
I4	1	2	2	2

→ Non linear pipelined process, execution sequence is depending on reservation station/table.

↳ don't know about time ✓.



* Before completion of 1st Iteration ×

(11) (a) $ET = (K + n - 1)\,tp$.

$\Rightarrow (4 + 1000 - 1)\,160\,ns$.

$\Rightarrow$

Date:
Page No.:

(15) No of clock cycles = 12

(a) If

(16) (b)

(19) (b)

20 (c)

CPU Control Design

(3) bbs, reg, pos, label

Emulation

→ If Instⁿ is not supported, what are the other possible ways to execute in environment

Simulatⁿ

→ If instⁿ itself supports with eg.

→ Instⁿ is not yet designed, but in need of functionality.

→ bbs → processor are not supporting bbs

↓

branch on bit set (no extra logic).

LSB

→ Reg 0 0 0 1 0 1 0 1 1

↑ Read the position, all the Reg bits undergoes evaluations

Date:
Page No.:

Mask indicates the position. The neg along position value reading onto temp.

31 ——— 0 (Right to Left).

Left representation

Read the 0, 1st, position ———.

31 ——— 0 position is pointed by Pos ie mask.

Pos.
1 1 0

Left Shift

a) mask ← 0x1 << pos.

Hexadecimal no.

(If any value followed by Hex → Hexadecimal value).

⇒ 0x1 position initially pointing to LSB is loaded to Mask.

Pos.
5 4 3 2 1 0 (Each time, we are reading one position).

Suppose we are reading 2oth position, control is transferred to 30th then check it is 1 or not If yes then load onto mask

Perform shift left operation on pos

pos → 1 → load onto mask

pos → 0 → No operation

Date:
Page No.:

Risc pipelining.

is a succesful pipeline it consists of 5 stages, one is

1. Instⁿ fetch
2. Instⁿ Decode
3. Execution
4. Memory access
5. Write Back.

→ Reason:

Risc m/c supports with 3 types of Instⁿs.

1. Data transfer (load & store)
   a) for memory reference.
   b) MBT for register reference
   Total = 3 ✓ Data Transfer Instⁿs.
2. Data manipulation instⁿs ⇒.
3. Unconditional & conditional branches.

Significance

1. Five Stage pipeline

① First stage → Instruction fetch
At this stage Instⁿ is transfered from memory to IR based on program counter. At the end of this stage, pc is incremented by step size.

PC ← PC + Step Sig

Based upon PC → Instⁿ transfered Memory

PC significance
loops
control loops
subprog calls

Assume fetched Instr^n is load

Date:
Page No.:

(II). 2nd Stage.

Decode stage performs 2 functionalities

(1) Decode the Instr^n

(2) Register file accessing.

(1) load r0, [2000]

(1). load :- Indexed register

load r0, 2 [Ri] — It designate it as r0 as destn. — accessing Index register content

↑ index register

At the time of Decode, it is read from memory.

displacement.

Register file accessing

(2) Store :-

Store reg, 3 [Ri]. — offset; Index register value

identified / accessed.

(3) Mov :→

MOV r0, r1. — destn; source

Data Xfer. Operatn

only Identification is done about source & destn

(4) Add

Add r0, r1 #23 — immediate value

Add r0, r1, r2 — identify, read, read

Date :
Page No. :

While executing the JMP, Decoder itself consists of comparators.

Jmp, branch.

branch 2000 → If it is JMP, PC ← 2000

whether unconditional or conditional → Execution is completed at the end of Decode stage.

→ Branch Not equal to Zero if

BNZ R0, 2000

R0 value is compared with BNZ.

PC ← 2000

Branch Instns are executed at the end of Decoding stage. The O/P is updated in the PC.

Execution The major work is done by ALU

It executes the Instn.

① Register → Immediate →

→ CU recognize Immediate value.

→ 2 data elements are given as I/P to ALU.

ADD R0, R1, #23.

↓ ↓ ↓

enable during second second

Date:
Page No.:

no register is not updated. The O/P line of execⁿ holds result.

② Reg - Reg:

Add r0 , r1 , r2
↑ ↑
read read

CPU recognise decoding

Execution is over, the o/p is not reflected at destⁿ. result is present at o/p line of execⁿ stage.

③. Memory reference

a) Load

b) Store

'+' Addⁿ b/w relative & content of Index register.
It generates EA

Result: The effective addr is the o/p of ALU

Points

① If the Instrⁿ is register register reference and reg - Immediate, the ALU executes that Instrⁿ but the o/p is not updated into the register.

② If the Instrⁿ is memory reference Instrⁿ effective addr is the o/p of ALU.

Date :
Page No. :

(4) Memory Access : Based on the O/P of ALU, (EA)
Memory Reference Instn are executed at this stage.

Execⁿ completed but result is not updated in Ro.

load ⇒ EA is already calculated, it enables the mem to actⁿ, read the contents.

Store : execⁿ is completed at the end of memory access.

Store

→ Store Ro, 3(Ri); ⇒ Store instn execⁿ is over at End of 4 stage.

→ No change in mem acc in Arthmetic.

Data transfer Instn are already completed. so ALU it is just forward from 1 latch to other latch.

only load & store is allowed.

Write Back :
The final o/p is updated into Destn register.

identified at Decoding level.

Page No.:

Q2.

S_4						I_1	I_1			I_2		I_3	I_3	I_4	I_4
S_3				I_1	I_1		I_2	I_2	I_2	I_3	I_3	I_4	–		
S_2			I_1	I_2	I_2	I_2	I_3	I_3	–	I_4	I_4				
S_1	I_1	I_1	I_2	I_3	I_3	–	I_4	–	–						
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$$\text{Efficiency } \varepsilon = \frac{\text{Execution time of non-pipelined}}{\text{Execution time of pipelined}} = \frac{29}{15} \simeq 2.$$

Q3

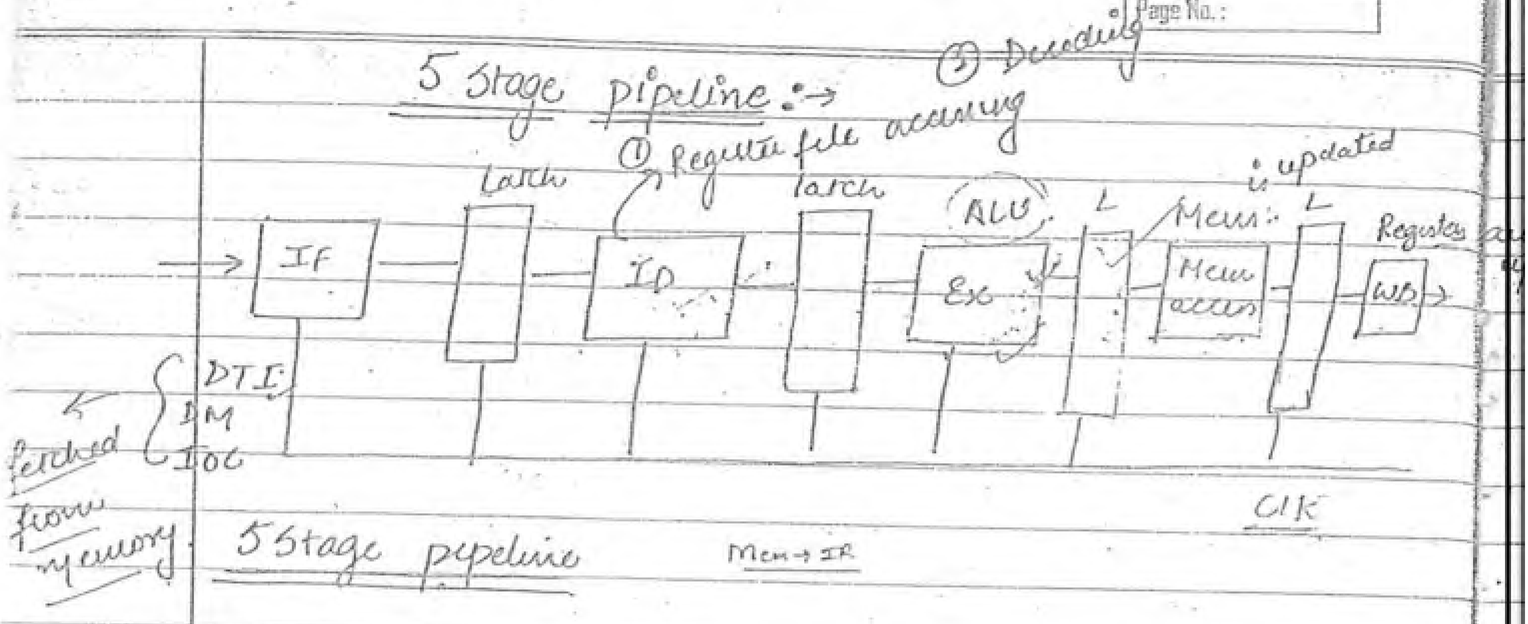
W				A			M	S			D	S
E			A	M	M	M	S	D	D	D	S	
D		A	M	S	–	–	D	S	–	–		
F	A	M	S	D			S					
	1	2	3	4	5	6	7	8	9	10	11	12

Q4

4/Jan/2011

Date :
Page No. :

5 Stage pipeline :→



5 Stage pipeline   Mem → IR

DMI, DTI, IOC → Which instrⁿ is executed at which stage is given by 5 stage pipeline.

→ Memory takes place at every part — (compulsory)

DTI → offset load/store EA.

(1) DMI → Register file along with operatⁿ

IOC → program counter is loaded with target addʳ (IOC is completed).

condⁿal — executed at the end of Decoding

Uncondⁿal

* IOC is not allowed in execⁿ stage only other instrⁿ are allowed

DTI → offset   Instrⁿ → load → Effective addr. / ↓ store

Destination ← access the memory to read ← load | store → store reg content into memory

DMI → reg file (opⁿ) → opⁿ is processed

latch o/p — forward regst → Another latch.

Date:
Page No.:

① Structural Dependencies

↳ means resource conflict This resource may be memory, functional unit or register Structural Dependencies causes Structural Hazard.

→ Hazard means "STALL" (no operation assigned in the particular clock)

Extra clock cycle is added.
(Stall inc, exec^n time inc)

	CC_1	CC_2	CC_3	CC_4	CC_5	CC_6	CC_7	CC_8
I_1	IF (Ultimate component → memory)	ID	Ex	MA (mem)	WB			
I_2		IF (mem)	ID	EX	MA			
I_3			IF (mem)	ID	EX			
I_4				IF (mem)	ID			
I_5					IF			

IM used to access Instn

MA used to access data

Pipeline for every new cycle, new I/P must present Here there are 5 Instn^s, 5 clock cycle so we can say it is succesful pipeline

Memory → conflict resource.
Here it is not clear picture about memory whether it is going to access data or Instn → it leads to more conflict

Date:
Page No.:

Data Manipulation Instⁿ

(Instⁿ)

Add r_0 r_1 r_2 — DM → regfile (Opⁿ) (+) → opⁿ is processed ($r_1 + r_2$)

related → forward right → Destination $r_0 = r_3$ lt

result is not reflected onto destⁿ. Same result is forwarded

Data transfr Instⁿ

load r_3, 2(Ri) — DT → offset → EA

load → Access the memory to read

store → Destination (R3) update

store → Store the req. content into regs

Point:

WB (operⁿ) only updating the register is completed at the end of ALU, and result is stored at the latch

Dependencies Among Instructions

The program tendency is Dependency. (No straight line scheming)

There are three types of Dependencies

Exist state b/w Instrⁿs.

- Structural Dependencies
- Data Dependencies
- Control Dependencies

Date:
Page No.:

~~In both cases pointer register & Memory space is different.~~

Unsuccessful operation → CCy is not allowed to fetch any operation, one clock cycle is wasted. This wastage clock cycle is known as stall as No. Instⁿ fetch takes place.

→ Same resource used by multiple Instⁿs that leads to conflict. Here we are considering about Von Neuman concept. It will lead to unsuccessful operation upto memory is not separated.

Point (1) In the clock cycle 4 Instruction 1 is accessing the memory to read the data.
In the same clock, Instrⁿ 4 is accessing the memory to ~~fetch~~ fetch the Instrⁿ.

(2) Two different operations pointing same memory. So memory conflict is there. So structural Hazard is present in the pipeline.

(3) To resolve the structural Hazards, use renaming mechanism.

Date :
Page No. :

(2) Data Dependency :→

Consider the program order $I_1, I_2, I_3 \ldots$ If I_2 one of the I/P is an o/p of I_1 until completion of I_1 Instrn I_2 instrn undergoes wait state

disadvantage of clock cycle

This wait state causes Stall (Hazard) bcoz of Lack of data.

I_i ↺ I_j one of the I/p. is an of o/p of I_i → Data Dependency

* This identification of Dependency is already implemented in H/W.

Instⁿ (1) Add r_0, r_1, r_2 → (1)

Instⁿ (2) Sub r_3, r_0, r_2 → (2)

Until completion of Instⁿ 1, no value is not available This is called as Data Dependency.

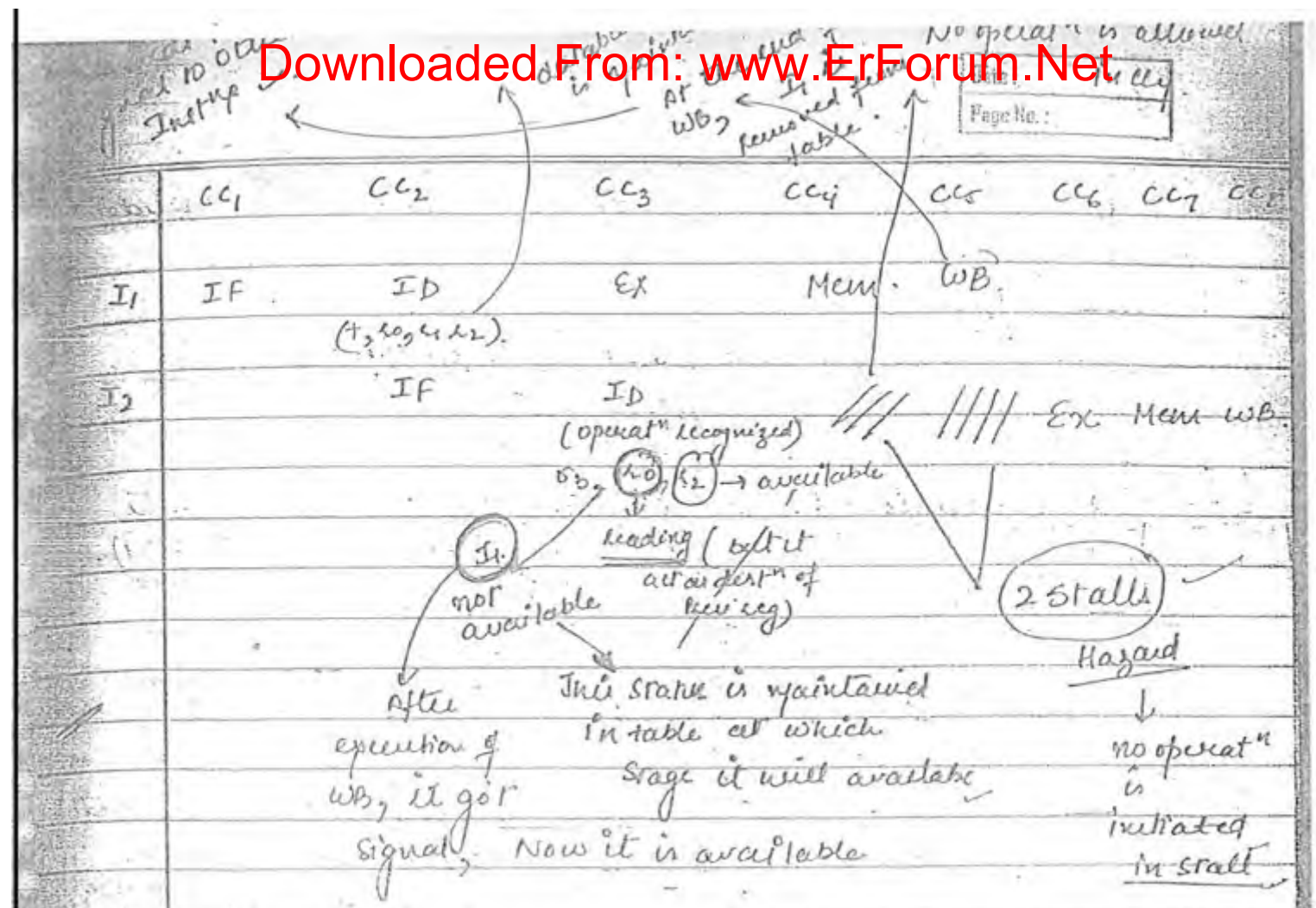
* Pipelining is implement in the H/w level

Internally processor is maintaining the database, where the database is present it is known as (Instⁿs status maintained).

(1) Reservation
(2) Reorder } maintain db instⁿs

* Completion of Instⁿ removes it form table

* At the time of Decoding R-F is there (Register file).



Now take one more Instⁿ I_3 ⇒

(3) MUL r_5, r_6, r_7 (Independent).

I_3 IF //// //// ID EX Mem WB

Once the stalls is present in Prog, it is present for all Instns whether it is dependent or independent

I_2 is Data Dependent on Instruction (I_1). Until completion of I_1, I_2 Undergoes waits The result is updated in the register at the end of Write Back Stage. So 2 clock cycles are wasted w/o any operation when the data is available, 2nd Instn continues.

Consider I_3 Instⁿ if it is Independent Instn when the processor executes the program it follows inorder execution i.e $I_1 - I_2 - I_3 \ldots\ldots$

Because of Dependency b/w I_1 & I_2 The independent Instns are also sharing the stall cycles.

Date
Page No.:

Resolving mechanism is reqd to remove the dependency in the program.

Data dependency is compulsory, then resolving mechanism is compulsory.

→ Resolving mechanisms :- (To minimize the stalls, when data dependency is encountered in prog).

→ is used in the data dependency in order to minimize the no of stalls. Called as operand forwarding or bypassing or short circuiting

→ Operand forwarding means one stage ALU o/p is connected as another stage ALU I/P

Intermediate result. Common data Bus carry. Intermediate result to the Dependent stage.

CC1 CC2 CC3 CC4 CC5 CC6 CC7 CC8

Result is present in latch

I_1 IF : [ ] ID [ ] : EX [ ] Mem WB.

one stage ALU o/p

Exec^n : (ALU I/P). EX Mem WB.

I_2 IF [ ] ID.

R_0 R_2

↓ depending on previous result but previous result in latch is forward to next

↓ Dependent stage

Dependency is resolved in this case as the intermediate result is forward to next

I_3 IF ID Ex Mem WB.

Only forwarding is possible, Bwd is not possible, it will cause stalls

Date:
Page No.:

Control Dependency :→

By execution of transfer of control Instn's control is transferred from one location to another.

→ In the pipeline process, the Instns are accepted into pipe based on the program counter in sequential order. When there is a transfer of control instns are executed the unsuccessful Instn is removed from the pipeline and the control is transferred to the successful Instns

(PC)		
1000 →	I_1 :	Add r_0, r_1, r_2
1001	I_2 :	Jmp 2000
1002	I_3 :	SUB r_3, r_4, r_5
		⋮
2000		Dec r_8.
2001		Add r_9, r_{10}, r_{11}

Consider the Instn Sequence :→

* → CPU want inorder execn.

$[I_1 \rightarrow I_2 \rightarrow [2000] \rightarrow [2001]$

Cond / Uncond Jmp

	CC1	CC2	CC3	CC4	CC5	CC6
I_1	IF	ID. (+, r_1, r_2)	Exc	Mem	WB	
I_2		IF.	ID PC=2000	Ex	Mem	WB
I_3			✗	IF [2000]	ID	
I_4					IF [2001]	

→ our Instn is uncond Jmp

→ No use of two stages

→ Removing unsuccessful Instn from pipe called FLUSH OR FREEZE

→ Nxt Instn is fetched from target addr. If leads to unsuccessful

U leads U.I so removed from PIPE

Target addr Instn

Date:
Page No.:

Here in this case In 5 cycles, 4 Instns are processing, there is wastage of one clock cycle i.e cc3 that leads to stall. Busy of

Point: For every Uncond^{n} JOC, it causes stalls.

→ If 20 Instns are there, 20 stalls are added (every subsequent) ↓ extra cycle

Points: The process of removing Unsuccessful instn. from the pipeline is called as FLUSH OR FREEZE

→ This process is used only to preserve the execution sequence.

→ Every Uncondnal JOC Instn causes one stall in the pipeline.

Conditional JOC

PC 1000 I_1 : Add r_0, r_1, r_2

1001 I_2, BNEZ r_3, 2000

$r_3 = 0$ ✓ → Nxt Instn is executed i.e 1002 (false).

$r_3 \neq 0$ → True. (control is transferred to 2000)

1002 I_3, SUB r_3, r_4, r_5

Nxt instn is fetched from 2000 as.

Page No.:

IF	ID	Ex	Mem	WB		
	IF	ID (True) PC=2000	Ex	Mem	WB	

is executed from 2000

No need to remove (IF) no stall is present

false

Remove it from pipe and control is transfered to target addr

already execute of Instn that leads to unsuccessful Instn.

IF

2

* Conanal toc causes the no of stalls as the condn evaluates to true

Power of the Instn is conanal toc, at the stage of Decoding it evaluates to true or false of the condn is true, remove the unsuccessful instn from pipe and fetch the successful instn based on target address.

* If the condn is false, no stall is present in the pipeline

No mechanism just prediction. (Guess work is done to remove/minimized the stalls when control dependency is depicted The target addr is stored or depicted in advance

To minimize the control dependency stalls Branch prediction Buffer is used or Branch target buffer

Date :
Page No. :

or delayed Branches are used.

→ SCHEDULING

→ When there is a dependency existed b/w 2 Instrns, if the rest of Instructions are independent then also stalls are shared among Instrns. (If the processor follows Inorder executions)

→ To avoid that problems, Use the out of order execn, that is Static Simultaneous execn along with dependent Instrn

Eg

I_1, I_2 → Dependency.

$I_1 - I_2 - I_3 - I_4$

→ Share the stalls which generated by I_1 I_2

I_3, I_4 → Independent Instrns.

$I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow I_4$.

Before accepting as I/P I_2 we will accept I_3 & I_4 bcoz data will be available at end.

I_1, I_3, I_4, I_2 → Arrangement of Instrns based upon dependency is known as Scheduling.

→ There is no stalls

Scheduling is possible when there is multiple final units

Name dependency is not present in Pipelining

Add1 Add2

I1 I2.

Scheduling is possible when there is no structural Hazards

I1 : Add r_0 r_1, r_2 → dependency. } Dependent Instⁿ

I2 : SUB r_3, r_0, r_1 }

I3 : MUL r_1, r_5, r_6. → no dependency } Independent

I4 : Div r_3, r_8, r_9. }

Because of this dependency all Instⁿ undergoes stalls

→ we move to scheduling concept.

Old data lost, new data is updated

1st Issue

$I_1 - I_3 - I_4 - I_2$

Anti dependency

I_2 r_3 r_0 r_1 → r_1 value is changed.

I_3 r_1, r_5 r_6

I2 modifies before I3 reads.

After execⁿ I_4, I_2 is coming.

2nd Issue

I_4 – r_3 r_8, r_9

old value

I_2 r_3, r_0, r_1

new value

Same register

output dependency.

$I_3 = I_2$

$I_3 = I_4$

Latest data is reflected (new). , an old value is removed in these cases

After scheduling the Instns, named Dependency are present, Name dependency occurs when 2

Date :
Page No. :

Inst^{n}s use the same register or memory location. There are 2 types of name dependencies :→ b/w instr^{n} (i) that precedes instruct^{n} (J) in the program order.

(1) Antidependency :→
When instruction (J) writes a register or memory location before Inst^{n} (i) reads that inst^{n}

(2) Output Dependency. :→ This dependency occurs when inst^{n}(i) and inst^{n} (J) my writes the same register or memory location.

Point → (1). Antidependency existed b/w I_2 & I_3 ie I_3 my writes the r_1 before I_2 reads.

(2) O/P Dependency existed b/w I_2 and I_4 ie I_4 my writes the data before I_2

These dependencies causes Hazards in the pipeline.

Types of Hazards :→

→ Hazards are created whenever there is dependence b/w Inst^{n}s.

→ Data Hazards are classified into 3 types :→ depending on the order of Read & write operat^{n}s.

→ Consider 2 inst^{n}s i & J with i occuring before J in

the program order.

Hazard.

" RAW " → Read after write Hazard.

↳ J tries to read data before i writes it so J incorrectly reads old value. This is called True Data Dependency.

" WAR " → write after read Hazard.

↳ J tries to write data to dest^n before it is read by i. So. & incorrectly gets new data. (Anti dependency).

" WAW " → Write after Write

↳ J tries to write data to dest^n before it wrote by i. (output Dependency)

(RAR) → cause no stall in the pipeline.

Performance of Pipeline With Stalls.

* SPEED UP : $\frac{\text{Execution time of Non pipeline}}{\text{Execution time of pipeline.}}$

$$\Rightarrow \frac{CPI_{\text{Unpipelined}} * \text{Clock cycle}_{\text{Unpipelined}}}{CPI_{\text{pipelined}} * \text{Clock cycle pipelined.}}$$

Assume ideal CPI on a pipelined processor is almost 1 (always). Hence CPI pipelined is equal to.

$CPI_{\text{pipelined}}$ = Ideal CPI + Pipeline Stall Cycles/Instn

(Stalls are present due to Dependency.

Date:
Page No.:

$CPI_{pipelined} = 1 +$ Pipeline stall cycles / Instⁿ.

⇒ If we ignore cycle time overhead (Skew typically) the clock cycle of unpipelined and pipelined is same.

(Skew is there is not there No overhead)

$$\text{Speedup} = \frac{CPI_{unpipelined}}{CPI_{pipelined}}$$

$$= \frac{CPI_{unpipelined}}{1 + \text{pipeline stall cycles / Instⁿ}}$$

* Important case: where all the Instⁿ take the same no of cycles which will also equal to no of pipelined stages (Pipelined depth).

Unpipelined CPI ⇒ Depth of pipeline

Hence Speedup = $\frac{\text{Pipelined Depth}}{1 + \text{pipeline stall cycles / Instⁿ}}$

If there is no stalls,

$$\boxed{\text{Speedup} = \text{pipeline Depth}}$$

→ Pipeline stall cycles from Branches:-

⇒ (Branch frequency (in part of prog)) * Branch Penalty

↑ need to identify with the help of space time Diag.

$ How many extra clock cycles are added for every Branch Instⁿ.

WB:

(7) $X = (S - R * (P + Q)) / T$

(a)

Add R5, R0, R1 ; R5 → R0 + R1 — RAW → 4.

Mul R6, R2, R5 ; R6 → R2 * R5 — WAR → 2.

Sub R5, R3, R6 ; R5 → R3 - R6 — WAW → 2.

Div R6, R5, R4 ; R6 → R5 / R4

Store R4, X

Same.

	F	D	Ex	W
I_1	1	1	1	1
I_2	1	1	3	1
I_3	1	1	1	1
I_4	1	1	3	1
I_5	1	1	1	1

ADD IF ID ID.

SUB. IF

STORE

MUL

DIV.

With operand forwarding (12 clock cycles) → Ans

→	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10 11 12
I_1	F	D	E	W						
I_2		F	D	E	E	E	W			
I_3			F	D	////	///	E			
I_4					////	///	D	E	E	E
I_5					////	////	F	D	///	//// E W

Date:
Page No.:

Q) Consider five stage pipelined which allows overlapping of all instns except branch instn, the target of branch instn are not available until the Branch instn is complete
let each stage delay is 20nsec (tp) and there are 30% Branch Instns (ignore the fact that some of them are condnal. What is the average Instn execution time & what is speedup?

CPU time = I + pipeline stall cycle / Instn

Pipeline stall cycle = Branch freq × Branch Penality

I1: ADD r_0, r_1, r_2
I2: Jmp 3000
I3: Sub r_7, r_8, r_9

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
I_1	S1	S2	S3	S4	S5.		
I_2		S1	S2 (D.I (no overlapped Instn)	S3	S4	S5	until S5, no further exec^n
I_3	(Unsuccessful Instn)		S1 flush.				

① Execution time = [1 + (30% * 4)] * 20nsec

② Speedup ⇒ pipeline depth / 1 + Pclock cycle stalls

⇒ $\frac{5}{1 + 30\% * 4}$ ⇒ $\frac{5}{1 + \frac{30}{100} * 4}$ ⇒ $\frac{5}{1 + \frac{120}{100}}$ ⇒ $\frac{5}{\frac{220}{100}}$ = 2.2

⇒ $\frac{5}{2.2 * 10^{-7}}$ ⇒ 1/44

Frequency = 1GHz
Cond^n al branches = 20%
Instn = 10^9.

Page No.:

(11) (12) CC1 CC2 CC3 CC4 CC5 (First Instn itself Branch).

S2 S3
Branch.

Flush ////

Branch penalty = 2.

(1 + Branch freq * Penalty) / rate

⇒ (1 + 2 * 20%) / (1 × 10^9) 10^9

⇒ 1.4 psec

(11) (b)

	IF	ID	EX	WB
I_1	1	1	1	1
I_2	1	1	1	1
I_3	1	1	3	1
I_4	1	1	1	1

Add R_2 R_1 R_0 $R_2 \leftarrow R_1 + R_0$
Mul R_4 R_3 R_2 $R_4 \leftarrow R_3 * R_2$

(5) (b)

Date:
Page No.:

(5) (b) (15) operand forwarding.

Ans → 15 clock cycles

(6) d

(10) c

(4) b

(8) d

	IF	ID	OF	PO	WO
I_0	1	1	1	3	1
I_1	1	1	1	6	1
I_2	1	1	1	1	1
I_3	1	1	1	1	1

CC1 CC2 CC3 CC4 CC5 CC6 CC7 CC8 CC9 CC10 0 1 2 3 4 5 6

I_0 IF ID OF PO PO PO WR

I_1 IF ID OF 111 111 PO PO PO PO PO

I_2 IF ID 111 111 OF 111 111 111 111 PO

I_3 IF 111 1111 ID 111 111 111 111 OF PO

Page No.:

Memory

→ Memory is used to store the program while exec^{n} of the program CPU frequently referencing the memory. So memory access time is countable to calculate CPU performance

→ Performance is dependence upon Access Time with inverse relat^{n}.

$$P \propto \frac{1}{AT}$$

→ To decrease the access time, memory hierarchy is used b/w memory and CPU.

Table shows:

Level	0	1	2	3
Name	reg	cache	M.M	S.M
Size	<1KB	<16MB	<16GB	>100GB
Implementation Technology	Custom memory with multiports	onchip/off chip SRAM	DRAM	Magnetic Disk.
Access time	0.25 – 0.5 AT (1000) Fast (High).	0.5-25	80-250	5,00,000 A.T. High as compared to SM
Bandwidth (Data transfer rate) (MB/sec)	20,000-1,00,000 [D.TR (High)] Due to register	5,000-10000	1,000-5,000	20 – 150
Managed by	Compiler	H/w	OS	OS
Packed by	Cache	MM	SM.	CD/ Floppy.

Date:
Page No.:

Cache Memory is a intermediate memory b/w CPU & main memory.

Cache Underlining principle is Locality of reference.

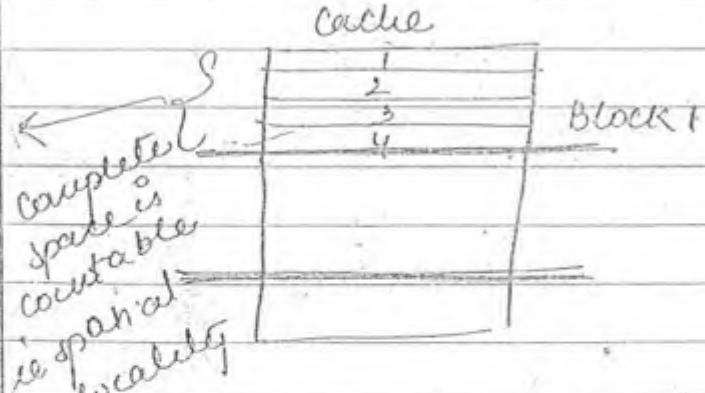
Locality of Reference

→ Data is placed in small area (which is needed) so Access time will be less. So performance is high.

→ Locality of reference is of 2 types:

① Spatial locality ② Temporal locality

→ Spatial locality: The adjacent words of referenced block is used by CPU in the near future ie is



space is referred word 1 word 2

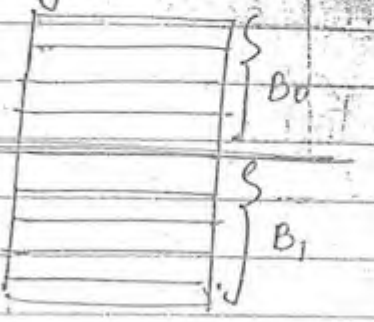
Reference is not on the same word, Reference is on same block for referring the adjacent word.

→ Temporal locality: Same word in the block referenced by the CPU in near future.

Cache memory is intermediate memory b/w CPU & main memory

→ Block means group of memory cells

* Main memory is divided into no of Blocks based on size of block.



→ No of blocks (Group of M.C) = $\frac{\text{Memory size}}{\text{Block size}}$

How cache memory is organized?

→ *Cache Memory size is less than main memory

↓

divided into no of parts based on block size.

Line size → divided no of parts Each part is known as line.

* Line size = Block size

↓

One line holding data

No of line = $\frac{\text{Cache Memory size}}{\text{Block size}}$

Date:
Page No.:

Points

① Cache memory is divided into equal size parts based on size of the block. Each partition is called as line. Line size is always equal to Block size. Main memory is also divided into equal partitions based on size of the block.

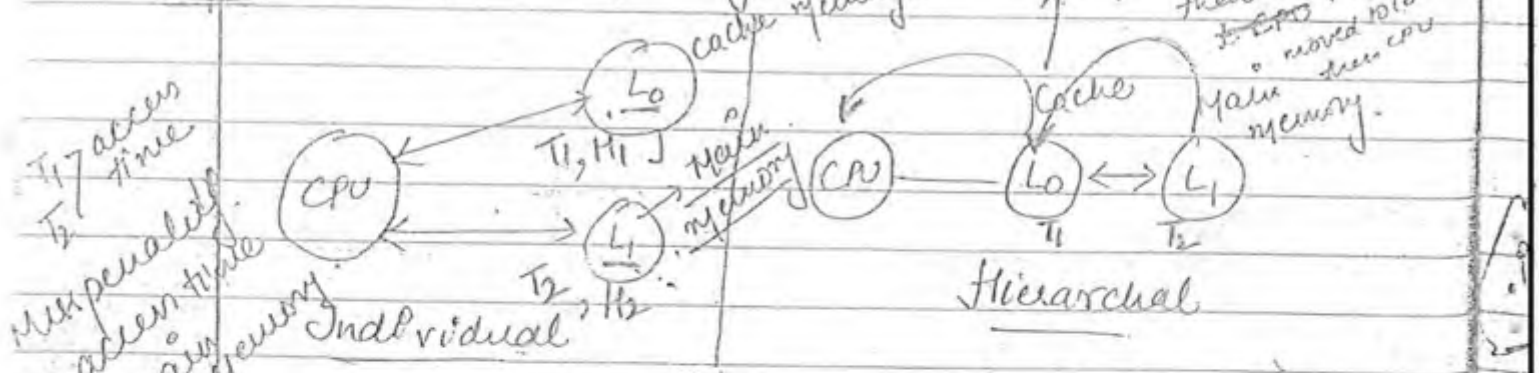
* No of blocks in main memory :→ is more than no of lines in cache memory.

② Memory is Organized in 2 ways :-

(1) Independent memory

(2) Hierarchal memory

Independent : Memory organization :-



→ When the CPU executes the program initial data reference takes place in the register
If the data is not present, next reference is to cache memory.

→ If the data is present in the cache memory operation is called hit, data is transfered to CPU

Harvard architecture, CPU enables the level 1 memory (or main memory) cell. and Immediately accesses data w/o involvement of cache

→ Average access time under this organization

* Miss penalty = access time of main memory.

Accessing time of 2 level memory Independent org

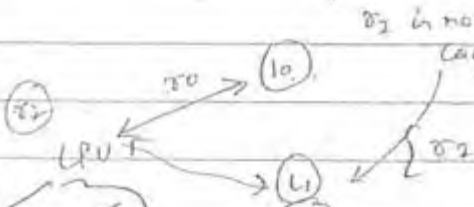
$T_{av} = T_1 H_1 + (1 - H_1) \cdot T_2$ ← hit, ← miss.

T_1: access time of C.M

T_2: access time of main memory.

{ No connection b/w L0 and L_1. }

{ Because of independent memory access, the Content of main memory is directly move to CPU. }



{ α_2 is not transfered to (always miss). }

Drawback of Independent arch

When miss operation takes place, there always a miss in other referenced data memory.

So we can say it is not suitable. Here access time Inc, perf dec.

* { Hit ratio = no of hits / (Hits + misses) = no of hits / total no of access }

Hit + miss = 1

Hit time / latency is not included

* No extra time is required to check operation whether it is hit or miss

* By default we need to consider Independent if ins quest is not mentioned about org

* In this organization if there is a miss operat^n takes placed ie always miss throughout program execution In L0 (Memory) because the image of L1 is not copied into L0

* Consider hierarchical memory concept where there is a miss operat^n in L0 cpu identifies the data in L1 and transfers that data into L0 and reads the data from L0.

So $T_{avg} = T_1 \cdot H_1 + (1 - H_1)(T_2 + T_1)$

* Principle of locality reference is possible due to Hierarchical as we can say the data in Hierarchical is dynamic (changing) But in case of Independent, the data is static.

In every part, extra storage is present

ABC's of Cache

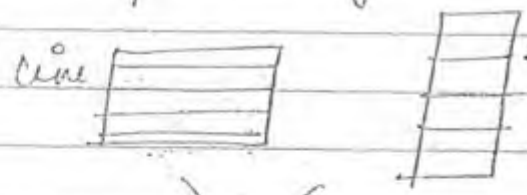
1. Mapping Technique
2. Replacement algo's
3. cache coherence
4. Levels of cache

Mapping Technique

→ The process of transferring data from main memory to cache memory is called as mapping. This mapping can be done with 3 techniques ⇒

1. Associative mapping
2. Direct mapping
3. Set associative mapping.

→ Associative mapping :- While transferring the data from main memory to cache lines, it checks the availability of line. If the line is free, immediately data is transferred w/o any formulae (rules). If line is not available with the help of replacement algorithm any one of line is replaced by new data.



no need of checking where Block B0 is loaded in cache. We just check the availability of line.

→ Direct mapping while transferring the data from main memory to cache memory, it uses the formula K mod N to replace any line. Where K indicates main memory block no. N indicates no of cache lines.

Date :
Page No. :

→ CPU want to cache data, but data is not present on cache line. Then it referred to main memory is refered to Block No such as B_8

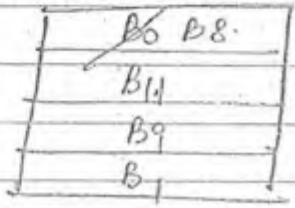
⇒ K Mod N

⇒ 8 Mod 4

⇒ 0 ✓ → B_0 → data

CPU want to read data from B_0, and we replace the existing data on B_0 with B_8.

Cache



Cache.

* With the help of this formula ie K MOD N there is association b/w cache line and main memory block.

→ Tag → How many no of main memory blocks are associated with one cache line.

⇒ tag = 2 ⇒ $2^2 = 4$

Point

* When the CPU generates the physical add^r, if the Cache memory supports Direct mapped the address is interpreted as 3 fields Info.

← Physical memory add^r →

Tag	Line offset	Word offset

2 2 = 4

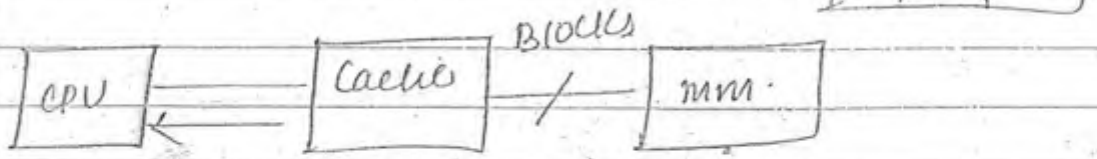
* Word offset = log Block size

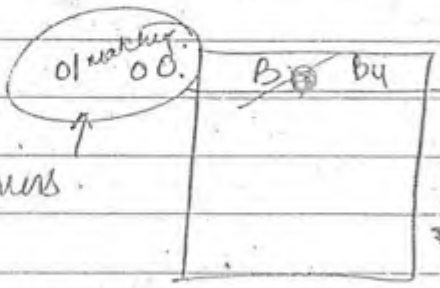
Problem of Direct mapped → One line to store one tag.
If same line uses 2 tags, no of miss operation increases

Date:
Page No.:

Suppose address 00 00 110110.

2 2 6.





01

Tag	line offset	word offset

→ Check onto main memory.

→ Consider main memory size is 1KB (2^{10}) and Cache memory size ($64 \times 4 = 256$ bytes), where Block size is equal to 64 bytes. The main memory is divided into

64 | 1 KB | 256

No of blocks ⇒ $\frac{1\,KB}{64} \Rightarrow \frac{\text{Memory size}}{\text{Block size}}$

$\Rightarrow \frac{2^{10}}{2^6} \; 2^4 \Rightarrow$ 16 blocks.

and denoted with $B_0 \rightarrow B_{15}$

Cache memory is divided into 4 lines denoted with lo to lg

Cache memory

00 → Lo (lines)

01 → L1

L3

Main memory

tag 0000 B_0 (Block)

→ 64 byte

$2^4 = 16$

0111 B_{15}

16 blocks are rep by 4 bits

Date :
Page No. :

* **line offset :** $\log_2$ # lines (→ no. of)

No. of lines = 4
$\log_2 2^2$ = 2 → bits
↓
it means four comb^n i.e. 00, 01, 10, 11

* **Tag bits** indicates the possible no. of main memory blocks present in cache line.

If tag is 00 → Block 0 → line 0 is present

01 → B4

line 01 — 10 → B8

11 → B12

Physical memory
⇒ tag + word offset
↓
line no. — Tag 10
due to direct mapped

line no	Tag no
00	00
00	01
⋮	⋮
11	11

~~Blocks in main memory~~
0000 → 1111

2	2	6
Tag	line offset	word offset

0 0000
1 0001
2 0010
3 0011
4 0100
5 0101
6 0110
7 0111
8 1000
9 1001
1010
1011
12 1100
B13 1101

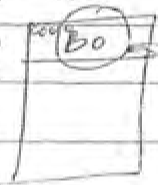
tag 00 — line no / Tag no 01 → B1, → B5, → B9, → B13 } These are blocks attached with line offset (01)

64 byte.

Tag matching

CPU address → 00 00 110110 → 00|00

110110



↓
CPU want to read data so add^r is generated

Size of block = 64 bytes

Cache line (Direct mapped cache)
↓
Memory interpretation is reqd.

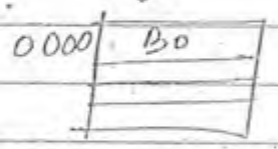
Bo

64K

among these cells we require 110110

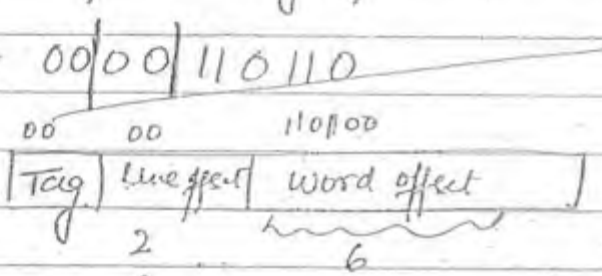
Page No.:

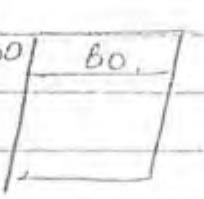
The direct mapping KMODN formula shows that which memory block is transferred to which line of cache. Transfer Bo block, i.e. {0 MOD 4} = 0. The data is transferred from main memory to cache memory line 0 along with tag



The CPU generates Physical addr to read the data from cache, memory addr is interpreted in the following format.

addr :- 00 00 110110

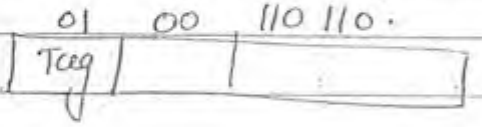




* Tag 00 is compared with existing tag's of cache memory. If it is matching, it enables the respective line along with word offset The byte is transferred from cache memory to CPU

* If it is not matching, it enables memory block with the help of KMODN formula, block is transferred from main memory to cache memory along with Tag.

01 00 110110.



& then it transfers the data to CPU.

Drawback of Direct mapped cache → One line holds only one tag.

Date:
Page No.:

This problem is overcome ~~typically~~ using set associative mapping.

Set Associative

While transferring the data from main memory to cache memory. This method uses K mod S formula to identify the cache line. where K indicates main memory reference no. and S indicates no of sets

K mod S.

↓ main mem ref.

↓ no of sets

No of sets = N → no of Cache line

(P-way)

↓

Value of how many sets are present in cache line

⇒ Once the CPU generates physical addr to read the data from set associative cache memory; the addr is interpreted as the following format

| tag | set offset | word offset |

No of cache lines = 4

Mapping Tec = 2 way set associative.

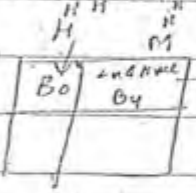
no of sets = $\frac{4}{2} = 2$.

Each cache line is divided into 2 sets

Page No.:

0	B_0	B_4
1	B_3	B_7

Same line is used to ... In Direct mapped B_0 is changed to B_4 then data is transferred but here there is no need of replacement. There is enough space for B_4.



B_4 is not there miss.

Only one time miss occurs other hit operation

Prog → B_0
Prog → B_4
" → B_0.
" → B_4
" → B_0.

(set associative mapping)

Direct mapped

only one time hit operation other time miss is happened.

* So we can conclude Set associative is better than Direct mapped

Neha Sharma

* Replacement Algo

When there is a miss operat^n in the cache, take the data from main memory. While transferring the data from main memory to cache, if the cache is full which line of cache is replaced with new block. This info is given by replacement algo's.

Replacement algo's are divided into 3 types →

1. Random replacement
2. First in first out replacement
3. Least recently used replacement.

Date:
Page No.:

① Random → In this method, no policy is used to replace cache line with new block.

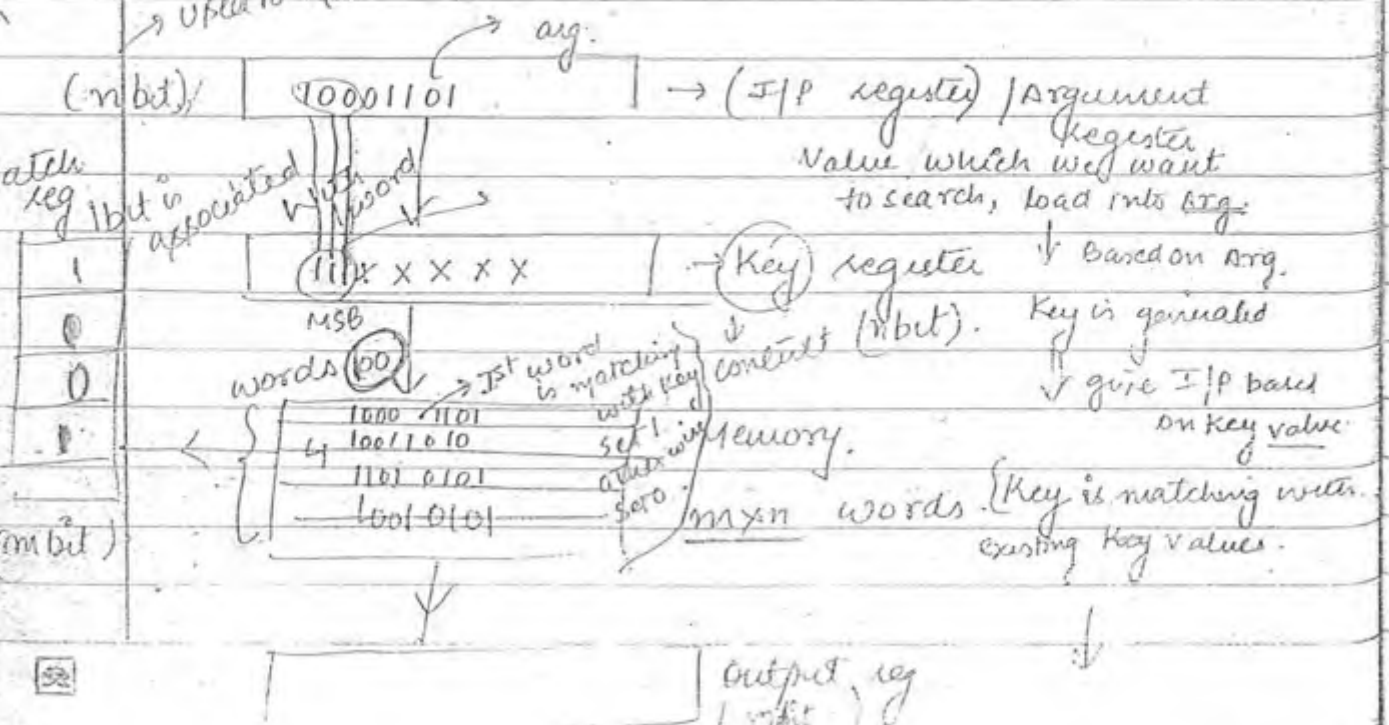
② FIFO → In this method, each line time stamp is taking into the account, replace the line with new block whose time stamp is high

③ LRV In this method, 2 parameters are considered into the account, no of references and time stamp.
Replace the line with new block where the line is having less no of ~~registers~~ references with highest time stamp.

Associative Mapping :- (Memory chip does not have address).

This memory is called as Content addressable memory or parallel search memory or multi access memory.

→ Used to maintain 1 bit at a time.



Count the no of 1's in match reg, those elements are retrieved from it.

111 X X X X

it indicates disable that bit

111 1111

enable the bit

we will do enable / disable acc to arg reg.

X → Disable.

1 → enable.

Respective bits of arg reg considered as Key

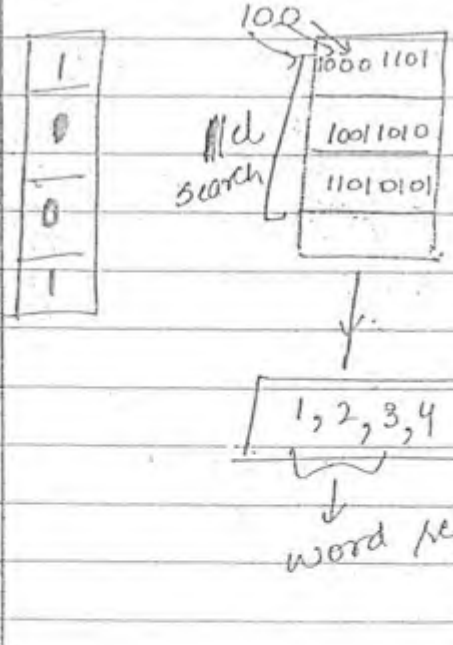
100

111

111

→ W/O address, Based on content, either read/write is possible.

Once the content is identified, all the words of memory undergoes RW search.



These are used in where search time is very critical

↓

Railway website

↓

Here Huge database is maintained

Date:
Page No.:

Numerical In a 2 level memory if the level 1 memory is 5 times faster than level 2 and its access time is 10nsec less than average access time let the level 1 memory access time (t_1) is 80nsec. what is the hit ratio. ?

level 1 memory = 5 times

$T_{av} = T_1 H_1 + (1-H_1) T_2$

$\Rightarrow$?

(1) Speedup = $\frac{\text{access time of } l_2}{\text{access time of } l_1}$

5 $\Rightarrow$ $\frac{T_2}{T_1}$

(2) $5 \times T_1 = T_2$

$100\text{nsec} = T_2$

$T_1 = T_{av} - 10$

$20 = T_{av} - 10$

$30 = T_{av}$

$T_{av} = 20(H_1) + (1-H_1)100$

$30 = 20H_1 + 100 - 100H_1$

$30 \Rightarrow -80H_1 + 100$

$80H_1 = 70$

$H_1 = \frac{70}{80}$

$H_1 \Rightarrow 0.8$

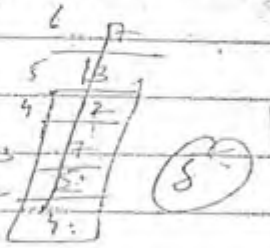
* Associative memory chip is expensive than RAM chip because of there is need of extra bit required for every cell (match logic) Size, Speed is same in both cases.

Page No.:

Ques Consider four block cache with following main memory block references

4, 5, 7, 12, 4, 5, 13, 4, 5, 7

(1) Assume cache is initially empty Identify hit ratios using (1) first in first out ⇒ 0.2
(2) LRU ⇒ 0.4
(3) Direct mapping ⇒ 0.5
(4) 2 way set associative with LRU



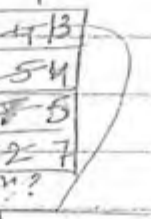
(1). First in first out

Hit ratio = No of Hits / Total access time

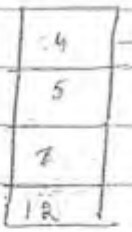
⇒ 2/10 = 1/5 = 0.2 FIFO

(2) How many blocks are there after completion?

Q1 Which block is not present after completion i.e 12



(2) LRU

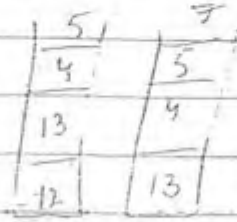


4 → M
5 → M
7 → M
12 → M

no of references by LRU along with

4 → M
5 → M
7 → M
12 → M
4 → H ✓
5 → H ✓
13 → M
4 → M
5 → M
7 → M

transfer

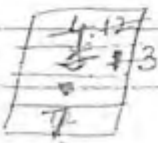


⇒ H = 4/10 = 0.4

12 is not there

(3) Direct mapping →

5 mod 4 = 1
13 mod 4 = 1
4 mod 4 = 0
5 mod 4 = 1
7 mod 4 = 3
12 mod 4 = 0



⇒ 3/10 = 0.5

Date:
Page No.:

4 2 way set associative cache

No of Sets $\Rightarrow \frac{N}{P\text{way}}$

$\Rightarrow \frac{4}{2}$

$= 2$

0	4	12
1	5	7

Formula = K mod S

(1) 4 mod 2 $\Rightarrow$ 0
(2) 5 mod 2 $\Rightarrow$ 1
(3) 7 mod 2 = 1
(4) 12 mod 2 = 0
(5) 4 mod 2 = 0 $\rightarrow$ Hit
(6) 5 mod 2 = 1 $\rightarrow$ Hit
Least $\rightarrow$ (7) 13 mod 2 = 1 $\rightarrow$ miss $\rightarrow$ 2 set anociative with LRU so 5 becomes latest we can't replace 5
7 becomes Least recent so replace it.
(8) 4 mod 2 = 0 $\Rightarrow$ Hit
Recent $\rightarrow$ (9) 5 mod 2 = 1 $\rightarrow$ Hit
(10) 7 mod 2 = 1 $\rightarrow$ Miss.

Hit ratio $\Rightarrow \frac{4}{10} = 0.4$

13 is not there in Cache

Qno Consider 1MB memory and 2KB cache memory both are partioned into 64 Byte word block, How many bits are read for the physical add^{n} what will be the total no of blocks in main memory and cache memory. How many tag bits are req if two cache mapped as direct mapped.

(23) 2^7 lines

(a) line size = 64 words $\Rightarrow 2^6$

Physical addr = 20 bit

| 9 | 5 | 6 |

Tag | Block offset | Word offset

20

S = No of lines

$\Rightarrow \frac{128}{4}$

$\Rightarrow 32$

$\Rightarrow 2^5$

(24) block size = 32 $\Rightarrow 2^5$

Physical addr = 32 bit = 2^{32}

(a) Cache size = 32 K bytes

No of lines = $\frac{32K}{32} = 1K \Rightarrow 2^{10}$.

32

| 17 | 10 | 5 |

Tag | line | word

(10, 17) Ans

(25) C

(26) C

32 K byte

32 bit

$\Rightarrow 32$ K byte

$\Rightarrow 16 \times 2^{10}$

$\Rightarrow 2^{14}$

32 bit

8 bit

32

| 15 | 12 | 5 |

6 Jan 2011

Date:
Page No.:

Cache Coherence —:

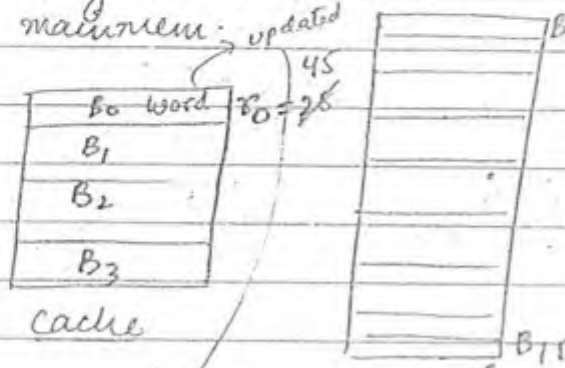
Mem

PA → Physical add

CPU — Cache — MM.

Write operation is a kind of add

Image of main mem: updated



This updation is not present in main memory

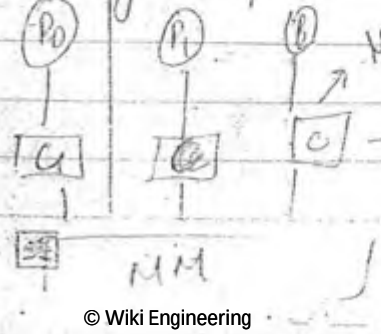
Same addr consists of multiple values ie Know Cache Coherence.

* After write operation, Cache memory is enabled so r0 value is changed to 45

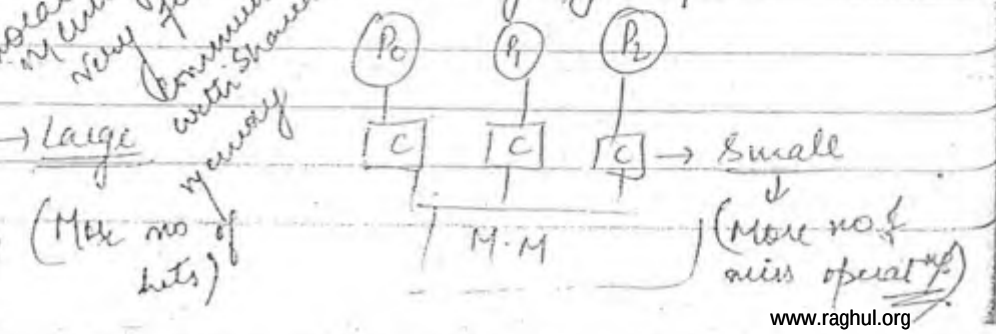
Modified data will be lost if we apply some mapping / replacement algo such as same addr consists of multiple values, then updated data will be lost. This is cache coherence. If we don't handle cache coherence, it leads to problems, such as it damages prog exec.

Multiprocessor.

Loosely coupled



Tightly coupled.



Date:
Page No.:

Cache Coherence :-

P_1	P_2	MM (x)
x = 0	—	0
x = 0	x = 0	0
x = 1	x = 0	x = 0.

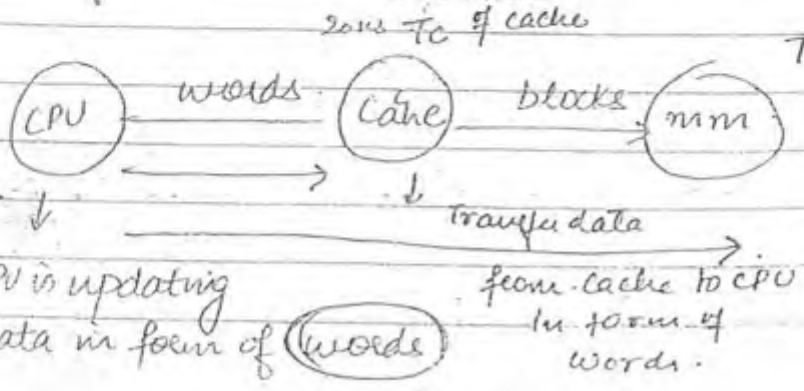
Coherence means same addr consists of multiple values

* In the Uniprocessor sys, when CPU executes a program, it performs write operation, only cache memory is updated. The respective addr in the main memory is not updated. So same addr consists of 2 values in cache and main memory. ie Cache Coherence.

If the cache coherence is not handled properly the new data will be missing to overcome cache coherence, there is a need of updating techniques.

* There are 2 techniques to avoid the coherence one is write through, write back policy

① write through → Simultaneous updation
② write Back



Date :
Page No. :

T_m = 200 ns.

↑ to access complete block

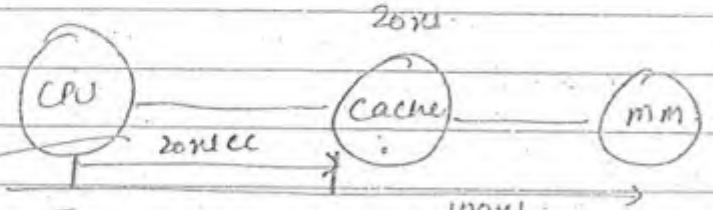
Block size = 2 words

T_w ⇒ word accessing time of main memory.

To access 1 word it require ⇒ $\frac{200}{2}$

⇒ 100 ns

Case 2:



20nsec are overlapped with 100nsec

T_w = 20ns, 100 ns

T_w = 100ns (max stage delay).

When there is non uniform delay then t_p = max stage delay.

For succesful updation, 100ns is reqd.

Write through policy :→ Indicates simultaneous updation ie when the CPU performs write operation it updated into cache memory and main memory at a time. Here (t_c) indicates cache memory access time (t_m) indicates main memory block access time (t_w) indicates main memory word accessing time

When there is a succesful write operation the time reqd to perform that is t_w ie maximum (cache word access time, main memory word access time)

⇒ Tmax (cache word access time , main memory access time)

⇒ Tmax (T_c , T_w)

⇒ T_w.

Date :
Page No. :

(11) 32×2^{10}

⇒

No of Chips

No of Chips ⇒ 32×2^{10} ⇒ 4

$2^5 \times 2^{10}$ 6

(11) 2. No of Chips ⇒ $\frac{256K}{32 \times 2^{10} \times 8}$ ⇒ $\frac{2^{18} \times 2^3}{2^{15}}$ ⇒ $2^6 = 64$

(13) Cache blocks. = 16 ⇒ 2^4

Main memory = 256 blocks. ⇒ 2^8

Main memory. Cache memory

(d) No of sets = $\frac{16}{4}$ ⇒ (4).

	4B	3V	2V	1V
0	0	4	8	216
1	15	133	127	73
2				
3	255	3	159	63

216 ✓

2V

(1) ⇒ K mod S.
⇒ 0 mod 4
⇒ 0.

(2) 255 mod 4
⇒ 3

(3) 1 mod 4
⇒ 1

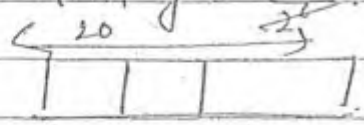
(4) 4 mod 4
⇒ 0

(5) 3 mod 4
⇒ 3

(6) 8 mod 4
⇒ 0

Date:
Page No.:

Physical memory = 2^{20} 2^{14}



① Physical memory = 20 bits

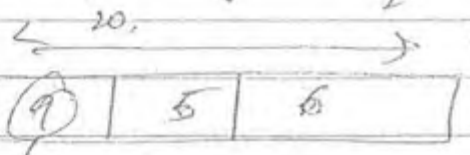
② no of blocks = $\frac{2^{20}}{2^6} = 2^{14}$ in main memory

③ word offset = $\log_2$ Block size. ⇒ $\log_2 2^6 = 6$ 2^{14}

Line offset = $\log_2$ no of lines $\log_2 2^5 = 5$

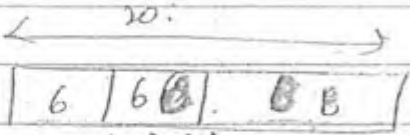
2×2^{10}
$= 2^{11}$

② Cache memory = $\frac{2^4}{2^0} = 2^5$ (no of blocks (lines)) ✓



④ tag

⑧ Memory = 20 bit address Cache = 2^8 bit Data size.



Tag. line size word offset

no of blocks in main memory ⇒ $\frac{\text{Memory size}}{\text{Block size}}$

⇒ $\frac{2^{20}}{2^8} = 2^{12} = 4096$ ✓

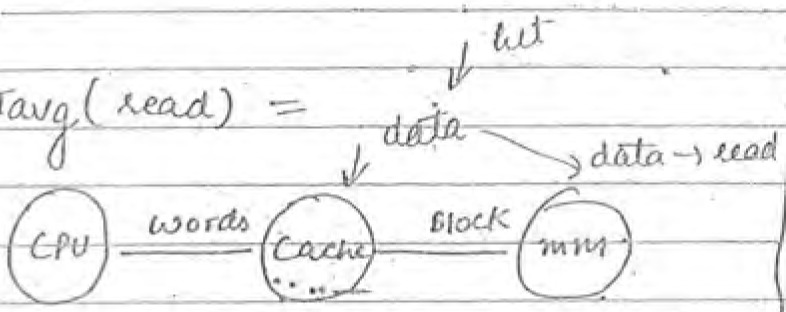
12
2
1024
10
4

no. of lines in cache = $\frac{2^8}{2^2} \Rightarrow 2^6 \Rightarrow 64$

System performance

read operat^n — write operat^n

Tavg(read) =

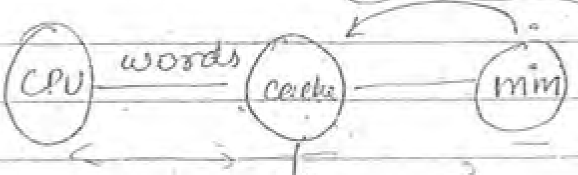


(No coherency due to write through itself states that update the value in cache as well as m.m. It means no possibility of having diff value for same addr.)

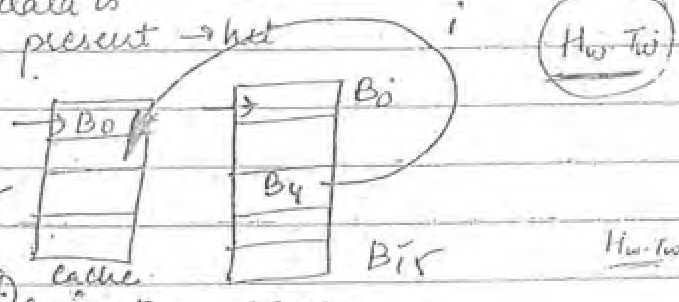
$$T_{av}(\text{read}) = T_c \cdot H_r + (1 - H_r)(T_m + T_c).$$
(Read operat^n)

Tavg(write)

First transfer block from mm to cache then perform write operat^n.



Block/data is present → hit



(a) Image is present in cache. now update both

(b) B4 is not present, i.e. miss then move to mm

$$T_{avg}(\text{write}) = \text{Successful write} + \text{Miss write}$$
$$= H_w \cdot T_w + (1 - H_w)(T_m + T_w).$$

Assume the block which is modified, that block is present in cache & M.M (max time need to be...)

T_m ↑ to get data from m.m

T_w ↑ Perform simultaneous write operat^n

Date:
Page No.:

Miss write is also known write allocate → The process of transferring the block from main memory to cache memory when only write operation is performed

Write OPERATION:-

In the case of write operation, first CPU checks the cache memory for the availability of block which is going to be modified. If the block is found in cache, that operation is called as hit operation related to write. The CPU immediately performs simultaneous updations on cache and main memory. So it will take time

Hit process $[tw * Hw]$ — (1)

(2) If the legal block which is going to be modified is not in the cache, operation is miss. so. there is a need of transferring the legal block from main memory to cache before write operation. This process is called as write allocate

) When the block is available in the cache, the CPU simultaneously update the block, this process requires the time

Miss process = $(1-Hw)(Tm+Tw)$ — (2)

↳ word update

↓

Block transfer

do (1) & (2)

TAV = Hit process + Miss process ie

$\Rightarrow tw * Hw + (1-Hw)(Tm+Tw)$

(1) $T_{(read)} = T_c H_R + (1-H_R)(T_m + T_c)$

$\Rightarrow 20 \times 0.80 + (1-0.80)(200+20)$

$\Rightarrow \frac{1600}{100} + 0.2 \times 220$

$\Rightarrow 16 + 44 = 60$

(2) $T_{(write)} \Rightarrow H_w \cdot T_w + (1-H_w)(T_m + T_w)$

$\Rightarrow 0.90 * 100 (1-0.9)(200+200)$

$\Rightarrow \frac{90}{100} \times 100 + 0.1 * 300$

$= 120$

$T_{avg.} = H_R \cdot T_c + (1-H_R)[\%\text{clean bits}(T_m + T_c) + 30\%[T_m + T_m + T_c]$

$\Rightarrow 0.80 * 20 + (1-0.80)[70\%(200+20) + 30\%[200+200+20)$

$= \frac{80}{100} \times 20 + (0.2)\left[\frac{70}{100} \times 220 + \frac{30}{100} \times 420\right]$

$\Rightarrow 16 + 0.2[154 + 126]$

$\Rightarrow 16 + \frac{2}{10}[280]$

2 word

$H_R = 80\%$

$H_W = 90\%$

30% updation

CAT = 20 sec

MAT = 100 sec

$T_{write} = H_w \cdot T_c + (1-H_w)[70\%(T_m + T_c) + 30\%(2T_m + T_c)]$

$\Rightarrow$ 4 min

COMPUTER ORGANIZATION