CO

Memory organisation : Hierarchy, cache memory - Address mapping
- updation
- Replacement techniques

principle of virtual and associate memories

2. Instructions, Addressing modes, Instruction pipeline
(Fixed point and Floating point arithmatic)

3. control unit design - μ operations, HW control unit design,
μ programmed control unit design,
Tracing assembly program

4. IO : addressing, data transfer technique, disk and tape memories.

Textbook:

computer architecture : J.P. Hayes
computer organisation : paul chowdary

⇒ computer architecture - It does not deal with physically touchable components just like civil architect.

- It concerns with conceptual design of the system
- It concerns instructions, addressing modes & data format.

Based on instructions, systems are

1) CISC (complex Instruction set)
2) RISC (Reduced Instruction set)
↳ Here every instruction completes in 1 clock cycle
ie clocks per Instruction (CPI) = 1

CISC - More instructions and CPI is different for different instructions

- programmer overhead is less
- compiler designer overhead is more

RISC - constant set of instructions
programmer overhead is more
compiler designer overhead is less

Addressing modes
↓
how operand reference is given in the instruction

more addressing modes ⇒ more flexibility

Immediate Direct Indirect Relative - -

CISC has more flexibility ~~to~~ as it has more addressing modes than RISC

Data format
↓
How to interpret/intercept the binary string

Eg: 1) some systems use 7 bits for character ⇒ ASCII
2) some systems use 8 bits for character ⇒ EBCDIC

computer organisation:
- It deals with logical design of the system
- It concerns with physical devices and their interconnection with the perspective of improving the performance

↳ goal of CO

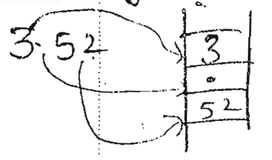
similar to library organisation (how to arrange books)
goal: Reducing average access time

Performance of a machine ⇒ Amount of work done in unit

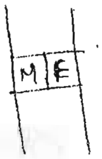
No. of instructions executed, Measured in MIPS (Million Instructions per second)

suppose MIPS = x then goal is increasing x.
This can be done by reducing the time for each individual instruction

floating point.



3 memory referencess } without changing H/w and s/w, performance is increase



1 memory reference to retrieve number

→ In RISC more no. of registers.

Accessing registers takes less time than accessing memory

∴ parameter sharing is easy in RISC

Eg: Nested subroutines

→ 90% to 10% rule:

90% of the program time is consumed by 10% of program code. This 10% program code is called as locality of reference. This locality of reference is supplied to the processor fastly so the execution time of program could be reduced

∴ Increase in performance

for this purpose 1st memory is introduced This is cache memory.

→ Every instruction

Hardware - Fetch (F)

H/w2 - Decode (D)

H/w3 - Execute (E)

H/w3 - Store (S)

} Execution cycle

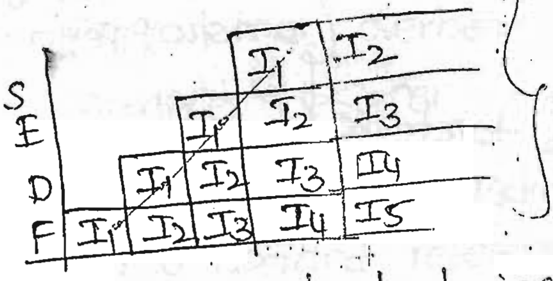
ie Each stage requires different hardware

At any time, an instruction utilises only one stage H/w

∴ idle H/w is $\frac{3}{4}$ (conventional way)

- So a technique that allows efficient utilization of resources - Instruction pipeline

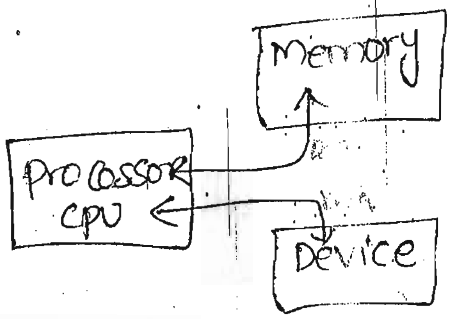
- overlapping of instruction stages:-



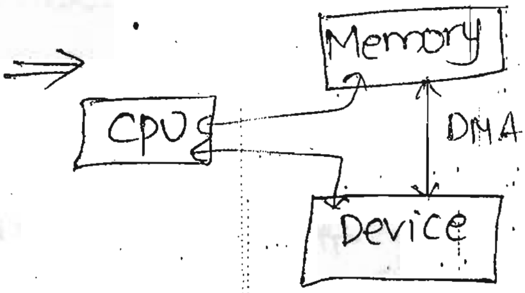
Here individual instruction time is not reduced.
But the average instruction time is reduced

Here all the resources are busy

- Data is either in memory (or) device finally



(conventional way)

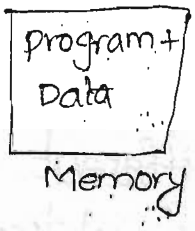


(Isolating processor from data transfer)

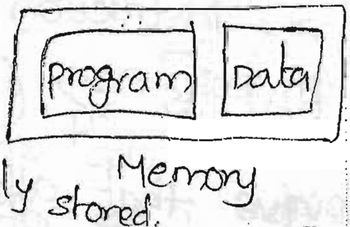
30/10/11

classification of computer Architecture :

1) memory based architecture
- von neumann
- stored program
- Inplace modification



2) Harvard Architecture
- similar to von neumann
- program & data are separately stored.
- parallel access of data & Instructions (program)
- modification of program does not disturb data and vice versa



Super scalar architecture

- separate processing of integers and floating point numbers

FPU IPU

- FLYN architecture

Basis: Instruction stream and data stream (CU, PEM, MU)

SISD - single Instruction stream & single data stream ⇐ serial processing

SIMD - " " " & Multiple " ← array processing (CU, N-PEM, memory modules)

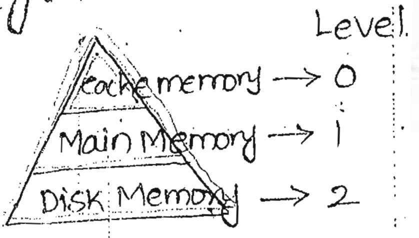
MISD -

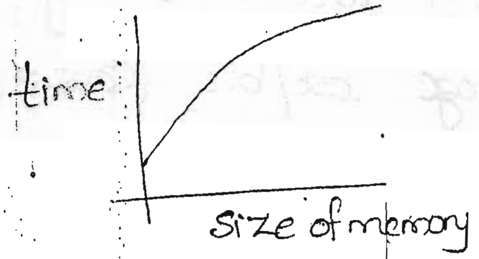
MIMD - parallel processing (n- control units (CU), m- processing element (PEM), n memory modules)

Memory organisation:

Not physically placing memories. But what information is to be stored in which memory — Memory organisation

Memory hierarchy:





In the memory hierarchy, ith level memory is placed higher than $(i+1)^{th}$ level memory. (Time of access $T_i < T_{i+1}$

size of memory $S_i < S_{i+1}$

cost/bit $C_i > C_{i+1}$

frequency of access $f_i > f_{i+1}$

Instructions $I_i \subset I_{i+1}$)

The performance of hierarchy is given by hit ratio. (Availability of referred information at referred level is called as hit).

The Hit ratio of bottom most memory is 1 ie Hit ratio of disk memory is 1.

Hitratio (H) $\propto$ size (S)

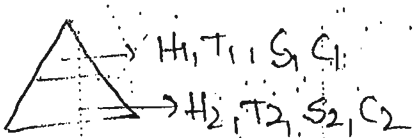
$H \propto \dfrac{1}{\text{avg.access time (Tave)}}$

$\left(\begin{array}{l} H \propto T \\ H \propto \frac{1}{T} \end{array} \right\}$ cant bederived Because individual access times are not affected by hit)

he side effect of memory hierarchy is data inconsistency same information is available differently at different levels) this problem is resolved using proper updation techniques.

nathematical expressions for memory hierarchy:

Let H_1, T_1, S_1, C_1 gives the specifications of level 1 memory of 2 level system. H_2, T_2, S_2, C_2 are the specifications of level2 memory

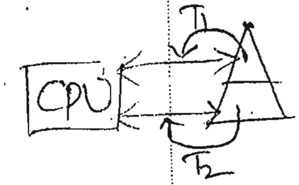


1) $H_2 = 1$

2) Size of total memory $= S = S_1 + S_2$

3) Average cost/bit $C_{ave} = \dfrac{C_1 S_1 + C_2 S_2}{S_1 + S_2}$ Total cost of level2 system, Total cost of level 1 system

Average access time $T_{ave} = H_1 T_1 + (1 - H_1) T_2$ (default)



If $H_1 = 1 \implies T_{ave} = T_1$

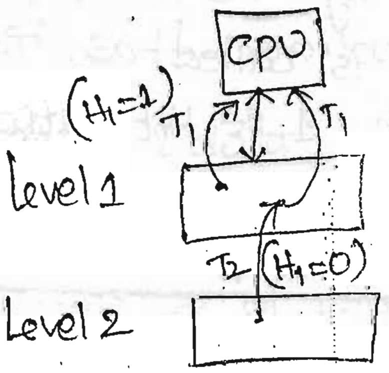
If $H_1 = 0 \implies T_{ave} = T_2$

↓ If it is available in level 1.

The above hierarchy is not strict hierarchy

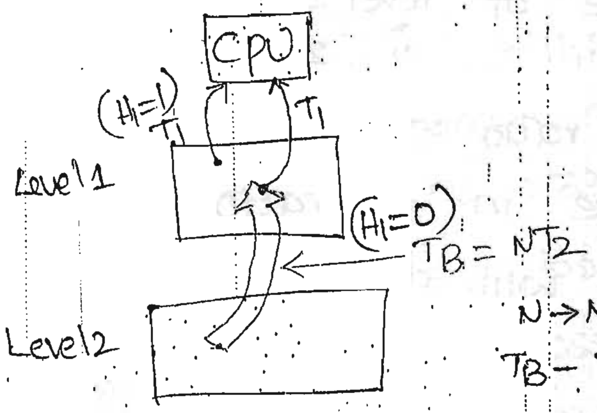
↓

processor can interact with only one level of memory



$T_{ave} = H_1T_1 + (1-H_1)(T_1+T_2) = T_1 + T_2(1-H_1)$

when there is miss in level 1, instead of taking a word from level 2 to level 1, a group of words are moved (Locality of reference)



$N \rightarrow$ No. of words.

T_B - Time to move block

$$\boxed{T_{ave} = H_1T_1 + (1-H_1)(T_B+T_1)} \Rightarrow T_1 + (1-H_1)T_B$$

4

From bus, data is moved word by word. But we are moving a block. At a time it can't be placed. so placement time is to be considered

placement time, $T_B = N(T_2+T_1)$

In general,

for n level $T_{ave} = T_1 + (1-H_1)T_{B_1} + (1-H_1)(1-H_2)T_{B_2} + \cdots\cdots +$

$+ (1-H_1)(1-H_2)\cdots(1-H_{N-1})T_{B_{N-1}}$

p) consider a 2-level memory system, in which the average access time without level 1 is 120ns, with level 1 it is 30ns the level 1 access time is 20ns. what is hit ratio?

Sol: for a 2 level system

$T_{ave} = H_1T_1 + (1-H_1)T_2$

Given $T_2 = 120ns$

$T_{ave} = 30ns$

$T_1 = 20ns$

$\Rightarrow 30 = H_1 \times 20 + (1-H_1) \times 120$

$100H_1 = 90 \Rightarrow H_1 = 0.9$

P) consider a 2level memory system in which the access speed of level 1 memory is 5 times faster than level 2. The hit ratio is 0.8. It is needed to increase the average access time by 20% from 60ns

1) what is the access time of level 1 memory
2) what is the " " " 2 "
3) what is the new hit ratio
4) what is the % of change in hit ratio
5) If Hit ratio = 100%, what will be the access time of T_1 & T_2

Sol: consider default. $T_{ave} = H_1T_1 + (1-H_1)T_2$.

Given $T_1 = \frac{T_2}{5}$ (level 1 is 5 times faster than level2)

Initially $H_1 = 0.8$

$T_{ave} = 60ns$

$\Rightarrow 60 = (0.8)T_1 + (1-0.8)5T_1$

$60 = 0.8T_1 + T_1$

$\Rightarrow T_1 = \frac{60}{1.8} = 33.3$ ns.

$\therefore T_2 = 166.5$ ns

∴ Access time of level 1 = 33.3ns

Access time of level 2 = 166.5ns

iii) $H_{New} = ?$, $T_{aven} = T_{ave} + (20\%)T_{ave}$

$= 60 + \frac{20}{100} \times 60$

$= 72$ ns

$\Rightarrow 72 = H_N(33.3) + (1-H_N)(166.5)$

$72 = -4 \times 33.3\, H_N + 166.5 \Rightarrow H_N = 0.71$

iv) % Hit ratio reduced

$$\overset{H_{old}}{0.8} \dashrightarrow \overset{H_{new}}{0.71}$$

ie for $0.8 \rightarrow 0.09$ reduction

$100 \longrightarrow ?$

$$= \frac{0.09}{0.8} * 100$$

$= 11.2\%$

v) Hit ratio does not change individual access times. Hence

$$T_1 = \frac{100}{3} \text{ ns} = 33.3 \text{ ns}$$

$$T_2 = \frac{500}{3} \text{ ns} = 166.5 \text{ ns}$$

* Hit ratio effects average access time.

5) consider 3-level memory system in which individual access times are $T_1 = 10$ ns/word $T_2 = 100$ns/word. $T_3 = 1000$ns/word and $H_1 = 0.8$, $H_2 = 0.9$, $H_3 = 1$. If the referred word is not available in level 1 memory, a 2 word block is moved from level2 to level1. and referred word is handed over to the processor. If it is not present in level 2, a 4 word block is first moved from level3 to level2 and the concerned block is taken to level 1. what will be the average access time and throughput of the system

Sol: without placement time

$$T_{ave} = T_1 + (1-H_1)T_{B_1} + (1-H_1)(1-H_2)T_{B_2}$$

$$T_{B_1} = 2 * T_2 \quad T_{B_2} = 4 * T_3$$

$$= 10 + (1-0.8)\, 2 * 100 + (0.2)(0.1)\, 4 * 1000$$

$T_{ave} = 130$ ns

throughput = No. of words accessed/transferred per second

$$= \frac{1}{T_{ave}} \text{ word/sec}$$

$= \frac{1}{n}$ = 7.6 Mwords/sec

suppose placement time is considered

$$T_{ave} = T_1 + (1-H_1)T_{B_1} + (1-H_1)(1-H_2)T_{B_2}$$

$$T_{B_1} = 2(T_2+T_1) \qquad T_{B_2} = 4(T_3+T_2)$$

44
56
10
2

$$T_{ave} = 10 + (0.2)*2*110 + (0.2)(0.1)\,4*1100$$

$$= 142\,ns$$

$$\text{Throughput} = \frac{1}{142\times10^{9}} = 7.04 \text{ Mwords/sec}$$

04/11/11

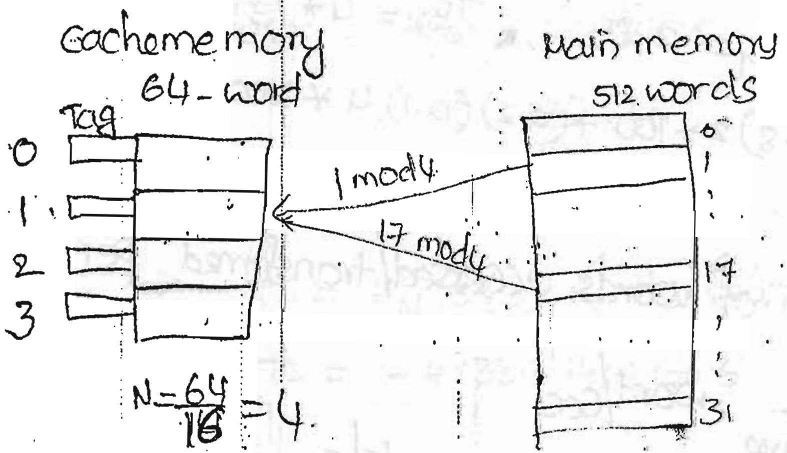
Address mapping

- cache memory and main memory are to be divided into equal sized partitions
- Address mapping decide which main memory block is to be placed where in the cache
  1. Direct mapping
  2. Associate mapping
  3. set-associate mapping

→ Direct mapping

- K^{th} main memory block in $(K \bmod n)^{th}$ cache location
- Tag bits denote which main memory block is currently residing in that cache position

Eg :



$N = \frac{64}{16} = 4.$

$N = \frac{512}{16} = 32$

main memory blocks

0, 4, 8, 12, 16, 20, 24, 28 compete for cache block '0'

$\downarrow$ 000, $\downarrow$ 001, ... $\downarrow$ 110, $\downarrow$ 111

$\therefore$ 8 blocks $\rightarrow$ 3 bits needed

for cache block 1 → 1, 5, 9, 13, 17, 21, 25, 29

for cache block 2 → 2, 6, 10, 14, 18, ~~22~~ 22, 26, 30

for cache block 3 → 3, 7, 11, 15, 19, 23, 27, 31.

Therefore 4 clusters & each cluster having 8 blocks

$\therefore$ No. of clusters = No. of cache blocks

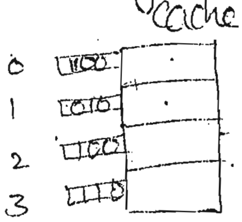
$$\text{No. of blocks in each cluster} = \frac{\text{No. of mainmemory blocks}}{\text{No. of clusters (or) cache blocks}}$$

$$= \frac{M}{N}$$

6

No. of tag bits(T) = $\log_2\left(\frac{M}{N}\right)$

→ suppose tag information



$\Rightarrow$ In cache block 0 — main memory block 16

" 1 — " 9

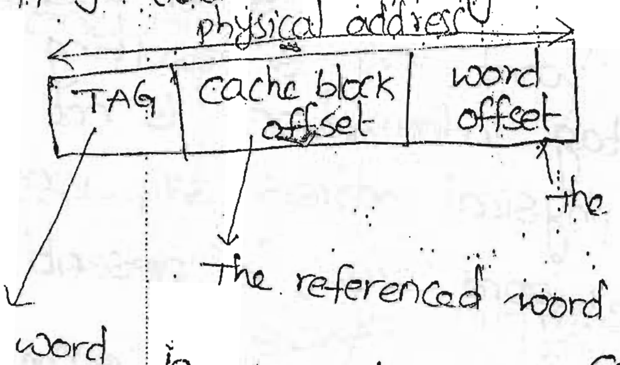
" 2 — " 18

" 3 — " 27

$\therefore$ Different main memory blocks are in cache

$\therefore$ Non contiguous

- direct mapping - address mapping is



the location of addressed word in the block.

The referenced word is likely to be in which cache block

If word is present in cache, this tag should match with tag of associated cache block

Suppose there are p words/block

word offset = $\log_2 P$

cache block offset = $\log_2 N$

Tag = $\log_2(\frac{M}{N})$

Eg: whether 374 word of main memory is present in cache block?

Sol: physical address requires 9 bits as it is 512 word main memory

$$374 = \overset{256}{1}\ \overset{128}{0}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{1}\ \overset{8}{0}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{0}$$

$\underbrace{1\ 0\ 1\ 1\ 1}$ → main memory block = $\overset{16\,8\,4\,2\,1}{10111}$

23rd block

⇓

23 % 4 = 3 block (cache)

Here

word offset = $\log_2 16$ = 4 bits

Cache block offset = $\log_2 4$ = 2 bits

tag = $\log_2 8$ = 3 bits

Tag	Cache block offset	word offset
101	11	0110

∴ This word has to be placed in cache block 3

The tag information of cache block is 110

But this physical address has tag 101

Not matched

The referred word is not present in cache

→ The reference to the word 374 is resulting in miss since the 3rd block tag information is not matching with tag portion of physical address 374 word

Eg: check whether the word 293 is present in cache (or) not

$$293 = \boxed{\overset{256}{1}\ \overset{128}{0}\ \overset{64}{0}\ |\ \overset{32}{1}\ \overset{16}{1}\ 0\ |\ \overset{8}{0}\ \overset{4}{1}\ \overset{2}{0}\ \overset{1}{1}}$$

→ cache position = 18 % 4 = 2 block

cache block 2

Tag matches ∴ word is present

Since the physical address of word 293 is referring the 2nd cache block and its tag address is matching with 2nd cache block tag address, the reference is resulting hit

H/w implementation for direct mapping:

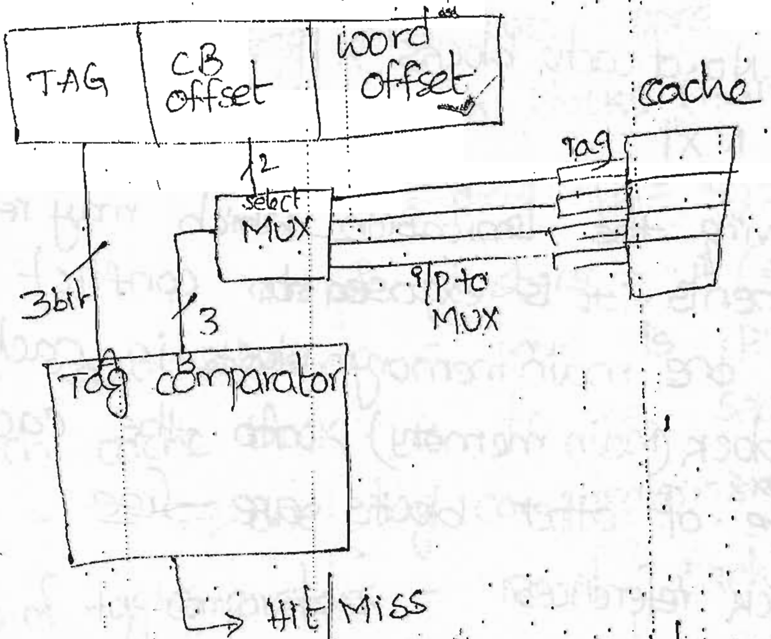
- To say whether the word is present (or) not we checked only one cache block i.e. compared with tag of a particular cache block ⇒ KmodN block

∴ Tag comparator is needed

one i/p is tag of physical address.

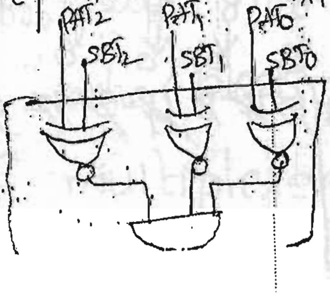
other i/p is tag of selected cache block. ⇒ For this MUX is needed

Its select lines are CB offset bits in physical address



Time for knowing hit|miss ⇒ $T_{Hittime} = T_{mux} + T_{comparator}$

Internally comparator is



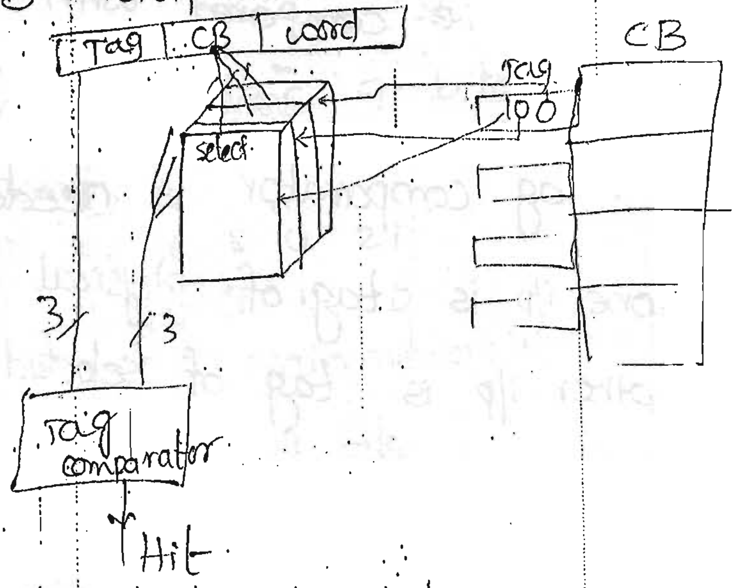
relay in tag comparator ⇒ $T_{comp} = T_{EX\text{-}NOR} + T_{AND}$

→ Here one problem

multiplexer can ~~be~~ give output only one bit at a time

But tag is 3 bits

Hence 3 multiplexers are needed



→ No.of multiplexers required = No. of tag bits

$$= \log_2\left(\frac{M}{N}\right)$$

Size of each mux = No. of cache blocks × 1

$$= N \times 1$$

⇒ The direct mapping is having the limitation which may result too many block replacements. It is exposed to conflict penalties i.e. the presence one main memory block in cache is obstructing incoming block (Main memory) into the cache for same position inspite of other blocks are free.

Eg: consider main memory block references

4 8 0 4 8 4 0 12 0 12

(K mod 4) 0 0 0 0 0 0 0 0 0 0

0 ~~4~~ ~~8~~ ~~0~~ ~~4~~ ~~8~~ ~~4~~ ~~0~~ ~~12~~ ~~0~~ 12

1

2

3

0 hits

10 faults/miss

- This problem is resolved using associate mapping

<u>Associate mapping</u>:

- In associate mapping, any block of main memory can be placed any where in the block

<u>eg</u>.

4	8	0	4	8	4	0	12	0	12
F	F	F	H	H	H	H	F	H	H

0	4
1	8
2	0
3	12

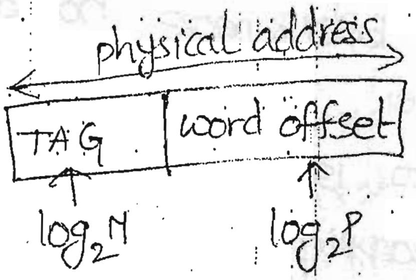
∴ 6 hits

4 faults / miss

Hit ratio $= \frac{6}{6+4} = 0.6$

In associate mapping, the physical address is divided into two portions

8



P words/sec

all the main memory blocks are competing for each cache block

$\Rightarrow$ tag bits are more

∵ M blocks (Main memory)

∴ tag bits $= \log_2 M$

No. of clusters = 1 (All main memory blocks)

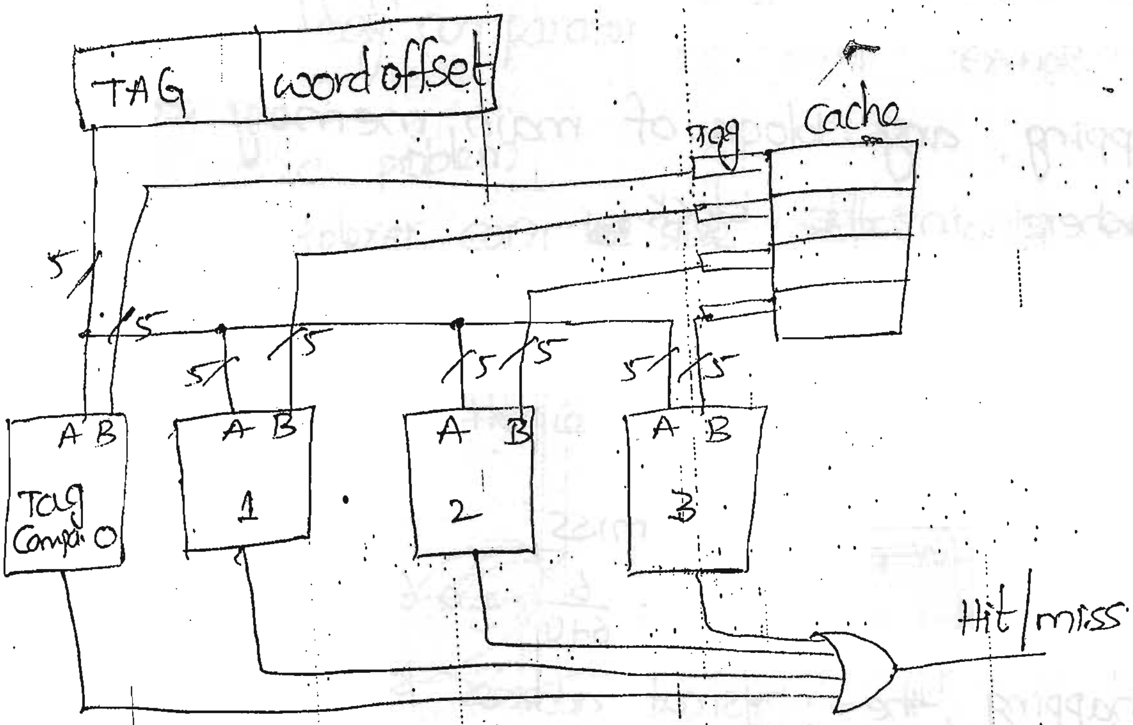
- To know whether a word is present (or) not in cache, each cache block is to be searched.

$\Rightarrow$ one tag comparator is needed for each cache block

No. of tag comparators = No. of cache blocks

If all tag comparators says miss then the word is not present in cache block

- There is no need of multiplexer



- It requires more tag bits, more no. of comparators
  ⇒ More H/w
  Hence cost is more

- It is fastest in mapping

Set-associate mapping:

The set-associate mapping contain advantages of both direct mapping and associate mapping

- The cache is divided into logical sets
- In K-way set-associate mapping, each set is allocated with K- cache blocks

$$\text{No. of sets} = S = \frac{\text{No. of cache blocks } (N)}{K}$$

- The physical address is divided into

physical address

Tag	set offset	word offset
$\log_2\left(\frac{M}{S}\right)$	$\log_2 S$	$\log_2 P$ bits

p words/block

Eg: N = 4 , 2-way set association, M = 32

$\therefore S = \frac{4}{2} = 2$

Tag	set	word
4	1	4

If it is one-way set associate mapping then it reduces to direct mapping (No. of sets = No. of blocks)

If it is N-way set associate mapping then it reduces to associate mapping

20/11/11

Q 2 million word main memory and 4K word cache are partitioned into 256 word blocks. The word addressable. memory has 32-bit words

1) what is the size of main memory & cache memory.

Main memory size $= 2\times2^{20}\times 32\ b$

$= 2\times2^{20}\times 2^{2}\ B$

$= 8\ MB$

9

cache memory size $= 4\times2^{10}\times 32\ b$

$= 4\times 2^{10}\times 2^{2}\ B$

$= 16\ KB$

2) How many bits are required to address physical memory?

It is word addressable

Total words $= 2\times2^{20}$ words

$= 2^{21}$

∴ 21 bits are required

3) How many blocks are present in cache memory & main memory.

Total no. of blocks in main memory

$$M = \frac{\text{Total words in main memory}}{\text{blocksize}}$$

$$= \frac{2^{21}}{2^{8}} = 2^{13} = 8K \text{ blocks}$$

Total number of blocks in cache memory

$$N = \frac{\text{Total words in cache memory}}{\text{blocksize}}$$

2^{12} 1K blocks

If it is one-way set associate mapping then it reduces to direct mapping (No of sets = No of blocks)

If it is N-way set associate mapping then it reduces to associate mapping

06/11/11

Q) 2 million word main memory and 4K word cache are partitioned into 256 word blocks. The word addressable memory has 32-bit words

1) what is the size of main memory & cache memory.

Main memory size $= 2 \times 2^{20} \times 32$ b

$= 2 \times 2^{20} \times 2^2$ B

$=$ 8 MB

cache memory size $= 4 \times 2^{10} \times 32$ b

$= 4 \times 2^{10} \times 2^2$ B

$=$ 16 KB

2) How many bits are required to address physical memory?

It is word addressable

Total words $= 2 \times 2^{20}$ words

$= 2^{21}$

$\therefore$ 21 bits are required

3) How many blocks are present in cache memory & main memory

Total no of blocks in main memory

$$M = \frac{\text{Total words in main memory}}{\text{blocksize}}$$

$$= \frac{2^{21}}{2^8} = 2^{13} = 8\text{K blocks}$$

Total number of blocks in cache memory

$$N = \frac{\text{Total words in cache memory}}{\text{blocksize}}$$

2^{12} 16 blocks

since the physical address of word 293 is referring the 2^{nd} cache block and its tag address is matching. with 2^{nd} cache block tag address, the reference is resulting hit

H/w implementation for direct mapping:

TO say whether the word is present (or) not we checked only one cache block

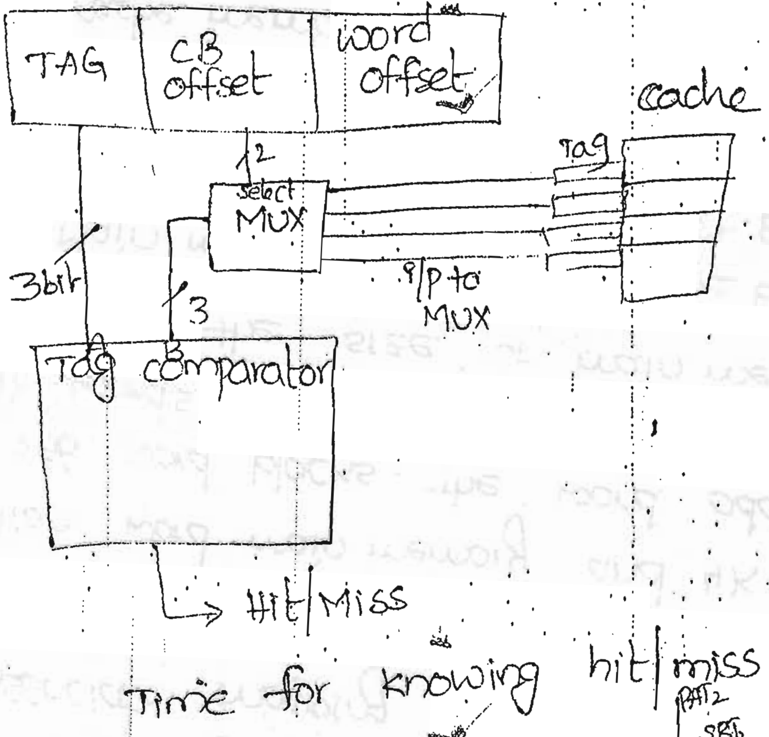
i.e. compared with tag of a particular cache block ⇓ KmodN block

∴ Tag comparator is needed

one i/p is tag of physical address.

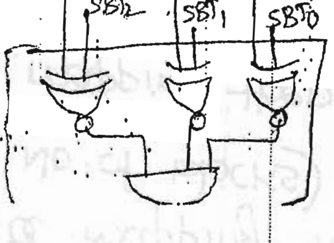
other i/p is tag of selected cache block. ⇓ For this ~~ca~~ MUX is needed

Its select lines are CB offset bits in physical address



Time for knowing hit/miss ⇒ $T_{Hittime} = T_{mux} + T_{comparator}$

Internally comparator is



he blocksize in cache2 is 64 words. A system of 32 bit main memory address with 16 bit word employees 256KB cache memory. compute the tag bits & tag memory for each case.

Sl: <u>cache 1 design</u>:

Main memory ~~blocks~~ size = $\frac{2^{32}}{2^4} = 2^{28}$ ~~blocks~~ $2^{32} \times 2 = 8GB$

cache 1 block size = 1K words = 2KB

cache size = 256 KB

$= \frac{256 KB}{2 B}$

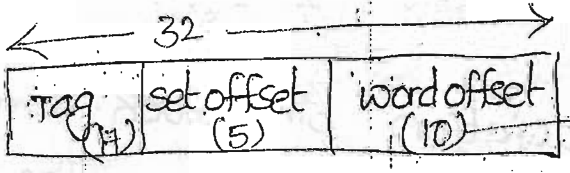
= 128 Kwords

∴ No. of blocks in cache 1 = N = $\frac{128 K}{1K}$ = 128 blocks

10

It is 4-way set association

∴ No. of sets = $\frac{128}{4}$ = 32 sets



Tag (17)	set offset (5)	word offset (10)

1K words & It is word addressable

∴ Tag bits = 17

Tag memory = N * tag

= 128 * 17

<u>cache 2 design</u>:

cache2 block size = 64 words = 128 Bytes

cache size = $\frac{256 KB}{128 B}$ (OR) $\frac{128 KW}{64 W}$

= 2K blocks

It is 8-way set association

∴ No. of sets = $\frac{2K}{8} = 2^8$ sets

Tag	set offset	word offset
(18)	(8)	(6)

→ 64 words & is word addressable

∴ Tag bits = 18

Tag memory = N * tag
= 2K * 18
= 36K bits

P) consider a main memory of 2C K blocks and 2-way set association is used. The position of pth block of main memory in cache ~~of~~ 2C block is

ⓐ p mod 2K set ⓑ p mod K set ⓒ p mod 2C set [✓ ⓓ p mod C set]

$$\text{No. of sets} = \frac{\text{No. of blocks in cache}}{K}$$

$$= \frac{2C}{2}$$

$$= C.$$

∴ In set association pth block is placed in p mod S set where S is total no. of sets

P) consider a direct mapped cache, where the no. of tag bits is equal to the no. of words in block. No. of words in cache equal to no. blocks of in cache. Let there are N blocks in cache, what is the the size of physical memory in words.

[✓ ⓐ $N^2 \cdot 2^N$] ⓑ $2N \cdot 2^{N^2}$ ⓒ $N \cdot 2^N$ ⓓ none

Sol: Direct mapping

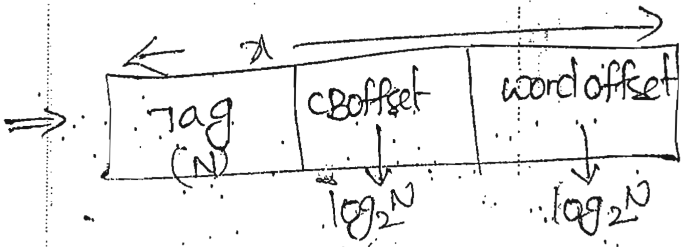
← physical memory address →

Tag	CBoffset	word offset

cache has N blocks

Each block has N words

∴ TAG bits = N



$x = N + \log_2 N + \log_2 N$

$= N + \log_2 N^2$

No. of bits in physical address $= N + \log_2 N^2$

∴ Total no. of words $= 2^{N + \log_2 N^2}$

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

$= 2^N \cdot N^2$

11

) A system employes 1000 references. Out of them 100 references result conflict penalities 150 references result compulsory penalities. Another 150 references result capacity penalty. what will be the hit ratio with direct mapping and associate mapping.

Sol: Direct Mapping → conflict penalities present ✓

Associate " → No " " ✓

Miss ratio in direct mapping $= \frac{100+150+150}{1000}$

$= \frac{400}{1000} = 0.4$

∴ Hit ratio $= 1 - 0.4 = 0.6$

Miss ratio in associate mapping $= \frac{150+150}{1000}$

$= \frac{300}{1000} = 0.3$

Hit ratio $= 1 - 0.3 = 0.7$

compulsory penalty are due to fixed block size. The more the size of the block, the less the compulsory penalities

Eg: If block size = 16 word
and we request 17 words ⟹ new block
∴ penality

If block size = 24 words then no penality.

capacity penalities are due to the fixed size of cache. The more the size of the cache, the less the capacity penalities

- If cache size = Main memory then all the penalities are absent

updation techniques:

- The write operation result cache coherence problem where as the read operation does not give any problem. In order to reduce/avoid this problem two updation techniques are proposed.
  1. write- Through
  2. write- Back
- In the write-through updation, the cache memory and main memory are simultaneously updated. The cache coherence problem is eliminated with write-through updation in a single processor system

the time for updation = max (T_c, T_m)
$= T_m$

T_c - cache memory access time
T_m - Main " "

The write through updation gives better performance for less number of updations

- In write-back updation, the updation of main memory is done only when the concerned block in cache is chosen for replacement. only last updation is visible ~~to~~ in the main memory. To know whether the block in the cache is ~~[illegible]~~ updated (or) not an additional bit (dirty bit) is allocated ⇒ At the time of replacement, the block that is chosen for replacement require either it is to be copied back into the main memory before getting the income block (dirty bit = 1) (OR) it is simply overwritten with incoming block (dirty bit = 0)

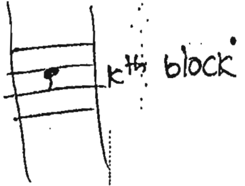
Time for updation is.

$$T_{updation} = \underbrace{2T_B}_{(T_{B\,copy\,back} + T_{B\,incoming})} + T_C \quad (\text{if dirty bit}=1)$$

12

$$= T_B + T_C \quad (\text{if dirty bit}=0)$$

eg: for (i=0; i<1000; i++)
{
=
}



In write through cache block updated 1000 times
main memory " 1000 "

In write back cache is updated 1000 times
Main memory " only 1 time

Q) consider a cache memory which has 80% hit for read operation and 90% hit for write operation. The cache is 5 times faster than main memory. The main memory access time is 100ns, Let there are 20% references for write operation. Answer the following questions using write through strategy.

1) what is the average read access time

2) " " write "

3) " " access time

4) what is the throughput of system

Note: whether it is read miss (or) ~~average~~ write miss a 2word block is first moved from main memory to cache memory

Sol:

1) Average read access time

$T_{ave_r} = H_r T_c + (1-H_r)(T_B + T_c)$ Here no replacement

T_m = Main memory access time = 100 ns

T_c = cache " " " $= \frac{100}{5} = 20$ ns

H_r = Read Hit ratio = 80% = 0.8

$T_{ave_r} = 0.8 * 20 + (1-0.8)[2*100+20]$

$= 16 + 44$

$= 60$ ns

2) Average write access time

$T_{ave_w} = H_w * T_{updation} + (1-H_w)(T_B + T_{updation})$

write-allocate – updated both in Cache & Main memory.
It is default

$T_{updation} = Max(T_c, T_m)$

$= 100$ ns

$T_{ave_w} = 0.9 * 100 + (1-0.9)[2*100+100]$

$= 90 + 30$

$= 120$ ns

3) Average access time ⇒ for a single word how much time?

Here reference can be write (OR) Read

$T_{ave} = f_r * T_{ave_r} + f_w * T_{ave_w}$

f_r – Read frequency = 80%

f_w – write frequency = 20%

$\therefore T_{ave} = 80\% * 60 + 20\% \; 120$

$= 48 + 24$

$T_{ave} = 72$ ns

4) $\text{Throughput} = \dfrac{\text{No. of words}}{\text{sec}}$

$= \dfrac{1}{T_{ave}}$

$= \dfrac{1}{72 * 10^{-9}}$ ns|word

$= \dfrac{10^9}{72}$ words/sec

$= \dfrac{1000}{72} * 10^6$ words/sec

= 13.8 million words/sec

13

20/11/11 **Block replacements:**

objective: Reduce the number of block movements

1) FIFO ---- Arrival time

2) LRU --- Basis : usage

3) Direct mapped --- (K mod N)th cache block

- performance of block replacement techniques is based on the number of hits

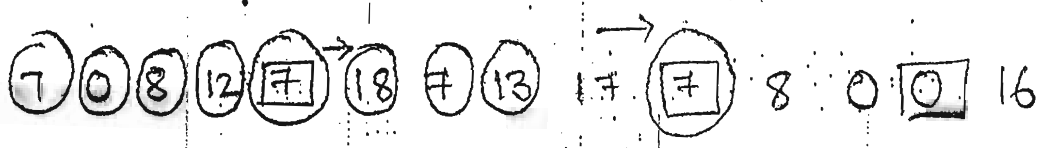
Eg:

consider a cache with 4 blocks

Main memory block references:

7 0 8 12 7 18 7 13 17 7 8 0 0 16

FIFO:



○ → moved out of cache

→ → movement

□ → Hits

∴ In FIFO,

Hits	Last block moved out of cache	status of cache
3	13	17 8 0 16

LRU :

◌ → old reference

7 ⓪ ⑧ ⑫ $\boxed{7}$ ⑱ $\boxed{7}$ ⑬ ⑰ $\boxed{7}$ 8 0 $\boxed{0}$ 16

Hits	Last block moved out of cache	status of cache
4	✓ 17	7 8 0 16

Direct Mapped :

7 ⓪ ⑧ ⑫ $\boxed{7}$ 18 $\boxed{7}$ ⑬ 17 $\boxed{7}$ ⑧ ⓪ $\boxed{0}$ 16

(%4)→ 3 0 0 0 3 2 3 1 1 3 0 0 0 0

Hits:	Last block moved out of cache	status of cache
4	0	16 17 18 7 14

7, 18, 17, 16 are not distorbed after their first arrival

→ Repeat the above using 2-way set associate mapping with LRU strategy

Sol: Each set is applied with LRU

→ (K mod s)th set in cache
=(K%2)

No. of sets = S = $\frac{N}{K}$ = $\frac{4}{2}$ = $\frac{2}{1}$

7 ⓪ ⑧ ⑫ $\boxed{7}$ ⑱ $\boxed{7}$ ⑬ 17 7 ⑧ 0 $\boxed{0}$ 16

Set: 1 0 0 0 1 0 1 1 1 1 0 0 0 0

Hits:	last block moved out of cache	status of cache
3	8	$\underbrace{0\ 16}_{\text{set-0}}$ $\underbrace{17\ 7}_{\text{set-1}}$

Principle of Virtual memory:

The virtual memory allows the construction of programme which exceeds physical memory size. The operating system takes care about swapin and swapout operations. Here the secondary memory and primary memory are divided into equisized partitions (frames/pages). The address mapping converts virtual address to physical address. Paging is the simplest one but it consumes more table space. The segmented paging reduces effective table space but uses two level address translation (segment table, page table)

P) consider a 32-bit virtual address and 4 MB mainmemory are partitioned into 64 KB blocks (frames/pages)

1) what is the table space required for paging technique?

2) what is the effective table space needed for segmented paging, with each segment having 64 pages

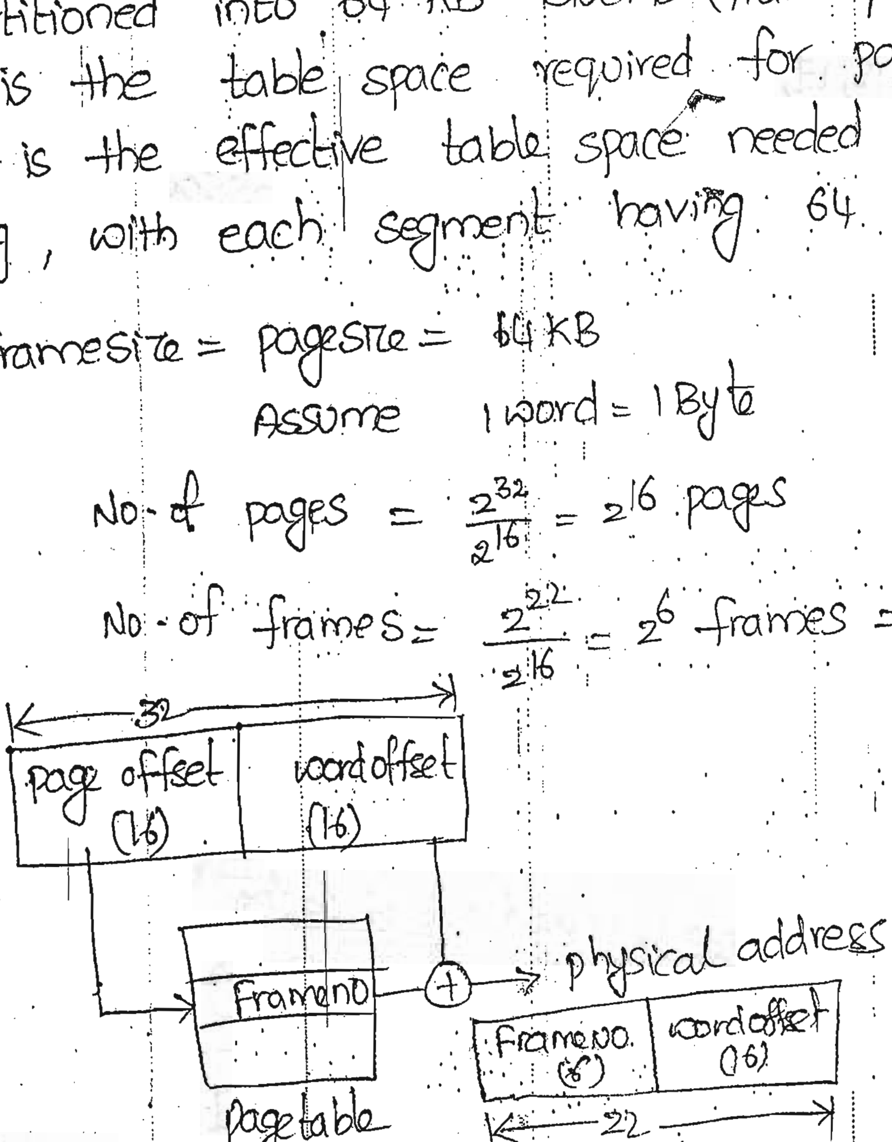
Sol: ⇒ Framesize = pagesize = 64 KB

Assume 1 word = 1 Byte

$$\text{No. of pages} = \frac{2^{32}}{2^{16}} = 2^{16} \text{ pages}$$

$$\text{No. of frames} = \frac{2^{22}}{2^{16}} = 2^{6} \text{ frames} = 64 \text{ frames.}$$

1)



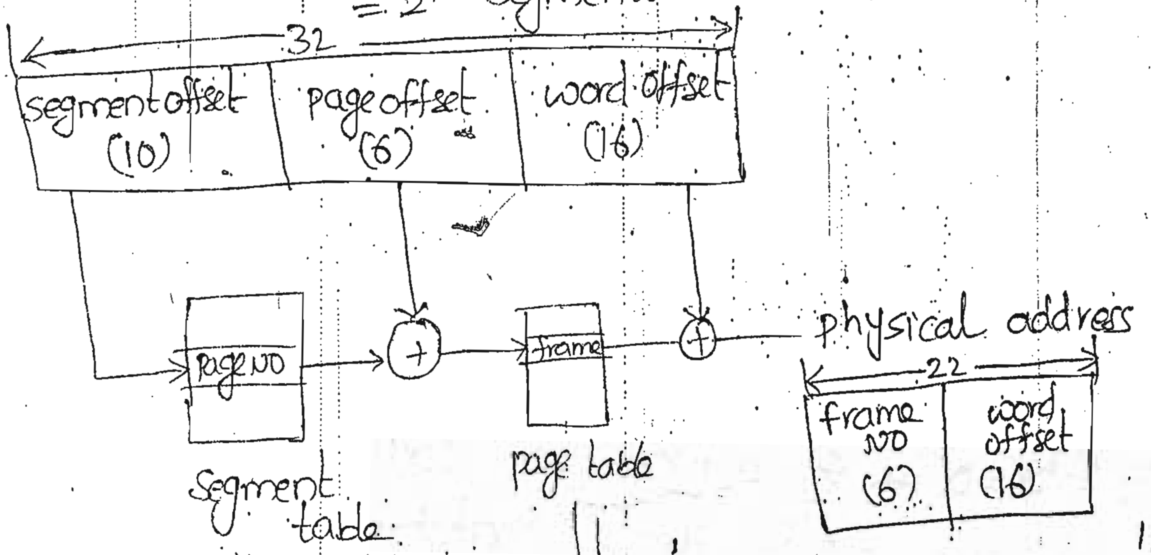
Page table size = $\frac{2^{16} \times 6}{8}$ Bytes

$= 48$ KB   Total 2 references

2) No. of segments = $\frac{\text{Total pages}}{\text{No. of pages per segment}}$

$= \frac{2^{16}}{2^{6}}$

$= 2^{10}$ segments



15

$\frac{2^{10} \times 6}{8} = 768$ B $\frac{64 \times 6}{8} = 48$ B

Effective table space = 768 + 48 = 1216 B
= 1.216 KB

Total 3 references

∴ segmented paging is slower than paging

Principle of associate memory:

The associate memory is the fastest memory and the information is retrieved using either the content (or) partial content. It is also called as content addressable memory (CAD).

The functional blocks are

1. Argument register (n)
2. Mask register (n)/Key register
3. Associate memory array (mxn)
4. Match logic

To perform the read operation, place the content (or) partial content in argument register. In case of partial matching set the appropriate mask register bits to "one". The match logic gives the concerned matched words. In case of multiple matching, access the words sequentially until the desired word is obtained.

- To perform the write operation, after ensuring that the word is not already present, transfer it into one of the free locations.
- The expression for match logic

$$M_i = \prod_{j=1}^{n} \left[K_j + (F_{ij} \odot A_j) \right] \quad \forall i$$

$\forall i$, if $K_j = 0 \rightarrow$ It reduces to complete matching
else partial matching

A_j – jth bit of argument register

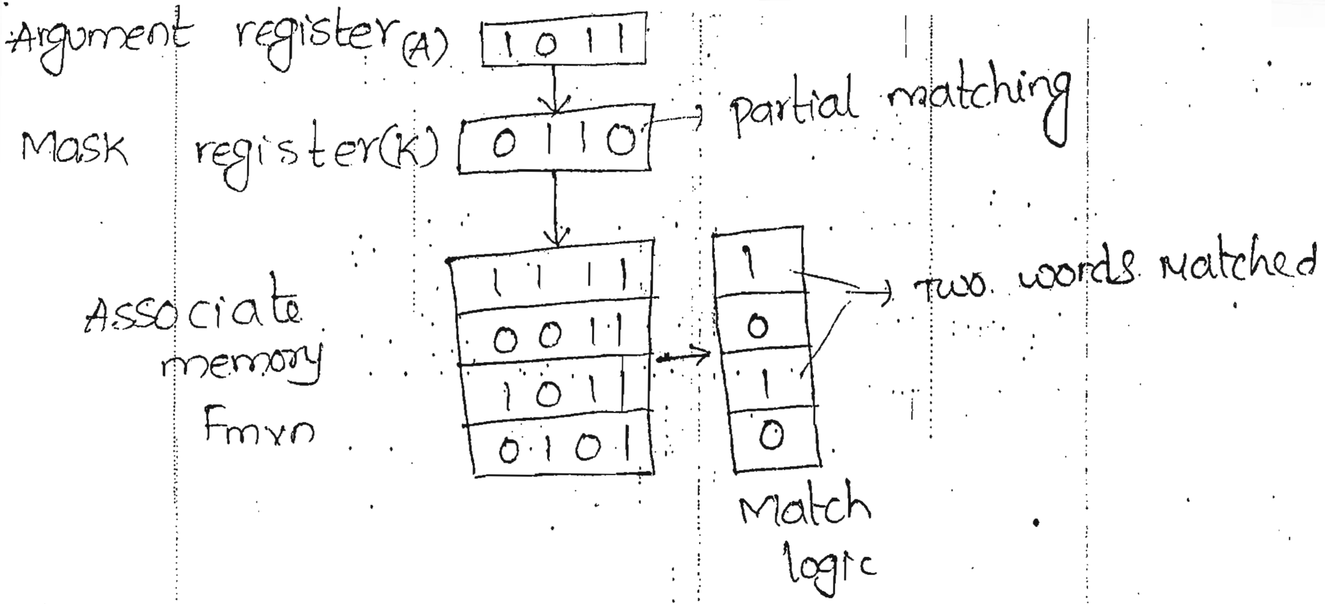
K_j – jth bit of Mask register

F_{ij} – jth bit in ith row of Associate memory

If $F_{i1} = A_1$ and
$F_{i2} = A_2$ and
$F_{i3} = A_3$ and
$F_{i4} = A_4$ then only word 'i' is called as matched one.

If we don't want to match a bit, that bit is masked with Mask register bit (K_j)

as $K_j = 1 \Rightarrow M_i = (1 + \underbrace{(F_{ij} \odot A_j)}_{1})$



- Associate memory does not require any address. Hence by construction, it is fastest memory

- Registers, cache require addresses. some time is spent in address decoding.

- Though the associate memory is the fastest, it needs more H/w to compare the words in parallel. Hence it is the most costliest memory

Q) A direct mapped cache implements the following 2- programs. It uses 512X512 array. Each element occupies 8 Bytes. The row-major order is used for placing the array in MM. Let the cache size is 32 KB and block size 16 B

```
P1: float A[512][512];
        for (i=0; i<512; i++)
            for (j=0; j<512; j++)
                A[i][j] = 5.0;
```

```
P2: float A[512][512];
        for (j=0; j<512; j++)
            for (i=0; i<512; i++)
                A[i][j] = 5.0
```

The miss ratio for P1 is M1 and for P2 is M2. find

1) M1 and 2) $\frac{M1}{M2}$

3) which element of array first time replaces A[0][0].

Sol: Direct Mapped ⇒ $(K \bmod N)^{th}$ block.

Each element of array is 8 Bytes

Block size = 16 Bytes

⇒ ** No. of elements in each block = 2

No of blocks in cache $= \frac{\text{cache size}}{\text{block size}}$

$= \frac{32\text{ KB}}{16\text{ B}}$

$= 2^{11}$ ~~blocks~~ blocks

= 2048 blocks = 2K blocks

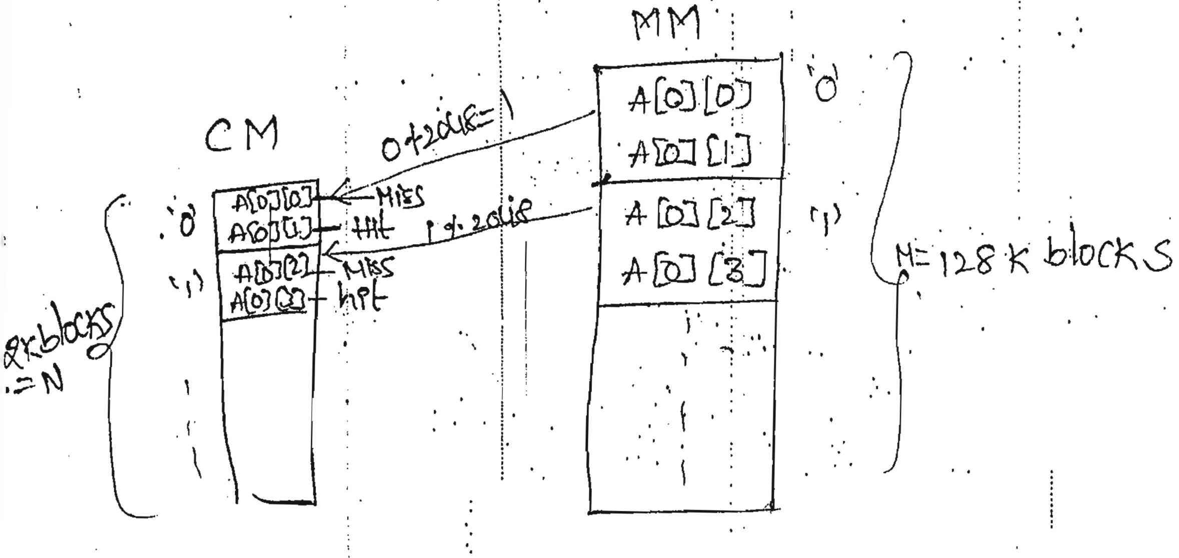
Total elements of array = 512*512

No. of elements in each block = 2

∴ No. of blocks for array in MM $= \frac{2^{18}}{2}$

$= 2^{17}$

= 128 K blocks



for program 1:

Block '0' referred twice

1 is miss & 1 is hit

program 1 refers

A[0][0], A[0][1], A[0][2], A[0][3], --

Miss hit Miss hit

Every block is referred twice and 1 is hit & 1 is miss

Miss ratio $M_1 = \frac{1}{2} = 50\%$.

Total blocks to be moved from MM to cache memory $= 128$ K blocks

Each block causes 1 hit & 1 Miss

$\therefore$ Total miss references $= 128$ K

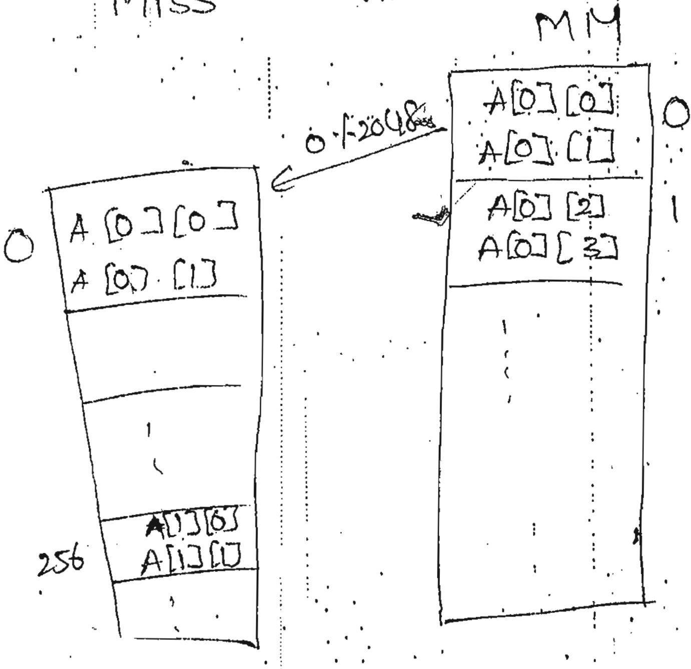
Total references $= 256$ K

for program 2:

program '2'

A[0][0], A[1][0], A[2][0], A[3][0], - - - -

Miss Miss Miss Miss



17

For every element reference a block is to be moved from main memory

Hence it is resulting miss

The total blocks that are to be moved = Total no. of elements

$\therefore$ Total Miss references $= 256$ K

Miss ratio $M_2 = \frac{\text{Total miss references}}{\text{Total references}} = \frac{256K}{256K} = 100\%$

$$\frac{1\times 11}{M_2} = \frac{50\%}{100\%} = \frac{1}{2}$$

3) No. of blocks occupied by each row = $\frac{512}{2}$ = 256 blocks

A[0][0] is in 0th block.

$0 \% 2048$ = 0th cache bloac

Next Main memory block = 2048

Total rows = $\frac{2048}{256}$ = 8 rows

∴ A[8][0] replaces A[0][0], first time

blocks	Row
0 - 255	0
256 - 511	1
512 - 767	2
768 - 1023	3
1024 - 1279	4
1280 - 1535	5
1536 - 1791	6
1792 - 2047	7
2048 -	8 ⟹ A[8][0]

Instructions, Addressing modes, Instruction pipelines:

Instruction has - 1) operation code (opcode)
- 2) operand reference (OPR ref)

Instruction | opcode | OPR ref |

Based on opcode (functional classification)

1. Data transfer instructions
2. Arithmatic "
3. Logical "

Based on the number of operands i.e. number of addresses

4-address instructions

ADD A_1, A_2, A_3, A_4

↓ contain address of next instruction

There is no program counter (PC)

It is equivalent to $M[A_1] \leftarrow M[A_2] + M[A_3]$

Advantage : simplicity

disadvantage : Lengthy

2) 3-address instructions

ADD A_1, A_2, A_3

PC is introduced.

Suppose 100 instructions ⇒ It required 100 memory references for ~~next~~ address of next instruction in 4-address 18

but in 3-address only PC.

adv: fast

disadv: More cost

3) 2-address instructions.

ADD A_1, A_2

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

adv: shorter length

disadv: one operand is overwritten with result.

i.e. one operand is permanantly lost.

4) 1-address instructions

adv: size is reduced

Memory operand is not disturbed

∴ They introduced accumulator (A) register

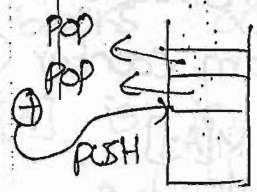
ADD A_2

$A \leftarrow (A) + M[A_2]$

4) 0-address instruction

stack address

ADD



Here CPI<1

ie More than one instruction in one clock cycle
ie one fetch

disadv: More complexity.

stack contents should be rigidly placed properly

P) A system supports 1-address and 2-address instructions. The 16 bit instruction is stored in 128 word memory. If there exists two 2-address instructions what will be the number of 1 address instructions supported by system

ⓐ 256 ⓑ 128 ⓒ 512 ⓓ 384

sol:

128 word ⟹ 7 bits required to address

16

one-address instruction

opcode	OPR ref
9	(7)

two-address instruction

opcode	OPR ref1	OPR ref2
2	7	7

If only one-address instructions then $2^9 = 512$ possible but two address also present

For one two-address instructions 7 bits are to pulled from opcode of 1-address instruction

ie for one 2-address instruction, $2^7 = 128$ 1-address instructions are lost

∴ for two 2-address instruction = 2×128, 1-address instructions are lost

∴ Total 1-address instructions = 256-128 = 128 instructions
(512 - 256 = 256)

i.e $\underbrace{P}_{\text{valid address instructions}} + \underbrace{Q * 2^7}_{\text{Invalid 1-address instructions due to the presence of Q two-address instructions}} = \underbrace{2^9}_{\text{Total possible 1-address instructions}}$

Max possible 1-address instructions are

i.e P is Maximum iff Q is minimum

$\Rightarrow Q = 0$

but 2-address instructions should be present

$\Rightarrow Q = 1$

$\therefore$ Max 1-address instructions = 384

and $1 \leq Q \leq 3$ 19

$Q \neq 4$ because $P = 0$

Addressing modes:

The addressing modes indicate how the reference of operand is made in the instruction. The reference of the operand is called effective address.

In non-computable addressing modes, effective address is to be accessed, while it is computed in computable addressing modes.

The non-computable addressing modes include

1) Immediate addressing mode
2) Direct addressing mode
3) Indirect addressing mode

Immediate addressing mode is the fastest one. There is no effective address for this mode. It is used for accessing program constants. However operand is limited.

In, the direct addressing mode effective address is given in the instruction. It is suitable for accessing static variables. It avoids limited range of operands but result limited address range

- In the indirect addressing mode, these two problems are avoided. It is used for accessing local variables. However it consumes more time compared to immediate and direct addressing modes

In the compotable addressing modes, effective address is computed by adding displacement of the instruction to the fixed processor register.

- If the base register is used in the computation then the addressing mode is called as base addressing mode and is useful for relocatable program
- If the fixed register is index register, the addressing mode is called as index addressing mode and it is suitable for accessing array elements
- The relative addressing mode is obtained using program counter in the computation. It is used for intra segment branching (Forward branching if displacement is +ve, backward branching if the displacement is -ve)

27/11/11

Instruction pipeline:

- Efficient usage of resources
- Time-overlap the instruction phase
- T_{ave} is ↓ ($CPI_{ave} = 1$)
- ∴ for k-stages $T_n = (k+n-1)$ clocks , $t_{clock} = \max(\text{stages delay})$

Release

n-1 clocks

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	

- Performance of the instruction pipeline is given by the speedup factor $(s>1)$

$$S = \frac{\text{Time without Pipeline}}{\text{Time with Pipeline}}$$

If T_1 is time for execution with pipeline & T_2 is time without pipeline

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

In ideal case $S=K$ (no. of stages)

- effective speed up factor $S_{eff} = \frac{S_{ideal}}{(1+\text{stall frequency} * \text{stall cycles})}$

- The performance of instruction pipelines is influenced by uneven stage delays, buffer overhead and dependencies among the instructions

20

- Due to dependencies the instructions in the pipeline are to be reorganised and hence it consumes more time which inturn reduces the ~~average~~ ~~the~~ performance.

ie For K-stage

If n instructions are to be executed

without pipeline, time = $(n*K)$ clocks

with pipeline, time = $(n+K-1)$ clocks

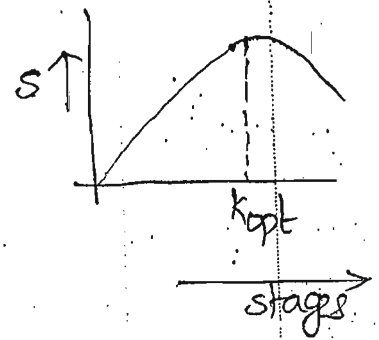
As $n >> K$

$$S_{ideal} = \frac{n*K}{n+K-1}$$

$$\cong \frac{n*K}{n}$$

$$\therefore S_{ideal} \cong K \checkmark$$

- If no. of stages are increased, Pipeline activity ↑. Hence Performance Increases i.e.



K_{opt} = 6 stages

- The data dependency occurs when one instruction in pipeline waiting for the result which is not yet computed by previous instruction. The data dependencies are dealt with instruction rescheduling, stall cycle introduction and operand forwarding. Arithmatic instructions are the main resources for the data dependencies.
- The control dependency occurs when the flow of instruction is changed due to the earlier instruction cycle. The branch instructions are the main resources for the control dependencies. The control dependency is tackled with multiple pipelines, prediction techniques, delayed load.
- The structural dependency occurs due to the nonavailability of resources. Resource duplication resolves this problem.

P) consider a 5-stage instruction pipeline which implement the two programs. program A contains 50 instructions. Program B contains 500 instructions. compute the speed up factor and pipeline efficiency for each case.

Sol: Given, No. of stages = K = 5

PA	PB
n = 50	n = 500

An K-stage pipeline $T_n = (K+n-1)$ clocks

∴ for program PA, $T_n = (5+50-1) = 54$ clocks

PB, $T_n = (5+500-1) = 504$ clocks

In non pipeline,

instructions in PA takes = 50×5 clocks

instructions in PB takes = 500×5 clocks

For program PA, speed up factor = $\frac{T_{NP_1}}{T_{P_1}} = \frac{50 \times 5}{54} = 4.6$

program PB, speed up factor = $\frac{T_{NP_2}}{T_{P_2}} = \frac{500 \times 5}{504} = 4.96$

$\therefore$ program B has better performance

Ignoring other factors, as the no. of instructions increases Performance also increases

we have, in ideal case, $S_{ideal} = K$

But PA $\Rightarrow$ $S = 4.6$

$\therefore$ efficiency = $\frac{5}{4.6} \times 100 = 92\%$

i.e. $S=5 \rightarrow$ efficiency = 100%

$S = 4.6 \rightarrow$?

$= 92\%$

PB $\rightarrow$ $S = 4.96$

$\therefore$ efficiency = $\frac{5}{4.96} \times 100 = 99.2\%$

i.e. $S=5 \rightarrow$ efficiency = 100%

$S = 4.96 \rightarrow$?

$= 99.2$

D) let the two pipelines are implemented by a system so that Pipeline 1 has 8 stages (phases) with uniform delay of 2ns. The pipeline 2 has 5 stages with the stage delays 3ns, 1ns, 1ns, 2ns, 3ns.

1) How much time is saved with pipeline 1 for 100 instructions compared to pipeline 2

Sol: with pipeline 1, time = $(K+n-1)$ clocks

$= (K+n-1)\, t_{clock}$

we have $t_{clock} = 2ns$

$\therefore T_1 = (8+100-1) \times 2 = 214\,ns$

with pipeline 2, t_{clock} = max (stage delay)

$= 3ns$

$T_2 = (5+100-1) * 3$

$= 312$ ns

∴ Time saved $= 312 - 214$

$= 98$ ns

For program P_1, speed up factor $= \frac{\text{Time without pipeline}}{\text{Time with pipeline}}$

$= \frac{8 * 100 * 2}{214}$

$= 7.47$

For program P_2, speed up factor $= \frac{\text{Time without pipeline}}{\text{Time with pipeline}}$

$= \frac{100 * (3+1+1+2+3)}{312}$

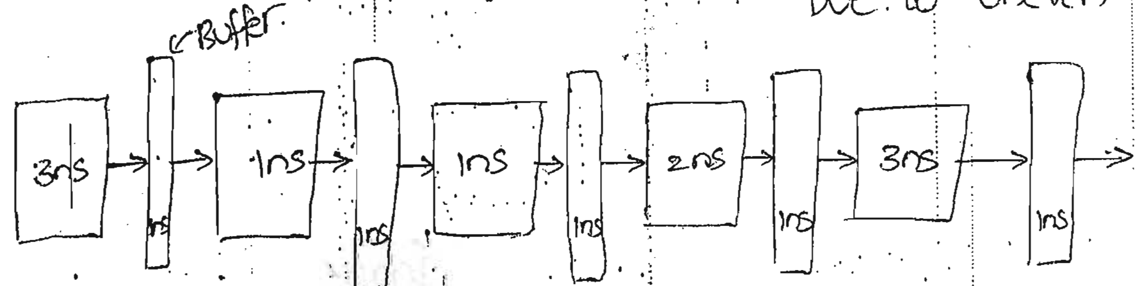
$= 3.2$

∴ η_1 (efficiency) $= \frac{8}{7.47} * 100 = 93.37\%$

η_2 (efficiency) $= \frac{5}{3.2} * 100 = 64\%$ ↓

Due to uneven delays

p)



find speed up factor.

Sol: Here No. of instructions are not given

∴ average time is to be considered

without pipeline, $T_{ave} = \sum$ stage delay

$= (3+1+1+2+3)$

$= 10$ ns

with pipeline $T_{ave} = 1$ clock

this is because CPI... = 1 clock

$T_{clock} = (3+\underset{\uparrow}{1})\ ns$

buffer time as it is operated by same clock

ie Buffers are not included in nonpipeline systems

$\therefore \quad S = \frac{10\ ns}{4\ ns} \Rightarrow 2.5$

) consider a 4-stage instruction pipeline which was aimed to implement the following 4 instructions. Here each instruction can spend different number of clocks at different stages. The following table gives the no. of clocks required per instruction per stage. what will be the no. of clocks required to complete the following 4 instructions.

22

	S_1	S_2	S_3	S_4
I_1	3	1	2	1
I_2	1	3	1	2
I_3	1	1	1	1
I_4	2	1	2	1

Sol:

1) The instruction has to complete its previous stage when it requests the next stage.

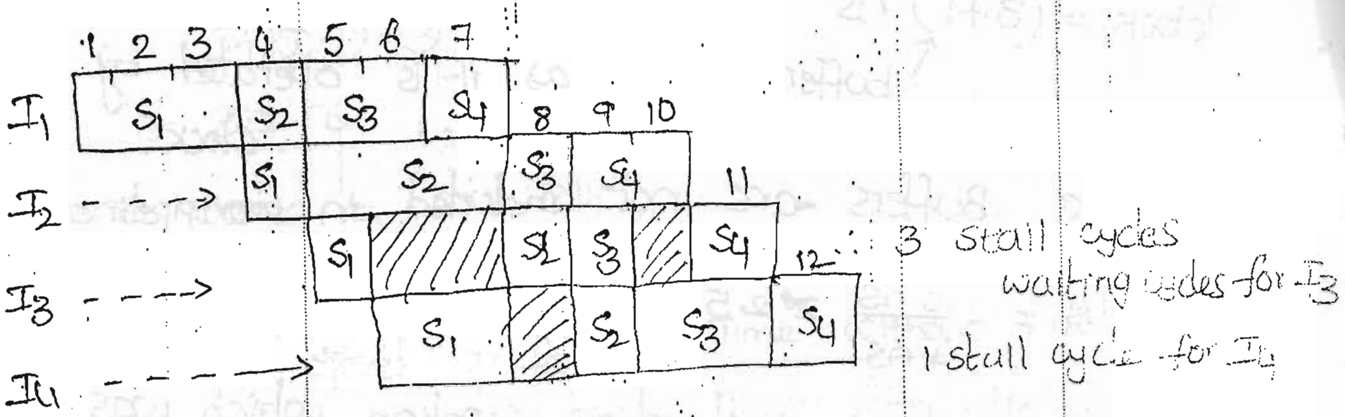
2) The stage is given to instruction if the stage is free ie the previous instruction has released that stage

ie stage q for instruction I_q is given if

1) I_q must complete S_{q-1} time $= x$

2) I_{q-1} must have released S_q time $= y$

$\therefore$ Time of allocation $= \max(x, y)$



∴ These 4 instructions are completed in 12 clock cycles.

P) for (i=1 to 2)
{ I_1; I_2; I_3; I_4; }

find total no. of clocks required for above problem.

Sol: suppose for above problem.

	S_1	S_2	S_3	S_4	S_1	S_2	S_2	S_4
I_1	3 3	1 4	2 6	2 7	10	[illegible]	13	14
I_2	1 4	3 7	1 3	2 10	11	[illegible]	15	17
I_3	1 5	1 8	1 9	1 11	12	15	16	18
I_4	2 7	1 9	2 11	1 12	14	16	18	19

It takes 19 clocks ~~cycles~~

At the end of first iteration

$$\text{speed up factor} = \frac{24 \text{ clocks}}{12 \text{ clocks}} = 2$$

At the end of two iterations

$$\text{speed up factor} = \frac{48}{19} = 2.53$$

The pipeline system contains 4 stages with stage delays 3 clocks ~~DBS~~, 2 clocks, 2 clocks, 3 clocks. The pipeline is operated with 1 GHz clock. The nonpipeline is implemented with 2GHz clock. what is the speed up factor.

Sol:

$$\text{speed up factor} = \frac{\text{Tave without pipeline}}{\text{Tave with pipeline}}$$

$$= \frac{(3+2+2+3) * \frac{1}{2\,\text{GHz}}}{3 * \frac{1}{1\,\text{GHz}}}$$

$$= \frac{5}{3}$$

$$\simeq 1.7$$ ✓

Data dependency:

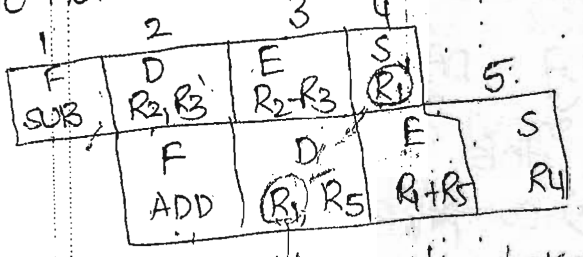
The data dependencies occur due to arithmatic instructions, where the result of one instruction are to be used in another instruction

23

Eg: SUB R1, R2, R3 ; $R_1 \leftarrow R_2 - R_3$

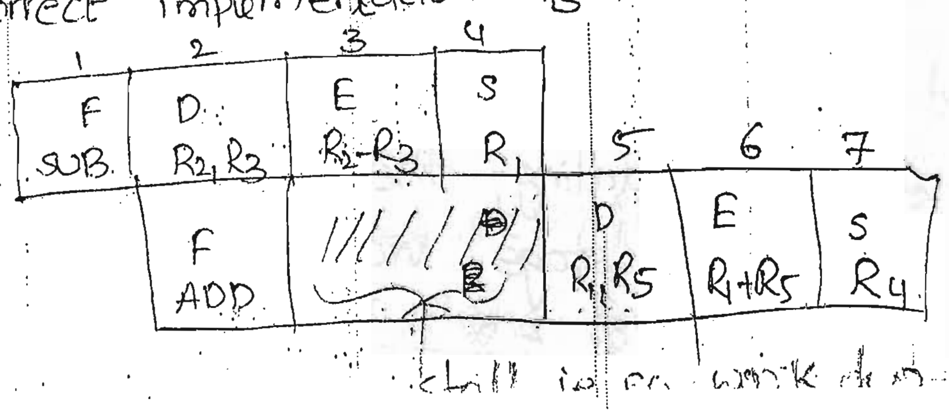
ADD R4, R1, R5 ; $R_4 \leftarrow R_1 + R_5$

suppose each instruction takes 1 clock per stage pipeline do not consider the data dependency



Here it takes old value ✓

The correct implementation is



stall is no work done

→) introduction of stall cycles solves all the problems But low performance

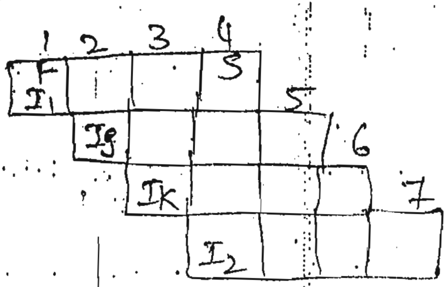
2) Instruction rescheduling:

Between the dependent instructions, independent instructio are scheduled

I_1

I_g } Independent instructions.
I_k

I_2



It also takes 7 clocks.

But how to know independent instructions?

I_g, I_k are to be independent among them and also independent of I_1, I_2 and their successo

Independent instructions, are known by compiler by constructing DAG

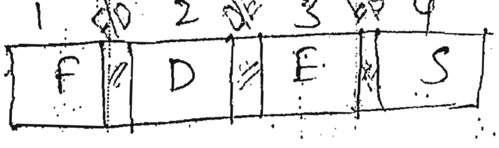
∴ efficiency of this method depends on

1) compiler
2) program

Instruction rescheduling does not solve all the problem

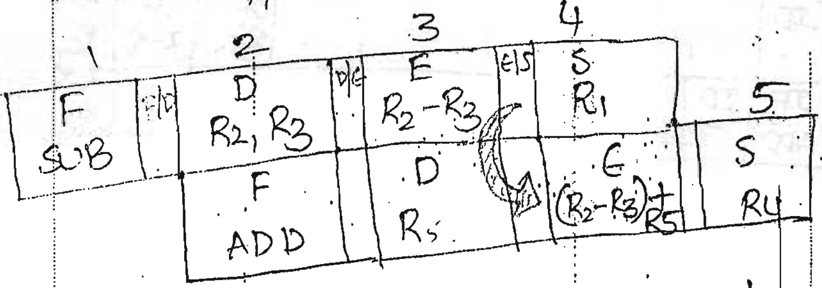
3) operand forwarding:

In operand forwarding, the value of the operand is given to required stage via interstage buffer registers.

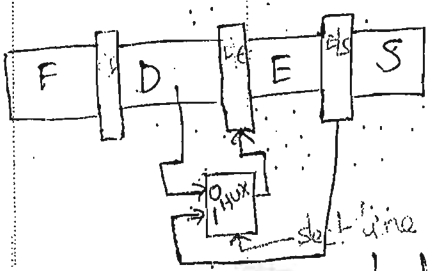


These buffer registers are shift registers operating in PIPO.

Eg: SUB R1, R2, R3
ADD R4, R1, R5



It requires 5 clocks.
extra H/w is needed



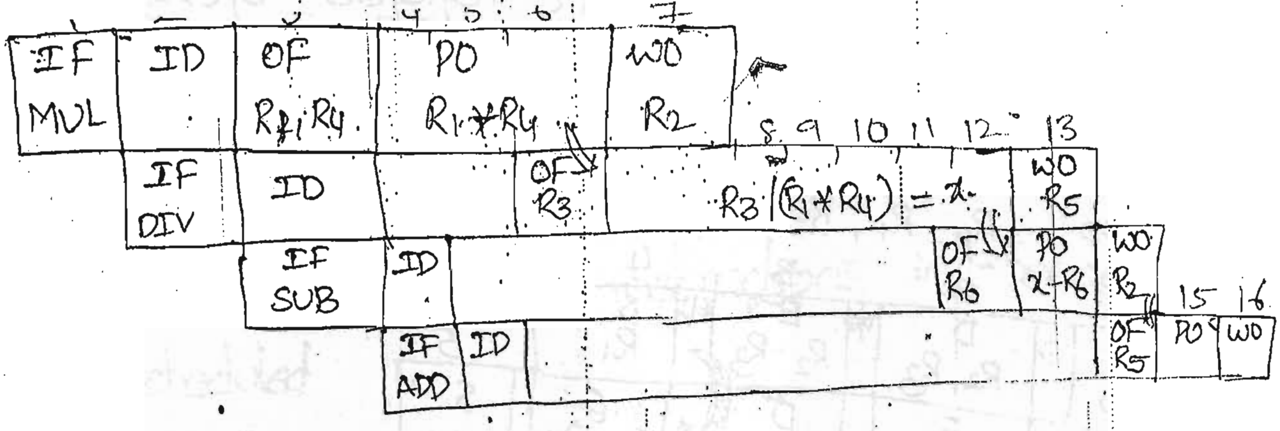
24

A path is needed between E|s & E. It should not be permanant path.
∴ Multiplexer is needed.
select line is activated by compiler as it knows dependencies
It solves all problems but more cost

b) IF- Instruction fetch, ID-Instruction Decode, OF-operand fetch, PO-Perform operation, WO-write operation are stages for the instruction to be executed.

MUL R2, R1, R4
DIV R5, R3, R2
SUB R2, R5, R6
ADD R4, R2, R5

Except PO all other stages take 1 clock/instruction-stage. PO takes 3 clock for MUL and 6 clocks for DIV, 1 clock for each ADD & SUB; Find total clocks for above execution



∴ 16 cycles needed

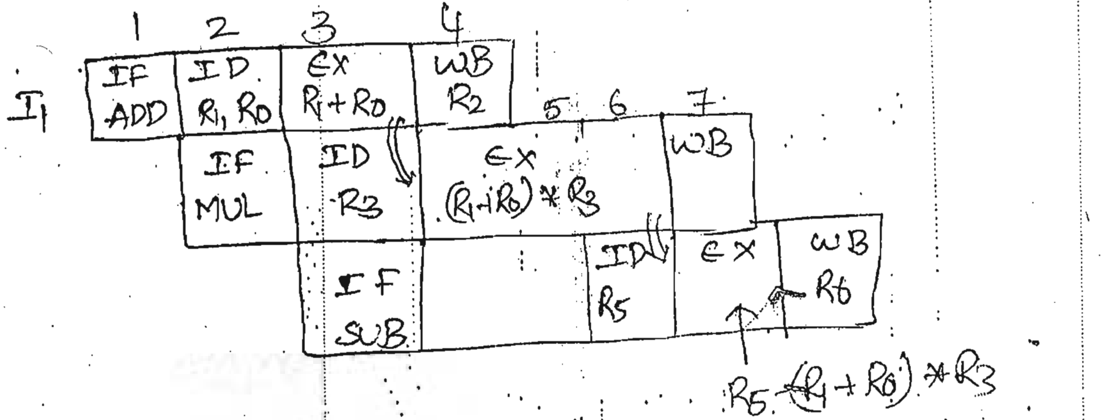
p) consider a 4-stage instruction pipeline (Fetch, Decode, Execute, write Back). The fetch, decode, write back will take 1 clock each for completion. The number of clocks required for execution is 1 clock for addition and subtraction and 3 clocks for multiplication. what will be the min. no. of clocks required to implement the following three instructions. operand forwarding is allowed.

ADD R_2, R_1, R_0 ; $R_2 \leftarrow R_1 + R_0$

MUL R_4, R_3, R_2 ; $R_4 \leftarrow R_3 * R_2$

SUB R_6, R_5, R_4 ; $R_6 \leftarrow R_5 - R_4$

Sol :



∴ 8 cycles i.e. 8 clocks are required

Types of data dependencies.

The data dependencies are classied into

1. Read After write (RAW) (OR) TRUE dependency
2. write After Read (WAR) (OR) ANTI dependency
3. write After write (WAW) (OR) OUTPUT " "
4. Read After Read (RAR) (OR) ~~Read~~ No dependency.

- The domain and range relations are used for detecting data dependencies. Domain indicates the source operand and range denotes destination operand
- Any violation to the order will cause hazard. Hence data dependencies also called as hazards

Eg:

	Domain	Range
I_1: ADD R_1, R_2, R_3	R_2, R_3	R_1
I_2: OR R_4, R_1, R_5.	R_1, R_5.	R_4

25

Read operation is done on Domain
write operation is done on Range

Suppose two pipelines then
OR completes before ADD as it is fast

∴ Error

∴ OR should Read R_1 only after ADD writes R_1

∴ It is Read After write

It is due to $\underset{\text{Domain}}{D_2} \cap \underset{\text{Range}}{R_1} \neq \emptyset$

II instruction ↑ I instruction

∴ RAW $\Longleftrightarrow$ $D_2 \cap R_1 \neq \emptyset$ ✓
WAR $\Longleftrightarrow$ $R_2 \cap D_1 \neq \emptyset$
WAW $\Longleftrightarrow$ $R_2 \cap R_2 \neq \emptyset$
RAR $\Longleftrightarrow$ $D_2 \cap D_1 \neq \emptyset$

Eg: I_1: ADD R_1, R_2, R_3
I_2: OR R_2, R_4, R_5

I_2 should write R_2 only After I_1 has read R_2

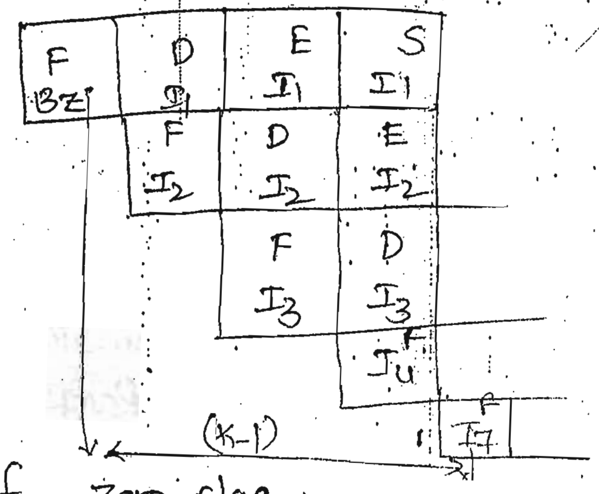
control dependencies:

- The control dependencies occur due to change in the flow of execution as a result of instruction cycle of earlier one
- The control dependency require reorganisation of pipeline. Hence performance will be reduced.

Eg: I_1: BZ I_7 BZ - Branch on Zero

Next sequential Instruction I_2:

Target I_7:



If zero flag=1

then the next instruction is I_7

Hence whatever work is done by pipeline is wast

Hence its contents are to be flushed out.

It requires (K-1) stalls.

If zero flag=0

then next instruction is I_2

∴ pipeline is useful

- This zero flag value can be known only after the execution of I_1

∴ This is control dependency

) consider a 5-stage instruction pipeline in which all the instructions can be overlapped except branch instructions. In case of branch instructions, the target is not available until the current branch instruction is completed. Let there are 20% branch instructions. Assume that each stage consumes 2ns delay

1) what is the average instruction time

sol: The stall cycles (waiting cycles) for target instruction

$$= K-1 = 5-1 = 4$$

Every instruction on an average takes 1 clock for completion

Total clocks required for an instruction = waiting time + completion time

$\therefore$ Branch instructions time $= 4+1 = 5$ clocks

other instructions time $= 0+1 = 1$ clock 2.6

$\therefore$ 20% branch instructions and 80% other instructions

$\therefore$ Average instruction time $= 80\% (0+1) + 20\% (4+1)$ clocks

$$= 0.8 + 1$$

$$= 1.8 \text{ clocks}$$

$$= 1.8 \times 2\text{ns}$$

$$= 3.6\text{ns}$$

(OR)

$$T_{ave.} = \left(1 + \underset{\text{freq}}{\text{stall}} \times \underset{\text{cycles}}{\text{stall}}\right) \times T_{clock}$$

$$= (1 + 20\% \times 4) \times 2\text{ns}$$

$$= 3.6\text{ns}$$

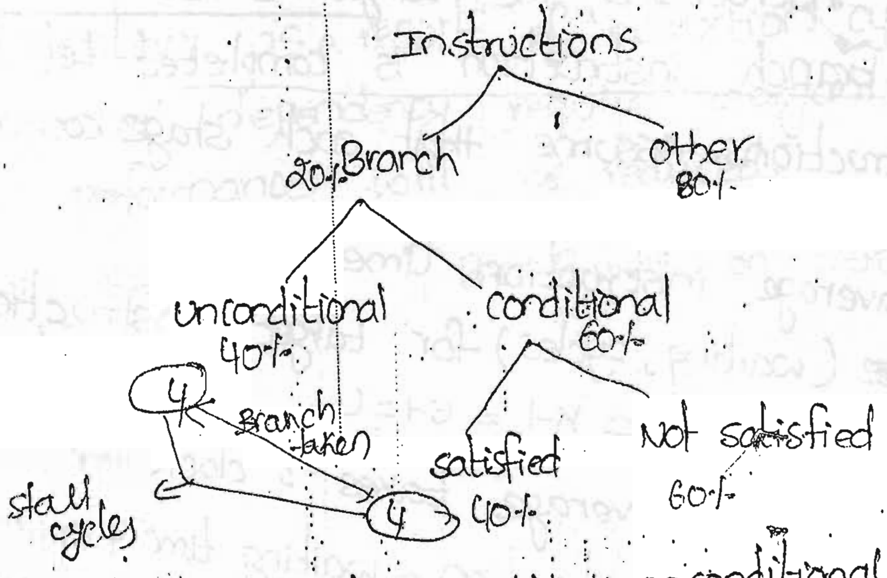
2) effective speed factor
$$S_{eff} = \frac{S_{ideal}}{1 + \underset{\text{freq}}{\text{stall}} \times \underset{\text{cycles}}{\text{stall}}}$$

$$= \frac{5}{1.8}$$

$$= 2.78$$

P) In the above problem, among the branch instructions, 40% are unconditional. 60% of conditional branch instructions does to satisfy the condition. If there is no penalty due to such branch instructions what is the average instruction time.

Sol:



Branch is taken in unconditional branch and/or if the condition is satisfied

stall cycles in case of branch = 4 clocks

$$T_{ave} = \left[1 + 20\%\,[40\% \times 4 + 60\%(40\% \times 4)]\right] * 2T_{clock}$$

$$= (1 + 0.2\,[1.6 + 0.6 * 0.4 * 4]) * 2\,ns$$

$$= [1 + 0.2\,[1.6 + .96]] * 2$$

$$= 3.024\ ns$$

2.56×2
5.1 1 ×1
1.512×2
3.024

Techniques to resolve the control dependencies:

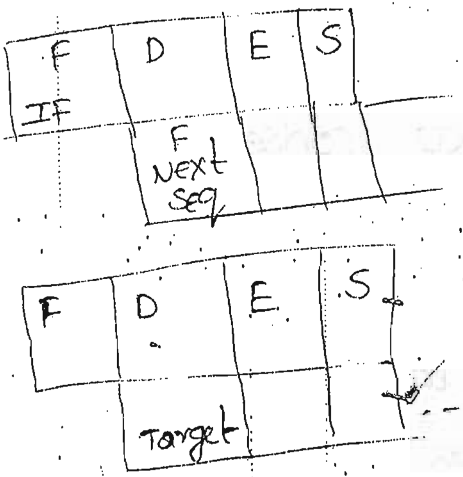
1. Multiple pipelines:

In a two pipeline system, the next sequential instruction is placed for the coming clock after the branch instruction. In second pipeline the target is placed for the next clock in the fetch stage. one of the pipelines always maintain correct flow of instructions. At the end of branch instruction one pipeline is allowed to continue while the other is stopped. This approach successfully deals IF-THEN-ELSE statement.

IF (COND) THEN(Ix) ELSE (Iy)

the two pipelinesystem fails for the nested If instructions

eg:



- If target itself is branch then three pipelines

multiple pipelines

- Implementation ~~is~~ cost is more 27

2) Delayed Load technique:

The load of the branch instruction is the next sequential (or) target instruction. Between the branch and its load independent instructions are scheduled. Here whether the branch is taken (or) not taken the load is getting delayed

3) prediction techniques:

The prediction techniques will, decide ~~whether~~ either the next sequential (or) target ~~te~~ instruction is to be placed in fetch stage for the next clock. The predictions can be static (or) dynamic. In dynamic prediction, the next prediction will depend upon the history of previous branch instructions.

The prediction may be
1) Branch never takes
2) Branch Always takes

In branch never takes, the pipeline is placed with next sequential instruction

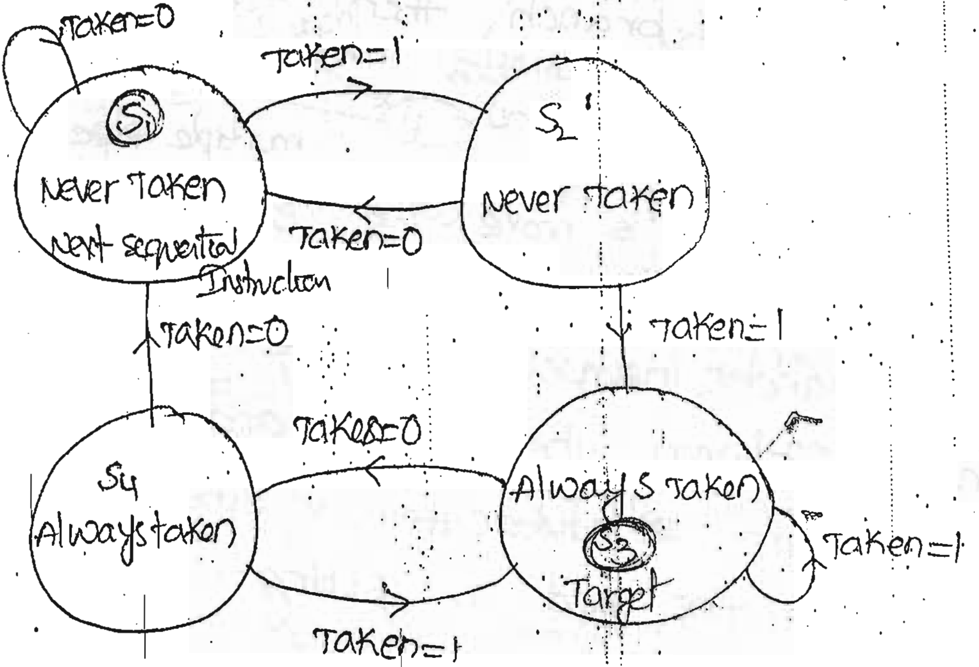
In branch always takes case, the target instruction is placed in the fetch stage

- The conventional pipeline follows branch never takes assumptic

Implementation of dynamic' prediction:

Here the prediction is changed.., if two consecutive previous predictions are wrong. The binary flag 'TAKEN' maintains the status of previous branch instructions.

Taken ─┬ 0 --- Branch never taken
       └ 1 — Branch Always taken.



→ S_1, S_3 are strong prediction states

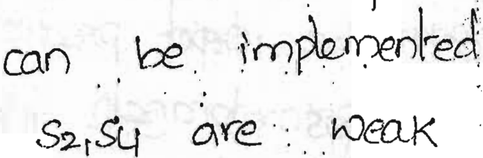
ie prediction is not changed whatever may be the outcome of its previous prediction.

It can be implemented using two T-flipflops

→ S_2, S_4 are weak prediction states

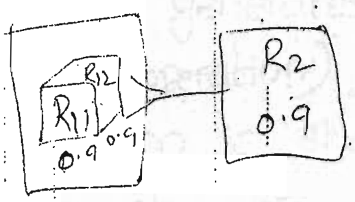
Structural dependencies:

- It is resulted due to non-availability of resources
- Tackled with resource duplication. Resource duplication is subject to the constraint of cost and reliability



$R = R_1 R_2 = 0.81$

suppose stage 1 has 2 copies of resource



$$R = R_1 \cdot R_2$$

$$R_1 = 1 - (1-R_{11})(1-R_{12})$$

$$= 0.99$$

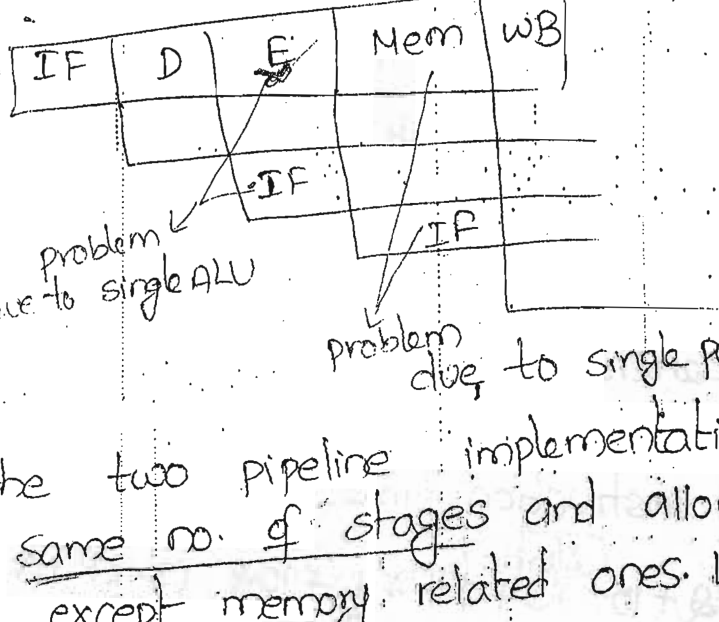
$$R = (0.99) * 0.9$$

$$R = 0.901$$

Q: If the system has a single port memory, instruction fetch and memory of another instruction cant be overlapped
ie storage

28

- In case of single ALU system we cannot overlap instruction fetch and Execution of another instruction if fetch requires updation of program counter (PC).



Q) consider the two pipeline implementations. Both of them are having same no. of stages and allow overlapping of all instructions except memory related ones. Let there are 20% instructions belong to memory. The pipeline 1 employes single port memory while pipeline 2 uses 2 port memory. If two memory operations cant be performed simultaneously there will be one stall cycle penality. Get the speed upfactor S_1 S_2 for two pipeline systems and find $\frac{S_1}{S_2}$

Sol.
$$S_{eff} = \frac{S_{ideal}}{(1+\text{stall freq} * \text{stall cycles})}$$

Let No. of stages $= K$

	Pipeline 1	Pipeline2
stages	K	K
Memory	1-port	2-port
stallcycles	1	0
stall freq.	20%	20%

$\therefore \quad S_1 = \frac{K}{1+0.2*1} \qquad S_2 = \frac{K}{1+0*0.2}$

$S_1 = \frac{K}{1.2} \qquad S_2 = K$

$\therefore \frac{S_1}{S_2} = \frac{1}{1.2} \Rightarrow \boxed{S_2 = 1.2 * S_1}$

i.e. performance of pipeline 2 is 1.2 times of pipeline 1

$\Longrightarrow$

P) A system employes a 5-stage instruction pipeline, in which 5% instructions result data dependencies, 10% instructions result control dependencies, 2% instructions result structural dependencies. 10% instructions are exposed for both data and control dependencies. If the penalities for control dependency and data dependency are 3 clocks and 2 clocks what is average instruction time

Sol: $T_{ave} = (1 + 5\% * 2 + 10\% * 3 + 2\% * \underset{\uparrow}{1} + 10 * (3+2)) *$

default for structural dependenc

$= 1.92$ clocks

control Unit

control unit generates control signals which implement micro-operations. (μoperation)

micro-operation is the elementary operation which results datamovement across registers

Register Transfer Language (RTL) is used to represent μ-operation

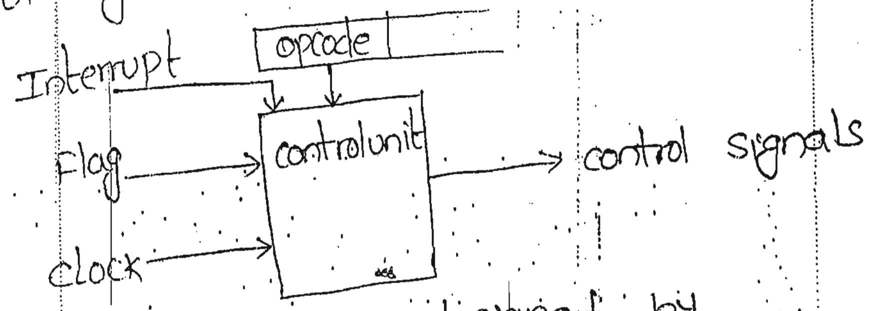
set of μoperations form μprogram

- control program implement μprogram.

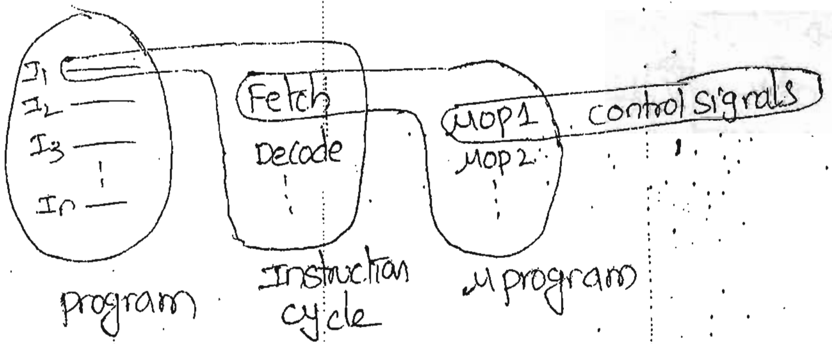
Instruction cycle contains

Fetch μprogram
Execute "
Interrupt "

control unit can be considered as u-state sequential machine (fetch, decode, execute, store) each state require different control signals.



control unit can be designed by
1) Hardware control unit (HCU)
2) microprogrammed control unit (μpcu)



each μoperation requires simultaneous control signals

- if control signals are generated only using H/ω then it is HCU

- If control signals are generated using memory and H/w then it μPCU

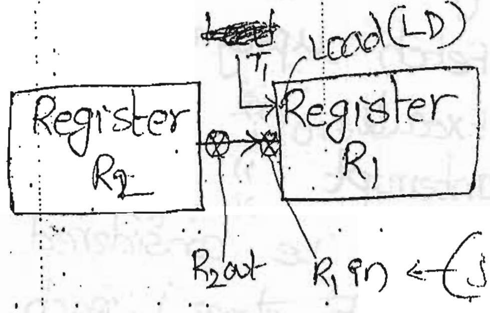
- RTL : syntax is condition : opcode

μoperation:

Eg: move data from Register R_2 to Register R_1 at clock 1
symbolically it is written as

$T_1 : R_1 \leftarrow R_2$ It is RTL

H/w is



← (just like taps for water)

∴ control signals required R_2out, R_1in

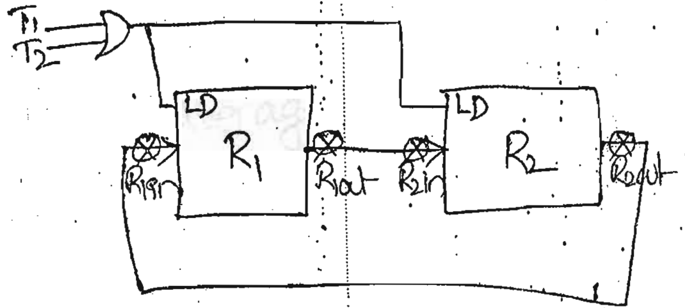
condition is at clock 1 (T1)

Eg: swap operation

H/w implementation require less resources than s/w implementation.

2 registers

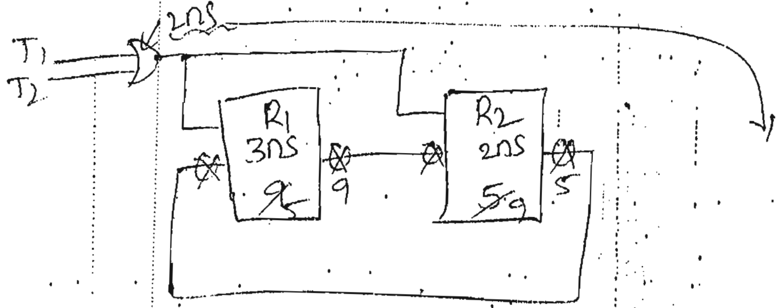
3 variables
i.e 3 registers



i.e swap R_1 and R_2 either at clock1 (T1) ~~and~~ or clock2 (T2)
R_1in, R_1out, R_2in and R_2out are to be opened simultaneously otherwise copy operation is done

$T_1+T_2 : R_1 \leftarrow R_2 , R_2 \leftarrow R_1$

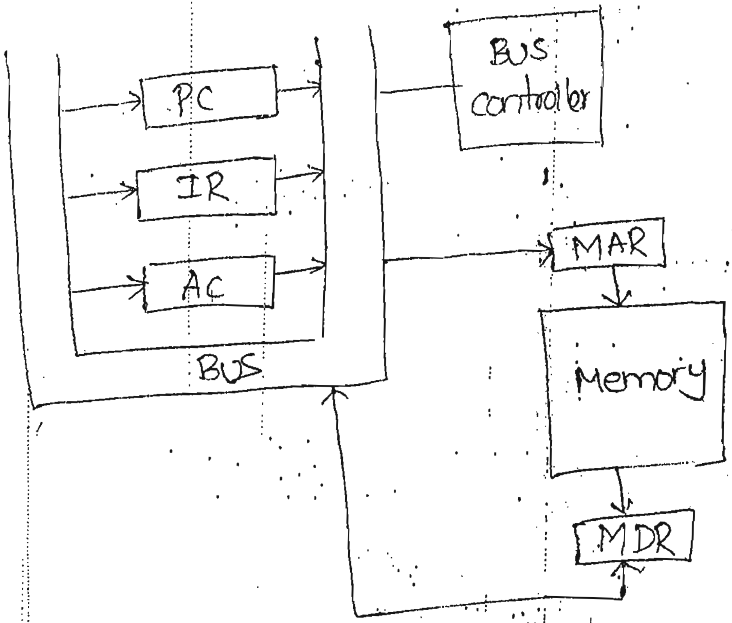
suppose in above diagram delays are given then, how much time is taken for swapping to be successful



(2ns + 3ns)

= 5ns

⇒ consider a single bus architecture in which the minimal register set of the system is connected Program counter(PC), Instruction Register (IR), Accumulator (AC) and memory.



30

MAR- Memory Address Register

MDR- Memory Data Register

Fetch:

Here, move the instruction from memory into IR.

- visit memory
- Get the instruction
- Transfer it into IR
- update PC

sequential actions ie. one after other

visit memory ⇒ place contents of PC into MAR - In first clock

Get instruction ⇒ Read it to MDR — In second clock

Transfer to IR ⇒ Move from MDR to IR — In third clock.

PC ⇐ PC + 1

∴ T1 : MAR ← PC
T2 : MDR ← M[MAR]
T3 : IR ← MDR
PC ← PC+1

fetch μprogram

In 8085, it requires 4 clocks (or) 4 T-states
(3 fetch + 1 stable)
beforee xecution

control signals ⇒ PCout , MARin ← T1 clock
(Address decoder)
&
MARout , MDRin ← T2 clock
(Address decoder)

~~PC~~ MDRout, PCin, IRin ← T3 clock
PCout

μprogram for execution phase of ~~an~~ instruction:
It varies from one instruction to other

Eg: ADD X
→ Add Accumulator contents to the contents of memory location and place the result in accumulator
This is done after fetch.
At the end of fetch IR contains

IR

opcode	OPR ref
ADD	X

∴ T1 : MAR ← IR (OPR ref/Address)
T2 : MDR ← M[MAR]
T3 : A ← A+MDR

∴ 3 clocks
execution phase μprogram

Eg: INC X   Assume incrementation possible at register level

T1 : MAR ← IR (Address)
T2 : MDR ← M[MAR]
T3 : MDR ← MDR+1
T4 : M[MAR] ← MDR

∴ 4 clocks

Eg: BUN X

The next instruction is taken from X location
⇓
placed in PC

T_1: PC ← IR(Address)

∴ 1 clock cycle

Hence different instructions require different no.of μoperations.

If we know which control signals are required at what
time, we can design control unit
↳ 1) HCU
2) μPCU

31

Hardware control unit design: each control signal is expressed sop
- In HCU design, each control signal is expressed sop expression of instructions, instructions timings and instruction phases.
- Each control signal is then derived from dedicated H/w
It provides all control signals parallelly at fixed time intervals (3 gate delays (NOT, AND, OR))
- It is used with RISC processors
- The only problem is any modification requires redesign and reconnection of H/w components and hence it is not flexible and it is not suitable for design and testing environment

Eg:
consider hypothetical control unit which supports only two instructions. The system is employing 4 control signals S_0, S_1, S_2, S_4. Each instruction requires 4 μoperations. The following table gives instructions and associated μ-operations. obtain expressions for S_1 and S_3

~~control~~ signals

	I_1	I_2
T_1		

control signals

μop	I_1	I_2
T_1	S_0, S_1	S_1, S_3
T_2	S_2, S_3	S_1, S_0
T_3	S_0, S_2	S_0, S_3
T_4	S_1, S_2	S_1, S_3

Sol: Each control signal $S_i = f(I_1, I_2, T_1, T_2, T_3, T_4)$

$$\therefore S_1 = (I_1T_1 + I_2T_1) + I_2T_2 + (I_1T_4 + I_2T_4)$$

Here it supports only 2 instructions and S_1 is required for both in T_1

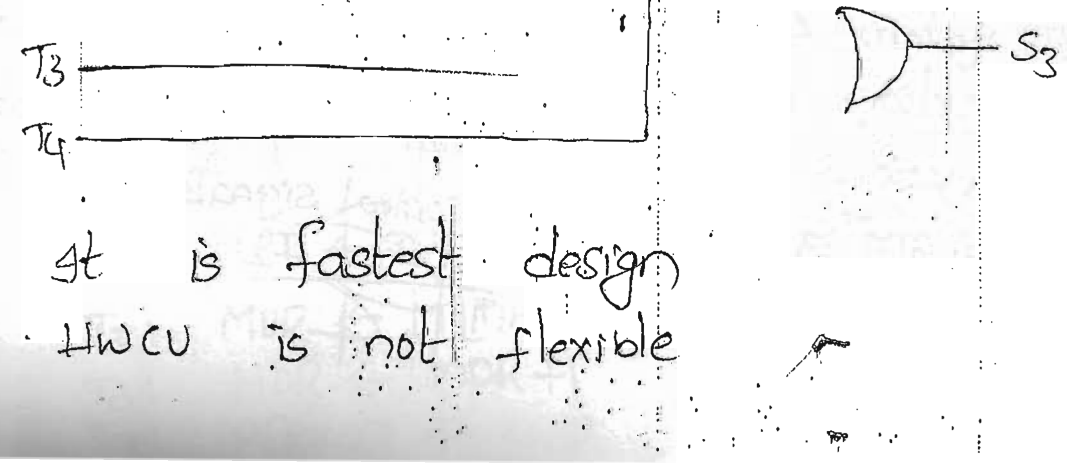
$$\Rightarrow S_1 = T_1 + I_2T_2 + T_4$$

$$S_3 = I_2T_1 + I_1T_2 + I_2T_3 + I_1T_4 + I_2T_4$$

$$= I_2T_1 + I_1T_2 + I_2T_3 + T_4$$

Similarly for S_0 and S_2

1 bit opcode is needed for 2 instructions.

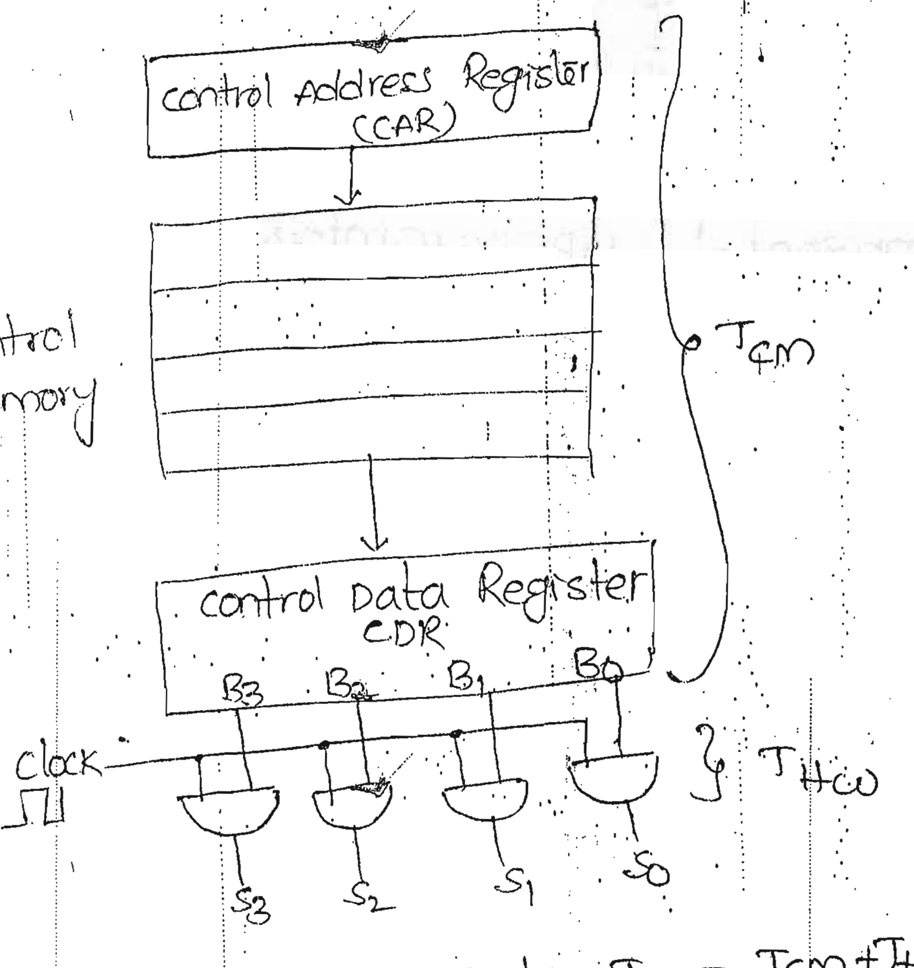


- It is fastest design
- HWCU is not flexible

3|12|11

μP control unit design

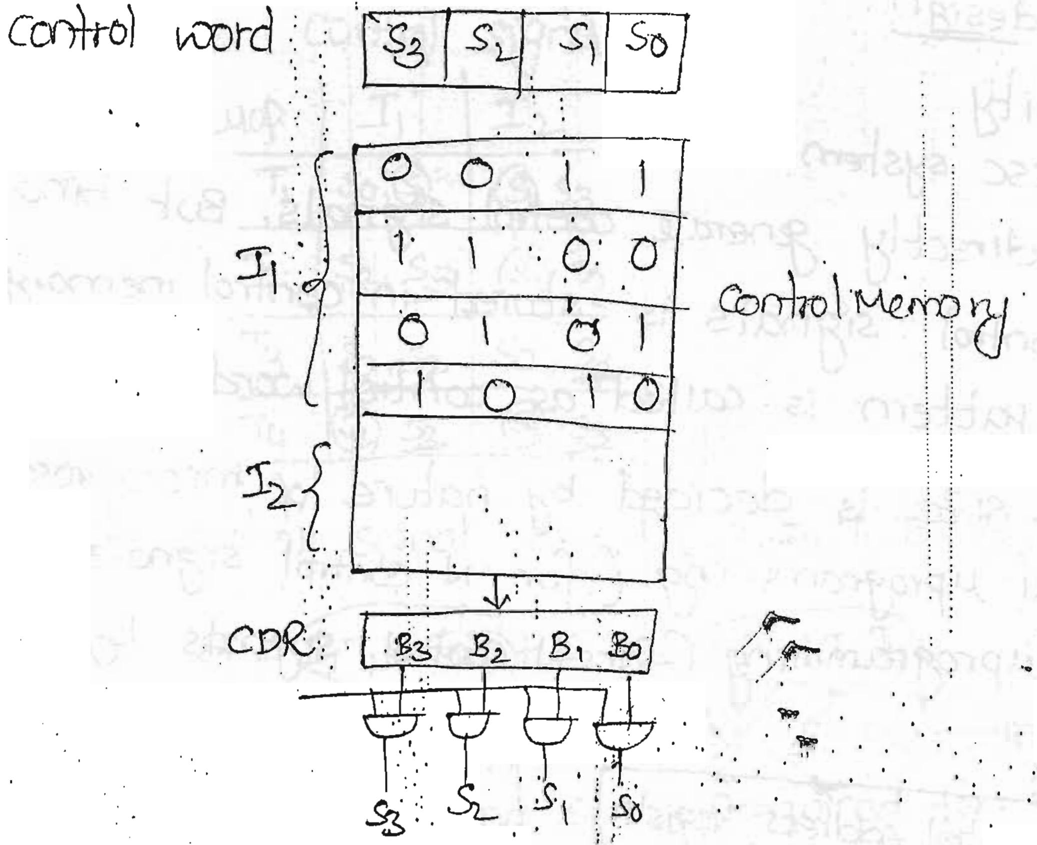
- offer flexibility
- used with CISC system
- It does not directly generate control signals. But binary pattern of control signals is stored in control memory.
- This binary pattern is called as control word
- control word size is decided by nature of microprogramming
  1) Horizontal μprogramming (for N control signals N bits)
  2) vertical μprogramming (for N control signals $\log_2 N$ bits)



- Time for control signal $T_{CS} = T_{CM} + T_{HW}$

Eg:-

	I_1
T_1	S_0, S_1
T_2	S_2, S_3
T_3	S_0, S_2
T_4	S_1, S_3



S_q = clock iff $B_q = 1$

- This is horizontal μprogramming. It offers higher degree of parallelism and is used with parallel processing

  No. of control signals generated by a single control word = N signal

  disadvantage:

  control word is too lengthy and most bits are zeros.

- vertical μprogramming
- Dont store binary pattern. Instead encoded binary pattern of control signals is stored in control memory.

  N signals require $\log_2 N$ bits

  Advantage: smaller control word

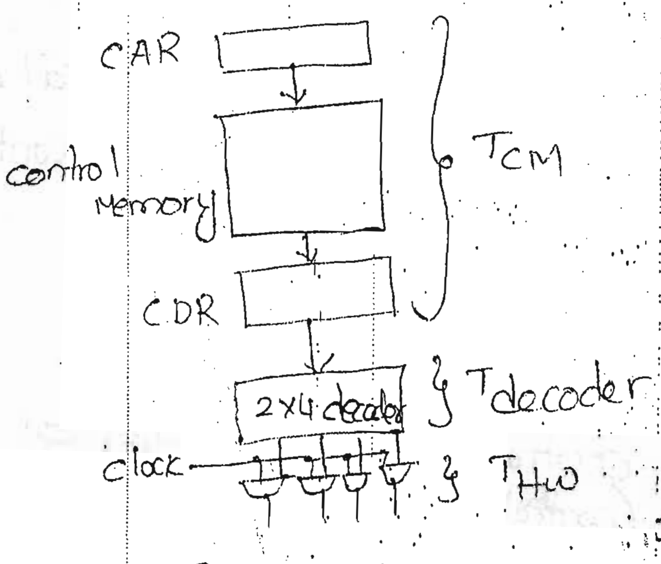
  Disadvantage: It requires a decoder

  It takes more time

  Max. degree of parallelism = 1

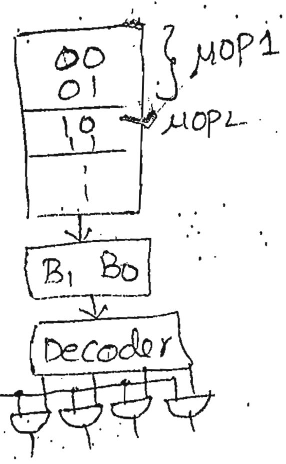
  $T_{CS} = T_{CM} + T_{Decoder} + T_{HW}$

- It is slowest control unit design



00	S0
01	S1
10	S2
11	S3

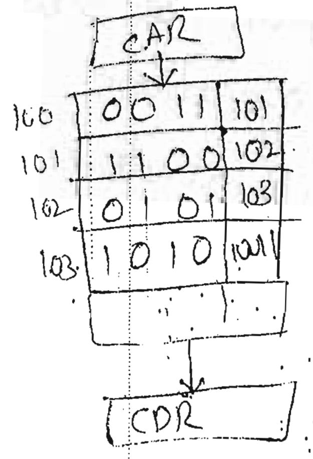
	I_1
T_1	S_0, S_1
T_2	S_2, S_3
T_3	S_0, S_2
T_4	S_1, S_3



sequencing the control words

1) using the another program counter in the system

2) using linked list
i.e having pointer to next location



100	0011	101
101	1100	102
102	0101	103
103	1010	1041

This is 1 address control Instruction

condition	control signals	Next address of control word

- 1- address control signal is same for both vertical and horizontal but they differ in the no. of bits for control signals

N bits – Horizontal

$\log_2 N$ bits – vertical

- The beginning address of control word is given by the opcode of instruction. (control program)

P) consider a micro programmed control unit which uses horizontal microprogramming and it supports 256 instructions. Each instruction on average requires 16 micro operations. The system is using 16 flag conditions and contains 64 control signals

1) How many bits are required for each control word

2) what is the size of control memory in bytes.

Let 1-address control instructions are required

Sol:

16 flag conditions.

we can use only one condition at a time

so these can be encoded

∴ 4 bits are required for encoding 16 conditions.

A is horizontal micro programming

∴ No. of bits for control signal = 64 bits.

Each instruction requires 16 μ operations

⇒ Each instruction require 16 words

Total instructions = 256

∴ Total words in control memory $= 256 \times 16$

$= 2^8 \times 2^4$

$= 2^{12}$.

∴ 12 bits are required for each word to address

condition 4 bits	control signal 64 bits	Address 12 bits

∴ Total bits for each controlword = 4+64+12
= 80 bits

size of control Memory = 2^{12} words × 80 bits/word

$= \frac{2^{12} \times 80}{8}$ Bytes

= 40 KB

P) Repeat the above using vertical µprogramming

condition 4	controlsignal 6	Address 12

34

∴ 22 bits

size = $\frac{2^{12} \text{ words} \times 22 \text{ bits/word}}{8 \text{ bits}}$

= 11 KB

P) consider a µprogrammed control unit that has to support the control signals such that it has to generate either 1 (or) none of the 63 signals and atmost four from the remaining ones. what will be the minimum number of bits needed in the ~~address~~ control signal field of one address control instruction

Sol: - either 1 (or) none of 63 signals
⇒ vertical microgramming
signals can be encoded
6 bits are needed
If all are 0 ⇒ No signal

- atmost four from the remaining
at may generate 1 (or) 2 (or) 3 (or) 4

Minimum bits required = 4 bits

Total bits in control signal field = 6+4 = 10 bits

flag	control signal (10 bits)	Address

P) consider a microprogrammed control unit design which has to support 6 groups mutually exclusive control signals the number control signals generated by each group are given below

G_1	G_2	G_3	G_4	G_5	G_6
7	11	12	3	6	10

what will be the minimum no. of bits stored per control word w.r.t. Horizontal μprogramming.

Sol: Mutually exclusive

$\Rightarrow$ either one (or) none of signals

Each group can use vertical μ programming

	G_1	G_2	G_3	G_4	G_5	G_6
signals	7	11	12	3	6	10
bits	3	4	4	2	3	4

Total bits = 3+4+4+2+3+4 = $\underline{20}$ bits

In Horizontal μprogramming = 7+11+12+3+6+10
= 49 bits

$\therefore$ No. of bits saved = 49 - 20 = 29 bits

The above approach is combinational μprograming.

within group - vertical μ programing

Between groups - Horizontal μprogramming ano programmed control unit design:

- The nano programmed control unit offers most flexibility. It follows horizontal µprogramming, at the same time it is reducing no. of bits per control word
- It follows two level control memory
- The micro control memory performs sequencing activity while the nano control memory generates the control signal. The time for accessing the control signals is

$$T_{CS} = T_{\mu CM} + T_{NCM} + T_{HW}$$

- It is used for firm-ware based applications
- The boot strap as program is also stored in nano control memory

25

- The total control memory size = µ control memory size + Nano Control Memory size

$$= H_M(\log_2 H_M + \log_2 H_N) + H_N * P$$

H_M - No. of words in Micro control memory

H_N - No. of words in Nano control Memory

P - Number of control signals.

- Any change is made to Nano control Memory. It is the slowest one

Eg:

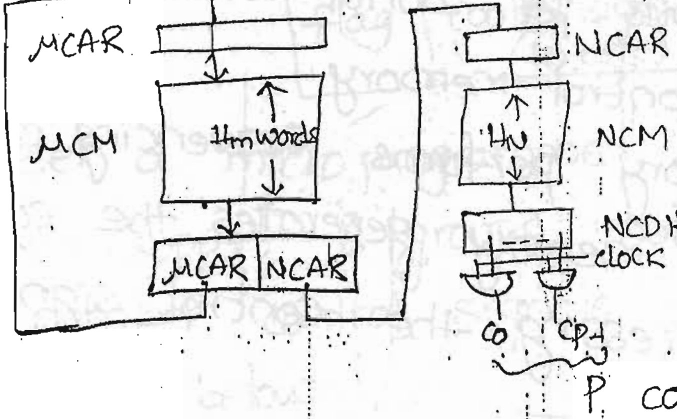
flexibility ascending order :- HW, HµP, VµP, NCM

accesstime ascending order :- HW, HµP, VµP, NCM

- Max degree of parallelism in Nano programmed control unit design is same as horizontal µprogramming = N

- Time is precious — Horizontal μP

~~Nano~~ Memory is precious - Nano control memory



IO (Input-output):

Assembly Language program tracing:

- Tracing the assembly language program concerns the computing the contents of registers (or) memory locations. and also computing about time and space requirements.
- The behaviour of the program is decided by the one (or) two key instructions.
- while tracing, status of flag conditions is to be noticed for these key instructions.

Eg:

	(64 bit word) size in words	Meaning
Load R$R_1, R_2(1000)$	4	$R_1 \leftarrow M[R_2+1000]$
ADD R_2, R_1	2	$R_2 \leftarrow R_1 + R_2$
STORE $R_1(1000), R_2$	4	$M[R_1+1000] \Leftarrow R_2$
HALT	1	

If initial contents of $R_1=10$, $R_2=1$, $M[1001]=1$

find the contents after executing the above program.

Sol:

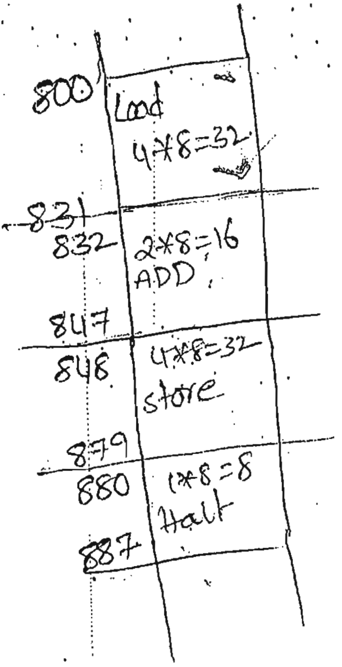
	R_1	R_2
Load R_1, $R_2(1000)$	1	1
$R_2 \leftarrow R_1 + R_2$	1	2
$[R_1 + 1000] \leftarrow R_2$		

∴ $R_1 = 1$ $R_2 = 2$

$M[1001] = 2$

Let the above program is stored in a byte organised emory from 800 decimal address onwards. If an interrupt ccurred after the fetch of ADD instruction, what ill be the address stored in stack. 36

Sol:



when interrupt occurs it completes present instruction, saves the address of next instruction in stack and branches to interrupt. After completing interrupt it gets return address from stack.

∴ Address of store instruction is stored on stack = 848 decimal value

b) If the interrupt is occured during the halt instruction what will be the saved address

Sol: Halt is implemented like

Halt ⟹ Here: JMP, Here

program counter is not incremented

∴ Address 880 is stored on stack

Q) Let the fetch, decode process consume two clocks per word for each instruction. Execution requires 4 clocks per memory instruction and 1 clock per arithmatic instruction. How many clocks are required to complete the program.

Sol :

	Subitword	F+D	E
LD R1, R2(1000)	4	8	4
ADD R2, R1	2	4	1
store R1(1000), R2	4	8	4
Halt	1	2	-

22 + 9 = 31 clocks

Q) consider the following program where R_1, R_2, R_3 are 32 bit registers. $R_2 = 8$, $R_1 = 0$. If the following program is implemented find their final values.

```
        MOV A, R3
   up:  RRC
        JNC DOWN
        INC R1
DOWN:   DEC R2
        JNZ UP
   Halt
```

Sol : ⓐ $R_1 = 8$ $R_2 = 0$ ⓑ $R_1 = 0$ $R_2 = 0$ ⓒ R_1 = No. of 0's of R_3, $R_2 = 0$

[✓ ⓓ R_1 = No. of 1s of R_3, $R_2 = 0$]

Sol :

									carry
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Ø
After RRC	Ø	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0

Q) what will be the instruction required at the blank place so that content of 'A' is restored back.

ⓐ NOP (No operation) ⓑ RLC [✓ ⓒ RRC] ⓓ ADD A, 01H consider $R_2 = 4$ $R_3 = (0110)_2$

	A	carry	R_2	R_1
1.	0110	∅	4	0
RRC	∅011	0	3	0
RRC	0∅01	1	2	1
RRC	10∅0	1	1	2
RRC	∅10∅	0	0	2

To get the content of A, we apply another RRC

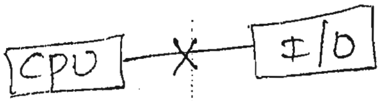
0110 = A

37

IO organisation:

- Addressing
- Data transfer techniques
- Disk and tape memories.

⟹ CPU is not directly connected to IOdevices



because

1) speed mismatch
2) different format
3) different device drivers

∴ so a blackbox is required between CPU & I/O.

Based on the capability, the blackbox can be called

1) Interface
2) Module — supports multiple interfaces
3) IO processor — Implements IO instructions
4) IO channel - contains IO processor as one component.

Interface

It varies from device to device based on the nature of device

1. serial interface

Eg: programmable communication Interface (PCI) 8251
It is also called as USART (universal synchronous Asynchronous Receiver Transmitter)

2. parallel interface

Eg: programmable peripheral interface 8255

- Serial interface has to do extra work
  1) Responsibility to convert serial to parallel (device to processor)
  2) Responsibility to convert parallel to serial (processor to device)

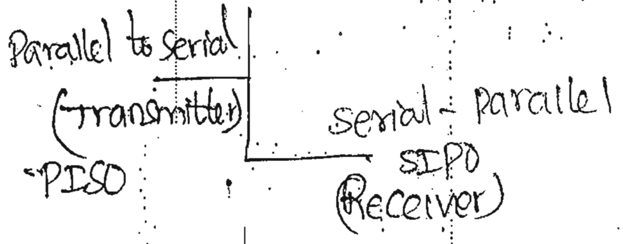
→ serial to parallel conversion is done when data is getting from device. This is at receiver section.

→ parallel to serial conversion is done when data is sending to device. This is at transmitter section.

→ Hence serial interface has two register

Serial In parallel out (SIPO) – Receiver section

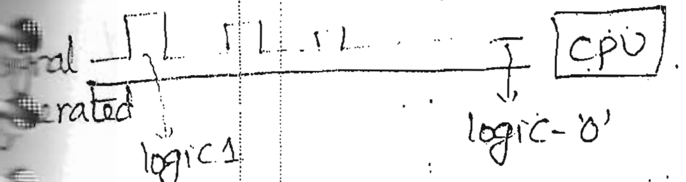
Parallel In serial out (PISO) – Transmitter section



- Every interface has to do
  1) buffering
  2) Latching

Buffering:

- Amplification of digital signals without changing the logic levels
- It is selective amplification because only logic-0 and logic-1 are amplified



The strength is decreasing. So buffering is needed

Latching:

- delaying | retaining
- To synchronise speed between processor and device

→ Interface cannot perform error control. It is only a media.

∴ Interface + Error control = Module

IO processor = Module + All IO instructions are executed

IO channel = separate IO processor and separate resources

→ selector channel ; maximum data transfer rate = $\max_{q} (D_q)$

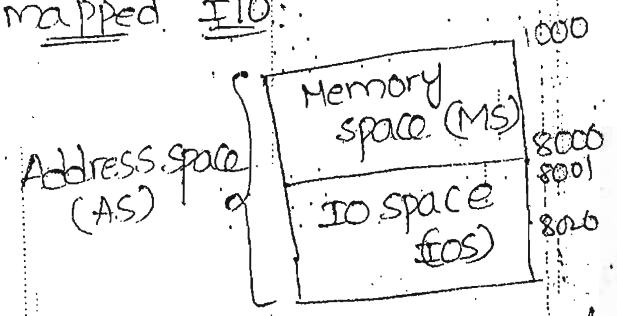
Multiplexer channel, max. data transfer rate = $\sum_{q} (D_q)$ 38

IO issues:

How to address device inorder to communicate?

There are two ways — 1) Memory mapped I/O
2) IO mapped I/O

Memory mapped I/O:



Adv: → Addresses are different
⇒ All memory transfer instructions can be extended to IO

LDA 1020 ⇐ A ← M[1020] } used to transfer data to a accumulator

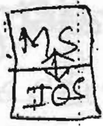
LDA 8020 ⇐ A ← (8020)

Memory device data

IO device data

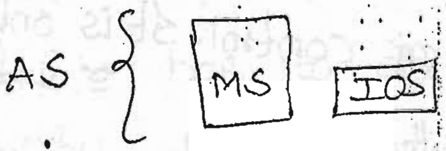
- More IO addressing modes. Hence more flexibility

disadv: change in IO space effect memory space



IO Mapped I/O:

- Also called as isolated I/O



- Hence we can decrease (or) increase IOS without effecti MS.
- Here the addresses are same for both MS and IOS. For finding the correct address mode bit (M) is used.

disadv:

1. IO/M is required to distinguish memory and I/o device addresses. so more no of address bits
2. Separate I/O instructions are needed
   Eg: IN, OUT
3. Addressing modes are less. So less flexibility

Data transfer techniques:

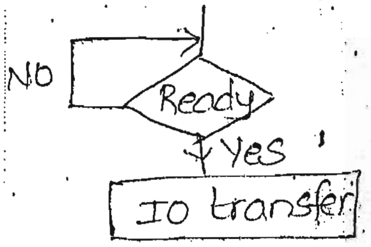
How to get data?

Three ways to get data (or) transfer data

1. program driven — processor control data transfer
2. Interrupt driven - device control data transfer
3. DMA

program driven:

one instruction continuously checks the status of device If it is ready then perform the data transfer. If it is not ready then it is kept in loop



Adv: It is simple to implement
No design complexity

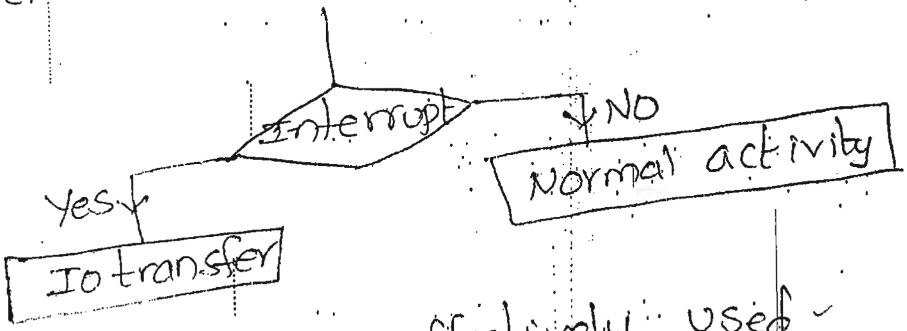
disadv Inefficient usage of CPU

Interrupt driven:

- Responsibility of device to tell processor whenever it is ready
  - If ready - Perform data transfer
  - If not ready - device will do its work.

29

- Device is controlling data transfer so it is peripheral transfer



Adv. processor is effectively used
disadv: More implementation complexity

Direct Memory Access (DMA):

- processor is isolated in the data transfer to avoid fetch, decode & execution overhead
  Normally, to perform IO transfer
  fetch the set of instructions
  Execute them
  this results to transfer data
- when bulk data is to be transferred this overhead is more [illegible] processing

∴ processor is isolated. Hence bulk data can be transferred at faster rate

eg: disk- Memory traffic

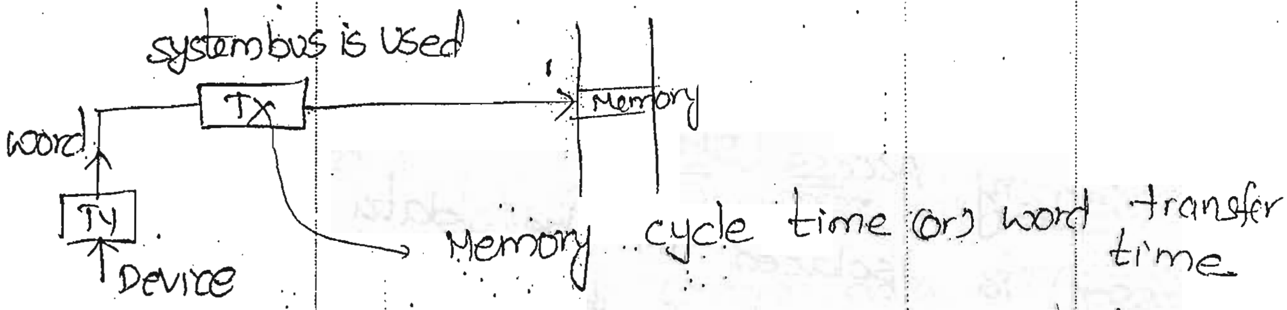
- To supervise (or) coordinate this activity, DMA is introdu
- DMA cant fetch, decode and execute. It only prepares a channel on which data can be transferred.
- To perform data transfer $\underbrace{\text{system bus}}_{\Downarrow}$ is required

  Address bus + data bus + control bus
- If two system buses, one for cpu and one for DMA it is more costly and is waste
- So sharing of only one system bus between CPU & DMA Hence both cant perform their work parallely
- So due to DMA, process is blocked ie performance is reduced.

% of time cpu blocked $= \frac{Tx}{Tx+Ty} * 100$

Tx: word transfer time

Ty: word preparation time



processor is blocked when no system bus

This cycle stealing Mode.

- Based on how the system bus shared between processor and DMA controller (or) when the system bus is returned, DMA operation modes can be

1. Burst mode
2. cycle stealing mode
3. Bulk transfer mode

In burst mode, system bus is returned to processor only after the entire data is transferred. So processor delay time is more. So processor performance is effected more

In cycle stealing mode the system bus is returned (or) exchanged after every word transfer and acquired after every instruction cycle. DMA waiting time is more (word transfer takes less time as it require only one memory reference. But instruction requires more than one memory reference) 40

- In Block transfer mode, DMA controller returns the system bus after transferring the block of data. If block size is one word then it reduces to cycle stealing mode. If blocksize is entire data then it reduces to burst mode

P) consider a 2MBps device operating in cycle stealing mode of DMA whenever 64 bit data is available the controller transfers it to memory in 2 µsec. what is the % of time the processor is blocked.

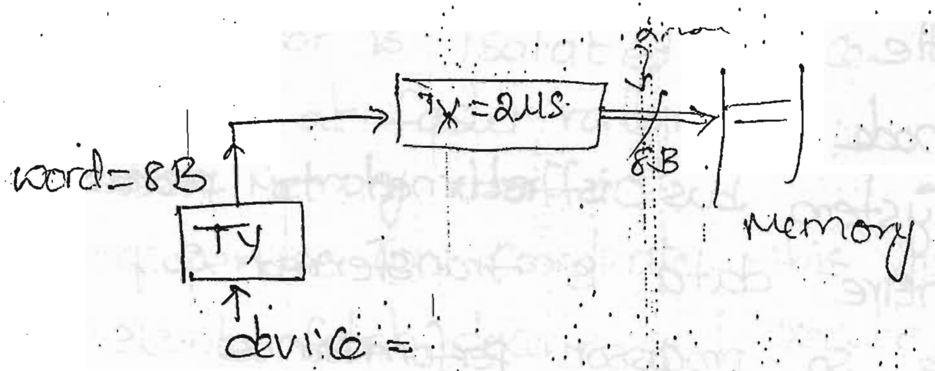
Sol:

2×10^6 B → 1 sec

8 B → 4 µsec

∴ T_y = 4 µsec

Given transfertime T_x = 2 µsec



$\therefore$ % of time CPU blocked $= \frac{2}{2+4} \times 100$

$= 33.33\%$

$\therefore$ efficiency of processor (CPU) $= 66.67\%$

i.e. processor is busy for 66.67 % of time

P) consider a system which uses 600 MHz clock. It is needed to transfer 500 Bytes. The DMA initialisation and overhead consumes 80 clocks. Every Byte will take 2 clocks per transfer. How much time taken for this DMA

Sol: = DMA initialisation time + Data transfer time

$= 80 + 500 \times 2$

$= 1080$ clocks ✓

$= \frac{1080}{600} \times 10^{-6}$ sec $= 1.8$ µ sec

P) In the above problem, If the interrupt driven transfer is used for every word, it requires 6 instruction overhead. Two of the instructions take 2 clocks and other instructions take 1 clock each. How much time is required ~~for~~ interrupt driven transfer?

Sol: 6 instructions $\left[\begin{array}{l} \text{4 instruction — each 1 clock} \\ \text{2 instructions — each 2 clocks} \end{array} \right.$

∴ Total $= 4\times1+2\times2$
$= 8$ clocks

Interrupt driven transfer time $= 8\times500$ clocks
$= \frac{4000}{600}\times10^{6}$ sec
$= 6.6\ \mu$sec

) what is the minimum performance gain possible w.r.t. DMA when compared to interrupt driven transfer

A: DMA performance $= \frac{\text{Interrupt time}}{\text{Total DMA time}} = \frac{6.6}{1.8} = 3.7$

) How many clocks are lost to the processor due to DMA. 41

A: these are lost during data transfer time
$= 1000$ clocks

% of time processor blocked $= \frac{1000}{1080}\times100$
$= 92.6\%$

Issues in interrupt implementation:

→ w.r.t. cpu
→ w.r.t. device
  → when to interrupt cpu
  → how to interrupt cpu

- The device can interrupt cpu at any time because interrupts are asynchronous in nature
- The device can interrupt cpu by changing the signal at hardware pin. This can
  1) level triggered interrupt
  2) edge triggered interrupt
- How much duration should the interrupt be maintained i.e how much time calling bell should be in pressed state

It is based on implementation
1) Maintain until it is recognised
2) Maintain some time until it is recorded in some place

w.r.t. CPU- Issues:

1) when to recognise interrupt?
After completion of current instruction

2) How to recognise interrupt?
By reading the status of interrupt flag

3) what to do?
Branch to Interrupt service Routine(ISR) whenever an interrupt is recognised. ISR address is given either by device (or) it is default.

device ⇓ Vector Interrupt

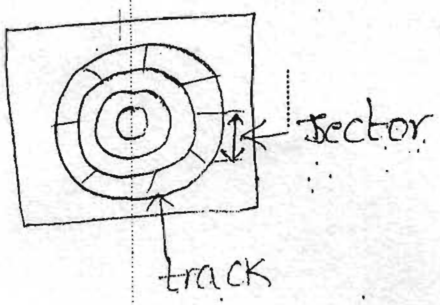
default ⇓ Scalar Interrupt

4) How to resolve in case of multiple interrupts?
- use priorities

Secondary Memories:

- It is often required to have larger programs which exceeds the size of main memory. Hence secondary memory is needed for this purpose.
- secondary memory can be
  1) disk (Magnetic disk)
  2) Tape
- Based on the access type, Memories can be
  1) random access
  2) semirandom access
  3) serial access

- Disk is a semi random access and data is distributed across circular tracks and sectors.



- on plastic base, magnetic material is coated in concentric circles — called tracks.
- Each track is divided into manageble units called as sectors. It is basic unit for accessing disk.

In general, there are equal no. of sectors per track. But the diameter of track is different so how to place data? 42

Based on method of placing the data, the disk can be

1) constant track * capacity disk
2) variable track capacity disk.

constant track capacity disk:

- Here diameter is different. But data is constant in each track.
- Here reading takes place constant time.
- To maintain constant capacity in each track, the recording density is varied i.e. the space between bits is varied

$$\text{Recording density } (e) = \frac{\text{No. of bits}}{\text{unit distance}}$$

e → Max. for inner track

e → Min. for outer track

- To perform reading, the disk has to be moved with angular velocity