Computer Organization

1. Number System & Representation

2. Memory Organization : Hierarchy
   Cache Memory – Address mapping
   – Updation
   – Replacement
   Principles of virtual memory & Associate memory

3. Fixed Point Arithmetic, Carry look ahead ae

4. Instructions, Addressing Modes, Instructior pipeline, control unit design

5. Addressing, Data Transfer Techniq
   Disk & tape memories

1,3,4 : Computer Architecture – J.P. Hayes
2,5 : Computer organization – Pault choudhar

CA: It deals with conceptual ~~things~~ view of [illegible]
- Instructⁿ set
- Addressing Modes
- Data format

~~CA~~

Instructⁿ Set
- CISC (Intel)
- RISC (~~Reduced~~ Risk Inst. set) (PowerPC) " Fixed length"

(Not advantage) Supports limited Addressing mode.

// If any instr. will not complete at at $CPI = 1$, then it is not a part of RISC. (clock per instr.)

// No more registers are required.

// Ltd provides only flexibility.

// Instr. have - | OPcode | Refr opr |

Reference of operand.

by Ref opr part, we can easily get the instr. addr.

// Data format deals with "How to interpret the binary string"

23|09|10

Instruction Pipelining

Register,

→ Efficient usage of resources (Instruction phase are overlapped)

→ Performance of pipelining is given by speedup factor, $S = \frac{\text{time without pipeline}}{\text{time with pipeline}}$

se. address

gister

→ Ideal case with K-stage Instructⁿ pipeline

$$T_n = (K+n-1)\, \cancel{T_b}\; T_{clockP} \quad S_{ideal} = K,$$
$$CPI_{avg} = 1,$$

is

ed in it

ing.

→ Parameters influencing the performance

- Un-even stage delays
- Buffer overhead
- Dependencies among the instructⁿ

→ Data dependencies occurs when the result of one instructⁿ is to be used by its successor.

→ Instructⁿ re-scheduling, stall cycles introduct & $\boxed{\text{operand forwarding}}$ techniques deal with data dependencies.

→ Control dependencies occur when flow of execu is altered by Instructⁿ cycles of another.

$\boxed{\text{Branch instructⁿs}}$ are the main resources for control dependencies.

→ Delayed-Branch, Multiple pipes & $\boxed{\text{prediction}}$ are used to deal control dependencies.

(1) If processor register will be

~~of operand~~

(1) Index - then it is Index Register

⟹ used for accessing the arrays.

(2) Base address of operand - Base address Register

→ Used for relocatable prog's.

→ Prob occurs with JUMP instr.

Used ~~for~~ in

~~(2)~~ (3) Relative Addressing - PC is involved in it.

Used for intrasegment branching.

(4) Based index Relative.

A(0,0)
A(1,0)
~~A(2,0)~~
A(3,0)

(8,0)
~~(9,0)~~
(0,1)
(1,1)

mn major
r is moved
ntal blocks
two
them

%)

Principle of Associate Memory

" The associate memory is also called a content addressable memory. To retrieve the info., the content or partial conten is used; since no address is involved, it is the fastest memory."

The associate memory contains:

(1) argument Register (n)
(2) Key Register (Mask register (mask).) (n)
(3) Associate memory array (m×n)
(4) Match Logic (M×1)

→ To perform the read oprn, place that cont or partial content in the argument regist The match logic will generate either singl or multiple match word.

→ In case of multiple matching, the matched words are accessed sequentially until the desired word is obtained.

→ To perform the write oprn, place the wo in argument register, if it is not already existing then it is to be moved into one of the free locatn. ~~th~~

→ The match logic expression for ith word, (Complete Matching) $M_i = \prod_{j=0}^{n-1} (F_{ij} \odot A_j)$

A(0,0) — M ← R   Column-Major
H ← W   Order

Block 0

A(0,0)
A(1,0)
A(2,0)
A(3,0)

A(0,1) — M ← R   A(0,0)
H ← W   A(0,1)
400 words
A(0,2) occupies

A(8,0)
A(9,0)
A(0,1)
A(1,1)

Hits : 100

If the array is arranged in Column major order for every element, a block is moved from main memory to cache, total blocks moved are 100. For every block two references are made, one of them result Hit. No. of Hits = 100.

Hit ratio = 0.5 (50%)

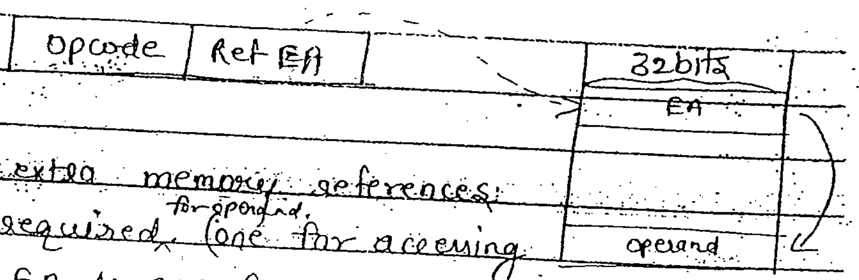
→ Requires 1- extra memomory Ref.

Adv. Range of operand is limited ~~or restricted~~ by memory word size not by instr. length.
Static variables can be accessed.

peroand)

DisAdv- Limited Range of Addr.

(3) Indirect Addressing Mode.



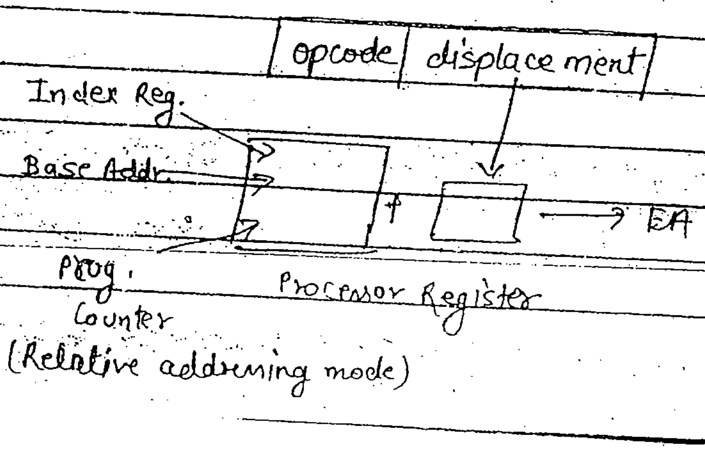
needed.

→ Two extra memory references are required (one for accessing Ref-EA & one for operand) (for operand)
→ Single indirectn requires 2 memory references
Hence, n-Indirectn requires n+1 extra memory ref.
→ Can access Local variable

2coz the
ed).

(4) Computable Addressing mode

addressn



~~32 bits~~
~~Words~~
~~1 to $2^{31}-1$~~
to $2^{31}-1$

1. Non - Computable Addressing Mode
1(effective)E Addr. is to be obtained.

2. Computatable Addressing mode
EA [(effective Addr.) + Address of operand]
is to be computed.

1

Non - computable Addressing mode

2 (1) Immediate Addressing mode

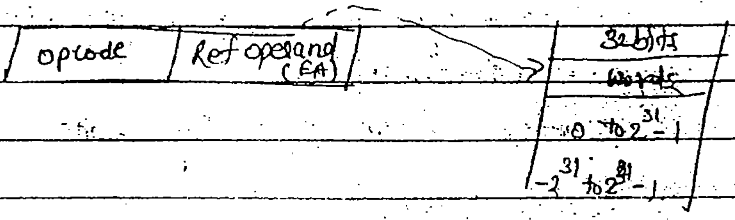
opcode	opreand

→ No extra memory ref. for operand is needed.
→ Fastest.
→ Suitable for accessing constants.

Dis - Limited Range of operand (bcoz the field given to operand is fixed).

(2) Direct Addressing mode

Gives ref. of operand = effective address (Addr.)
Known as



A
Dis
(3)
Hence,
(4)
Index
Base
Prog Coun
(Relati

112
128
4

$P \rightarrow P_{max}$ iff $Q \rightarrow Q_{min}$.

384 | $P \rightarrow P_{min}$ iff $Q \rightarrow Q_{max}$

Q_{min} & Q_{max} here, $1 \le Q \le 3$

6 instr. | min. $Q=1$ $\quad P + 1 \times 2^7 = 2^9$

$P = 384$

max. $Q=3$ $\quad P + 3 \times 128 = 512$

$P = 512 - 384$

$P = 128$

1-address bits are ence of idatn of ed 1-Add.

$$\underset{\text{Zero Addr. instr.}}{P} + \underset{\substack{\text{no. of 1-Addr. instr.} \\ \text{(Invalid zero address inst}^n\text{ due to the presence of 1-Address instr.)}}}{Q} \times 2^7 + \underset{\substack{\text{no. of 2-Addr. instr.} \\ \text{(Invalid zero Address instr. due to the presence of 2-Address instr.)}}}{R} \times 2^{14} = 2^{16}$$

– Total possible zero-Addr.-instr.

address instr.

ence of instr.)

Addressing Modes

"The way reference of operand is given in the instr."

→ The addressing mode provides flexibility ~~afor~~ for program constructn

→ CISC offer more addressing modes.

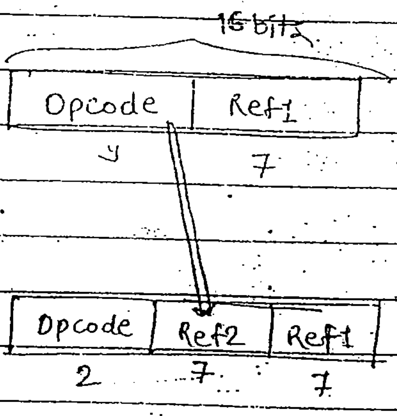
by the above machine?

(a) 128, 512 (b) 0, 384 [✓ (c) 128, 384]
(d) 256, 512

Solⁿ-

For
1 Address -

For
2 - Address



To accomodate 1-2-address instr. 2^7 1-address instr. are to be sacrificed (As 7-bits are allocated from opcode). The presence of 2 - 2 Address instr. results invalidatⁿ of 256 1-Address instr. Thus, Valid 1-Add. instr. are 512 − 256 ⇒ 256 instr.

$\underbrace{P}_{\text{Valid 1-Addr. instr.}} + \underbrace{Q \cdot 2^7}_{\text{Invalid 1-Addr. Instr.}} = 2^9$ Total one address instr.

(due to the presence of Q Two-address instr.)

Q = 2 , P = ?

$P = 2^9 - 2^8 = 256$

stn one after

256 instr.

1024 Words

023

included)

)

ts)

r. } Adv.
ey.

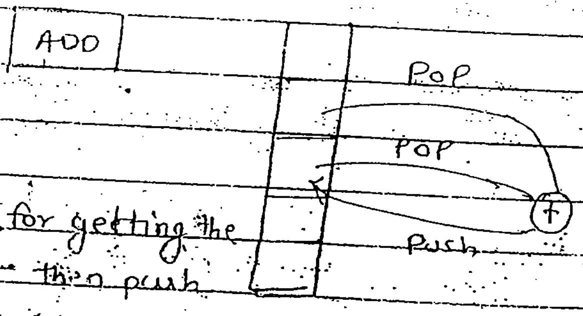
will permanently

$A = [A] + M[A_1]$

→ Smallest length (Adv.)

→ More cost (disadv.)

→ Zero-Address instr. (or stack Addressing)



→ Uses Stacks.

→ Perform pop (for getting the operands) first then push

→ Adv. - CPI < 1
↓
Clock pulse Instr. (getting little faster)

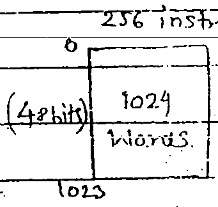
Disadv. - More Complexity in the implementat?
- Rigid requirement on stack requirements (placing proper

Q. A Hypothetical system support 1-Address & 2-address instr. The 16-bit instr. is stored in 128-word memory. If there exis 2-2 address instr. (A) what will be the no. 1-Address instr. supported by the machines?

(a) 128 bits
~~(b)~~ 256 "
(c) 384 "
(d) 512 "

(B) What will be the minm/maxm 1-Address instr. supports

Prog - carrying sequence of instⁿ one after another



4 Address ~~Opts~~ instructⁿ

→ Adv - Simplicity

disAdv - Too lengthy

→ 3 - Address instr. (Prog. Counter included)

$\boxed{ADD}\boxed{A_1}\boxed{A_2}\boxed{A_3}$ + PC (36 bits)

→ length become Reduced (Adv.)

Cost increased (disAdv)

→ 2 - Address Instr.

$\boxed{ADD}\boxed{A_1}\boxed{A_2}$ + PC

$M[A_1] \leftarrow M[A_1] + M[A_2]$ (28 bits)

→ Faster than above 2-Address instr. }

→ Length Reduced more than 3-address. } Adv.

→ Overwriting the operand & Result will permanently lost. (disAdv

→ 1-Address Instr

PC, Accumulator $\boxed{ADD}\boxed{A_1}$ $A = [A] + M[A_1]$

→ Use

→ Per

Oper

→ Ac

Di

Q. A

2-

st

2-

1-

(a)

~~(b)~~

(c)

(d)

(B)

$X - 1$

X + 2's complement

001
110

$\boxed{111} = X - 1$

Z_0

Z_1

E.g. $M_1 M_2 = 01$

$Z = X + 0001$

$Z = X + 1$

Z_2

$M_2 = 1$, all the Adder gives no.

Z_3

E.g. $M_1 M_2 = 11$

$Z = X + \bar{Y} + 1$

$= X - Y$

Instructns, Addressing Mode, μ-operatn - - - - -

• Execu
Finished

Instructns
- opcode ↓
- (Operand) Ref. OPR

Classificatn based on func -
- Data Xfer
- Arithmetic
- logical
- Branching & Condn

Number of references
4 - Address Instructn (No PC)

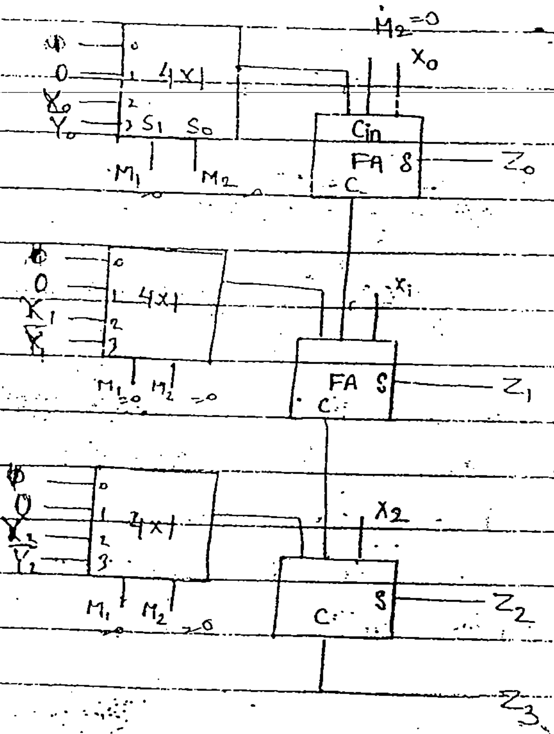
ADD	A_1	A_2	A_3	A_4

$M[A_1] \leftarrow M[A_2] + M[A_3]$

$A_4 \rightarrow$ Null bcoz
Using Linked List.
Ref to nxt instructn (Selfreferencing)

put
$M_1 = 00$, then
et 1 to all
its
ratn therefore,

21-09-10



M_1 M_2	Operat^n $\Rightarrow Z$
0 0	X−1
0 1	X+1
1 0	X+Y
1 1	X−Y

Eg. $M_1 M_2 = 0 0$ (6) 110 − Input

111 − $M_1 M_2 = 00$, then

(5) 101 (5) add 1 to all bits

∴ When add anything it performs subtract^n therefore, X−1.

generated using

$T_{cg} = 3T_g$

$T_{sg} = T_{EX-OR}$

$T_{CLA} = 3T_g + T_{EX-OR}$

$= 4T_g \; (= T_g)$

$= 6T_g \; (= 3T_g)$

Y_0

$X_0 + Y_0$

$+ C_{in}(P_0)$

$C_0 P_i$

$+ C_{i-1} P_i$

$X_i + Y_i$

Q. Consider a 16-bit addition, here the behav of EX-OR to similar to that of any other gate & consumes 5 ns. Wat is the time taken with-

① 16-bit CLA

② 8-bit CLA followed by 4-bit CLA.

(1)

$$\boxed{\text{16-bit CLA}} \xrightarrow{4T_g} = 20ns$$

Costliest - More h/w require bcoz gate delay is ~~far~~ less

9

(2)

$$\boxed{\text{8bit CLA}} \rightarrow \boxed{\text{4-bit CLA}} \rightarrow 40ns$$

$4T_g + 4T_g = 8T_g \Rightarrow 8 \times 5 \Rightarrow 40ns$

$P_i, G'_n)$

(3)

$$\boxed{\text{6-bit CLA}} \rightarrow \boxed{\text{6bit CLA}} \rightarrow 40ns$$

$4T_g + 4T_g = 8T_g = 8 \times 5 \Rightarrow 40ns.$

(4)

$$\boxed{\text{4bit CLA}} \rightarrow \boxed{\text{4bit CLA}} \rightarrow \boxed{\text{4-bit CLA}} \rightarrow 60ns$$

(cheapest)

$4T_g + 4T_g + 4T_g = 12T_g \Rightarrow 12 \times 5 \Rightarrow 60ns$

less h/w required, bcoz gate delay is high

Carry Look Ahead Adder

- The CLA provides constant time for addition.
- It contains carry generatn stage & sum generatn stage.

$$T_{CLA} = T_{cg} + T_{sg} \underbrace{(4T_g / 6T_g)}_{\text{gate delays}}$$

- It improves the speed of addition.
- With CLA, the performance of ALU will be increased.
- The CLA requires more h/w. Hence, it is the costliest adder.

$$\text{H/w Complexity} = O(2^n)$$

Limitatn of CLA -

- The fan-in constraints of the gate violates constant time requirement.
- The combinatn of CLA or RCA (Ripple carry Adder) is used for larger size operands.

ted, for
exponent
wires
floating
pipelining).
ires
Smaller
e, variable
a^3
erat^n (addit^n)
malizat^n

$$X = X_s(\cdot X_m)_2 * 2^{E_X - Bias}$$

$$Y = Y_s(\cdot Y_m)_2 * 2^{E_Y - Bias}$$

$X * Y = Z$, Z_s = sign bit
Z_m = Mantissa

$$Z = Z_s(\cdot Z_m)_2 * 2^{E_Z - Bias}$$

$$Z = (X_s \oplus Y_s)$$

if operands sign are same, ~~there~~ ~~wi~~ operat^n resul ~~with~~ zero.

$$Z_m = X_m * Y_m$$
$$E_Z = E_X + E_Y$$

$$\boxed{Z = (X_s \oplus Y_s)(\cdot X_m * \cdot Y_m)_2 \; 2^{E_X + E_Y - 2Bias}}$$

in Exponent Bias field.
Bias

for $E_X + E_Y$, we add external Bias,
It will be, E_Z - Bias
for the Biased exponent field, in Subtract Biased will be subtracted.

$$Z = X \overset{\text{subtract}^n}{/} Y = Z_s(\cdot Z_m)_2 * Z^{E_Z - Bias}$$

Floating Pt Arithmetic

- The floating pt. arithmetic implemented for the operands which are in biased exponent form.
- The addition - subtractn process requires
(i) Exponent Alignment
(ii) Operatn (Add. / Sub.)
(iii) Normalizatn of the result.

- To improve the speed of computatn, floating pt. pipelines are used (functional pipelining).
- The multiplicatn and division requires correctn to the biased exponent.

Exponent Alignment - Equating the smaller variable to the larger variable.

0.05×10^3

$5 \times 10^{-1} \Rightarrow \cancel{x} \times 10^3$

$.5 \times 10^0,\ 0.05 \times 10^1,\ 0.005 \times 10^2,\ 0.0005 \times 10^3$

Ans - 0.0505×10^3 // After operatn (additn)

$\boxed{.505 \times 10^2}$ // After Normalizatn

, no any

n-bit
divisor

Successful $n/2$ If unsuccessful, restore it.

$n/2$
↓ Restoratn
Addition

ng ~~[illegible]~~
oratn div

$\frac{n}{2}$, $\frac{n}{2}$ + $\xrightarrow{\text{Restoratn}}$ $\frac{n}{2}$ (addition)

Successful   Unsuccessful

& consumed

$\frac{3n}{2}$ (addition / Subtractn).

pe

Non-Restoratn

operatns

Dividend digit
&
Division digit

LQ

compared before any subtractn.
// If dividend digit > divisor digit, then subtractn is performed otherwise not. ~~[illegible]~~ only n operatn will be performed in this.

toratn is

// for all 0's & all 1's ie. 000 or 111, no any operatn will performed

~~Examp~~

FIXED POINT DIVISION

The fixed pt. div. is implemented using ~~one~~ 1. Restoratn division algo. 2. non-restoratn div. algo.

- The Restoratn div. algo. is simple & consumed $3n/2$ addition / Subtractns on average.
- The non-Restoratn requires only n operatns but consumes more h/w.

Restoratn

$$V = Q*D + R$$

$$D\overline{)V}(Q$$
$$R$$

Restore

$$\begin{array}{r} 87 \\ 6 \\ \hline -3 \\ +6 \\ \hline \end{array}$$

getting the dividend back if the restoratn is not succesful.

Adv.

① Sign of the multiplicatn process is protected.

② It enables to reuse the result. (A reused MSB result & Q reused LSB result)

③ It is placed the current multiplier bit in Q position.

0111 (7)

Operatn

A ← A − M

A. R. Shift AQQ₋₁ (7)

A.R.S. AQQ₋₁ (7)

A ← A+M

A. Right S. AQQ₋₁

hift

the sign

-64 32 16 8 4 2 1
1101011 = $(-21)_{10}$

// Count denotes How many A.R.S. will performed

The Above Booth's Algo is called as Radix-2 Booth's Algo. In the Radix-4 Booth's Algo, 3-multiplier Bits are considered and based on their pattern the arithmetic operatns A ± 2M or A ± M are performed.

If All the 3-bits are same

$\boxed{Q_1 Q_0 Q_{-1} = 000 \text{ or } 111}$

then there is no Arithmetic operatn is performed.

If $Q_1 Q_0 Q_{-1} = 011$ $A \leftarrow A+2M$

$Q_1 Q_0 Q_{-1} = 001$ } $A \leftarrow A+M$
010 }

If complement the above bits

$Q_1 Q_0 Q_{-1} = 100$ $A \leftarrow A-2M$

$Q_1 Q_0 Q_{-1} = 110 \text{ or } 10\phi$ $A \leftarrow A-M$

$Q_2\,Q_1\,Q_0\,Q_{-1}$

1001 * $\boxed{0}\boxed{1}\boxed{1}\boxed{0}$

1001 ← $M * 2^0$

1001 ← $M * 2^1$

0000 ← 0

$-7 \times 3 = (-21)_{10}$

$M = 1001 (-7)$ (2's complement of M) $(M) = 0111 (7)$

Count	A	Q	Q_{-1}	$Q_0 Q_{-1}$	Operatn
3	0000	011	0	10	A ← A − M //
	0111	011	0		A. R. Shift AQQ_{-1} (7)
2	0011	101	1		
2	0011	101	1	11	A.R.S. AQQ_{-1} (7)
1	0001	110	1		
1	0001	110	1	01	A ← A + M
	1001				
	1010	110	1		A. Right S. AQQ_{-1}
0	$\boxed{1101}$	$\boxed{011}$	0		

0100 +4

0010 +2 // right shift

Sign bit → $\textcircled{1}$100 (−4)

will not change (Should protected) $\textcircled{1}$110 (−2)

// Arithmetic right shift preserved the sign bit.

19|09|10

arst the

pattern
oth's Algo.

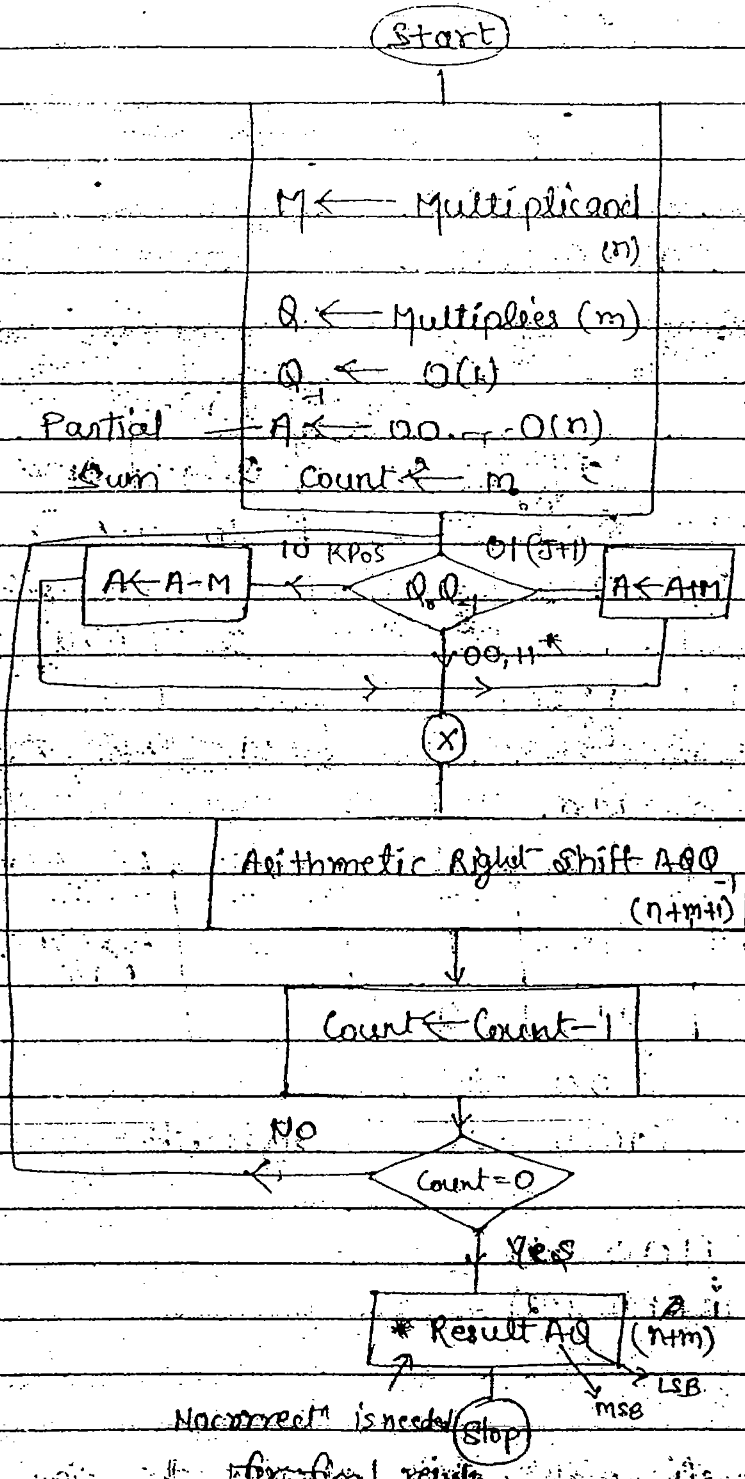
lock

lock

e Q(d)

shift
8
2
6
8

patterns



// LSB $\xrightarrow{to}$ MSB is the ~~process~~ direction of multiplication

// The more the blocks of 1's, the worst the performance with booths Algorithm.

e.g. Which of the following multiplier pattern gives the best performance with the booth's Algo.

```
     7 6 5 4   3 2 1 0  pos
(a)  0 1 1 1   1 1 1 0   // having 1 complete block.
(b)  1 1 1 1   1 1 0 0   // partial Block
(c)  0 0 1 1   1 1 1 1   // 1 Complete block
(d)  1 1 1 0   0 1 1 1   // 1 complete, 1 partial block
```

a (b) is giving the best performance & (d) is giving the worst performance.

		Add	Sub	shift
(a)	$2^7 - 2^1 = +126$	1	1	8
(b)	$-2^2 = -4$	0	1	2
(c)	$2^6 - 2^0 \Rightarrow 63$	1	1	6
(d)	$(+2^3 - 2^0) + (-2^5) \Rightarrow 8 - 1 - 32$ $\Rightarrow -25$	1	2	8

Least performance = (d), (a), (c), (b)
(in ~~ascending~~ ascending order)

Ascending order performance of multiplier patterns, wrt booth's algo.

Booth's Algorithm

ng

be

— Notation

— Recording Process

— Algorithm.

substract^n.

↳ Features

~~0110~~

(-ve or +ve) Sign bit $\underline{0}$ 111 1110 +ve no.

= 126

7 6 5 4 3 2 1 0 Position (Pos)

$0\boxed{111\ 111}0 = 2^{J+1} - 2^{K}$ // 1-addition, 1-substract, 8-shift Operat^n.

J = 6, K = 1

$= 2^{6+1} - 2^{1}$

$= 2^{7} - 2$

$= 128 - 2$

$= 126$

OR

$2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 = (126)_{10}$ // Conventional Method.

conventional

6 additions, 21 shift operat^{ns}

only

e. done

3 2 1 0

$0101 \Rightarrow 2^2 + 2^1 \Rightarrow (+5)_{10}$

2 additions, 3 shift operat^n

~~3 2 1 0~~ Pos

$0\boxed{1}0\boxed{1} = +2^{J_2+1} - 2^{K_2} + 2^{J_1+1} - 2^{K_1}$

$J_2\ K_2\ J_1\ K_1$

2 2 0 0

$= 2^3 - 2^2 + 2^1 - 2^0$

$= (+5)_{10}$

operat^{ns} & additions Booth's reporting pattern is convenient for converting the signed no. into decimal.

- Adding the two complement no's are substract^{n}.

```
   (-6)        3
   (10)        3
   1010  *  0011
   -------------

     1010
    1010
   0000
  0000
   0011110


          32 16 8 4 2 1
        0 0 1 1 1 1 0   = 30
```

OR

- We cannot use $\overset{\text{Normal}}{\text{multiplicat}^{n}}$ or conventional multiplicat^{n} for signed arithmetic. only addition & ~~divi~~ Substract^{n} can be done

&

exceeds

Saturatⁿ Arithmetic -

- for this 0111 ⇒ 7

Results taken betⁿ -7 to +7 & -8 to +7

```
 0110
 0010
-----
  000
```

ving.

Result

- In satⁿ arithmetic, whenever the results exceeds it will be set to largest represe table value, no correctⁿ to the overflow

- In Non-saturatⁿ Arithmetic, correctⁿ will be performed in case of overflow, Here the result is wrap-around with more bits.

- The Addition & Subtractⁿ can be extended for both signed no's & unsigned no's ; if signed no's are denoted to 2's complement notatⁿ. These is not possible for multiplica & division. The sign bit is getting disturbed in the process of multiplicatⁿ & division.

- Booth's Algo. Is proposed for implementing fixed point signed multiplicatⁿ. The booth's notatⁿ is similar to that of 2's complemen Notatⁿ but reduced effective no of shift

Whatever be the Notatⁿ, if addition exceeds the limit -127 to +127 or
-128 to 127, then
There is a overflow.

Correctⁿ to the overflow-
In the presence of overflow, the following correctⁿ is to be taken,
(1) Complement the sign bit of the Result
(2) Add + or -2^{n-1} to the Result.

```
  0110 0000
  0100 0001
  ---------
  1010 0001
Complement(1)
  0010 0001 = +33
```

Correctⁿ ⇒ (+33) + 128

```
  33
 128
 ---
 161
```

Arithmetic —
(1) Saturation Arithmetic
(2) Non-Saturation (Wrap-around)

Results

ins=1) = Cins=1
outs=1) Couts=1
plement | 2's Complem.
s=2s

2s=1 | Cins=1 ✓ Cout=0

① 1111 1111 ⇒ (−0)

EAC End Around Carry ① 1111 1111 ⇒ (−0)

1111 1110

→ 1

1111 1111 ⇒ (−0)

② 0111 1111
0000 0000
1111 1111

③ 64 32 16 8 4 2 1
0110 0000 96
0100 0001 65
1010 0001 161 beyond the range −127 to +127

If Add the no's of diff. sign there ~~are~~ no overflow.
Ans- Resulting overflow.

④

If 2's complement ignored EAC & its range is −128 to 127

① 1111 1111 −1
EAC ① ← 1111 1111 +1
1111 1110 −2

② 0110 0000
X ← 0100 0001
1010 0001

Q Which of the following sign arithmetic Results overflow ? (Signed Magnitude Notatⁿ) $\left(\begin{matrix} C_{in_s} = 1 \\ C_{out_s} = 1 \end{matrix}\right)$ $\begin{matrix} C_{in_s} \neq \\ C_{out_s} \end{matrix}$

	Signed M N	1's Complement	2's Complem
① 1111 1111 + 1111 1111	✓	$X_s = Y_s = Z_s$ —	—
② 0111 1111 + 1000 0000	?	—	—
③ 0110 0000 + 0100 0001	✓	$X_s = Y_s = 0$ ✓ $Z_s = 1$	C_{in_s} ✓ $C_{out_s} = 0$
④ 0000 0000 + 1111 1111	—	—	

① – Sign position

```
   1111 1111  }  Signed
 ①←1111 1111  }  Magnitude
   1111 1110  }
```

②

```
  0111 1111
  1000 0000
  1111 1111
```

③

```
   110 0000
 + 100 0001
  0010 0001
```

④

```
  111 1111
  000 0000
  111 1111
```

1's Complement

4 bits

Unsigned Number Range : 0 to 2^4-1 ⇒ 0 to 15.

$$\begin{array}{r} 1010 \\ \text{Carry} \rightarrow \textcircled{1}\ \underline{0110} \\ 0000 \end{array}$$

Signed Number Range : −7 to +7 / −8 to +7

4 + 5

0100	X_S
0101	Y_S
1001	Z_S

→ Add two +ve no. results −ve no., then it is overflow.

X_S	Y_S	Z_S	V
0	0	1	1
1	1	0	1

Range

$$V(X_S\,Y_S\,Z_S) = X_S'Y_S'Z_S + X_SY_SZ_S'$$

C_{ins} ← Carry into sign bit

C_{outs} ← Carry out of sign bit

$$V(C_{ins}, C_{outs}) = C_{ins} \mp C_{outs}$$
$$= C_{ins} \oplus C_{outs}$$

~~into sign~~
denote overflow
is having opp. sign
ry bits into
ions & out of
are not same.

Fixed point Arithmetic

— Boolean Notation Booths Notation
— Booths Algorithm

Arithmetic
- Addition
- Substraction
- Multiplicat^n
- Division

2's complement Notat^n

unsigned		signed (-8,4,2,1)
(10)	1010	-6
1	0001	1
(11)	← 1011 →	-5

Arithmetic addition ~~(Signed)~~ can Exceed the Range
It may be
overflow
correction for overflow

Un-Signed Numbers - If arithmetic result carry → overflow

Signed Numbers
- Signed Magnitude Notation - Carry into sign bit denote overflow
- 1's/2's complement Notation
  - (i) Operands of sign & result is having opp. sign
  - (ii) Carry bits into sign-positions & out of sign position are not same.

ion $M = 01110. \quad 0$

ve format $E - 127 = 3$

$E = (130)_{10}$

=0 $= 10000010$

0

128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	0

tes

si denotes

.0

-4E3

Q. Consider, IEEE 754, Single precision format, the following 32 bit no. is interpreted as —— value with the above format

S Bias M

32 bit No. – $1\ \underbrace{1000\ 0011}\ \underbrace{11000\ldots\ldots 0}$

31 30 ~~6~~ 23 22 0

MSB $E = (131)_{10}$

~~is bet~~

As $E \neq 0$ to 255 ; this pattern denotes implicit normalized no.

$V = (-1)^1 (1 + .75) * 2^{131-127}$

$(.1100\ldots0)$

$= -(1.75) \times 2^4$

$= -1.75$ ~~* 26~~

$= -$ ~~(28)~~ $-(28)_{10}$

Q. What is the hexadecimal pattern denotes 11.5 in IEEE 754 format ?

0 10000010 01110..0

S	E	M

~~4F3~~

0 x 41380000

$(11.5)_{10} = (1011.1)_2$

$= (1.0111) * 2^3$

4%
floating
ored in
on
ating
ying
be
t norma-
exponent
)(not a no.)

Single Precision (32 bits) Base : 2

S (1)	E (8)	M (23)	Value
0/1	000...0	000...0	±0
0/1	111...1	000...0	±∞
0/1	If E≠0 & E≠255 ↓	xxx...x	Implicit normalized no. $V=(-1)^S(\overset{\text{implied bit}}{1}.M)_2 * 2^{E-\cancel{1}127}$
0/1	E=0,	M≠0	fractional form $V=(-1)(.f)_2 * 2^{-126}$
~~[illegible]~~ other combinatⁿ	E=1	M≠0	Not a Number (NAN)

// Bias is not 128, it is 127 ~~bog~~ Certain special patterns are used for exponent therefore, Bias is reduced.

// E is not used in fractional part ~~[illegible]~~

Percentage of error

$67 \rightarrow 2$

$100 \rightarrow \frac{100 \times 2}{67}$ %

$\frac{200}{67} \rightarrow 4.4\%$

IEEE 754

→ It provides the standards for floating point no.

→ The floating point no. is stored in either a single precision (32 bits) or in double precision (64 bits).

→ It gives the provision for denoting $\pm$ or ± 0 and $\pm\infty$ by sacrifying certain mantissa Exponent pattern.

→ The floating point no. is can be denoted ~~with~~ either with implicit normalizatn or fractional form.

→ Certain combinatns of mantissa exponent doesn't denote any no. (a(NAN) (not a no.)

$M = (1111)_2$

$)_8$

$E - 8 = 1$

base 8,

$E = 9$

$(37)_{10}$

32 8 4 3 2 1

1 0 1 0 0 1

$(67)_{10}$

64 32 8 4 3 2 1

1 0 0 0 0 1 1

$(.\underbrace{001}\underbrace{000}\underbrace{011})_2 \times 2^9$

$(.103)_8 \times 8^3$

in the

$E - 8 = 3$

$E = 11 = (1011)_2$

$(237)_8$

$M = 0010$

$\overbrace{0}\ \overbrace{1011}\ \overbrace{0010}$

S	E	M

$= (262)_8 = (67)_{10}$

8 $\frac{1}{7}+\frac{1}{64}$ $\frac{8+1}{64}$ $=\frac{9}{64}$ 0.5 + 0.2 (0.7) 8 8

0	1010	1000	
S	E	M	$=(250)_8$

(4) It is normalized w.r.t. to base 8,

S = 0 as the number is +ve,

$$(\cdot 1875)_{10} \times 2^5 = (\cdot 0011)_2 \times 2^5$$
$$= (\cdot 011)_2 \times 2^4$$
$$= (\cdot 11)_2 \times 2^3$$
$$= (\cdot 6)_8 \times 8^1$$

$\Rightarrow$ $E-8=1$ $\Rightarrow$ $E=9=(1001)_2$
$M=(1100)_2$

0	1001	1100	
S	E	M	$(234)_8$

Q. What is the pattern for 7.5 in the above register ?

S	E	M	$=(237)_8$
0	1001	1111	

$(7.5)_{10} = (111.1)_2 \Rightarrow (\cdot 1111)_2 \times 2^3$
$(\cdot 1111)_2 \times 8^1$

oint

overflow

max $= 2^{63}$ can't overlap here.

rs.

en it

uniform

ery close

towards

towards

x^m value

ude fract^n

(3) What is the pattern with implicit normalize

$$V = (-1)^S (1.M)_2 \times 2^{E-Bias}$$

(4) Pattern if Base of the system is 8.

$$V = (-1)^S (.M)_8 \times 8^{E-Bias}$$

Excess - 8 Exponent

(i)(2) S=0 as the number is +ve

$(0.1875)_{10} \times 2^5 = (.0011)_2 \times 2^5$

$(.11)_2 \times (2)^{-2} \times 2^5 = (.11)_2 \times 2^3$

$= \cancel{(.1100)_2 \times 2^3}$

M = 1100

$E-8=3 \Rightarrow E = (11)_{10} = (1011)_2$

E=11

0	1011	1100
S	E	M

(ii) $\Rightarrow (274)_8$

(iii)(3) S=0 as the number is +ve,

$(.1875)_{10} \times 2^5 = (.0011)_2 \times 2^5$

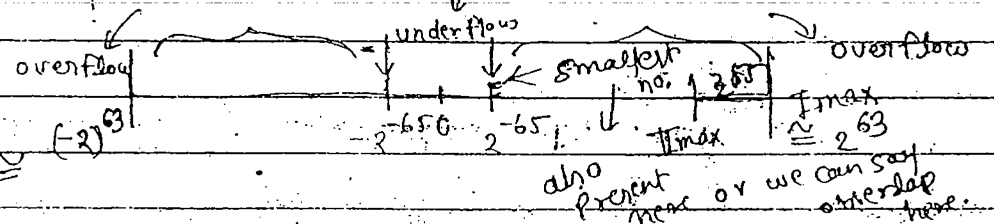
$= (1.1)_2 \times 2^{-3} \times 2^5$

$= (1.1)_2 \times 2^2$

M = ~~1.1~~ 1000

$E - 8 = 2 \Rightarrow E = (10)_{10} = (1010)_2$

The No. distribum of floating point representatn



floating point repr. is not continuous.

//if Value comes between -2^{-65} to 2^{+65} then it is underflow.

→ 0 Cannot be covered.

→ The no. distributn is also not uniform the distance betn the no's is very close towards zero & widely spread towards maxm Value

→ The effect of error is negligible towards zero & it is dominents towards maxm value

(4)

Q. | S (1) | E (4) | M (4) |

Mantissa is Normalized Sign Magnitude fractn
Exponent in Biased form
Base : 2

(1) Expression
(2) Pattern for $\pm$ ~~0.375~~ 0.1875 $* 2^5$
(Octal)

$V_{max} = (-1)^0 (1-2^{-8}) * 2^{127-64}$

$V_{max} = (1-2^{-8}) * 2^{63} \cong 2^{63} (2^{63} - 2^{55})$

truncated 2nd largest No.

what

$\underset{S}{0}\ \underbrace{1111110}_{E}\ \underbrace{1111110}_{M}$

05) to

$\hookrightarrow 127 \qquad 1-2^{-7}$

$V_{max} = (-1)^0 (1-2^{-7}) * 2^{127-64}$

$= (1-2^{-7}) * 2^{63}$

$\therefore V_{max} = 2^{63} - 2^{56}$

Difference betn 1st & 2nd largest No.

$V_{max} = (2^{63} - 2^{55}) - (2^{63} - 2^{56})$

$= 2^{56} - 2^{55}$

bits $V_{max} = 2^{56}$

maxm

Pattern for smallest positive No.

$\underset{S}{0}\ \underset{E}{0000000}\ \underset{M}{10000000}$

$V_{min} = (-1)^0 (0.5) * 2^{0-64}$

$= 2^{-65}$

1	1000111	1101111
0X	C 7 D F	
S	E	M

Since one of the mantissa bit is truncated the repre. is having the error. What was stored in the system is $-(111.25)$

Percentage of error is equal to

$-111.75 \quad = -0.25$

$100 \quad = \frac{100 \times 0.25}{111.75}$ ~~2235~~ 447

$= \frac{2500}{11175}\%$

$= \frac{100}{447}\%$

$= 0.22\%$

(c) Sign should be zero & all the bits are 1 then the value will be max^m.

(d) Pattern for max^m value is

1st largest (+ve) No.

$\underset{S}{0} \quad \underbrace{1111111}_{E} \quad \underbrace{11111111}_{M}$

E = 127

$(1-2^{-8})$

ent the
many
ting
for

ister
no.
agnitude
Base

ing the
represent
in the

(1) $V = (-1)^S (.M)_B * B^{E-Bias}$

expression $V = (-1)^S (.M)_2 * 2^{E-64}$

$Bias = 64 = 2^{K-1}$

$2^6 = 2^{K-1}$

$K-1 = 6$

$K = 7$ bits

S	E	M
1	7	8

$0 \le E \le 2^7 \Rightarrow 127$

(2) ~~(a)~~ $-(111.75)_{10}$

$S = 1$ ('−' sign) or (as the number is −ve)

$(111.75)_{10}$

Weight of (111.75)	64	32	16	8	4	2	1	0.5	0.25
	1	1	0	1	1	1	1	1	1

Integer portion | Fract^n Portion

$(.110111111)_2 * 2^7$ → How many places (.) is shifted

$E - 64 = 7$

$E = (71)_{10} = (1000111)_2$

$M = 11011111$ → truncated bcoz M has only 8 bits.

The above representatⁿ cannot represent the value '0' which is required for many computation. The IEEE 754, floating point standard has a provision for (± 0 & $\pm\infty$)

4. IBM machine base is 16.
4. Intel " " " 2.

Q. Consider the following 16 bit register representing the floating point no. with mantissa in normalized sign magnitude form. Exponent in excess-64 form. Base of the system is 2.

S	E (Biased exponent)	M

(a) What is the expression that is giving the value 2?
(b) What is the 16-bit pattern that represent the value $-(111.75)_{10}$?
(c) What is the largest value stored in the above register?

4. FLOATING POINT REPRESENTATION

256 MB
64 KB
size of
used for

The floating point no contains integer portion & fractional portion. In the system, It is stored as mantissa & exponent pair. Most of the systems represent the mantissa in normalized in sign-magnitude fract. The exponent is denoted in Biased form. The biased exponent is an unsigned no. which denotes signed true exponent (unsigned no.) If the Biased field is having K bits, then

$\boxed{\text{biased} = 2^{K-1}}$

S	E (K)	M

Mantissa & exponent pair (± M, ± E)

Value of the no. is

$$V = (-1)^S (\cdot M)_B * B^{E-\text{Bias}}$$ (Explicit Normalize)

where, B = Base of the system
(It is not stored in Memory)

The implicit normalizatⁿ increases the accuracy & value of expression is

$$V = (-1)^S \underset{\uparrow}{(1.M)_B} * B^{E\text{-Bias}}$$
"Implied bit"

~~x12~~ 3
2
12 ⇒ $2^{13} \times 12$
×3 ⇒ 96 KB

// Hard disk is the virtual memory.

Q. Consider a 32-bit VA which refer 256 MB MM. Both are partitioned into 64 KB pages / frames. What will be the size of page table in bytes if paging is used for Address translatⁿ ?

VA = 32 bit

pages / frames = 64 KB

$= 2^{16}$

Word offset = 16 bits

page offset = 16 bits

MM = 256 MB

$= 2^{28}$ ~~16~~

MM = 28 bits

No. of frames = $\frac{2^{28}}{2^{16}}$

$= 2^{12}$

No. of frames = 4 KB

page table contains 2^{16} words.

Page table size (in bytes) = $\frac{2^{\cancel{16}} \times \cancel{12}^{3}}{\cancel{8}_{2}}$

$= \frac{2^{16} \times 12}{8} \Rightarrow 2^{13} \times 12$

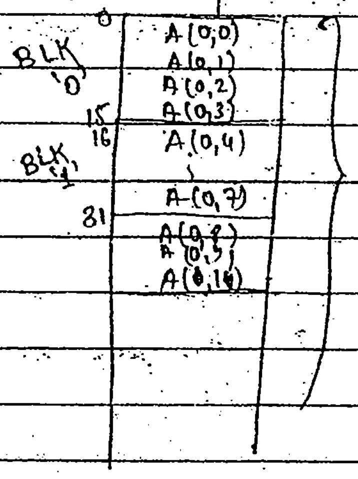
$= 2^{15} \times 3 \Rightarrow 96$ KB

Q. Consider a direct-mapped cache with 64-words & the MM has to accomodate 10 x 10 float array. Each floating point element occupies 4 words, the cache & main memory are partitioned into 16 word blocks. What will be the no. of hits if the following prog. segment is executed?

Let the array is stored in Row major order

```
float [10] [10]
for ( i=0 ; i<10, i++)
    for (j=0, j<10, j++)
        A[i] [j] = A[i][j] +2;
```

row major order



400 words
A (10x10)
float array

10 elements in each row.

ache
che
ts of
cks are

29, 63, 8,

block is
LRU

n (K Mod S)

3 159
3
↑

73 92
1 0
↑

Q. Consider a 2-way set-associative cache with 4-blocks the following MM reference are made, what will be the min. no. of faults with LRU strategy?

4 8 5 12 4 5 8

Assume that cache is empty initially.

(a) 4 (b) 5 [✓ (c) 6] (d) 7

P = 2, N = 4 blocks

No. of set = $\frac{4}{2}$ = 2

set 0	
set 1	

K Mod 2 set

$$\begin{array}{ccccccc} F & F & F & F & H & H & F \\ 4 & 8 & 5 & 12 & 4 & 5 & 8 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}$$

Q. Consider a 4-way set-associative cache which was empty initially the cache contains 16 blocks and ~~M~~M consists of 256 Blocks. Let the following MM blocks are referred.

0, 255, 1, 4, 3, 8, 133, 159, 216, 129, 63, 8, 48, 32, 73, 92, 155

Which of the following main memory block is not available in cache at the end of LRU replacement policy?

(a) 3 (b) 8 (c) 129 [✓ (d) 216]

P = 4 , N = 16 blocks

No. of Sets s = 4

K^{th} block of MM has to placed in (K Mod S)

K Mod 4 Set.

~~0~~ ~~255~~

	0	255	1	4	3	8	133	159
KMod 4	0	3	1	0	3	0	1	3
	↑	↑	↑	↑	↑	↑	↑	↑
	216	129	63	8	48	32	73	92
	0	1	3	0	0	0	1	0
	↑	↑	↑	Hit			↑	
	155							
	3							
	↑							

LRU : 4, 5, 7, 12, $\boxed{4}$, $\boxed{5}$, 13, $\boxed{4}$, $\boxed{5}$, 7, 12, 13
Hit Hit Hit Hit

replacement

Block present : 5, 7, 12, 13, ~~5, 7, 12, 13~~,
last Replacement : 4
Hits : 4

During MM
nent

Direct Mapping :

9

K Mod N = K Mod 4

4, 5 7, 12, 13

4	5	7	12	4	$\boxed{5}$	13	4	5	$\boxed{7}$	12	13
0	1	3	0	0	1	1	0	1	3	0	1
					Hit		Hit		Hit		

ost Recent
eference.

Hits = 3
Blocks present - 12, 13, __, 7
Last replacement - 5

, 12, 13

The ascending order of the performance of mapping scheme w.r.t the above problem:

FIFO, Direct Mapped Cache, LRU.

(iii) Direct Mapping

(K Mod N)th Block is chosen for replacement.

K = recently refferred from MM

Q. Consider a 4-block cache, the following MM blocks are referred, which replacement strategy use the best performance ?

ref : 4, 5, 7, 12, 4, 5, 13, 4, 5, 7, 12, 13

	Cache
0	
1	
2	
3	

◯ replaced.

□ Hit

◌ - Most Recent Reference.

FIFO: $\textcircled{4}$ $\textcircled{5}$ $\textcircled{7}$ $\textcircled{12}$ $\underset{\text{Hit}}{\boxed{4}}$ $\underset{\text{Hit}}{\boxed{5}}$ $\textcircled{13}$, $\textcircled{4}$, 5, 7, 12, 13

4 left mode side will be replaced.

Hits = 2

finally Blocks present are 5, 7, 12, 13

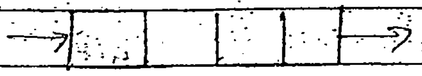
Last Replacement = 4

The Block replacement Techniques

The block replacement techniques are aimed to choose such a cache block, for replacement that may result less or no penal (xtra time, performance get reduced due to the

(1) FIFO

The block that spent longer time in cach is chosen for replacement. The Basis is Arrival time.
The data structure used in this is queue

Queue :- 

Blocks are entered from front & goes ou from rear.

(2) LRU

The block that is not recently used is choosen for replacement. Queue with slight adjustment on hit (recently used).
If there is not hit, LRU & FIFO are b same.
It can be applied for both associate as as set-associative mapping.

[illegible]

T

T

t

m

(

(1)

is

f

Que

B

F

(2)

ch

a

I

a

$T_{avg} = \frac{3}{\cancel{4}\,2}\overset{53}{\cancel{106}} + \frac{1}{\cancel{4}\,2}\overset{29}{\cancel{58}}$

$= \cancel{164} \quad \frac{159}{2} + \frac{29}{2}$

clxeare

$= \frac{188}{2}$

$\boxed{T_{avg} = 94 \text{ ns.}}$

Performance $= \frac{1}{T_{avg}}$ words/sec.

c 80% +

$= \frac{1}{94}$

$+ T_c)]$

$\simeq \frac{10^9}{94}$ million words/sec

ag Block.

$\Rightarrow$ 10.63 million words/sec.

ellom + 0.2)]

if $f_w = 15\%$, $f_r = 85\%$

$T_{avgr} =$

$T_c) + 20\% +$

$T_{B_{new}} + T_c)]$

For write back oprⁿ:

At any point of time, 20% cache blocks are modified and cache is full.

$$T_{avg} = f_r * T_{avgr} + f_w * T_{avgw}$$

$$\text{Performance} = \frac{1}{T_{avg}} \text{ words/sec}$$

$$T_{avgr} = H_r * T_c + (1-H_r)\left[\underbrace{(T_B+T_c)}_{\text{clean}} * 80\% + \overbrace{20\% * (T_{B_{old}} + T_{B_{new}} + T_c)}^{\text{dirty}}\right]$$

// If it is clean, overwrite on the existing Block.

$$= 0.8 * 10ns + 0.2[410ns * 0.8 + 810ns * 0.2]$$
$$= 8 + 0.2[328 + 162]$$
$$= 8 + 0.2 \times 490$$
$$= 8 + 98$$
$$\boxed{T_{avgr} = 106}$$

$$T_{avgw} = H_w * T_c + (1-H_w)\left[80\% * (T_B+T_c) + 20\% * (T_{B_{old}} + T_{B_{new}} + T_c)\right]$$

$$= 0.9 \times 10 + 0.1 \times 490$$
$$= 9 + 49$$
$$\boxed{T_{avgw} = 58}$$

which
updatn
$\}$ 0.8 (Read hit)
le, then
from
or
e of
strategy?
see
(c)
10)

$T_{avgw} = H_w T_{\text{updatn}} + (1-H_w)(T_B + T_{\text{updatn}})$

$= 0.9 \times 100\,ns + (1-0.9)(400+100)$

$\boxed{T_{\text{updatn}}\, T_m = \text{Max}(T_c, T_m) = 100\,ns}$

$= 90 + 0.1 \times 500$

$= 90 + 50$

$T_{avgw} = 140\,ns$

$\cancel{T_{avg} = 25\% \times 90 + 75\% \times 140}$

$= \frac{90 \times 25}{100} + \frac{75 \times 140}{100}$

$= 22.5 + 105$

$T_{avg} = \cancel{127.5}$

$T_{avg} = 75\% \times 90 + 25\% \times 140$

$= \frac{75}{100} \times 90 + \frac{25}{100} \times 140$

$= 67.5 + 35$

$\boxed{T_{avg} = 102.5\,ns}$

Performance $\simeq \frac{1}{100\,ns/\text{word}}$

$\simeq \frac{10^9}{100}$ words/sec

$= 10$ million words/sec.

Q Consider, a Hypothetical pocessor which issues 25% of references for updatn. It uses two level memory hieraracy with $T_c = 10 ns$, $T_m = 100ns$, $H_R = 0.8$ (Read hit) $H_w = 0.9$ (write hit)
If the referred word is not available, then a 4 word block is to be moved from MM to cache (Either for read oprn or write oprn) What is the performance of this memory with write through strategy?

$$\text{Performance} = \frac{1}{T_{avg}} \text{ Words/sec}$$

~~$T_{avg} = 25\% \, T_{avgw} + 75\% \, T_{avgr}$~~

~~$= f_r \times T_{avgr} + f_w \times T_{avgw}$~~

$T_{avgr} = H_R \times T_c + (1-H_R)(T_B + T_c)$

$T_B = 400 ns$

$= 0.8 \times 10 + (1-0.8)(400+10)$

$= 8 + 0.2 \times 410$

$= 8 + 82$

$T_{avgr} = 90 ns$

be
f two
entire

→ It gives the better performance for less no. of updatn

→ In write-back updatn, the MM updatn is done only when concerned updated block is chosen for replacement. If the block chosen for replacement is not mod. then incoming block can be simply overwrit. In the cache. The Dirty bit is used to indicate the status of the block in cache

$$\boxed{T_{updat^n} = \underset{\text{New block}}{T_B} + T_C}$$

~~current updat~~ in case of clean block

// Current updatn always reflected ~~to done~~ in cache memory, only.

ced In

$$T_{updat^n} = \underbrace{2T_B}_{(T_{BOLD} + T_{Bnew})} + T_C$$ in case of dirty block

Copied back

dealt
update

In MM, current updatn is reflected when T_{Bold} is copied back.

Q. The K^{th} block of MM has to be placed in which cache set, if two way associatⁿ is used for the entire $2 \cdot C$ cache blocks.

[✓ (a)] K Mod C.
(b) K Mod 2C.
(c) 2C Mod K.
(d) C Mod K.

No. of Cache blocks $N = 2 \cdot C$

$$S = \frac{N}{P} = \frac{2C}{2} = C$$

~~Cache block = $\log_2 C$~~

K^{th} block of MM has to be placed in K Mod C set.

UPdatⁿ Techniques

The updatⁿ technique are used to dealt with cache coherence problem. The ~~ω~~ write through updatⁿ simultaneously update the cache memory & MM.

~~The~~ $$\boxed{T_{updat^n} = \text{Max}(T_c, T_m)}$$

Mapping)

Mapping)

ciative Mapping)

tes

bytes

Set-Associative Memory

$1MB + \left(\frac{2^{13} \times 13}{8}\right)$ bytes

Q. Consider a cache memory which is applied with direct mapping. The no. TAG bits is equal to no of blocks in cache. Each block contains N words wt N is the total cache blocks. How mar words are there in MM?

(a) $N^2 \times \log N$
(b) $N \times 2^{2n}$
[✓ (c)] $N^2 \times 2^n$
(d) None

Direct Mapping

	$\log_2 N$		
Physical Address Say K	TAG	C. Block offset	Word offset
	N	$\log_2 N$	N

Block offset $= \log_2 N$

$K = \log_2 N + \log_2 N + N \Rightarrow N + \log_2 N^2$

The no. of words in MM $= 2^K$

$= 2^{N + \log_2 N^2}$

$= 2^N \cdot 2^{\log_2 N^2}$

$= \boxed{2^N \cdot N^2}$ Ans

(3) One TAG comp.

Size = 10 bits (Direct Mapping)

8 K TAG comp

Size = 23 bits (Associate Mapping)

8 TAG comp

Size = 13 bits (Set-Associative Mapping)

(4) Actual size of the Cache

data memory + TAG Memory

N × Tag bits

Direct Mapping

$$1MB + \left(\frac{2^{23} \times 10}{8}\right) \text{ Bytes}$$

$$1MB + \left(\frac{2^{23} \times 10}{2^{3}}\right)$$

$$1MB + 2^{20} \times 10$$

Associate Memory

$$1MB + \left(\frac{2^{23} \times 23}{8}\right) \text{ bytes}$$

MM

4 GB = 2^{30}

$\frac{2^{30}}{2^{2}}$ (4 bytes) per word

2^{28}

256 Million words

(1) Physical Address (bits)

$= \log_2 2^{28} \Rightarrow 28$ bits

(2) Direct Mapping

$$\underbrace{\overset{10}{\boxed{\text{TAG}}}\overset{13\ (\log_2 N)}{\boxed{\text{C. Block offset}}}\overset{5}{\boxed{\text{Word offset}}}}_{28\text{ bits}}$$

Associate Mapping

$$\underbrace{\overset{23\text{ bits}}{\boxed{\text{TAG}}}\overset{5}{\boxed{\text{Word offset}}}}_{28\text{ bits}}$$

8-way Set Associate Mapping

(M)

$$\underbrace{\overset{13}{\boxed{\text{TAG}}}\overset{10\ (\log_2 S)}{\boxed{\text{Set offset}}}\overset{5}{\boxed{\text{Word offset}}}}_{28\text{ bits}}$$

P = 8-way set-associative mapping

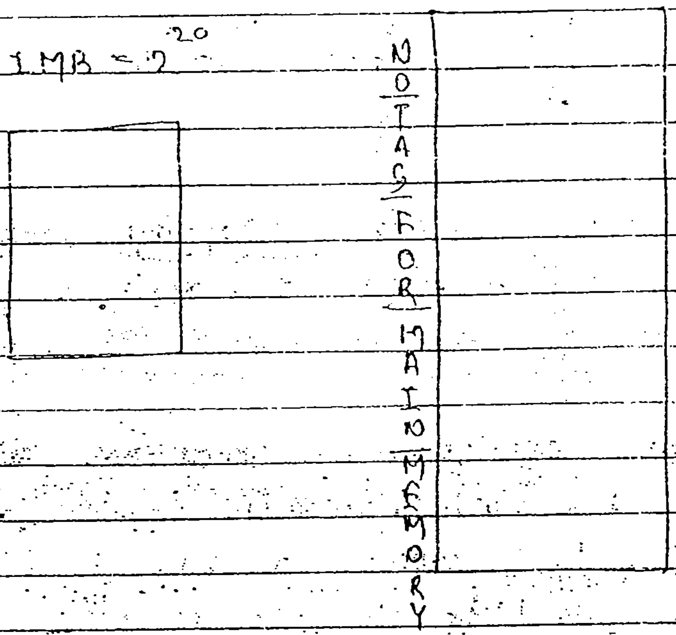
$S = \frac{N}{P} = \frac{2^{13}}{2^{3}} = 2^{10} = 1$ K bit

// The higher the associatn ; More the TAG bits.

8 K blocks.

$\frac{2^{13}}{2^{3}} = 2^{10}$

CM : 1MB = 2^{20}



MM

4 GB = 2^{30}

$\frac{2^{30}}{2^{2}\ (4\,\text{bytes})\ \text{per word}}$

2^{28}

256 Million words

M = $\frac{2^{28}\ \text{words}}{32\ \text{words per block}}$

$= \frac{2^{28}}{2^{5}}$

M = 2^{23} blocks

M = 8 Million blocks (in MM)

8

CM :

1MB = 2^{20}

$= \frac{2^{20}}{2^{2}}$

$= 2^{18}$ words

$= 256$ K words

N = $\frac{2^{18}}{32\ \text{words per block}}$

$= \frac{2^{18}}{2^{5}} = 2^{13}$ blocks => 8 K blocks.

H

th Block of
od S) set
in cache
t associatn
che
ed cmy
of the set
y see
physical
x required)

TAG	Set offset	Word offset

↑ $\log_2 S$

if $P = 1 \Rightarrow$ Direct Mapping , $S = N$
if $P = N \Rightarrow$ Associate Mapping , $S = 1$

Q. Consider ~~one ending~~ 1MB cache memory & 1GB MM, Both are divided into 32 word Blocks. Each word contains 32 bits.
(1) What will be the no. of bits required address the MM?
(2) What will be the no. of TAG bits for the following Mapping Techniques?
  (a) Direct Mapping
  (b) Associate Mapping
  (c) 8-way ~~a~~set associate mapping
(3) How many comparator & of what size required for each case?
(4) What is the Actual size of the cache memory in ~~b~~ required for the above mapping?

CM = 1MB , MM = 1GB

Size of Word blocks = 32
~~Wo~~ Size of each word = 32 bits

Set-Associative Mapping

In the set-Associate Mapping K^{th} Block of MM has to be placed in (K Mod S) set where,

S = Total No of sets in cache

$S = \frac{N}{P}$ for P-way set associatn

N = total No. of Blocks in Cache.
P = Gives the No. of sets

within the set, it can be placed any-where.
The No. of TAG comparator = Size of the set (P)

i/ P Blocks are accomodated in every see.

~~no. Fig~~ ee

For Normal Mapping, P < N.

The P-way set associatn divides the physical Address into 3 portions.
(1) Word offset. (Size of the blocks are required)
(2) Set offset ($\log_2 S$ bits required)
(3) TAG

extra
l a more
very

= Blocks in
2

ntition.

ts (512 words

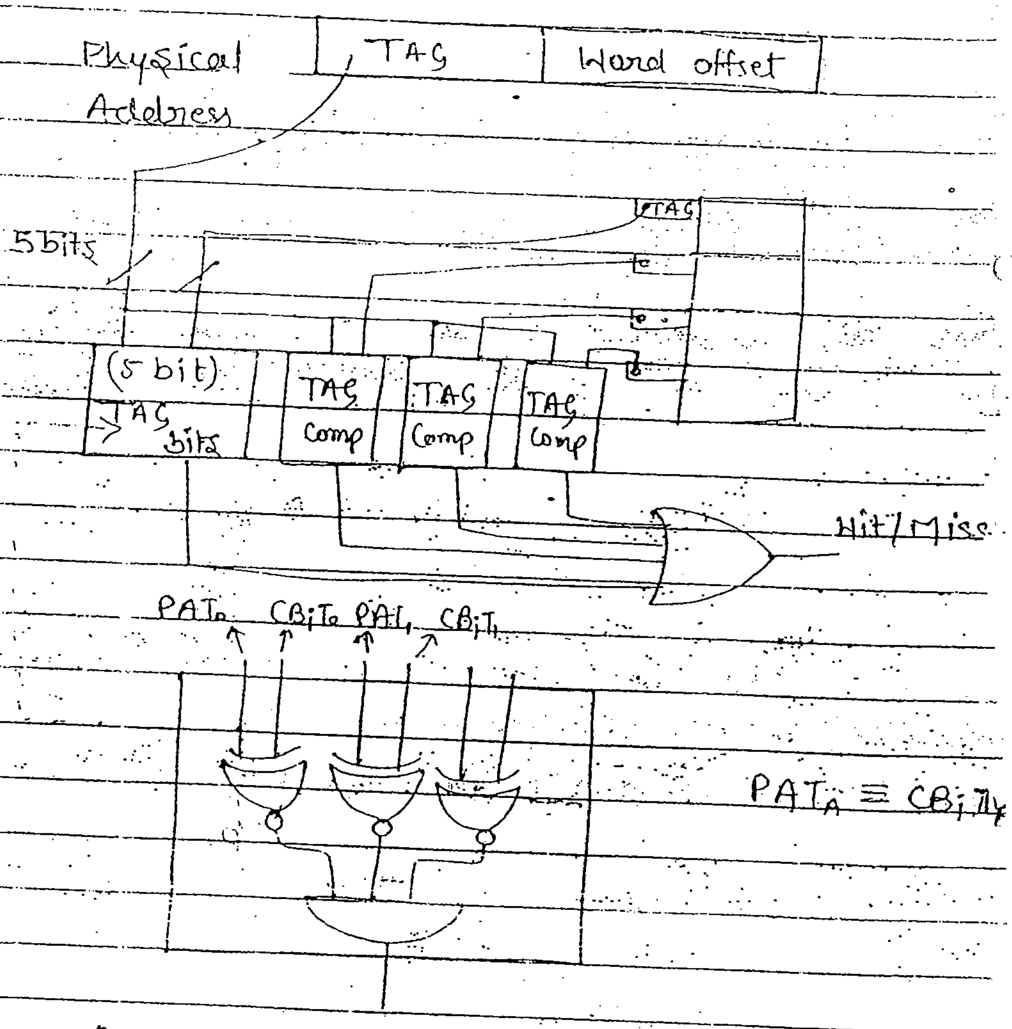
TAG bits.

+ TAG memory

els

is:

this is fixed-
Cache)



$PAT_A \equiv CB_iT_A$

PAT_0 - Physical Address Tag bit
CB_iT_0 - Cache Block Tag

$$\bigwedge_{i=0}^{k} (PAT_i \odot CB_iT_i)$$

~~The Set Associati~~

→ We have ~~more parallel~~ comparators are ~~to be~~ referred & also more h/w is required. Therefore, it is very costiliest.

No. of tag comparators Needed ~~for mapping~~ = No. of Blocks in Cache.

Physical Add. having only two partition.

$$\underbrace{\overset{5}{\boxed{\text{TAG}}}\overset{4}{\boxed{\text{Word offset}}}}_{\text{Physical Address = 9 bits (512 words)}}$$

→ Associative mapping requires more TAG bits.

Actual Size of Cache = Data memory + TAG memory

// No. of words in cache i.e. 64 words is fixed.

TAG Memory needed to be like this:

$\boxed{\text{TAG Memory = N * TAG bits}}$

where; N = No. of blocks (in this is fixed in Cache)

$$\begin{array}{r} 72 \\ 64 \\ \hline 28 \end{array}$$

09|03|10

Direct mapping is very simple and required less no. of tag bits.

1
0

Eg. 4-Blocks

Blok fault F F F F F F F

CPU ref: 0 4 0 8 4 0 8 0

0 mod 4

0	$\cancel{0}$	$\cancel{4}\ \cancel{0}\ 8\ \cancel{4}\ \cancel{0}\ \cancel{8}\ 0$
1		
2		
3		

→ Block ~~fault~~: 3/4 blocks of cache is not used.
→ It requires more time bcoz it is slower.
→ It ~~may~~ requires many Block replacement.

Soln—
Any block of MM can be placed anywhere i the cache.

F F H F H H H H
CPU: 0 4 0 8 4 0 8 0

0	0
1	4
2	8
3	

"Accessing from cache is takes less time, known as Associative mapping.
"Any Block of MM can be place any where."
It is very fastest mapping.

$\frac{32}{60}$ 28 = (39)

Q. 92

256 128 64 32 16 8 4 2 1

$$\underbrace{0\ \ 0\ \ 1}_{\text{Tag info.}}\ \ \underbrace{1\ \ 1\ \ 1}_{\text{Cache '3' block}}\ \ \underbrace{1\ \ 0\ \ 0}$$

Not available.

placed in
ory.
ed in
he same
em is placed
k is
n the MM
r 1" - cache
contain
ntly.
pointed by
: 0 to 7.

Q. Check whether 200 word ~~of~~ of main memory available in cache or not.

200 word in
9 bits for MM

256 128 64 32 16 8 4 2 1

$\underbrace{0\ 1\ 1}_{\text{TAG info}}\ \underbrace{0\ 0}_{\text{Cache '0' Block}}\ \underbrace{1\ 0\ 0\ 0}_{\text{Word off}}$

128 + 64 + 8 = 200 word.

Word 200 is not available in the cache bcoz physical address tag is not match with TAG bits of associated cache block

Q. Whether the word 121 is present In cache not ?

256 128 64 32 16 8 4 2 1

$\underbrace{0\ 0\ 1}_{\text{Taginfo}}\ \underbrace{1\ 1}_{\text{Cache '3' block}}\ \underbrace{1\ 0}\ \underbrace{0\ 1}$

121 is not available.

Q. 96

256 128 64 32 16 8 4 2 1

$\underbrace{0\ 0\ 1}\ \underbrace{1\ 0}_{\text{Cache '2' block}}\ \underbrace{0\ 0\ 0\ 0}$

available

96
64
32

// Any block of main memory can be placed in one of the block of cache memory.

→ K^{th} block of MM has to be placed in $(K \bmod N)^{th}$ cache position

→ Many MM blocks compete for the same cache position only one among them is placed in cache.

→ TAG bits denote which MM block is currently placed.

Address Mapping (Practically)

$$\underbrace{\boxed{\text{TAG}}\boxed{\text{C. Block offset}}\boxed{\text{Word offset}}}_{\text{Physical Address}}$$

9 bit

Word offset - position of word within the MM Block.

C. Block offset - " " " " cache Block. which cache block contain main mem. block currently.

e.g.
(KMOD4)

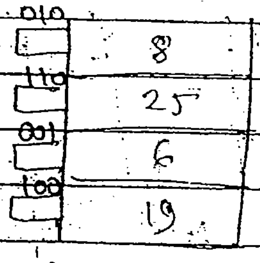
TAG 000 001 010

MM Blocks {0, 4, 8,
12, 16, 20, 24, 28} ← 0
110 111

{1, 5, 9, 13, 17, 21, 25, 29} ← 1

{2, 6, 10, 14, 18, 22, 26, 30} ← 2

{3, 7, 11, 15, 19, 23, 27, 31} ← 3



0(0, - - - 28) pointed by 000 to 111 i.e. 0 to 7.

{6, 8, 19, 25} N=4

ADDRESS MAPPING

1250
9

The cache & MM are divided into fixe size partitions (Blocks). The Address mappi decides which block of main memory (MM is to be placed in which cache position. Th direct Mapping is the simplest one & requ less no. of TAG bits. The direct mapping re many block replacements.

o) 138.88

a)

Associate mapping is the fastest one but i requires more h/w. The set associate mappi used the advantages of direct mapping & associa mapping. In four-way set associate mapping the cache is logically partitioned into sets with each set containing four cache blocks.

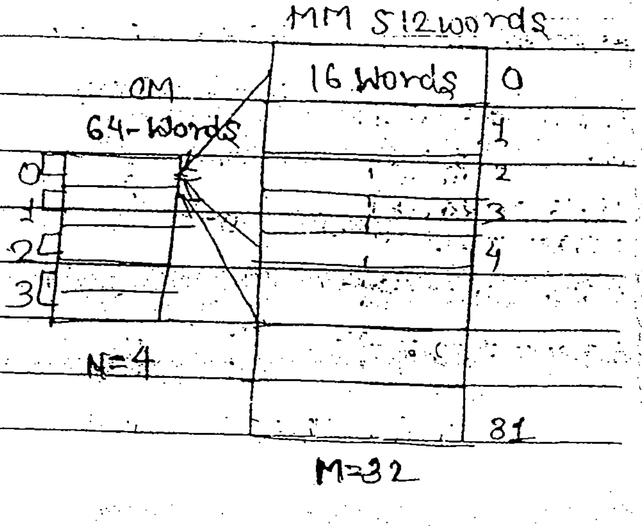
)/4

The one-way set associatn reduces to direct mapping, N-way set associatn reduces to assoc mapping.

At any point of time, only 64 words are placed in cache memory.

Direct Mapping

e



~~$T_c = \frac{T_m}{5} = \frac{138.88}{5 \times 100}$~~

~~$T_{avgnew} = 60x$~~

~~$60 = H_{new} \times \frac{1250}{9} + (1-H_{new})\frac{1250}{9}$~~

~~$= \frac{1250}{9}(H_{new} +$~~

~~T_e~~ $60 = H_{new} \times \frac{138.88}{5} + (1-H_{new})\, 138.88$

$111.104 H_{new} =$

$60 = 138.88\left(\frac{H_{new}}{5} + 1 - H_{new}\right)$

$(H_{new} + 5 - 5H_{new}) = \frac{5 \times 60}{138.88}$

$-4H_{new} = 5 - \frac{5 \times 60}{138.88}$

$H_{new} = \left(5 - \frac{5 \times 60}{138.88}\right) / 4$

$H_{new} = 0.71$

$0.8 \rightarrow 0.08$

$100 \rightarrow ?$

$\frac{100 \times 0.08}{0.8} \Rightarrow 10\%$ ↓ decrease

T
of
d
is
de
les
mas
A
re
used
map
the
with
Th
mapp
mapp
At
pla
Direc

identify
ced.

block will

rsently
l.

placed.

faster than
ccess time
from 50ns

$T_c = \frac{T_m}{5}$, $H = 0.8$

$T_{avg} = 20\%$ ~~of 50ns~~

$\frac{20}{100}$ ~~$\times 50 \times 10^{-9}$~~

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

$T_{avg_{old}} = 50ns$

avg time increased so hit ratio

$H_{new} = ?$ (< 0.8) is reduced

$T_{avg_{new}} = T_{avg_{old}} + 20\%$ of $T_{avg_{old}}$

$= 50ns + 10ns$

$T_{avg_{new}} = 60ns$

T_c & T_m will remain same.

$T_{avg_{old}} = H \times T_c + (1 - H_c) T_m$

$50 = 0.8 \times \frac{T_m}{5} + (1 - 0.8) T_m$

$50 = \cancel{0.8} T_m \left(\frac{0.8}{5} + 1 - 0.8 \right)$

$T_m = \frac{50}{\frac{0.8}{5} + 1 - 0.8}$

$= \frac{50 \times 5}{0.8 + 5 - 4} \Rightarrow \frac{250 \times 10}{\cancel{18}\,9} \, 5$

$= \frac{1250}{9} \Rightarrow 138.88$ ~~[illegible]~~

$$\begin{array}{r} 138.8 \\ 9 \overline{)1250} \\ 9 \\ \hline 35 \\ -27 \\ \hline 80 \\ 72 \\ \hline 8 \end{array}$$

→ Memory management is aimed to identify the block that is to be replaced.

- Memory management decides which block will be replaced.

Techniques to identify -
- FIFO
- LRU
- Direct Mapping

FIFO - Arrival time

LRU - The block which is not currently used that will be replaced.

Direct Mapping - $\underset{\substack{\downarrow \\ \text{MM Block No. which is to be replaced}}}{K} \text{ Mod } \underset{\substack{\downarrow \\ \text{Total Blocks in cache}}}{N}$

(K Mod N) = ? → This block will be replaced.

(1) Consider a cache which is 5 times faster than MM with hit ratio 80%, the avg. access time for this cache is increased by 20% from 50ns

(a) What is the cache access time?

(b) What is the MM access time?

(c) What is the new hit ratio?

→ Cache used in client side memory for mail-addressing as well as server side memory.

hierarchy

→ Cache Coherence problem is dealt with
- write through updatⁿ
- Write back updatⁿ

ldress to

coherentey - similar

e)
e)
)

// A Variable having same value in both main & cache memory.

* Cache Coherency - data inconsistency with main & cache memory.

ariable

(1) Write through updatⁿ - not suitable for more write operatⁿˢ. Problem is resolved or avoided bcoz no data inconsistency when MM & cache are update at same time.

(2) Write back updatⁿ - It is postponing the write operatⁿ for some time, main memory is not updated with cache memory.

running after 1st

Main memory is updated when the concerned block is to be taken out of cache then MM updated.

memory)
= 11)

Eg. for bulk write operatⁿ it is suitable.

lity
n

Data consistency is reduced in this updatⁿ.

8/09/10

Cache Memory

→ Smallest & fastest memory component in hierarchy and maintain Locality of reference
→ Address mapping converts physical address to Cache address
 — Direct Mapping (Small cache size)
 → Associate " (large cache size)
 — Set-Associate " (Medium " ")

Variables are of two types-

(1) live variable ← ~~required for used in m~~
(2) Dead variable ←

Register Allocatn Register Check — Active variable / Dead

```
for ( i=1 ; i<1000 ; i++)
{
    P(i) = q(i) + r
```

Stored in cache memory for run 999 times after 1st running
 1st time - slow (fetch from main memory)
 999 times - fast (" " Cache - ")

~~for(i=~~

- Spatial locality by Position
- temporal locality by executn

pred mag

Complement

Complement

for -6

signed mag. - 1110

1's complement - 1001

2's " - $\overset{-8}{1}\,0\,\overset{2}{1}\,0$

$-8+2$

	$+0$	-0
S.M. -	0000	1000
1's C. -	0000	1111
2's C -	0000	0000

for S.M.

n-bit $\Rightarrow (a_{n-1}\,\underbrace{a_{n-2} \cdots\cdots a_0}_{\text{for magnitude}})_2$

$\sum_{i=0}^{n-2} a_i \times 2^i$ is +ve if $a_{n-1}=0$.

weight coeff

$$\underbrace{\left[\left(-2^{n-1}\right) \times a_{n-1}\right]}_{\text{weight for sign bit}} + \sum_{i=0}^{n-2} a_i \times 2^i$$

Range (n-bits)

V_{min}	to	V_{max}	
$-(2^{n-1}-1)$		$(2^{n-1}-1)$	Signed mag
"	"	"	1's complement
-2^{n-1}		$2^{n-1}-1$	2's complement

Largest -ve no. are min. value.

denoted
mplement
significant
the
ttern
e repr.
ent
the
code.
notatn
it (the

Signed magnitude

$\overset{8421}{10111} = (-7)_{10}$

→ The 1's complement & 2's complement arithmetic uses the same h/w for both addition & subtractn whereas separate h/w is needed for sign-magnitude arithmetic.

→ The 2's complement arithmetic is faster compared to 1's complement arithmetic bcoz it doesn't require the correctn to end around carry.

→ The signed magnitude & 1's complement cover the same range & have two pattern for zero, (−0 & +0) whereas ~~two's~~ 2's complem repr. has single pattern for zero bcoz of this the 2's complement cover one xtra mag-n no. compare to the other.

Eg. for +6

wt.	signed mag.	-	0110	All +ve no.'s have same repr. in all 3 patterns
Non wt. co.	1's complement	-	0110	
wt. code	2's "	-	0110	

//

Fixed point Signed Repr.

The fixed point signed no.'s can be denoted in 1- Signed magnitude repr. ~~2o~~ 1's complement 3- 2's complement

→ All the three repr. gives the most significant bits to denote the sign, if it is 1 all the repr. declared the no. is -ve.

→ All +ve no.'s are having identical pattern & identical value in all the three repr.

→ The signed magnitude & 2's complement notatⁿ are the weighted codes whereas the 1's complement repr. is non-weighted code.

→ The 2's complement is the only notatⁿ that essence the weight to sign bit (the weight of the sign bit is -ve.

E.g. 2's complement repr.

$$\begin{array}{ccccc} 1 & 0 & 1 & 1 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ -16 & 8 & 4 & 2 & 1 \end{array} = (-9)_{10}$$

we

then if

its then

at

$$\begin{tikzcd}
 & & \text{Integer (Radix posit}^n \text{ LSB)} \\
 & \text{Fixed Point (Position of Radix is fixed)} \arrow[ur] \arrow[r] \arrow[dr] \arrow[ddr] & \text{Fract}^n\text{ (Radix position before MSB)} \\
\text{Number} \arrow[ur] \arrow[dr] & & \text{Unsigned} \\
 & \text{Floating Point} \arrow[ur] & \text{Signed}
\end{tikzcd}$$

$(110.101)_2$
$(11.101)_2$

Value Expr^n

$(a_{n-1}\ a_{n-2} - - - - - - a_1\ a_0)_2 \rightarrow$ fixed point unsigned intege

$$V = \sum_{i=0}^{n-1} a_i \times 2^i$$

E.g:

$\overset{16\,8\,4\,2\,1}{10\,111} = (23)_{10}$

$V \rightarrow V_{min}$ iff $\forall_i ; a_i = 0$

$V_{min} \rightarrow 0 \times 2^0 + 0 \times 2^1 + \ldots\ldots + 0 \times 2^{n-1} = 0$

$V \rightarrow V_{max}$ iff $\forall_i ; a_i = 1$

$V_{max} = 2^0 + 2^1 + 2^2 + \ldots\ldots + 2^{n-1} = 2^n - 1$

Range : V_{min} to $V_{max} \Rightarrow (0$ to $2^n - 1)$ if n =

0 to 25

NUMBER REPRESENTATION (Radix-2)

Range : V_{min} to V_{max} V- value

Value Expression

// If the value is in betn limits then it it is underflow and if outer the limits then it is overflow.

$$\begin{tikzcd} & \text{Numbers} \arrow[dl] \arrow[dr] & \\ \text{Fixed point} & & \text{Floating point} \end{tikzcd}$$

Fixed point (Position of radix is fixed)

↓(1) Integer

(Radix position after LSB)

/ Integer itself is a ~~floating~~ fixed point no.

(2) Fraction (Radix position before MSB)

Floating point E.g. $(110.101)_2$

$(11.101)_2$

5×32

101×100000

10100000

(6)

Max^m decimal value of 3-digit binary no. $= (7)$

E.g. $(777)_8 = 8^3 - 1 \Rightarrow (511)_{10}$

n - digit no.
radix - r

max^m decimal value $= (r^n - 1)_{10}$

Let us say in que it requires k-bits, then

$$(2^K - 1)_{10} = (10^{20} - 1)_{10}$$

$\Rightarrow$

$$2^K = 10^{20}$$

$$\therefore K = 20 \log_2 10$$

$$K = 66$$

~~If we have a~~

Any radix can be used as borrow in their subtractⁿ like above or as example.

in decimal, 10 can take as borrow.

(6)

E.g.

$$\begin{array}{r} 10111.10 \\ 1001.01 \\ \hline 01110.00 \end{array}$$

(7) $(9003)_{16} \rightarrow (16^3 * 9 + 3*16^0)_{10}$

weighted sum of the digits is always radix no.

9 0 0 3

$(1001\ 0000\ 0000\ 0011)_2$

Z = 12 // No. of 0's

N = 4 // No. of 1's

$(32^5 * 5 + 32*3 + 3)_{10}$

(2) (3) (2) = 7

// When above expression is converted into binary, then the no. of 1's in it is 7.

ed to binary

of 1's
ression :

X<Y

When we increasing the radix the value should be decreased.

(5) $(x^2+2x+3)_{10} = (9+6+x)_{10}$

$x^2+x-12=0$

$x \to -4$ ✗, 3 ✗ } digit value & radix value relatⁿ is not maintained

(2) (1) Convert the operands into octal.
(2) Perform octal substractⁿ.

opr1: $C012.25_H =$

$(\underline{1100}\ \underline{0000}\ \underline{0001}\ \underline{0010}\ .\ \underline{0010}\ \underline{0101})_2$

~~1 4 0 0 1 2 2~~

140022.112_8

opr2. $10\,111\,001\,110.101$

2716.5_8

Octal subtractⁿ

(8-1) borrow

$$\begin{array}{r} 140022.112 \\ -\ \ \ 2716.5 \\ \hline 135103.412 \end{array}$$

E.g. ① $(212\ 121\ 112)_3 = (\qquad)_9$

$3^{2} \xrightarrow{\text{digit } 2 \to 1} 9^{1}$

$(2\ 12\ 12\ \boxed{1\ 1}\ \boxed{2})$

$= (2\ 5\ 5\ 4\ 5)_9$

$\overset{3^1\ 3^0}{(12)_3} = (5)_{10} = (5)_9$

② $(101110011)_2 = (173)_{16}$

$2^4 \to 16^1$

// If there is any relatⁿ betⁿ the radixes, we can use them like above examples.

1 2 1 7

000 001 010 001 111

0 ~~2~~ 8 F

(3) 10

(1)

(2)

~~(a)~~

(c)

(3) Pos

r =

(4)

(a) (1

(5)

x

(a) 3

ssm

rcessing
ement

rocessing
ctional

E.g. $(41.4)_8 = (\quad)_5$

Conversion –
One radix to another radix
(1) Radix-r to decimal
(2) decimal to radix-r

(1) Radix-r to decimal

$$\sum_i w_i d_i$$

$$w_i \begin{cases} \Rightarrow r^i & \text{if } d_i \text{ is integer} \\ \rightarrow r^{-i} & \text{if } d_i \text{ is fractn} \end{cases}$$

$$\begin{matrix} 4 & 1 & . & \overset{4}{\downarrow} \\ 8^1 & 8^0 & & 8^{-1} \end{matrix}$$

$8\times4+1+4\times8^{-1} \Rightarrow 32+1+\frac{4}{8}\,0.5 \Rightarrow 33.5$

(2) Decimal to Radix-r
divide the decimal no. (Integer) successively by radix-r and arrange reminders in reverse order (LIFO

$5\overline{)33}(6 \quad 5\overline{)6}(1 \quad 5\overline{)1}(0$
$\underline{-30} \rightarrow \underline{5} \rightarrow \underline{0}$
$\times 3 \qquad 1 \qquad 1$

$0.5\times5 \Rightarrow \boxed{2}.5 \Rightarrow 2$
$0.5\times5 \Rightarrow \boxed{2}.5 \Rightarrow 2$

$(41.4)_8 = (113.222\ldots)_5$

(6) Approximate number of bits needed to represent 20-digit decimal number in binary

(a) 20 [✓ (b) 66] (c) 86 (d) 96

(7) What is the relatⁿ betⁿ number of 1's & 0's if the following decimal expression is converted into binary

$$(16^3 + 9 + 3)_{10}$$

[✓ (a)] 8 zeros are more than 1's

(b) 7 — " — " —

(c) 6 — " — " —

(d) none

Solⁿ:-

(3) $(42)_9 = (XY)_9$

$$\begin{tikzcd} 4 \arrow[d] & 2 \arrow[d] \\ 9^1 & 9^0 \end{tikzcd}$$

$36 + 2 \Rightarrow 38$

$(38)_{10} = (XY + 3)$ $\quad \because X < Y$

$\Rightarrow XY = 5, 7$

3|09|10

(1) $(101011)_2 = (x4y4)_3$

x, y, z value (a) 1,1,1 [✓ (b) 1,1,2] (c) 1,2,1 (d) 2,2,2

(2) $C012.25_{H}$ $_{16}$ -10111000110.101_{8} $_{2}$

[✓ (a) 135103.412_8] (b) 135103.205_8 (c) 564411.412_8 (d) 564411.205_8 → octal.

(3) Possible values of X if

$(42)_y = (X3)_y$

x = [✓ (b) 1,5] (a) 5,7 (d) 3,5 (c) 7,35

s, we can

(4) $(1217)_8 =$

(a) $(1217)_{16}$ (b) $(0B17)_{16}$ [✓ (c) $(028F)_{16}$] (d) $(2297)_{10}$

(5) $(123)_x = (12x)_3$

x =

(a) 3 (b) −3, 4 (c) −4, 3 [✓ (d) None]

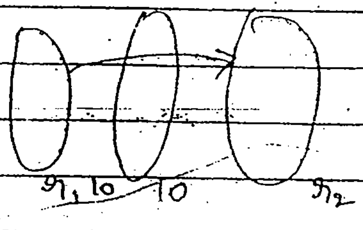
SISD - Single inst. single data stream
(Array) SIMD - — " — " — Multiple — " —
MISD - Multiple — " — Single — " —
(parallel) MIMD - Multiple — " — Multiple — " —

SISD - 1CU, MU, 1PEM
SIMD - 1CU, MU, N-PEM PE - Processing element
MISD - N-CU, MU, 1-PEM
MIMD - N-CU, MU, M-PEM.

- Super scalar Arch contains separate processing unit for ~~inger~~ integer manipulatn & fractional manipulatn.

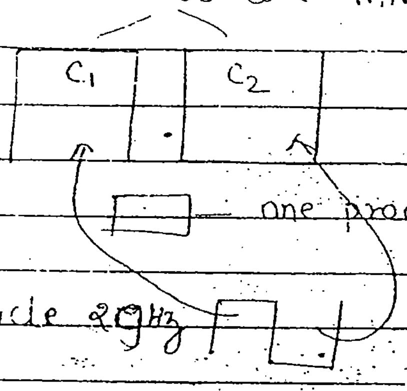
Number System -

$\boxed{\text{Digit value} < \text{radix}}$



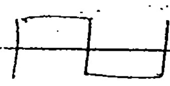
required

Dual Core ~~two core mirror images~~



Having clock cycle 2GHz

4 GHz Non-dual core



More frequency more heat Hence, Dual core is effective

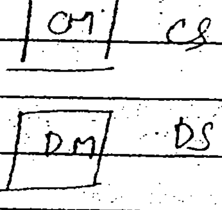
Classificatn of Computer Architecture

First Memory Architecture is proposed by Von neuman which supports the stored prog. concept (prog. & data are to be stored prior to processing).

The Harvard arch. is similar to that of ~~Von~~-neuman arch., the only diff. is the placement of prog. & data is done in diff. memories

The basis of FLYN arch. is instr. stream & data stream.

Harvard



// Locality of reference or frequently required info are stored in cache memory.

// Aim of CLA (Carry look ahead) -
→ reduced the time of addition

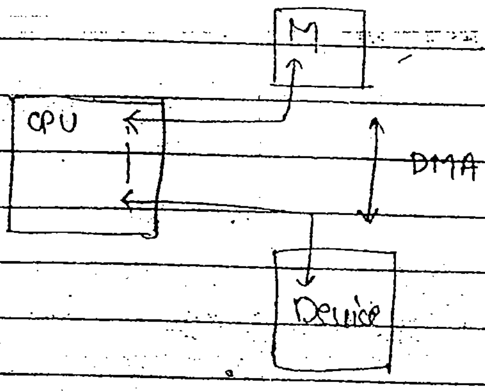
Cycle oprn		H/w required	
Fetch	F	HW_F	
Decoding	D	HW_D	I_1;
Execution	E	HW_E	I_2;
Storing	S	HW_S	I_3;

// Overlap the instructn phases

S				I_1
E			I_1	I_2
D		I_1	I_2	I_3
F	I_1	I_2	I_3	I_4
	T_1	T_2	T_3	T_4

// Goal of Instr pipeline -
→ "Efficient usage of Resources"

Data transfer



s of system

l Inst. set)
xed length"
ddressing mode.
CPI=1,
↓
clock per
instr.

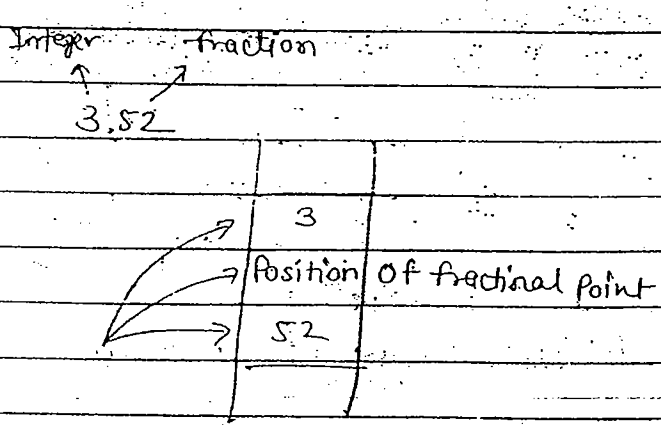
addr.
nterpret

CO: Deals with logical design of the system

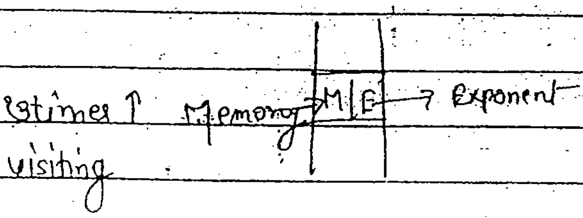
→ It consider the physical devices & their interconnectn with the perspective of ~~approv~~ improv the performance

Performance of system –
// Amount of work doing in unit tim or ①instn. per second (MIPS) &
②floating pt. opr. per sec (FLOPS)

// The main aim of CO is to reduced time & more work is done at that tim



exponent represents memory ~~reference~~ locatn of fraction point.



→ Structural dependency is due to resource limitatn.

→ Resource duplicatn is used to deal with it.

Three types of dependency

1. Data
2. Control
3. Structural

We don't have any program without Branch instructn.

K-clocks n-1 clock

S				I_1	I_2
E			I_1	I_2	I_3
D		I_1	I_2	I_3	I_4
F	I_1	I_2	I_3	I_4	I_5
	1	2	3	4	5 ... clock

Total clocks = $(K+(n-1))$ clocks for n instructn

T_{avg} of instructn cycle is reduced.

$T_n = (K+n-1) T_{clockp}$

$T_{clockp} =$ Max (stage delay) + Buffer overhead resource

eal with it.

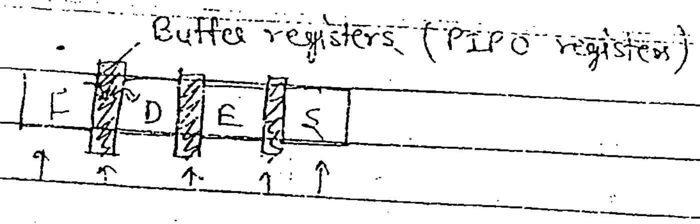
out Branch

K

Total clocks=
+(n-1) clocks
r n instruct^n

rk

reeleceel.



→ All data ~~Synchronizily~~ circuits

If 100 clock is $S = \frac{97 \times 4 \text{ clocks}}{100 \text{ clocks}} \simeq \underline{3.88}$ → No. of stages in ~
there,

If 1000 clock is $S = \frac{997 \times 4}{1000} \sim \underline{3.986}$
there,

∴ More instruct^n in pipeline → More the better performance.

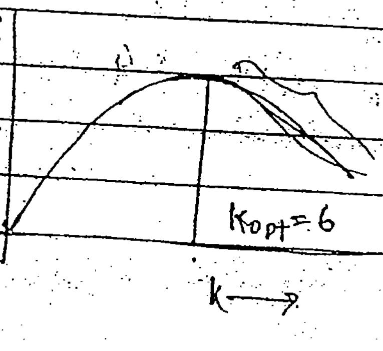
$\boxed{S_{ideal} = 4}$

Pipeline system gives 4 times more better perfor then non-pipeline system.

K = 20

If the no of stages is increasing after at par' time speedup factor is decreased i.e; perf decreases.

No. of optimal stages is always 6. it is always fixed for any processor.



Q. Consider 2-pipelines a & b. The pipeline a has 8-stages of uniform delay 2ns. The pipeline b has 5-stages ~~of~~

5stages with
4ns, 1ns, 2ns, 3ns, 1ns

(A) How ~~many~~ much time is saved for 100 instr. if pipeline a is used instead of b?
(B) What will be the respective speedup factors & efficiencies?

Uniform delay - ~~same time at every~~ ~~time~~. stages ~~uniform~~
same time delay at every stage.

T_{clockP} = Max(stage delay) + Buffer overhead.

clock of pipeline

$T_{clockPA} = 2ns$, $T_{clockPB} = 4ns$

① $n = 100$ (Instr.)

$T_n = (K + n - 1) T_{clockP}$

$T_{nA} = (8+99) * 2 ns = 214 ns$
$T_{nB} = (5+99) * 4 ns = 416 ns$

(2)

Every
in

Amore
or

The pipeline
ay 2ns
~~of~~

ns

no instr. if
b?
up factors

~~uniform~~
~~every~~ ~~time~~.
lay at every
overhead stage.

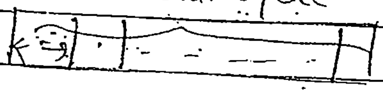
4ns

Time saved if A is used instead of B $= 202ns$ (416 −

(2) $S_A = \frac{\text{Time without Pipeline}}{\text{Time with pipeline}}$

Every Instr. has to perform 8 phases in 2ns Instr. cycle.

Instr. cycle



Amount of work is not changed in Time as without pipeline.

stages in A

$(8 \times 2ns)$

$$S_A = \frac{16ns \times 100}{214ns}$$

$$S_A = 7.476$$

$$S_B = \frac{\text{time widout pipeline}}{\text{time wid pipeline}}$$

$$= \frac{11\,ns \times 100}{416\,ns}$$

$$S_B = 2.64$$

S_{ideal} = Number of stages
$\Updownarrow$
100% efficiency

100% w.r.t pipeline A = 8
? % — " — = 7.46

$= \frac{7.46 \times 100\%}{8}$

efficiency for A, $\eta_A = \boxed{93.25\%}$

100% w.r.t pipeline B = 5
? — " — = 2.64

$= \frac{2.64}{5}$

$= \frac{52.8}{100}$

efficiency for pipeline B, $\eta_B = \boxed{52.8\%}$

Q. Consi
each
diff.
give:
in

Q. An ~~10 stage~~ Instr. pipeline is having speedup factor 10 while operating with 80% efficiency. What will be the no. of stages for the pipeline?

$S_p = 10$
$\eta_p = 80\%$

100% =
80% — ~~?~~ 10
100% $= \frac{100\% \times 10}{80\%}$

$\frac{10}{8}$ 2)

24 (b)
check

S_{ideal} = No. of stages.

$= \frac{1000}{80} \rightarrow \frac{100}{8}$
$= 12.5$
$\boxed{\approx 13 \text{ stages}}$

0%

Q. Consider a 4 stage instr. pipeline in which each instr. spends diff. no. of clocks at diff. no. of stages. The following table gives 4 instr. and associated required clock in the respective stages.

Instruct^n	F	D	E	S	
	clocks				
I_1	2 adag	1	3	1	4
I_2	1	2	2	1	6
I_3	1	1	2	3	7
I_4	1	1	1	1	4
					$\boxed{24}$

edup factor
ficiency.
- the

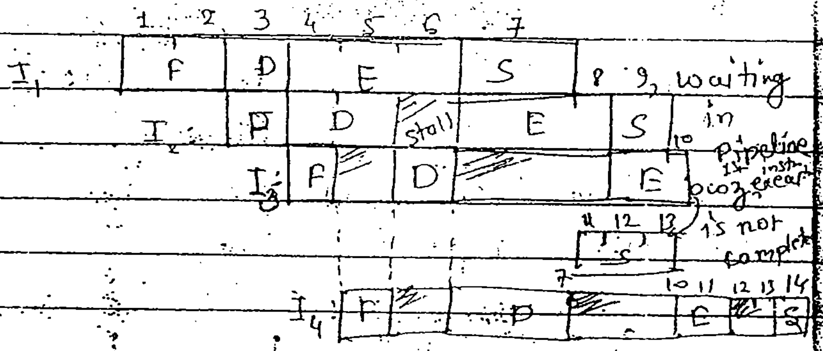
for (i=1 ; i<2 ; i++)
{ I_1 ; I_2 ; I_3 ; I_4 }

(a) 24 (b) 28 (c) 22 (d) none

• clocks required per instr. per stage is changed here.

Thumb Rule -

→ Resource should not be allocated unless it freed by previous one
→ No instructⁿ acquires the next before the completⁿ of previous stage



Here, $S_{ideal} = 4$

$S = \frac{\text{time without pipeline}}{\text{time with pipeline}} = \frac{24}{14} = 1.71$

$4^{\text{stages}} = 100\%$ eff.

$1.71 = \frac{1.71}{4} \times 100$

$= 42.7\%$

$\eta_p = 43\%$

Pipeline producing not any activity, we call it as stall.

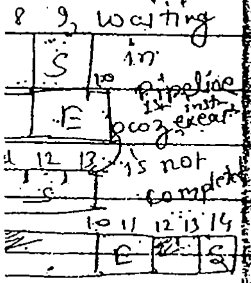
IInd
Exa
Stt
More
& H

cated unless

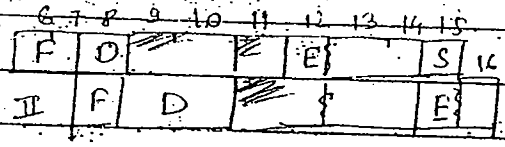
before the

for {i = 1 : 2}

{ I_1 ; I_2 ;
I_3 ; I_4 ;
}



IInd Iteratn



Static Method — Ist Iteratn

	I_1	I_2	I_3	I_4	I_1	I_2	I_3
F	2	3	4	5	7	8	9
D	3 (2+1)	5 stall	6	7	8	10	11
E	6 (3+3)	8 (6+2)	10	11	14	16	18
S	7 (6+1)	9	13	14	15	17	21

$\frac{24}{14} = 1.71$

6 is max^n then 5, add (6+2)

4, we call

$S = \frac{48}{22} = 2.18$

More instr. in pipeline, more the speedup fact & better performance.

Dependencies -

→ The dependencies among the instr. requires re-organisatn of instr. pipeline.

→ This inturn consumes extra time. Hence, the speedup factor reduce.

→ The data dependencies are four types -

-	RAW	" True	dependency"
-	WAR	" Anti	" "
-	WAW	" Output	" "
-	RAR	" NO	" "

instr. → II I

Read After write - RAW

// II^nd instr. has to read only after I^st instr. is read.

(1) RAW

E.g. ADD R_1, R_2, R_3 ; $R_1 \leftarrow (R_2)+(R_3)$
ADD R_4, R_1, R_5 ; $R_4 \leftarrow (R_1)+(R_5)$

	1	2	3	4	
I_1 :	F ADD	D R_2, R_3	E $(R_2)+(R_3)$	S R_1	
I_2 :		F ADD	D R_1, R_5	E $(R_1)+(R_5)$	S R_4

R_1 is ~~going to be~~ available in 4th clock cycle & Instr 2 is used it in 3rd cl " . It's used its previous value i.e; calledas dependency requires

Hence,

ypes –

instr. Is

+(R$_3$)
)+(R$_5$)

le N Instr
. It is
las dependency

Rectificatn:

	1	2	3	4	5	6	7
I_1:	F ADD	D R_2, R_3	E $(R_2)+(R_3)$	S R_1			
I_2:		F ADD	Stall	Stall	D R_1, R_5	E $(R_1)+(R_5)$	S (R_4)

5|09|10

Data dependency

RAW True $D \cap R_1 \neq \phi$

WAR ANTI $R_2 \cap D_1 \neq \phi$

WAW output $R_2 \cap R_1 \neq \phi$

~~w~~ $D_2 \cap D_1 \neq \phi$

RAR NO - It takes xtra times than others

WAW - IInd instructn has to write when Ist instructn has to read it.
(All are like this statement)

			domain	Range
SUB	R_1, R_2, R_3	$R_1 \leftarrow (R_2) - (R_3)$	R_2, R_3	R_4
OR	R_2, R_4, R_5	$R_2 \leftarrow (R_4) \vee (R_5)$	$R_4 R_5$	R_2

logical instructn takes less time.

Every stmt has 2 instructn

(1) domain D - getting the operand (Read) operⁿ
(2) Range R - storing the operand (Write opⁿ)

If $\neq \phi$, then ~~it will~~ there will be dependency

Sch

I) Inst

sche

→ Eff

(Per

II) Sta

i.

III) Opes

Before

→ It c

→ Most

eg. ADD R

ADD R

No penality

Solutns

I) Instructn Re-scheduling

Scheduling – Instructn taken in order to perf oprns.

→ Efficiency of compiler & program.
(Performance is depends upon it).

tra times

If $I_1 + I_2$ pipeline Causing stalls

I_1 :
I_j : } Independent 4 stage
I_k : } Instructns
I_2 :

instructn

new Range
R_3 R_4
R_5 R_2

II) Stall cycle instruction: No operatn cycle introduced in betn instructns.

III) Operand forwarding : Value of operand is provided to needy stag before its is stored.

→ It consume additional H/w.

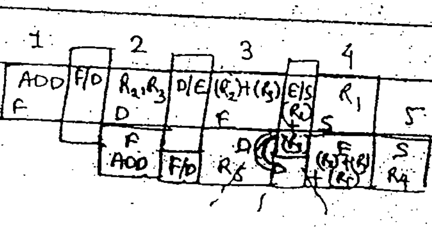
→ Most of the data dependencies are resolved

opern
(Read)
(Write opn)

Eg. ADD R_1, R_2, R_3
ADD R_4, R_1, R_5

be

No penality



$(R_2)+(R_3)$ – called as value or result data to should be consumed in same clock, not going in other clock cycle.

Control Dependency

Instruct^n cycle of ~~other~~ any instruct^n is altered ~~by~~

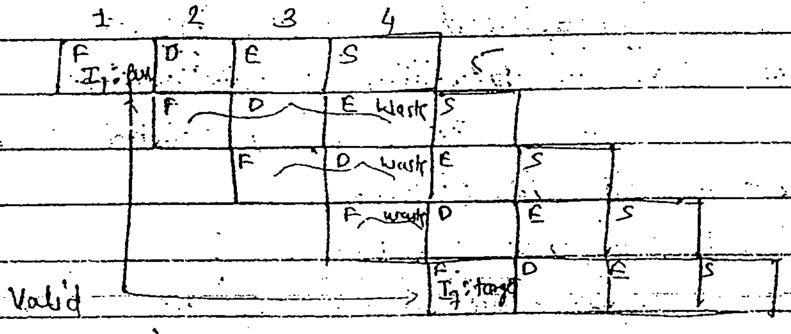
Sequence of execut^n is altered by Instruct^n cycle of earlier one in pipeline.

I_1 : BUN I7

I_2 :

I_3 :

I_7 :



3-stall cycles

$\underbrace{\qquad}_{(k-1)\text{ clocks}}$

No. of stages in pipeline

// Speed up factor is reduced.

ock, not

No $S_{eff} = \dfrac{S_{ideal}}{(1 + \cancel{\text{Stall}}, \cancel{\text{Stall}} \text{ Stall freq} * \text{Stall cycle})}$

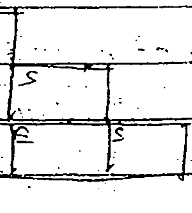
$$\boxed{S_{eff} = \dfrac{K}{\left(1 + \text{stall freq} * \text{Stall cycles}\right)}}$$

~~altered by~~

Instruct? $\longrightarrow$ Opcode - gives nature of instr.

Que Consider 5-stage instr. pipeline (F, D, E, write Back $\rightarrow$ WB)

The instr. pipeline allows overlapping o all instr^ns except the branch instr. in the case of branch instr. The target i not available until the current bra instr. is completed. Let there are 20% br insts. (Conditional & unconditional) each stage consumes 10 ns delay



(a) What is the avg. instr. time? if There is no special consideratn for this conditional branch instr.

(b) Let there are 50% instr^ns are conditional among the branch once & 30% doesn't satis the condtn, if there is no penality for conditional branch instr. whose condtn is met (branch is not taken). Wat is the

ine

effective speedup factor?

Sol^n (a) All other except branch instr consume (on average) 1 clock for completion. for each branch instr. the target has to wait (K-1) clocks for scheduling & take 1 - clock (on average) to complete.

$K = 5$

T_{clockP} = Max stage delay = 10ns

stall cycles for branch instr = (K-1) clocks
(waiting time) = (5-1)
= 4 clocks (b)

take for scheduling (waiting time)

$T_{avg} = [80\% * 1 clock + 20\% (4 clocks + 1 clock)]$
= 1.8 clocks ⇒ 18 ns

OR T_{avg} = (1 + stall freq. * stall cycles) clocks

= (1 + 20% * 4) clocks

= 1.8 clock

= 1.8 × 10ns

T_{avg} = 18ns

consume
sletion.
t has to
l take 1-

ions

(K-1) clocks
(S-1)
- 4 clocks
uling (waiting time)
s + 1clock)]

1 clocks

Effective Speed up factor $S = \frac{k}{1+(\text{stall freq} \times \text{St}}$

$= \frac{5}{1+20\% \times 4}$

$= \frac{5}{1+0.2\times 4}$

$= \frac{5}{1+0.8}$

$= \frac{5}{1.8}$

$S = \frac{\cancel{50}^{25}}{\cancel{18}_{\cancel{9}}}$ 3

$= 1$

(b) branch instr. 20% (decision tree)

$$\begin{tikzcd} & 20\% \arrow[dl] \arrow[dr] & \\ 50\% & & 50\% \end{tikzcd}$$

~~un-conditional~~ (Branch is taken) stalls : 4

~~Conditional~~ ~~(Branch is taken)~~
- Condition satisfact 70%
- Cond^n not satisfied 30% (Branch is not taken) stalls: 0

$T_{avg} = (1 + 20\% [50\% \times 4 + 50\% (70\% \times 4 + 30\% \times 0$ clock

$= (1 + 0.2 [0.5 \times 4 + 0.5 (0.7 \times 4)]]$

$= (1 + 0.2 [2 + 0.5 \times 2.8]] \Rightarrow (1 + 0.2 [2 + 1.4]]$

$= 1 + 0.2 \times 3.4 \Rightarrow 1.68$ clocks $\Rightarrow 1.68 \times 10\,ns$
$\Rightarrow 16.8\,ns$

$\frac{3.4}{.68}$ $\frac{2.8}{1.40}$

Here, Branch instr. is present ~~th~~ which doesn't want penality, or causing trouble. So, the avg. time is less here.

$$S = \frac{K}{1.88}$$

$$= \frac{5}{1.88}$$

$$\boxed{S = 2.97}$$

$$\text{efficiency } \eta = \frac{1}{T_{avg}} \rightarrow \frac{1}{16.8\text{ns per instr.}}$$

$$= \frac{10^9}{16.8} \text{ instr./sec}$$

$$= \frac{10^3 \times 10^6}{16.8} \text{ MIPS}$$

$$\eta = 59.5 \text{ MIPS}$$

Techniques to deal with ctrl dependencies–

(1) Delayed load

The load of the branch instr. is either the next sequential one or the target insts. betn the branch & its load independent instr^ns are scheduled since, the load

is

Loa

No

No

Lo

W

is

(2) Mul

The

menu

the

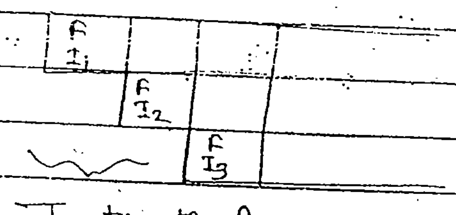
one

doesn't is getting delayed
is

I_1
I_j
I_k
I_l
I_m

Load I_2 or I_7

/ No. of instr. delayed depends upon stall cycles no.
// Load is taken at end bcoz load is delayed
/ Whether the condn is satisfied or not, penality is there ~~sk~~ will be.



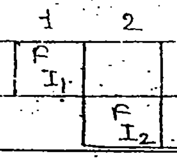
Instructn Queue.

dencies-

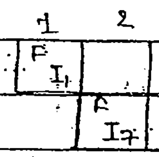
(2) Multiple Pipelines

s either The two pipeline sys. successfully resolve
target memry ~~data~~ dependency. Both pipeline maintains
2) indep- the branch instr. in the beginning clock, &
the load one pipeline places I_2 fetches in the (next sequential) ~~Another~~

places I_7 (target for the next clock). Based on the condⁿ, one of the pipeline is allowed to continue & other one is stop.
This 2 pipeline sys. is resolved
IF-THEN-ELSE statement. If face for Nested if conditions.



IF-THEN-ELSE



) Based
ne is
s stop.
for
EN-ELSE

1/10/10

Control dependency

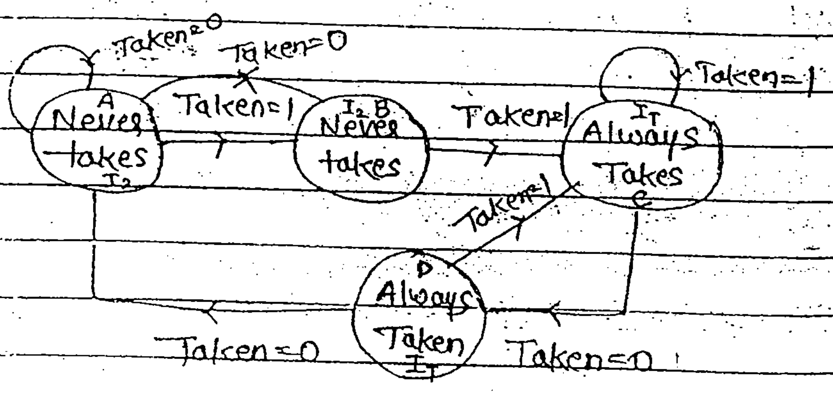
(3) Prediction Techniques

- Static
- Dynamic
  - decision when to change the predi
  - E.g. when-ever previous Two consecutive prediction are wrong then change pred

- Branch Never takes
  Fetch: Next Sequential (Next clock)
- Branch Always taken
  Fetch: Target (Next clock)

Implementation of Dynamic Prediction:

Flag : Taken = 0 (Branch is Not taken)
(denote status of prediction) = 1 (Branch is Taken)



A, C = strong predictⁿ status
B, D = weak predictⁿ status

When the outcome is come & the predictⁿ is not changed then it is strong predictⁿ.

~~@~~

Control dependency can't resolve with predictⁿ tech.

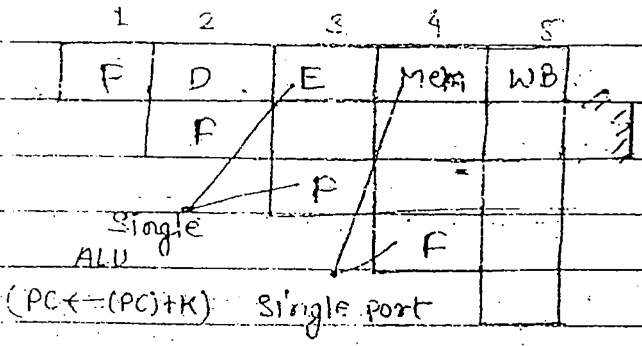
Structural Dependencies

The structural dependency result due to the limited availability of resources. Resource duplication is used to deal with structural dependency. The duplicatⁿ is subjected to cost and reliability constraints.

E.g. (1) In case of single port memory, fetching the instructⁿ cannot be overlapped with storing the result for another instructⁿ.

(2) In case of single ALU unit, execution of one instruction can't be overlapped with fetch of another instructⁿ if the fetch requires updating the prog. counter, (PC←(PC)+K).

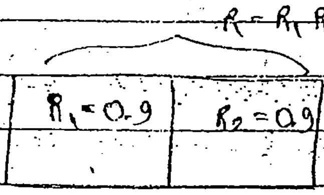
predicto
redicto
predictn
to the
esource
tural
to cost
ory,
be
for
petition
ed with
h requires
K).



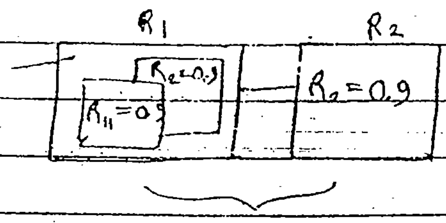
memory
(have only one address & data bus)

Due to less address & data bus, delayed in data & address sending which caused problem in efficiency. As well as due to ALU only one operatn is performed at a time which is also caused delayed.

~~$R = R_1 R_2 = 0.89$~~



It one module fail other works.



$R_1 = 1 - f_1$

$f_1 = f_{11} \cdot f_{12}$

$= (1 - R_{11}) \cdot (1 - R_{12})$

$= 0.1 \times 0.1 = 0.01$

$R_1 = 0.99$

$R = R_1 \cdot R_2 = 0.99 \times 0.9 \cong 0.9$

Q. Consider two instr. pipelines A & B, the pipeline having single port memory while B is having 2-port memory. Both pipelines are having equal no. of stages & allow all the instrucᵗⁿˢ to be pipelined without penality except the memory instr. In case of memory instr. if two memory operatⁿˢ can't done simultaneously there will be 1-stall penality. Let there are 20% instrucᵗⁿˢ are memory related. How many times the performance is enhanced if pipeline B is chosen instead of A.

Pipeline	stages	Memory	stall freq.
A	K	Single-port	20%
B	K	two-port	20%

stall cycles

1

0

(efficiency) $S_{eff} = \dfrac{S_{ideal}}{\left[1 + \left(\text{stall freq.}\right) \times \left(\text{stall freq.}\right)\right]}$

$= \dfrac{K}{1 + 20\% * \text{stall cycles}}$

B, the
while B is
s are having
instruc^{ns}
ept the memory
tcns
simultaneously
let there
ry related
is
sen instead

stay freq

20%
20%

les

$S_A = \frac{K}{1+20\% \times 1}$, $S_B = \frac{K}{1+20\% \times 0}$

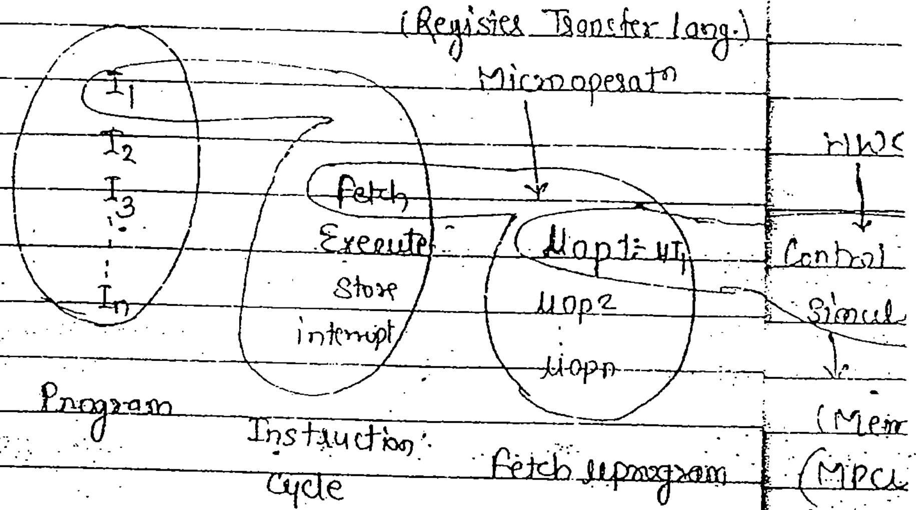
$\frac{S_B}{S_A} = \frac{K/1}{K/1.2} \Rightarrow 1.2$

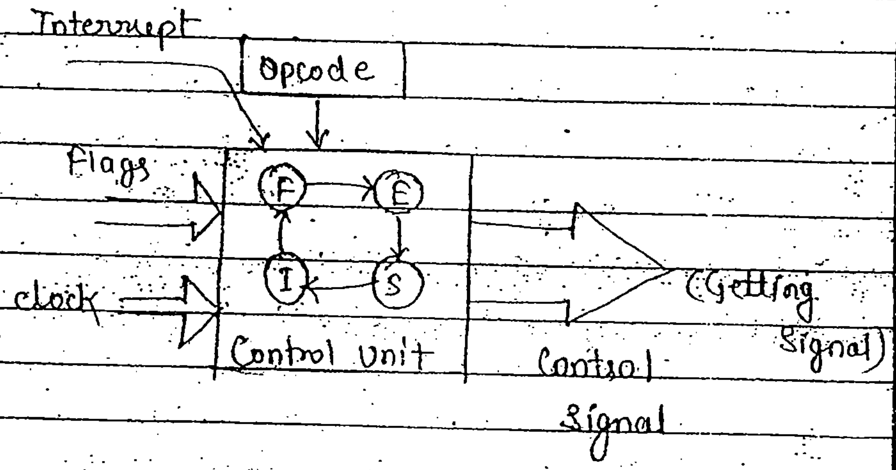
$\boxed{S_B = 1.2 S_A}$

Micro Operations

There are the elementary opr^{nts} (atomic) which are used to move the data across registers. A group of microoperat^{ns} imple a subphase of instruct^{ns} cycle. The reg transfer lang. is used to encode the mi operat^{ns}. The encoded version is called as micro program. Each micro operat^{n} in term required a set of simultaneous getting sig (ctrl signals). The ctrl signals can be gene using h/w (H/w ctrl unit design), Memory t (Microprogrammed ctrl unit design).

A program is a collect^{n} of sequentia instr.





H/

Rout
(C

Control signal never generates any thing only helps to perform the ~~of~~ signal & in flow of signals.

E.g. ~~wbr~~ ^or^ taps in water piplines helps to flow the water

Transfer lang.)
ro operat^n
$\downarrow$
μop1: μt1
μop2
μopn
tch μprogram

HWCU design
$\downarrow$ (H/W)
Control Signals
Simultaneous
$\downarrow$
(Memory + H/W)
(MPCU design)

Register Transfer language

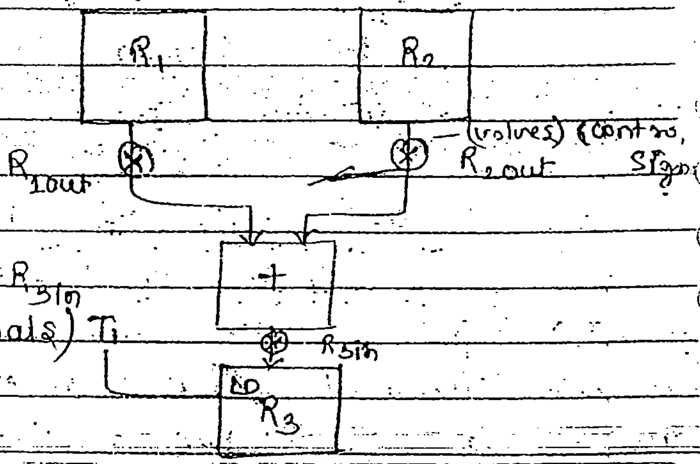
E.g. Addition of R_1 & R_2 and place the result in R_3 at clock 1

$\Downarrow$

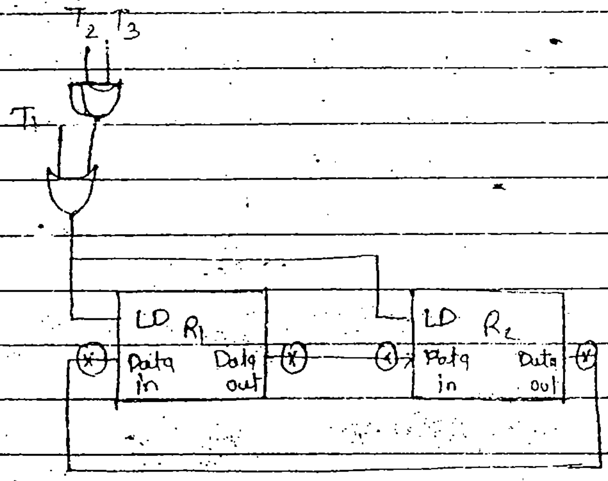
Cond : μ operat^n

$T_1 : R_3 \leftarrow R_1 + R_2$

H/W implementat^n -



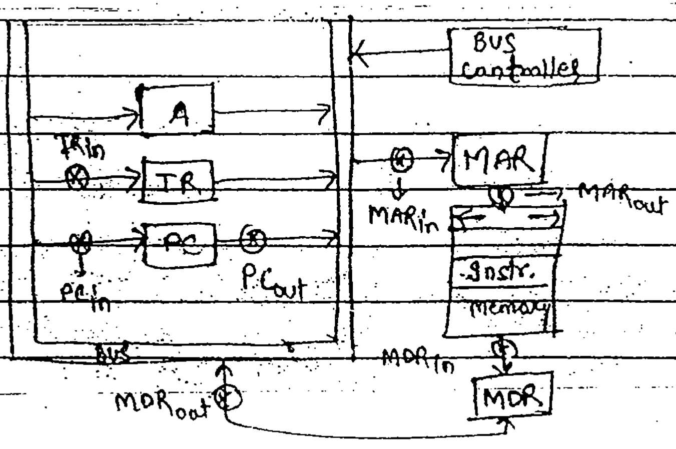
> (Getting Signal)
|
al
thing
signal is
s to flow



$T_1 + T_2T_3$: $R_1 \leftarrow R_2$, $R_2 \leftarrow R_1$; R_{1in}, R_{1out}, R_{2in}, R_{2out}

↑ Both of them are done parallely

Q. Consider a basic sys. which has accumulator, instr. register, prog. counter, memory address & data register, which are connected by single bus architecture. The bus controller will decide who has to use the bus at that time.



R_1out
R_2out
ssallely

accumulator
ry address
by single
ll decide
ime

BUS controller

MAR
→ MARout
instr. memory
MDR

→ fetch - get the instr into instr. register.
It contains 3-oppr^n -
Prog. ~~counter~~ (1) place the address in MAR
(2) Get the instr. into MDR
(3) Place it in IR.

Fetch Micro Program -

T_1 : MAR $\leftarrow$ (PC) [Clock1]
T_2 : MDR $\leftarrow$ M[MAR] (Clock 2)
T_3 : IR $\leftarrow$ (MDR) (clock3)
PC $\leftarrow$ (PC)+1 (5 values are ope.

// 4 clocks are used in fetch instead of 3
// clock 4 is used to isolate Fetch & Exe opr^n

→ The Execut^n of phase μprog. for ADD in
The ADD R_1 instr. adds the content of memory locat^n R_1 to the accumulator & the results are placed in accumulator.

ADD R_1
Accum $\leftarrow$ M[R_1] + Accum.

(1) Visit the memory to locat~~e the~~ R_1.
(2) Get that content into MDR.
(3) Add it to Accumulator

opcode	ref	
ADD	R_1	IR

Fetch

T_1 : MAR ← IR (Address)
T_2 : MDR ← M[MAR]
T_3 : A ← (A) + (MDR)

3-10-1

← IS

Bur

// Inc

Han

In

are

&

ctrl

for

it

All

Get the content of x
Increment
Save Back

3-10-10

← ISZ X Increment & Skip on Zero — Increm Contents
— Skip the instruct^n if result is ze

BUN X Branch to X

	Opcode	Ref
IR	ISZ	X

/Increment to memory locat^n x

T_1 : MAR ← IR(Ref)
T_2 : MDR ← M[MAR]
T_3 : MDR ← (MDR)+1
T_4 : M[MAR] ← (MDR)
T_5 : IF [(MDR)==0] PC ← (PC)+1

IR	BUN	X
	Opcode	Ref

T_1 : PC ← IR(Ref)

Hardware Control Unit Design

In the h/w ctrl unit design the ctrl signal are expressed as a sum of product expressi & realised using dedicated h/w. The h/ ctrl unit offer fastest response & its su for real time applicat^n as it is not fl it can't be used for design & testing places. All the ~~RISC~~ (RISC) processor employ the h/w c

//

unit design

Consider a CU (Ctrl unit) which has to support two instructⁿˢ I_1 & I_2 and uses ~~2~~ 3-8 bit registers A, B, C.

The following table gives the ctrl signal request for each μ operatⁿ :-

μ operatⁿ	I_1	I_2
T_1	$\boxed{A_{in}}$, B_{out}	$\boxed{A_{in}}$, A_{out}
T_2	B_{in}, C_{out}	C_{in}, B_{out}
T_3	C_{in}, C_{out}, B_{in}	$\boxed{A_{in}}$, A_{out}
T_4	$\boxed{A_{in}}$, C_{out}	B_{in}, C_{out}
T_5	END	END

1) Mo

NO

$$Control_x = f(I_1, I_2, T_1, T_2, T_3, T_4, T_5)$$

// Every ctrl signal is SOP of Instructⁿ & clocks.

Both instructⁿ perform at clock 1

SOP Expʳ – $A_{in} = \boxed{I_1 * T_1 + I_2 * T_1} + I_2 * T_3 + I_1 * T_4$

Hence, $A_{in} = T_1 + I_2 T_3 + I_1 * T_4$

$$C_{out} = T_4 + I_1 * T_3 + I_1 * T_2$$

has to
ad uses ~~3~~ 8-8 bit

-1 signal

~~out~~
~~out~~
~~out~~
~~out~~
2

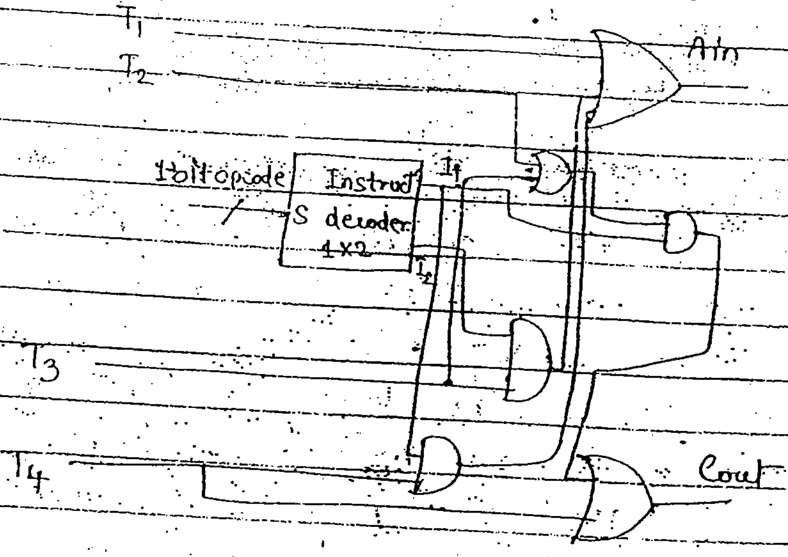
k1

$I_2 * T_3 +$

T_4

$* T_2$

// If 1 bit opcode = 1 , denotes Instrct^n I_2 otherwise I_1.



// Max degree of parallelism
No. of control signal simultaneously generated.

$$Bin = I_1T_2 + I_1T_3 + I_2T_4$$

$$f(P, Q, R) = \Sigma(3, 5, 6)$$
$$= \Sigma(3, 4, 5, 6, 7)$$

Micropregrammed Ctrl Unit Design

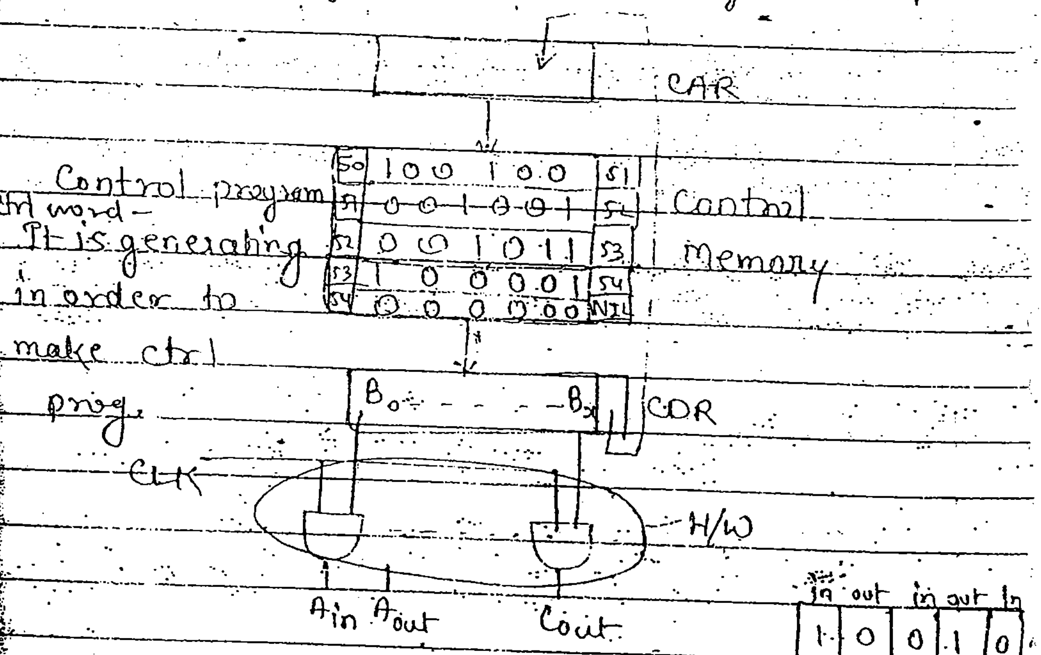
It ~~is~~ offers flexibility. Here, the binary pattern of ctrl signals (either with encoding or without encoding) stored in a memory (ctrl memory) & h/w is used to realize the ctrl signals any modificatn requires changing the binary pattern the h/w remain intact. Based on how many bits are used ~~with~~ for each ctrl words, the μ-programming can be termed as $\boxed{1}$ Horizontal μ-programming ~~is~~ $\boxed{2}$ Vertical μ-prog. The Horizontal μ-programming ~~ctrl~~ ~~co~~ consumes more bits for each ctrl word but offer maxm degree of parallelism.

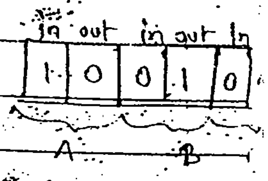
In a vertical μ-prog. the ctrl signals are first encoded. The encoded patterns is stored in ctrl memory. It is required to decode this ctrl word before the signals are generated.

e binary
with
stored
w is
ls any
e binary
t Based
~~th~~ for each
be
ramming
zontal
nore bits
max^m.

rl signals
atterns.
is required
e the

(1 bit / Ctrl signals) Horizonal μ-p





let control word - ~~Ain Bout~~

// H/w will never changed while binary patterns are changed.

The time taken to generate ctrl signal.

$$T_{cg} = T_{cm} + T_{HW} \text{ - using h/w.}$$

Time to access (T_{cm}); ~~time take for~~

// System requires 6-ctrl signals. Hence, Every ctrl word in CDR is of 6-bits

// 1 ctrl word can generate more signals.

Adv - Max degree of parallelism.

Disadv - lengthy ctrl word.

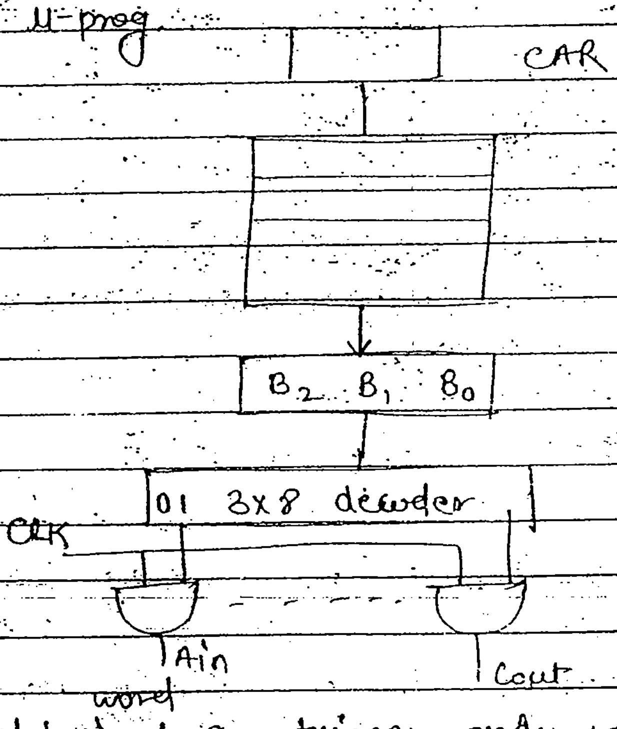
More no. of zero's, not efficient use of memo

// ctrl bit or ctrl signal = 0 is no operatⁿ is performed or no signal is generated.

6-ctrl signals encoded in 3-bits — encoding

A_{in}	001
A_{out}	010
B_{in}	011
B_{out}	100
C_{in}	101
C_{out}	110

Vertical μ-prog.



// 1 ctrl ~~signal~~ word can trigger only one signal. (generate)

$T_{cg} = T_{CM} + T_{decoder} + T_{HW}$

I
T
i
is
d
Dis
Se
to
o
te
f
dis
s
— On
req
Condtⁿ
here

n operatn
nerated.

The ctrl signals are encoded & stored & then decoded, so there is need for a decod in vertical u-prog. Hence, more time is consumed than Horizontal while in Hori decoded ctrl signal is stored so, no externa decoder is needed Here.
H/W is not changed in both u-prog.

Disadv - ① More time required to access.
② one word can generate only one signal.

Sequencing the ctrl words - it is required to access the ctrl word in specific order, then only the instructn is implemented properly. Here the ctrl words are following the concept of node of link list.
Sequencing is done by PC.
- One address instr is used to perform sequencing the ctrl words.

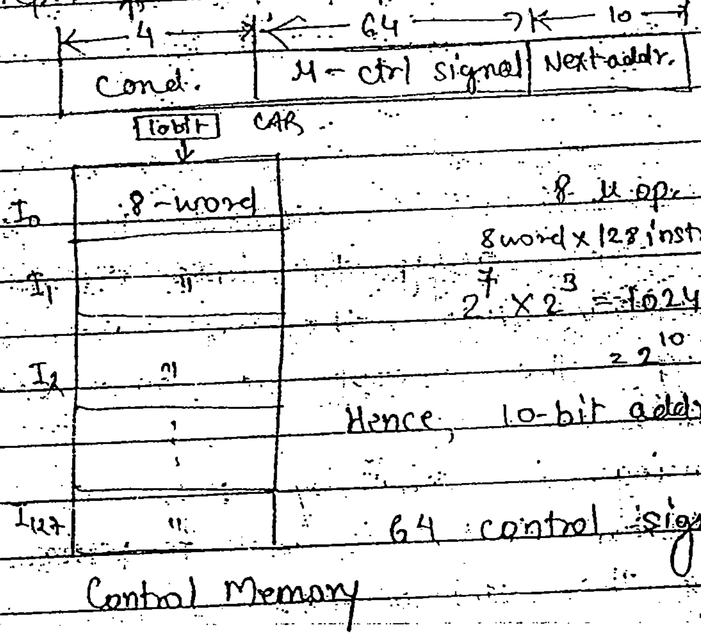
Cond field	ctrl signal field	Next Addr field

$\nearrow$ Condtn added here

gnal.

Que- Consider a μ-programmed CU design, which has to support 128 instrucⁿs. Each instrⁿ on average consumes 8-μoperⁿ. The system has to support 16-flag condⁿs with horizontal μ-prog. How many bits are required for each ctrl word & wat is the size of ctrl memory (in bytes) required. The one-addr. ctrl word is used for instr. sequencing.

4bits repr. the cond. 2^6 bits → 1 ctrl word - 6 bits



8 μ op.

8 word × 128 instr.

$2^7 \times 2^3 = 1024$ words (total)

$= 2^{10}$

Hence, 10-bit address.

64 control signals

Control Memory

(i) Control word - 78 bits

(ii) Control memory - $\frac{1024 \times 78 \text{ bits}}{8 \text{ bits / Byte}}$

$= 128 \times 78$

$= 9984$ B

128
×78
1024
896×
9984

lesign, nstrucTMS
8- uoper?
- flag
Haw-
b ctrl
memory &
ctrl

Q. Repeat the above with vertical μ-prog.

Cond	μ Control Signal	Next Addr
4	[illegible]	10

(i) Ctrl word = 20 bits

(ii) Ctrl memory = $\frac{\cancel{1024}^{\cancel{512}\ 256\ 128} \times 20}{\cancel{8 \times 2}}$

= 128 × 20

= 2560B

words (total)

ss

us

4) Neither horizontal μprog, Nor vertical μ-prog is preferred bcoz the first one required more control memory & the 2nd requires more " signals, ie; degree of parallelism is ↑.

The Hybrid μ-prog is used in which the ctrl signals are partitioned based on mutually exclusive property. Sum of the fields follow the horizontal μ-prog while other follow vertical μ-prog.

E.g: Consider 1-addr ctrl instr which has to support the control signals of the following behaviour

1. either 1 or none of the 63 ctrl signals

n.l.r
128
×8
10.24
[illegible]
64

2. At most 5 ~~of~~ from the remaining ones, wat will be the minⁿ bits required for ctrl bit?

Cond.	Control field		Next
	F_1 6	[illegible]	9

bit 5 bit

1st signal 000001
last " 111111

~~fields~~

F_1 - either 1 or 63 (follows) V up

F_2 - At most 5 from remaining ones (follows H up)

ctrl field = 11 bits

H Hup - 1 bit / ctrl signal.

Que- A ctrl instructⁿ has to support 10 groups of control signals

G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}
3	6	5	11	1	6	13	3	5	7

Minⁿ no. of bits saved for Each control ~~signal~~ word with respect to Horizontal ~~M~~ prog.

Mimᵐ no. of bits need in HUP = 3+6+5+11+1+6+13+3+5+7
= 60 bits

wat will
ctrl bit?

In hybrid up

G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9
2	3	3	4	1	3	4	2	3

~~total~~

min^n bits are saved for H up = 32 bits.

up)

Max^m degree of parallelism = 10
~~[illegible]~~ // 10 ctrl signals, each for each grp.
Min^m " " " = 0

Nano-programmed CU design.

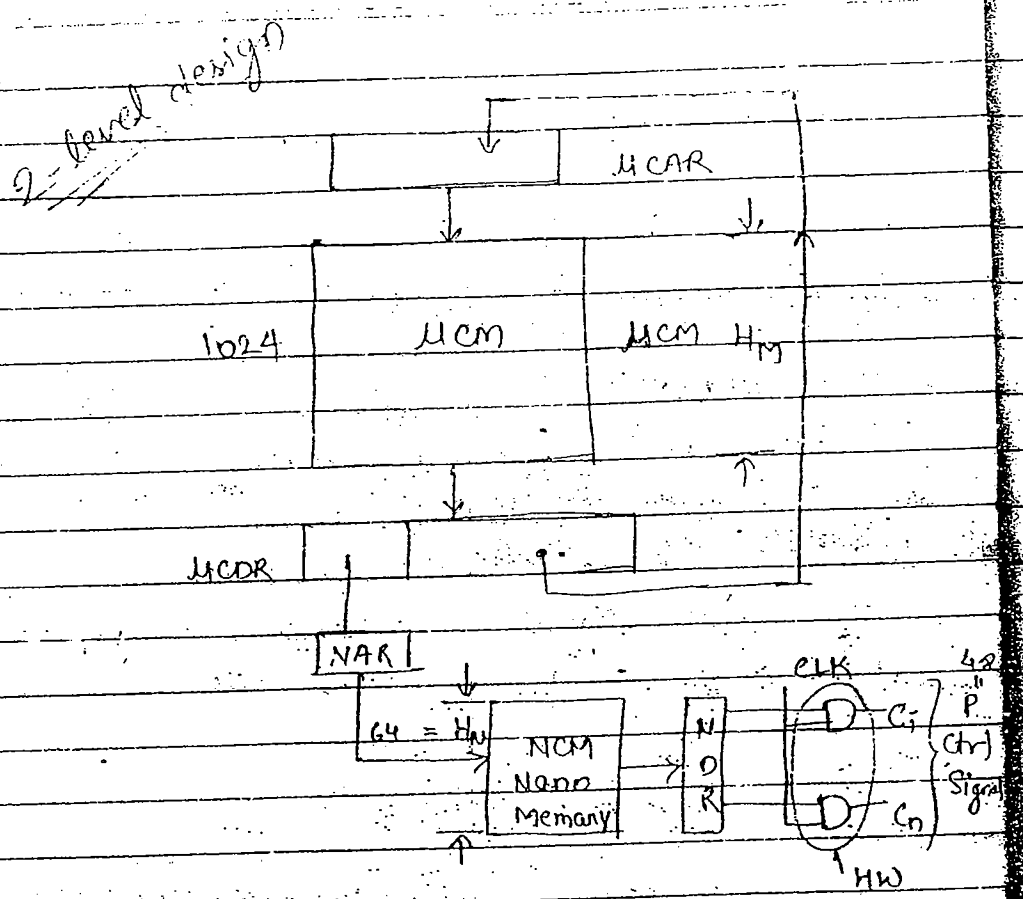
The nano-programmed CU design uses 2-
Ctrl memory (µ-ctrl memory, nano ctrl memo
The ctrl words are sequenced in µ-ctrl
memory & generated in nano ctrl mem. It
Supports max^m flexibility & modular pr
The firmware based app^n uses nano-pr
CU, the bootstrap prog (BIOS) is stored i
nano-programmed CU Since it uses two-le
controlled memory. It is the slowest ctrl
unit design.

o groups

G_9	G_{10}
5	7

rol ~~signal~~ word

3+6+8+11+
13+3+5+7
50 bits



bits are required to address H_N words = $\log_2 H_N$

" " " H_m " = $\log_2 H_M$

∴ Control Memory Size = μCM size + NCM size

$$H_m \left[\log_2 H_m * H_N \right] + H_N * P$$

// ctrl word accessed from T_m, T_N (Nano ctrl M)

* Follow the Horizontal up and reduce effective control memory

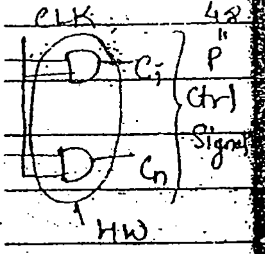
$$\boxed{T_{cs} = T_{\mu cm} + T_{ncm} + T_{HW}}$$

Q. let $H_m = 1024$, $H_N = 64$, $P = 48$

cm size $= \frac{1024\left[\frac{16}{\text{bits}}\right] + 64 \times 48 \text{ bits}}{8 \text{ bits / bytes}}$

$= (2048 + 384)$ B

$= 2432$ B



words = $\log_2 H_N$

$n = \log_2 H_M$

e + Ncm size

$4 \cdot H_N \times P$

(Nano Ctrl M)

// Using HMP for max degree of paralleli

Que- How many bytes are saved w.r.to 1-ctrl memory, which has to offer some degree of parallelism (Indirectly H.MP)

Single level cm $= 1024\,[48+10]$ bits

in bytes $= \frac{1024 \times 58}{8}$

$= 7424$ B

// More bits are required for HMP compar to NMP

HMP - 1-level design of memory

NMP - 2-level " " "

Ascending order of flexibility of ctrl unit design –

H W, H μP, V μP, Nano

If it will reversed, it repr. ~~access time~~ ascending order of the speed of operatn.

Nano, V μP, H μP, HW.

~~time~~

Assembly Language Program Tracing

Tracing
- finding the contents of specifying register
- Time requirement (time to complete) or clocks
- Space Requirement

‖ behaviour of program dictate by Key instr.

	Size (words)	F	F Cc
(1) LD R_0, R_1 (100)	4	2clock/word	4
(2) ADD R_1, R_0, R_1	2	1 " / "	2
(3) ADD R_0, R_1, R_0	2	1 " / "	2
(4) ST R_0[100], R_0	4	2 " / "	4

ctr) let $R_1 = 100$

$M[100] = 200$

$M[200] = 100$

~~time~~ (1) $R_0 \leftarrow M(R_1 + 100)$

zeram

R_1	R_0
100	100

(2) $R_1 \leftarrow (R_1) + (R_0)$

Tracing

R_1	R_0
200	100

s. of

or clocks (3) $R_0 \leftarrow (R_1) + (R_0)$

time to

R_1	R_0
200	300

(4) $M(R_0 + 100) \leftarrow R_0$

by $M(400) \leftarrow 300$

s)

R_1	R_0
200	300

F	F (clocks)
2 clock/word	4
1	2
1	2
2	4

The tracing of the prog. result R_1 ~~contents~~ contains 200, R_0 contains 300 &

$M(400) = 300$.

Space Req:

Let Each word is occupying 32 bits & the prog. is stored in a byte organised memory from the locatn 1000 onwards (decimal address). If an interrupt is occurred during the 1st ADD Instr., wat will be the addr. saved in the stack (take Previous records)

		Size (Words)
		4
LD 1000 – 1015	4B = 2^4 = 16 locatns	2
	Address of Next instr. i.e;	2
① ADD 1016 – 1023	1024 is saved which	4
② ADD 1024 – 1031	is to be return.	12 Words
ST 1032 – 1047		12×4 = 48B

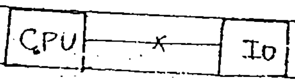
How many clocks are required to complete above prog. (Time complexity)

Size		F		E	Total (Clocks)
4	*	2 clock/word	+	4	8+4 = 12
2	*	1	+	2	2+2 = 4
2	*	1	+	2	2+2 = 4
4	*	2	+	4	8+4 = 12
					32 clocks

32 bits
e organised
words
= is
astr, Wat
stack
'Words)
words
=488
complete
s)
2 clocks

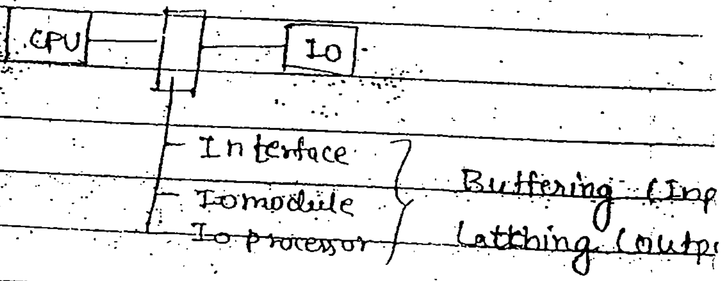
10-10-10

IO Organization



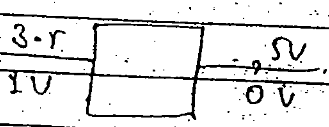
CPU Can't connect directly to Io bcoz of –
① Speed Mismatch
② Diff. device drivers
③ Diff. data format (8-bit patter and some 7-bit,
Sum take

Black box is placed betn them for their commu
to each others.



// Processor gives data then it is latched & when
~~$~~ take data it is buffering.
// Buffering is nothing but selective amplificat
Logic '1' signal gets amplified & logic '0' is
get reduced.

0 → 1 ... '0'
3.5 to 5 ... '1'



After buffering ~~log~~ '0' retained to '0' & '1' retained t logics will not change only quantity will change

Interface (No Error Correctn)

Based on Connecting device	Serial 8251 USART, PCI (Programmable (universal syn. Asyn.) Directn Comm. Interface)
	Parallel 8255 PPI (Programmable providing connectivity peripheral Interface)

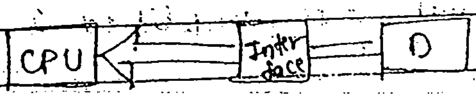
By placing the ctrl word into ctrl word register, the interface is programmed.

Device is serial & processor is parallel in Serial interface & it has to do additional conversions -

(1) Serial to parallel (Placed in receiver sectn)

(2) parallel to serial (placed in xmitter sectn)

// device given data in serial & change it to parallel



// Placed SIPO register in receiver sectn for ser. S to P conversion.

// Placed PISO register in xmitter sectn for P to S.

will charge

Module Error Correctⁿ

Interface can't do error correctⁿ but mod. does

(Programmable

mmm. Interface)

mmable

eral Interface)

IOP : If implement IO instructⁿ
(IO processor) (capable to deal with IO)

System has two processor:
① CPU processor
② IO processor

ord register,

allel in

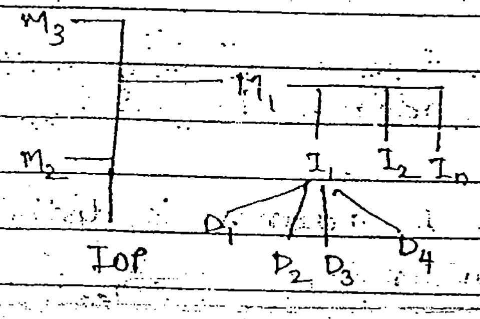
o additional

// IO processor deal with several module. Each module connected to several Interfaces & e. interface connected to several devices.

eceives sectⁿ)

nitter sectⁿ)

ange it to



ISSUES in IO

es sectⁿ for

① Addressing (How to address IO device)

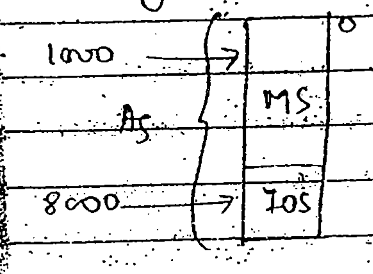
er sectⁿ for

② Data Xfer (Xfer from data device)

Addressing — Memory mapped IO
— IO mapped IO

Data transfer — Program Driven
— Interrupt Driven
— DMA

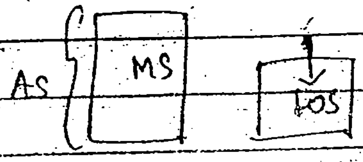
(Addressing) A_S → Partitioned into MS & IOS (memory space)



Can't have same address for MS & IOS.

Adv — ① Memory & IO address is distinct.
② All memory transfer Modes can be used in IO } Flexibility
~~LDA 8000 A ← [8000]~~
LDA 1000 A ← M[1000]

Disadv — Δ IOS effect memory space.
Soln —

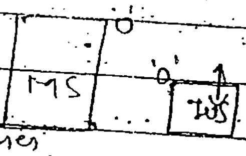


napped IO
IO
niven
Driven
& IOS
listinct.
xibility

Above solⁿ is named as Isolated IO (I
I

Adv of Above solⁿ-
△ IOS doesn't effect MS

disadv- Addresses are not distinct.



both addresses starts at '0'.

Solⁿ- IO/$\overline{M}$ is used (n+1 Address line
It ~~ctrl~~ bit is 1 address used for IO device
" " " " 0 " " Memory.
It separates IO instructns
IN OUT
& provides limited flexibility.

Program Driven-

One of the instrc checking the device status
If device is ready; it performs IO operatn
Above techinque called as program driven.

Adv- Simplicity (need not take any h/w).

disadv- processor is forced to wait till the device is ready i.e; inefficient usage of CPU.

③ Block transfer mode.

Burst Mode -
The Burst Mode delays the processor for longer time (System bus is returned only aft entire data has been transfered).

Cycle Stealing Mode -
The cycle stealing mode makes DMA to wait longer time. (the sys. bus is returned by DMA controller after every word xfer & acquired after every instr. cycle.

Block Xfer Mode -
The Block xfer mode maintains the advant of Burst mode & cycle stealing mode. If the block size is 1 word then it reduces to cycle stealing, if block covers entire data then it reduces to burst mode.

The processor performance is reduced due to DMA.

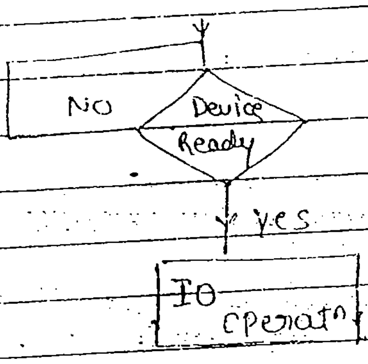
% of time processor blocked is

$$\boxed{= \frac{T_x}{(T_x + T_y)} * 100}$$

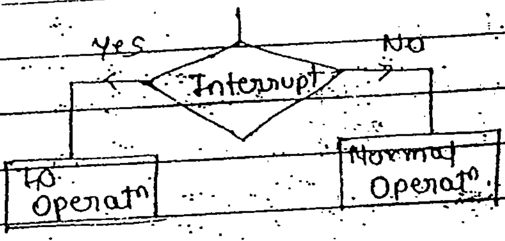
where,

T_x = word transfer time

T_y = word preparatn time



Interrupt Driven - (Device Controlled)

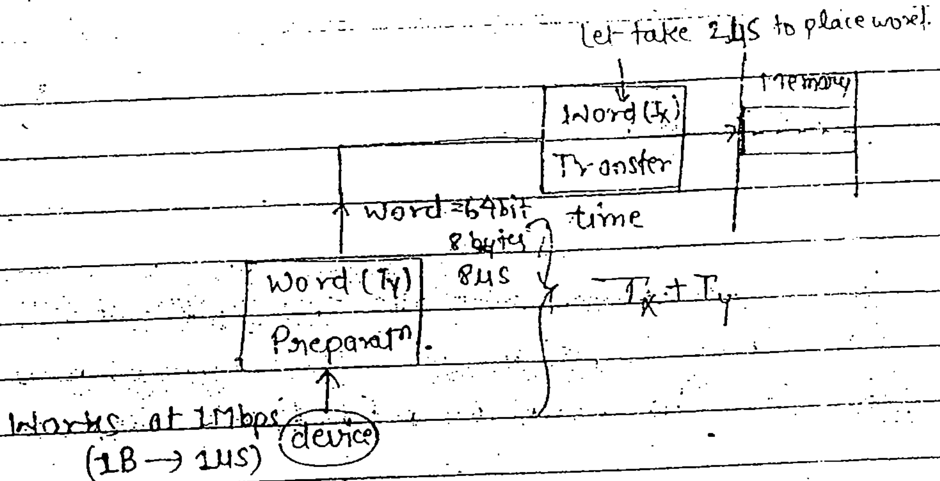


Adv - efficient usage of CPU
disadv - More complexity

DMA -
In a DMA, the processor is isolated from data path. The DMA controller coordinate the data Xfer; the bulk data can be Xfered in a rapid rate using DMA. the sys. bus is shared betn processor & DMA controller in master - slave mode.

Based on when the system bus is returned the DMA operating modes can be 1.

① Burst Mode
② Cycle stealing mode



System bus is not available at $T_x + T_y$ time i.e; processor gets blocked.

$\frac{2}{10} \times 100\%$

CPU blocked = 20%

If sys. uses 2 system buses theoretically, sys doesn't get blocked but practically it is having some waiting time & the costs are also get increased.

Issues in Interrupt driven - Implementatn

// processor efficiency is increased but complexity is also increased.

① Wher
② How
③ What
④ H
// In
① Ot
② I
w
③ I
④
Bra

is to place word.

memory

x+Ty time

oretically,
practically
& the

lementatn

CPU	Device
① When it should recognize	When the device shou
② How " " "	interrupt // At any time
③ What it's recognized	How to interrupt
	⊢ change the signal le
④ How to resolve	(level triggered
	⌊ change the edge.
	(edge triggeri

// Interrupt is Asynchronous.

TRAP – both level & edge triggered.

① After the completn of ~~1st~~ current instructn.

② INTR flag is used to recognize when – interrupt occur.

③ Branch to ISR (Interrupt service routi

④ Use priorities to resolve.

Branch to ISR

~~Use Device informatn "Vector interrupt"~~
↓
direct

device itself giving the address, it is known a,

Default location – "Scalar interrupt"

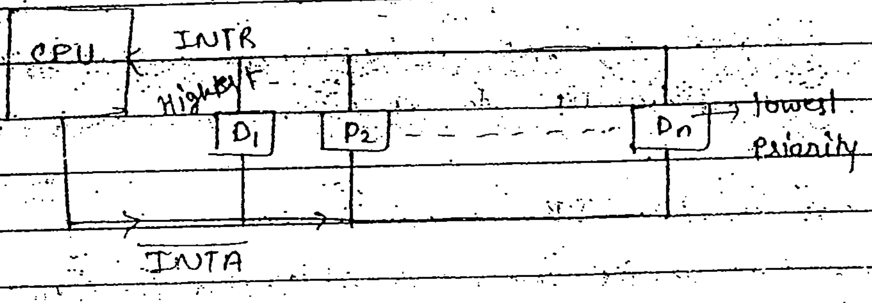
Implementing the priorities -

The priorities are implemented either by using :-

① Daisy chaining (Device chaining)
② Polling

Daisy chaining

Devices are positioned such a way the highest priority is closer to processor & lowest priority is farther to processor.



due to lowest priority, starvatn occurs.
Positions of priority are fixed so, it's also known as static or fixed priority.

Polling

Each device has INTR & INTA lines. The buses are polled device by device until it receives interrupted one. By changing the order then by

he highest
owest priority

→ lowest
priority

so, it's

lines . The
e until it
ging the order

of pulling, dynamic priorities get changed.
due to this starvatn will reduced but
is not economical.

Secondary Memory.

(1) Disk Memory - the magnetic disk is a
random access memory. The unit of xfer
one sector. The disk sys. is associating with
disks stayed together cylinder is the unit
accessed in the disk system, the disk addr
refers the concerned sector.

Drive offset	Surface offset	Track offset	Sector offset

Sector is addressed by either logical or lin
address

$$\langle \underset{\substack{\uparrow \\ \text{cylinder}}}{C}, \overset{\substack{\text{Surface} \\ \uparrow}}{S}, \text{Sector} \rangle \rightleftarrows \text{linear add}$$

logical address
- ~~which cylinder it belongs~~
- which surface it belongs
- & sector is in which surface

Based on the track capacity, the disk can be
constructed using constant & ~~linear~~ recording capa
density.
Variable

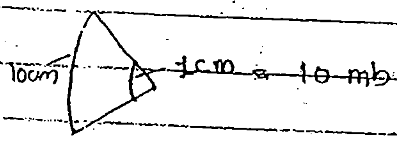
Angular Speed is going to changed acc. to
Velocity tracks
for inner track – less velocity
" outer " – ~~More~~ more velocity
Concerned with variable recording density.
linear Velocity ← It is concerned with constant recording density.
Capacity of track is changed but distance betn bits is ~~eq~~ constant

I Data xfer time

Q Consider A disk which is having 10 equidistance track. The inner track diameter is 1 c m & outer track dia. is 10 cm.
① What is the capacity of the disk if it is using constant linear velocity
② Constant angular velocity

Header

Answer – (a) 100 MB, 550 MB (c) 100MB, 100 MB
✓(b) 550MB, 100 MB (d) 550MB, 550 MB

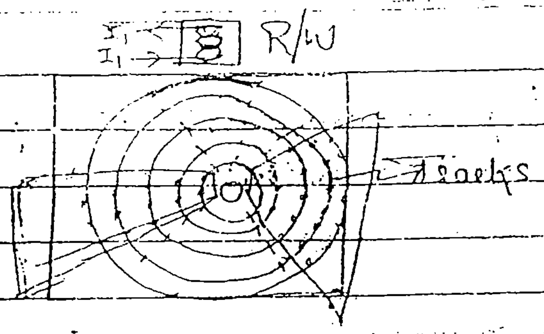


apacity

anged i.e

acity

6-8 dbms
8.30 – 10 – LP



The perimeter of track is varied.
R/W Contains the conductor

$$T = T_{seek} + T_{rotational\,latency} + T_{Data\,xfer\,time}$$
(Fixed)

seek time & rotational time are dominating
Semi-random access
Rotational latency - getting the sector under R/W head : (Serial access)

~~As the perimeter is changing~~
No. of bits varies track to track.

Constant recording density - Track capacity (distance in betn bits is going to be changed) varies
Variable " " - Row is changed i.e

e ⟨ e_{min} (for outer track) / e_{max} (for inner track)

It is having constant track capacity.