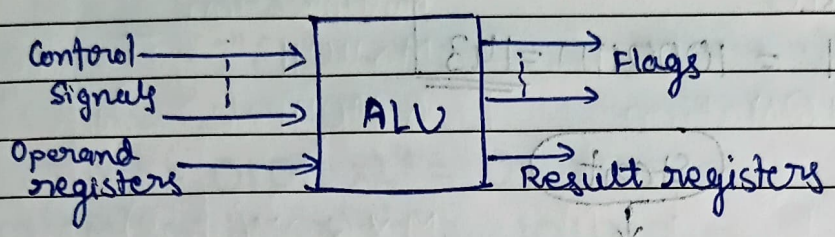
COMPUTER ORG.

UNIT 2: Arithmetic for Computers

Multiplication

- Ignoring sign bits, Multiplicand = n-bit, Multiplier = m-bit } Product = n+m bits

- Overflow Rule:
  While addition: Both nos. with same sign.
  While subtraction: Both nos. with different sign.



- 3+4
  0011
  +0100
  0111 = 7

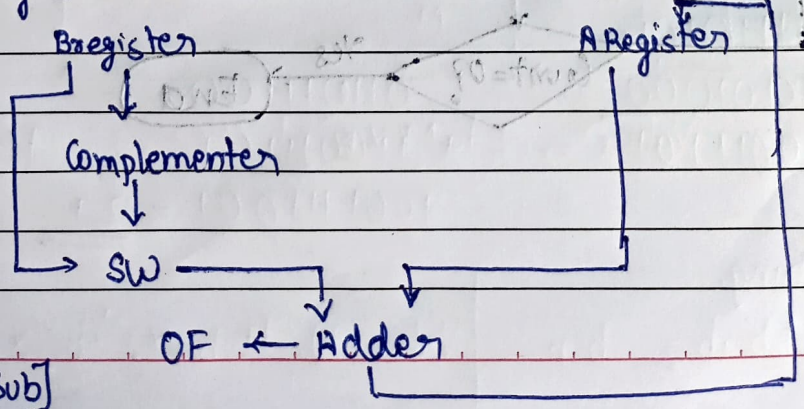
  -7 +5 ⇒ 7 = 0111
  -7 = 2' comp of 7
  = 1001
  1001
  +0101
  1110 = -2

  8 4 2 1
  0001
  +1
  0010 = 2

5+4
0101
+0100
1001 = Overflow

-7-6 = -7+(-6)
1001
+1010
10011 = Overflow.

- Block diagram of Hardware for addition and subtraction.



SW = Switch
[select Addition/Sub]

- Multiplication.

11 x 13 ⇒ 1011 x 1101

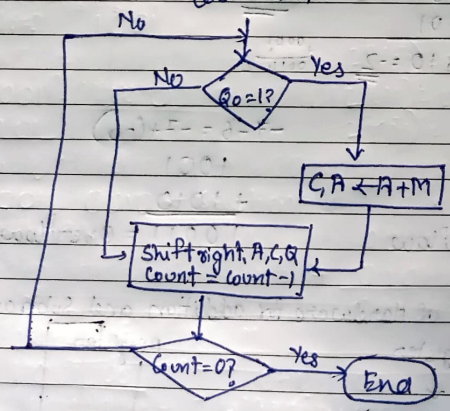
```
      1011
    x 1101
    -----
      1011
     0000x
    1011xx
   1011xxx
   --------
   10001111 = 10001111 = 143
```

-7X+3

7 = 0111
-7 = 1001

```
      1001
     x0011
    -------
    11111001
   11110010
   --------
   11101011 = -21
```

00100



~~1011~~

- Booth's Algorithm

000111 - - - - - 10000 ⇒ $2^{i+1} - 2^{i}$

M ⇒ 01011 Multiplicand ⇒

Q ⇒ 01110 Multiplier ⇒ $2^4 - 2^1$

$\overline{M}$ ⇒ 10101

M = 01011
$\overline{M}$ = 10100
+1
10101

$M \times Q : M \times (2^4 - 2^1)$

$= 2^4(M) - 2^1(M) = 2^4(M) + 2^1(-M) = 2^4(M) + 2^1(\overline{M})$

$2^4(M) = 01011 \times 2^4 = 010110000$

$2^1(\overline{M}) = 10101 \times 2^1 = 101010 \Rightarrow 111101010$

```
   010110000
 + 111101010
 -----------
 1|010011010 ⇒ Result.
```

ignore

- Booth Multiplier recoding table

Multiplier Bit i	Bit i-1	
0	0	0XM
0	1	+1XM
1	0	-1XM
1	1	0XM

0 1 1 0 1 0 1 1 0 0 1 0 1
1 0 -1 -1 -1 1 0 -1 0 1 -1 -1 -1

• Division 7/3

$3 = 0011$

1100

$+ \quad 1$

$1101 \rightarrow$ 2's comp

A	Q	M
0	0111	0011

① 0000 1110
+ 1101
1101 1110 Restore, set $Q_0 = 0$
0000 1110

② 0001 1100
+ 1101
1110
0001 1100

③ 0011 1000
+ 1101 $Q_0 = 1$
0000 1001

④ 0001 0010
+ 1101
1110
0001 0010 ← Quotient
Remainder

Multiplication

$M \rightarrow 01011 \quad \bar{M} = 10101$

$Q \rightarrow 1011 \rightarrow (2^2 - 2^0) + (2^4 - 2^3)$

$M \times Q \rightarrow M(2^2 - 2^0) + M(2^4 - 2^3)$

$= 2^2(M) - 2^0(M) + 2^4(M) - 2^3(M)$

$= 2^2(M) + 2^0(\bar{M}) + 2^4(M) + 2^3(\bar{M})$

$2^2(M) = 2^2(01011) = 0101100$

$= 000101100$

$2^0(\bar{M}) = 2^0(10101) = 111110101$

$2^4(M) = 2^4(01011) = 010110000$

$2^3(\bar{M}) = 2^3(10101) = 10101000$

$= 110101000$

000101100
$+ \quad 010110000$
011011100

111110101
$+ \quad 110101000$
110011101

011011100
$+ \quad 110011101$
$1001111001 = 21$

11×11

$+17/5 \qquad 17 \Rightarrow 10001 \qquad 5 \rightarrow 00101$

2's comp $\rightarrow 11011$

A	Q	M
0	10001	11011

① 00001 00010
+ 11011
11100
00001 00010

② 00010 00100
+ 11011
11101
00010 00100

③ 00100 01000
+ 11011
11111
00100 01000

④ 01000 10000
+ 11011
00011
00011 10001

⑤ 00111 00010
+ 11011
00010 00011 ← Quotient = 3
Remainder = 2

- Floating Point

$(5.625)_{10} \rightarrow \underset{\text{Mantissa}}{0.5625} \times 10^{1} \rightarrow$ Exponent.

$(101.101)_2 \rightarrow 0.\underline{101101} \times 2^{3} \rightarrow$ Exponent.

Normal Format → | S | Exponent | Mantissa |

- 32-bit Representation [Single Precision]

32 →

S	E'	M

Sign: 1 → -ve, 0 → +ve

E': 8-bit signed exponent in excess-127 repr.

M: 23-bit Mantissa Fraction

$E' = E + 127$
$0 \leq E' \leq 255 \Rightarrow 1 \leq E' \leq 254$
$-126 \leq E' \leq 127$

Value $\Rightarrow \pm 1.M \times 2^{E'-127}$

Ex.

$0 \mid \underbrace{00101000}_{(40)_{10}} \mid 001010 \ldots 0$

Valued repr. $\rightarrow 1.001010\ldots0 \times 2^{40-127}$
$= 1.001010\ldots0 \times 2^{-87}$

- Double Precision. 64-bit Excess-1023

64 →

S	E'	M
← 11 →	← 52 →	

$E' = E + 1023$
$1 \leq E' \leq 2046$
$-1022 \leq E \leq 1023$

Value $\Rightarrow \pm 1.M \times 2^{E'-1023}$

- n-bit string.

$B = 1.M = 1.b_{-1}b_{-2}\ldots b_{-n}$ | Value(B) $= 1 + b_{-1}\times 2^{-1} + b_{-2}\times 2^{-2} \ldots b_{-n}\times 2^{-n}$

0
1
-7
-8
-7
-1

- Normalization

- Special Values

End values ⇒ 0 and 255 of excess 127 exponent E'

When $E'=0$ and $M=0$; value = 0.

$E'=255$, $M=0$; value = ∞.

$E'=255$, $M\neq0$; NaN ⇒ 0/0 or $\sqrt{-1}$

$E'=0$, $M\neq0$; denormal numbers, $\pm 0.M \times 2^{-126}$ smaller nos., allow for gradual underflow.

- Exception

Underflow, Overflow, divide by zero, inexact and Invalid.

Inexact → needs rounding in order to be represented in one of normal Format.

Invalid → 0/0, $\sqrt{-1}$

- Explicit Normalization

$(101.101)_2 \rightarrow 0.101101 \times 2^{3}$

Implicit Normalization → Better w.r.t. precision

$(101.101)_2 \rightarrow 1.01101 \times 2^{2}$

- Biasing.

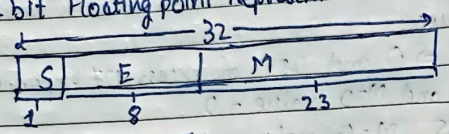
We cannot save signed exponential value. So we have to make signed to unsigned repr.

-8	-7	-6	-5	—	0	1	2	—	—	7
(+8)	—			—	(+8)	—			—	(+8)
↓	↓	↓	↓	—	↓	↓	↓	—	—	↓
0	1	2	3	—	8	9	10	—	—	15

∴ Called Exponent: Excess(8)

- Examples

32 bit Floating point Representation



Excess 127

Value $\Rightarrow \pm 1.M \times 2^{E'-127}$ $\qquad E' = E + 127$

① -5

$\Downarrow$

| 1 | 10000001 | 0100 ---- 0 |

- 101.00

$\Downarrow$

$1.0100 \times 10^{2} \rightarrow E \therefore E' = E + 127 = \underline{129} \Rightarrow 10000001$

② $-6 \Rightarrow 110.00 \Rightarrow 1.1000 \times 10^{2}$

| 1 | 10000001 | 1000 ---- 0 |

S E M

③ $-1.5 \Rightarrow 01.100 \Rightarrow 1.100 \times 2^{0}$ $\qquad 0 + 127 = \underline{127}$

$0.5 \times 2 \rightarrow 1.0 \quad 1$

$0.0 \times 2 \rightarrow 0.0 \quad 0 \downarrow$

$0.0 \times 2 \rightarrow 0.0 \quad 0$

| 1 | 01111111 | 1000 ---- 0 |

S E M

④ 384

⑤ $1/16 = 0.0625 = 0.0001 = 1.0 \times 2^{-4}$

$0.0625 \times 2 = 0.125$ $\qquad$ ~~1274~~ $127 - 4 = 123$

$0.125 \times 2 = 0.25$

$0.25 \times 2 = 0.5$

$0.5 \times 2 = 1.0$

| 0 | 01111011 | 00000000 ---- |

⑥ $-1/32$

- Add $\qquad$ A → 414C0000H $\qquad$ B → 40E80000H

A → 0100 0001 0100 1100 0000 0000 0000 0000

B → 0100 0000 1110 1000 0000 0000 0000 0000

$E'_A = 130$, $E'_B = 129$ | Make $E'_B = 130$ by shifting right.

$\therefore$ B → 0100 0001 0111 0100 0000 0000 0000 0000

Adding mantissa of A and B:

	S	E	M
A	0	100 0001 0	100 1100 0000 0000 0000 0000
+ B	0	100 0001 0	111 0100 0000 0000 0000 0000
	0	100 0001 0	100 0000 0000 0000 0000 0000

→ 41

- IEEE Standard for Binary Floating Point Repr.
  1. Arithmetic format
  2. Basic Format → 5; 3 for binary, 2 decimal
  3. Interchange format

Additional Formats:-
1. Extended Precision Format
2. Extendable Precision Format

- Division
  1. Positive Nos.

N/M — 2's complement of M should be added to A

A Q

[N no. divided] [equal digits]

Keep on shifting left.
Add M to A every time.
if MSB = 1, then restore and $Q_0 = 0$
if MSB = 0, then $Q_0 = 1$

2. If Negative no.

-N/M → 2's complement of N

A Q

[Prefix 1, signbits, -N]

Keep shifting
Keep on adding M to N.
If MSB = 1, then $Q_0 = 0$
if MSB = 0, restore and $Q_0 = 1$

Operations to be done for no. of bits in Q times.

- Multiplication
  1. Booth's algo +ve nos.

M×Q — Takes 2's comp of M, $\overline{M}$

break Q in terms of $2^{j+1} - 2^i$ $j > i$

j = starting index of consecutive 1's and i is last index of that set.

Indexing is 0-based, LSB = 0 index.

For ex: Q = 11101 ⇒ $(2^{4+1} - 2^2) + (2^{0+1} - 2^0)$
4 3 2 1 0
$= (2^5 - 2^2) + (2^1 - 2^0)$

Multiple M with Q.
For -ve sign multiply with $\overline{M}$.
Now for adding all nos.
$2^n(M)$ = add n 0's at LSBs
$2^n(\overline{M})$ = —"— but to get size of all nos. equal append 1's to start.

Now add all nos., ignore carry.

- Addition

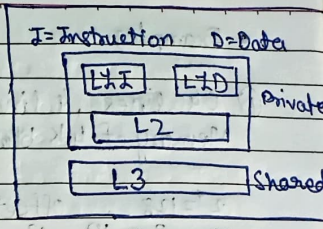
① Represent A, B in IEEE 32-bit format
First bit = sign, to next 8 = expo, rest mantissa.

Here to make the expos equal.
Find difference and smaller needs to be made equal to greater one. Write expos equal and difference no. of zeros to be added at starting of Mantissa.
Add mantissa.
While taking out mantissa it is 1. given., so also while shifting that 1 should be considered. And also ~~th~~ while placing final answer that 1 is ignored.

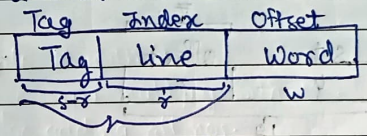
UNIT-3

- Direct Mapping

Maps each block of main memory into only one possible cache line.



The mapping is expressed as
$i = j \text{ modulo } m$
i = cache line no. j = main memory block no.
m = no. of lines in cache



⇐ Memory address

$s \rightarrow 2^s$ blocks of main memory

Address length $= (s+w)$ bits
No. of addressable units $= 2^{(s+w)}$
Block size = line size $= 2^w$ words or bytes
No. of blocks in main memory $= \frac{2^{s+w}}{2^w} = 2^s$

No. of lines in cache $= m = 2^r$
Size of cache $= 2^{r+w}$
Size of tag $= (s-r)$ bits.

Least Flexible
~~Longest~~ Shortest search time $O(1)$

Valid bit ⇒ 0 → Not present
1 → Present

Dirty bit ⇒ 0: Block is not recently modified.
1: Block is recently modified so write back to main memory first then replace it.

- Examples

① 64 lines, 4 lines in each set.
Memory of 4K blocks of each 128 words.

$2^7 = 128$. offset = 7 bits.
$4k = 2^2 \times 2^{10} = 2^{12}$

$\therefore$ No. of sets $= \frac{64}{4} = \underline{\underline{16}}$ (2^4)

Total $\Rightarrow 2^{12} \times 2^7 = 2^{19}$

19 bits

Tag	set	offset
8	4	7

② 2way set association
Lines of 16 bytes . Total size = 8kbs $= 2^3 \times 2^{10} = 2^{13}$
MM = 64MB $= 2^6 \times 2^{20} = 2^{26}$
~~Total size $\Rightarrow 2^{13} \times 2^{26} = 2^{39}$~~
Total no. of lines $= \frac{8kbs}{16bytes} = \frac{2^3 \times 2^{10}}{2^4} = 2^9 = \underline{\underline{512}}$
2way set $\Rightarrow \therefore$ Cache consists 256 sets of 2 lines each.
$\therefore$ To identify set $\underline{\underline{8}}$ bits.

64 MB main memory $\Rightarrow$ 26 bits.
64MB/16 bytes $= 2^{22}$ blocks $\Rightarrow$ 22 bits.
$\therefore$ Tag + set $= \underline{\underline{22}}$ $\therefore$ Tag $= \underline{\underline{14}}$

Offset $= 26 - 22 = \underline{\underline{4}}$

Tag	set	offset
14	8	4

26 bits

- Set Associative (k-way set associative) — only search in k locations

Tag	Set	offset
s-d	d	w

Search time = O(k)

$m = v \times k$ $i = j \bmod v$

i = cache set no j = main memory block number
m = no. of lines in cache v = number of sets
k = no. of lines in each set
— No. of sets $= v = 2^d$ — d = No. of set bits
No. of lines in cache $= m = kv = k \times 2^d$
size of cache $= k \times 2^{d+w}$
Size of tag $= (s-d)$ bits.

- Full associative Mapping

Tag	Byte
s	w

Most flexible. Longest search time = O(N)
No. of lines in cache is undetermined
Size of tag = s. bits.

Examples

③ MM = 4GB Cache size = 1MB Blocksize = 4KB

→ MM = 4 GB = $2^2 \times 2^{30} = 2^{32}$ 32 bits total size.

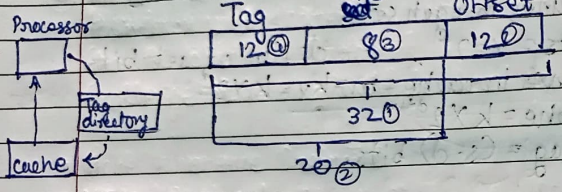
No. of blocks in MM = $2^{32}/4KB = 2^{32}/2^{12} = 2^{20}$

20 ⇒ Line + Tag bits

Blocksize = 4kb = 2^{12} — 12 bits offset.

No. of lines in cache = $1MB/2^{12}$ = 2^8 → Bits for line

↳ Blocksize



Tag directory size = No. of lines in cache × No. of tag bits

= $2^8 \times 12$ bits = 3072 bits

④ MM size = 256 MB Cache size = 512 KB

Not needed → Block size = 256 B Word size = 1 Byte ← NN

→ MM size = $2^8 \times 2^{20} = 2^{28}$

Cache size = 512 KB = 2^{19}

No. of tag bits = $\log_2$(MM size / Cache size)

= $\log_2(2^{28}/2^{19}) = 9$

⑤ MM size = 16 GB = $2^4 \times 2^{30} = 2^{34}$ ⇒ Total = 34 bits.

Block size = 16 KB = $2^4 \times 2^{10} = 2^{14}$ ⇒ offset = 14 bits

No. of tag bits = 10

∴ Cache size = lines × offset = 2^{24} = 16 MB

Tag = 10	Line = 10	Offset = 14

⑥ Main memory = 2^{16} bytes Block size = 8 bytes

Assume that a direct mapped cache consisting of 32 lin

a) 16 bit memory address

→

Tag = 8	Line = 5	Offset = 3

b) In which line?

0001 0001 0001 1011 → Line 3

Tag Line offset

1100 0011 0011 0100 → Line 6

c) Byte with address 0001 1010 0001 1010 in cache

Other bytes stored along with it? use offset

→ 0001 1010 0001 1000 to

0001 1010 0001 1111

d) How many total bytes of memory can be stored in cach

→ $2^5 \times 2^3 = 2^8 = 256$ bytes.

Write through → Cache & Main Memory are update simultaneously.
Write back → Cache updated in real time and use dirty bit

- Write strategies
  1. Write Hit : Write through, write back
  2. Write Miss : Write Allocate
     No-write Allocate

- Cache Coherence [Snooping Based / Directory Based] → Shared bus / Broadcast messging
  ↳ Uniformity of shared resource data
  Cache Coherence Problem:
  Challenge of keeping multiple local cache synchronized.

- Cache Coherence Protocols
- MESI Protocol
  Modified Exclusive Shared Invalid

Modified → line in cache has been modified
Exclusive → line in cache is the same as that in main memory.
Shared → line in cache is the same as that in MM and may be present in another cache.
Invalid → line in cache does not contain valid data.

	M	E	S	I
1) Cache line valid?	Yes	Yes	Yes	No
2) The memory copy is...	out-of-date	valid	valid	-
3) Copies exist in other caches?	No	No	Maybe	Maybe
4) A write to this line	does not go to bus.	—	goes to bus & updates cache	goes directly to bus

- Examples

(1)

P0

Coherency state	Address Tag	Data
B0 I	100	0010
B1 S	108	0008
B2 M	110	0030
B3 I	118	0010

P1

CS	AT	D
B0 I	100	0010
B1 M	128	0028
B2 I	110	0010
B3 S	118	0018

P3

CS	AT	D
B0 S	120	0020
B1 S	108	0008
B2 I	110	0010
B3 I	118	0010

on-chip interconnect [with coherency manager]

Memory

Address	Data
100	0010
108	0008
110	0010
118	0018
120	0020
128	0028
130	0030

(a) P0 : read 120
→ 120 not in P0 but it is in P3 in shared state
B0 is invalid in P0 and can be used.
P0.B0 [S, 120, 0020]
B0 from P3 will come to P0 at B0 and B0 from P0 will get evicted.

(b) P0 : ~~read~~ write 120 ← 80
P0.B0 [M, 120, 0080] ⇒ Modified
P3.B0 [I, 120, 0020] ⇒ Invalidated as shared it was

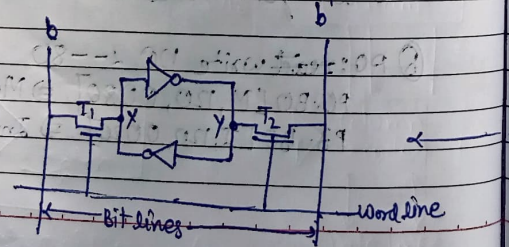
(c) P3: write 120 <-- 80
→ P3.B0 [M, 120, 0080]

(d) P1: read 110
→ 110 in P1 is invalid, so cannot be read.
But 110 is M in P0.
P1.B2 [S, 110, 0030]

(e) P0: write 108 <-- 48
→ P0.B1 [M, 108, 0048]
P3.B1 [I, 108, 0008]

(f) P0: write 130 <-- 78
→ Not in any cache, so now use replacement.
Target the Modified once while bringing data from main memory. Set dirty bit = 1.
① P0.B2 [M, 130, 0078]
② M ← 110 ⇒ Write back to main memory.

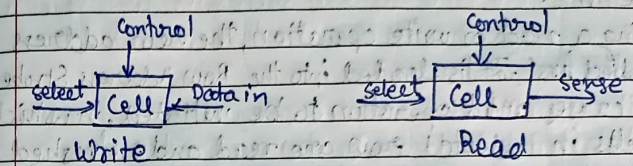
(g) P3: write 130 <-- 78
→ No M states, now target I states.
P3.B2 : [M, 130, 0078]
Write back to Main memory not needed as we have modified the invalid states data.



RAM
- Static [SRAM]
- Dynamic [DRAM]
  - Syne [SDRAM]
  - Asyne

Internal Memory

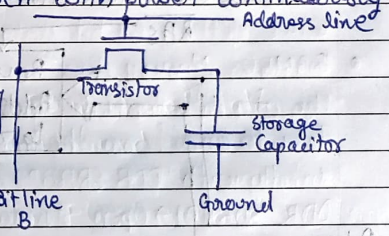
- Memory Cell Operation



Basic element a semiconductor memory is a memory cell.

- Dynamic RAM
Made with cells that store data as charge on capacitors. The presence or absence of charge in a capacitor is interpreted as a binary 1 or 0. Because capacitors have a natural tendency to discharge, dynamic RAMs require periodic charge refreshing to maintain data storage. The term dynamic refers to this tendency of the stored charge to leak away, even with power continuously applied.

DRAM stores a single bit [0/1]



- Static RAM
Binary values are stored using traditional flip-flop logic gate configurations. A static RAM will hold its data as long as power is supplied to it.

SRAM faster than DRAM. SRAM are used for cache memory and DRAM is used for Main memory.

DRAM

Reading the contents of a cell automatically refreshes its contents. Since the wordline is common to all cells in a row, all cells in a selected row are read and refreshed at the same time.

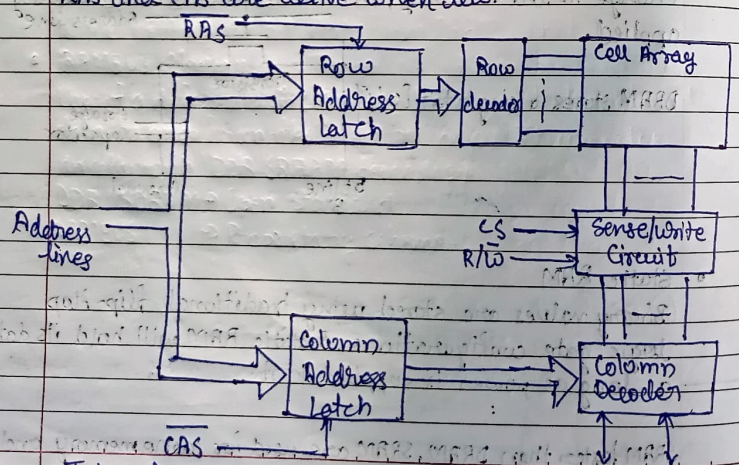
During a read or write operation, the row address is applied first. It is loaded into the Row Address Strobe [RAS]. This causes read operation to be initiated, in which all cells in selected row are read and refreshed. Shortly after row address is loaded, the column address is applied to address pins and loaded into the column address latch under ~~control~~ control of Column Address Strobe [CAS]. The information in this latch is decoded and appropriate group of 8 sense/write circuits are selected.

R/W control signal:

~~Read~~ Read → Selected circuit $\xrightarrow{\text{data}}$ ~~D0 - D7~~ Data lines

Write → Data lines $\xrightarrow{\text{data}}$ selected circuit

RAS and CAS are active when low.



• Internal Organization of Memory Chip

Fast Page Mode

Possible to access multiple bytes/circuits in same row without having to reselect the row.

Each sense amplifier also acts as a Latch.

• Synchronous DRAM

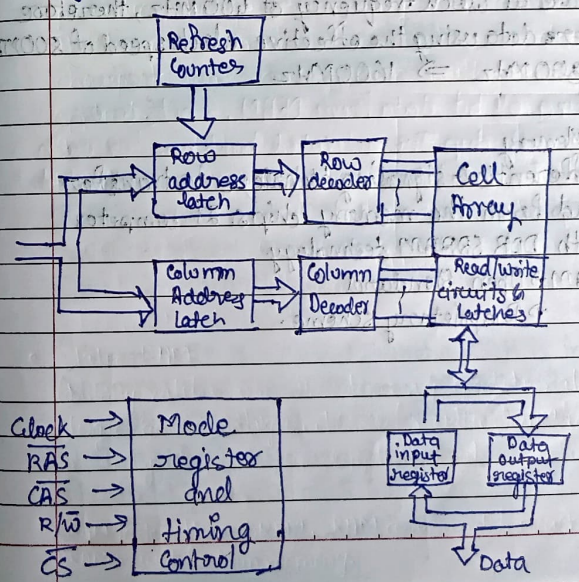
Synchronized with Clock Signal.

SDRAM have build-in refresh circuitry, with a refresh counter to provide the addresses of the row to be selected for refreshing.

Sense/Write amplifiers function as Latches.

SDRAMs have several different modes of operation which can be selected by writing control information into a mode register.

For selecting successive columns, column counter and clock signal are used.



Latency and Bandwidth
During block transfers, memory latency is the amount of time it takes to transfer the first word of a block. The time between successive words of a block is much shorter than the time needed to transfer the first word.
Another useful performance measure is the number of bits/bytes that can be transferred in one second. This measure is often referred as memory bandwidth. Interconnections used alway ensures that the bandwidth available for data transfers between the processor and the memory is very high.

- Double-Data-Rate SDRAM
Data are transferred externally on both rising and falling edges of the clock.
DDR2 operate at clock frequency of 400 MHz, therefore they transfers data using the effective clock speed of 800MHz
DDR3 at 800 MHz ⇒ 1600MHz.

- RAMBUS Memory
Use of a differential-signaling technique to transfer data to and from the memory chips. It competes directly with DDR SDRAM technology.
DDR SDRAM → Open Standard
RAMBUS → Proprietary scheme.

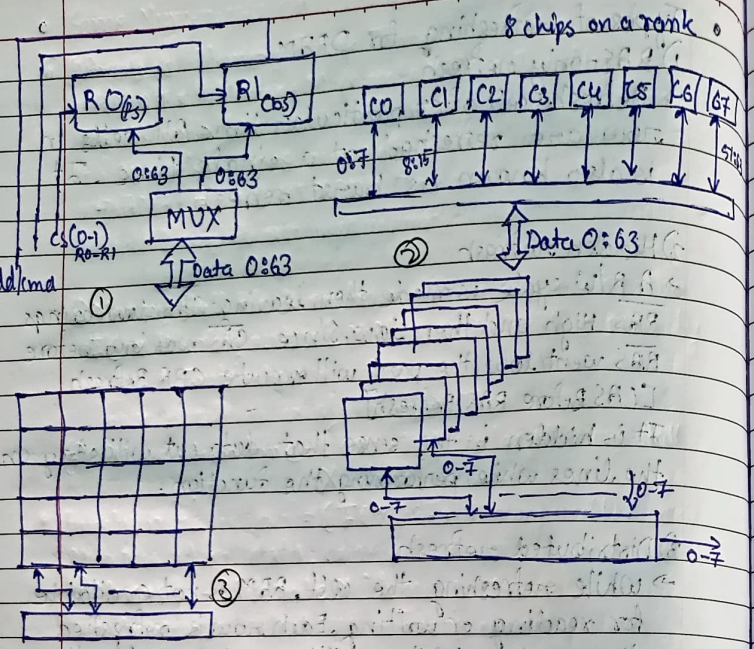
- Types of refreshing for DRAM
1) RAS-only refresh
→ Refreshes cell after particular time. All cells in row are refreshed at a single time. It is also known as burst refresh.

2) Hidden refresh
→ A R/W cycle is made then, leaving $\overline{CAS}$ low, brings $\overline{RAS}$ High and then low. Since $\overline{CAS}$ was low before $\overline{RAS}$ went low, the part will execute CBR Refresh. [CAS Before RAS Refresh]
It is hidden in the sense that data out will stay on the lines while performing the function.

3) Distributed refresh
→ While refreshing the cell, RAM is not available for reading or writing. Each row is refreshed after sometime. While waiting period, read/write action can take place.

- Synchronous DRAM [Class notes]
A chip present on DDR. One side of it is called Rank0 and another Rank1.
On each RAM, there are 8 chips working simultaneously.

1) Channel - It is a socket where DDR is inserted
2) DIMM [Dual Inline Memory Module] - Actual packet/Module
3) Rank - 0 and 1. Surfaces on which chip are present.
4) Each side 8 chips are present.
5) Each chip consists of 8 Banks
6) In Bank, there are row and column addresses.



While getting 64 bit data from DRAM, 8 bits are taken from each chip. The activated bank no. for each chip is same. ~~For 8 bits~~ Row and column no. activated in each chip is same. For 8-bits from each bank, 8 columns of that bank are read.

Example ① Assume DDR with storage capacity 1GB, it consists of 1 Rank, 8 chips, 8 banks per chip. If there are 14 rows, find no. of columns.

Storage capacity = No. of chips X banks X rows X columns.

$2^{30} = 2^3 \times 2^3 \times 2^{14} \times 2^{x}$ → No. of columns

$2^{x} = 2^{30}/2^{20}$

$\therefore x = 10$ → No. of columns

② Storage capacity = 4 GB = $2^2 \times 2^{30} = 2^{32}$

2 Ranks, 8 chips, 8 banks per chip, columns = 10.

x = No. of rows

$2^{32} = 2^1 \times 2^3 \times 2^3 \times 2^{10} \times 2^{x}$

$2^{x} = 2^{32}/2^{17}$

$\therefore x = 15$ → No. of rows.

③ 1GB x 32 built of 32MB x 16 chips

→ No. of chips = ~~1GB/32MB = $2^{30}/2^{25}$~~

1G x 32 / 32M x 16

Date / /

- Some Numericals on Cache

① Average Memory Access Time

AMAT = Hit time + Miss rate X Miss Penalty

Ex. CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, cache miss rate = 5%

→ AMAT = 1 + 0.05 X 20 = 2ns

2 cycles per instruction.

② Memory stall clock cycles =

IC X $\frac{\text{Memory accesses}}{\text{Instruction}}$ X Miss rate X Miss penalty

IC → Instruction count

③ CPU execution time = (CPU clock cycles + Memory stall cycles) X Clock cycles

IC X CPI

= (IC X CPI + Memory stall) X clock cycle

④ $\frac{\text{Misses}}{\text{Instruction}}$ = Miss rate X $\frac{\text{Memory accesses}}{\text{Instruction}}$

Error detection

① Hardware failures - Permanent physical defect so that the memory cell or cells affected cannot reliably store data but become stuck at 0 or 1 or switch erratically between 0 and 1. Caused by harsh environment, manufacturing defects and wear.

② Soft errors - Random, non-destructive event that alters the contents of one or more memory cells without damaging the memory. Caused by power supply problems or alpha particles.

- function 'f' is performed on data to produce a code. M-bit word of data, code of K bits; actual size of the stored word will be M+K bits.

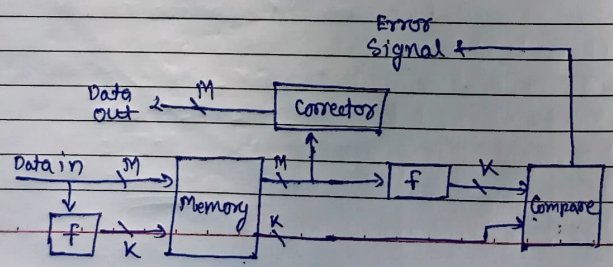
Code is used to detect and possibly correct errors

3 conditions (tests/outputs) are yielded:-

① No errors are detected. Fetched data bits are sent out.

② An error is detected, and it is possible to correct it. The data bits + error correction bits are fed into a corrector, which produces a corrected set of M bits to be sent out.

③ An error is detected, but it is not possible to correct it. This condition is reported.



- Hamming Code

Syndrome word is of K bits and has range 0 to 2^K-1. Value 0 → No error detected; leaving 2^K-1 values to indicate, if there is an error, which bit was in error.

$$2^K-1 \geq M+K$$

Generate a 4-bit syndrome for an 8-bit data word with foll. characteristics:-

① If syndrome contains all zeros, no error has been detected

② If syndrome contains one and only one bit as 1, then error has occured in one of 4 check bits, no correction is needed.

③ If syndrome contains more one bit set to 1, then the numerical value of syndrome indicates position of the data bit in error. This data bit is inverted for correction.

Check bits:-

$C1 = D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7$ — First LSB = 1

$C2 = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7$ — 2nd LSB = 1

$C4 = D2 \oplus D3 \oplus D4 \oplus D8$ — 3rd LSB = 1

$C8 = D5 \oplus D6 \oplus D7 \oplus D8$ — 4th LSB = 1

8-bit input word = 00111001

$C1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$C2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$

$C4 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$

$C8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$

Bit Pos.	12	11	10	9	8	7	6	5	4	3	2	1
Pos. No.	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Checkbit					C8				C4		C2	C1

Suppose D3 sustains an error and is changed from 0→1.
C1=1, C2=0, C4=0, C8=0.

Syndrome ⇒

	C8	C4	C2	C1
	0	1	1	1
⊕	0	0	0	1
	0	1	1	0

Indicating that bit position 6, which contains D3, is in error.

① HDD

Harddisk contains no. of disks on which data is stored. Each disk is called as platter. Platter is made up of magnetic material/glass/plastic.
On this non-magnetic disk, magnetic material is coated in order to make track available.
In track there are sectors. 512 bytes in each sector.
Each platter having top and bottom surface, on both side data is stored. Documentation can be done on any platter and surface.
There is head on surfaces. 4 platters will have 8 heads.
Depending on from which track we want to read/write on that disk, surface head is moved to that particular track and particular sector on that track.
A motor is used to spin the platter.

Disk performance: seek time, rotational delay, transfer time

① Seek time - Time required to move disk arm to req. track.
② Rotational delay - Time it takes for head to reach at the beginning of sector.
③ Access time = seek time + rotational delay
④ Transfer time - To or from the disk depending on the rotation speed of disk

$$T = b / rN$$

T = Transfer time b = No. of bytes to be transferred
N = No. of bytes on a track r = rotation speed in rpm.

⑤ Total average read/write time = T_{total}

$$T_{total} = T_s + \frac{1}{2r} + \frac{b}{rN}$$

T_s = avg. seek time

Examples ① For reading a file upto 2000 sectors with 500 sectors per track, 512 bytes per sector, with rotational speed of 15000 rpm and avg. seek time = 4 msec. Assume file stored in consecutive sectors.

a) How file stored as compact as possible, find time.

b) Find time for storing a file that can be randomly distributed over a disk.

$$15000 \text{ rpm} \Rightarrow \frac{60}{15000} = \frac{1}{250} \text{ sec} = 0.004 \text{ sec} = 4 \text{ ms}$$

a) → $4\text{ms} + 4\text{ms} * 0.5 + \left(\frac{2000}{500} \times 4\text{ms}\right) = 22\text{ms}$ [Optimized]

avg. seek time + Rot. delay + transfer time

b) → $2000 \times \left(4\text{ms} + 2\text{ms} + \frac{4\text{ms}}{500}\right)$ = Avg. seek time

② Consider a magnetic disk drive with 8 surfaces, 512 tracks on each surface, 64 sectors per track. Sector size is 1kb. Avg. seek time = 8ms, track to track access time = 1.5 ms and drive rotates at 3600 rpm. Successive cylinders can be read without head movement.

ⓐ What is disk capacity?

disk capacity = No. of surfaces X no. of track per surface X sectors per track X sector size

$= 8 \times 512 \times 64 \times 1\text{KB}$

$= 2^3 \times 2^9 \times 2^6 \times 2^{10}$

$= 2^{28} = 2^8 \times 2^{20}$

$= 256\text{MB}$

ⓑ What is avg. access time? Assume file is stored in successive sectors and tracks of successive cylinder. Starting of sector is 0th location, track 0 of ith cylinder

→ Access time = seek time + rotational delay.

$$\text{rotational delay} = \frac{60}{2 \times 3600} = 8.3\text{ms}$$

∴ access time = 8.3 + 8 = 16.33 ms

ⓒ Estimated time required to transfer a 5MB file.

→ Each cylinder is made of 8 tracks, each of 64KB

64 sector each of 1kb

Total memory = 8 X 64KB = 512KB

∴ To store 5MB file, we need 10 cylinders.

Disk will take seek time of 8ms to find the cylinder and 8.33 ms to find the sector [rotational delay]

Time taken to move to next cylinder = 1.5 ms

Access time = 8 + 8.33 + 133.333 + 9 × (1.5 X 8.333 + 133.333)

= 1438.16 ms

Time taken to read cylinder is time to read its tracks, $\frac{8 \times 60}{3600} = 133.33\text{ms}$

ⓓ What is burst transfer rate?

Burst rate = revolution per second X sectors per revolution X bytes per sector

$= \frac{60}{3600} \times 64 \times 1\text{kb} = 3.4\text{ MB/s}$

③ Consider a single platter disk
rotation speed = 7200 rpm
No. of track on 1 side = 30000
Sector per track = 600
Seek time = 1 msec for every 100 tracks

a) Avg seek time
→ Assume head starts at track 0
Request track is 0 then seek time = 0
—— " —— 29999 —— " —— = 29,999

For random req. on avg, the no. of tracks travelled is
$\frac{29999}{2} = 14999.5$ tracks

At 1ms. per 100 tracks.
Avg seek time $= \frac{14999.5}{100} = 149.995$ ms

b) Avg rotational latency
speed = 7200 rpm $\frac{60}{7200} = 8.3$ ms

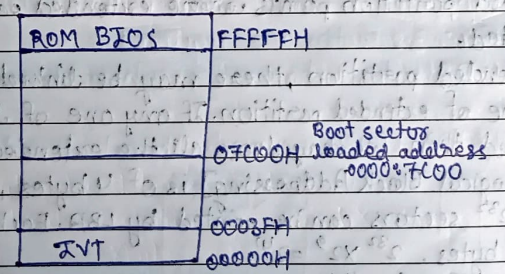
Avg. rotational latency $= \frac{8.333}{2} = 4.167$ ms

c) Transfer time for sector $= \frac{8.333}{600} = 0.01389$ ms

d) Total avg. time to satisfy a req.
= avg. seek time + avg. ~~rotational time~~ rotational latency + transfer time
= 149.995 + 4.167 + 0.01389
= 154.17589 ms

⑥ MBR [Master Boot Record]

• Booting Process
① System initialization with firmware is called BIOS.
② BIOS looks for bootloader on MBR, then executes it.
③ Bootloader reads partition table, conventional Windows/Disk operating system MBR bootloader search for one primary partition.
④ Load the OS
ⓐ As we power on, CS = FFFFH and IP = 0000H
ⓑ ROM BIOS present at FFFF0H.
ⓒ Once power on, reading of ROM BIOS starts.
ⓓ While reading ROM BIOS, it is made to read 1st sector of HDD.
ⓔ Very first sector of HDD is referred to as bootsector. Boot sector is loaded from
07C00H [CS = 0000H, IP = 7C00 H]
ⓕ In this the very first sector [512 bytes], there is small boot program. Boot Program finds out on HDD where is the active partition.
ⓖ Active partition, the OS program which needs to be loaded.
MBR that is located on 1st sector of HDD.



Basic Structure of MBR sectors

1. Boot Code
2. Partition Table
3. Boot Record Signature

000-445	000H-1BDH	446 bytes code area
446-509	1BEH-1FDH	64 bytes Master Partition table
510-511	1FEH-1FFH	2 bytes boot record signature [AA55]

			Primary Partition
446-461	1BE-1CD	16 bytes	#1
462-477	1CE-1DD	16 bytes	#2
478-493	1DE-1ED	16 bytes	#3
494-509	1EE-1FD	16 bytes	#4

If it is MBR HD, then atmost 4 primary partition are possible. If more partition are needed, then one of the primary partition is used as extended partition. And that partition points where extended partition is located.

In extended partition, there may be linked-list like structure of extended partition. If any one of link is destroyed then we may loose all the extended partitions.

LBA [Logical Block Addressing] is of 4 bytes, then at max 2^{32} sectors can be pointed by LBA. Each sector of 2^9 bytes. $2^{32} \times 2^9 = 2^{41}$

Hence $2^{32} \times 2^9 = 2^{40} \times 2 = 2TB$ entire capacity.

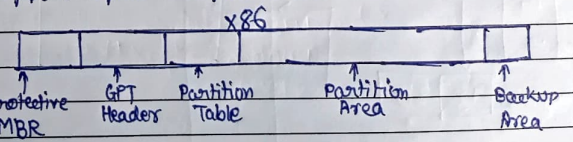
Any MBR disk more than 2TB, the MBR cannot access if going above 2TB.

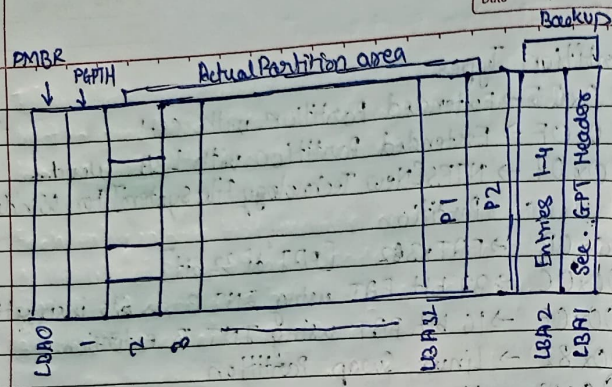
- Partition Types
  1. 0x05 → Extended Partition with CSH
  2. 0xOF → Extended Partition with LBA
  3. 0x07 → NTFS [New Technology File System] For Windows Partition
  4. 0xOB → FAT x32 [FAT of 32 bit]
  5. 0xOC → 32 bit FAT using INT 13 as its instructions.
  6. 0xOE → 16 bit FAT using INT 13 extension.
  7. 0x82 → Linux Swap Partition
  8. 0x83 → Linux Native File System [EXT3 / EXT4]
  9. 0xEE → GPT protective MBR
  10. 0xEF → EFI/UEFI System Partition

- GUID [Globally unique identifier]
- GPT [GUID partition table]

1. Initially INTEL came with idea that will remove limitation of 2 TB disk. The proposed model is known as extensible Firmware Interface [EFI]
   EFI idea is accepted by all industries and now is available as UEFI [Unified EFI]

1. GPT is a default partition style for x86 based system
2. It uses EFI to boot the drive.
3. Uses primary and backup partition table to improve integrity.
4. Support upto 128 partitions.





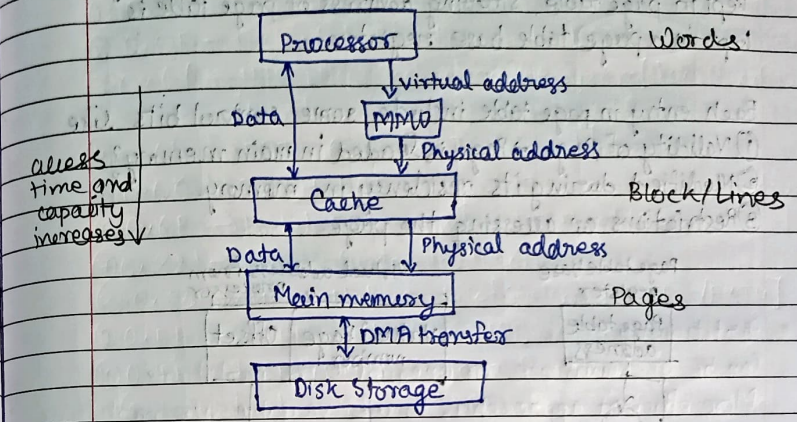
First sector is protected MBR
Next sector is GPT Header
Next few sectors are LBA [Logical Block Addresses]
— " — are actual partition area
At last few sectors are called backup area.

Backup of primary GPT Header is in secondary GPT Header.
LBA 2-33 in upper part is copied as it is in
LBA 2'-33' of lower block near secondary GPT for backup.

At max total 128 partitions and each sector store information about 4 partition that is LBA2-LBA33
So 128/4 = 32 sectors are required.
512/4 = 128 bytes [storage for each partition's information]

After LBA33, LBA34 onwards the actual partition starts.

Virtual Memory. [Demand Paging]



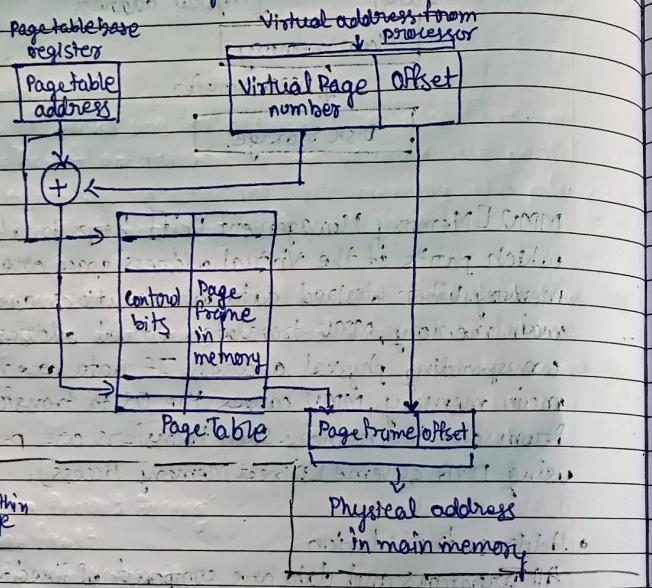
MMU [Memory Management Unit] keeps track of which parts of the virtual address space are in physical memory. When desired data or instructions are in the main memory, MMU translates virtual address into corresponding physical address. If data are not in main memory, MMU causes the OS to transfer data from disk to memory. Such transfers are performed using DMA scheme [Direct Memory Access].

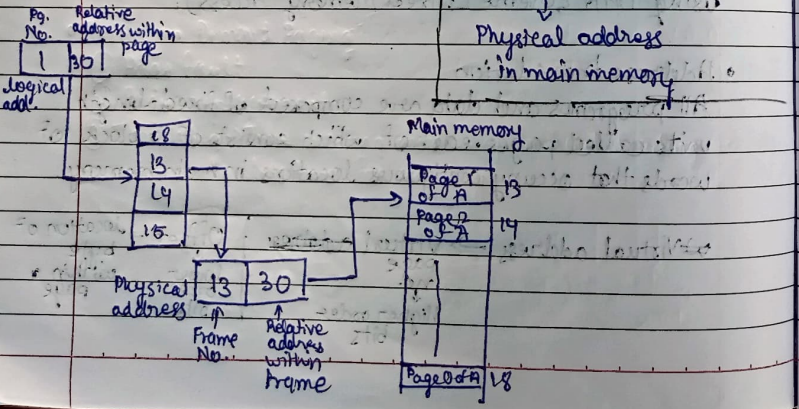
- Address translation
  All programs and data are composed of fixed-length units called pages, each of which consists of a block of words that occupy contiguous locations in main memory.

Virtual address → virtual ~~address~~ page no. → higher order bits | Offset → lower order bits → location of byte within a page

Information about main memory location of each page is kept in page table. Starting address of page table is kept in a page table base register.

Each entry in page table include some control bits like
① Validity of Page → Page is loaded in main memory?
② Modified during its residency in memory.
③ Restrictions on accessing the page.

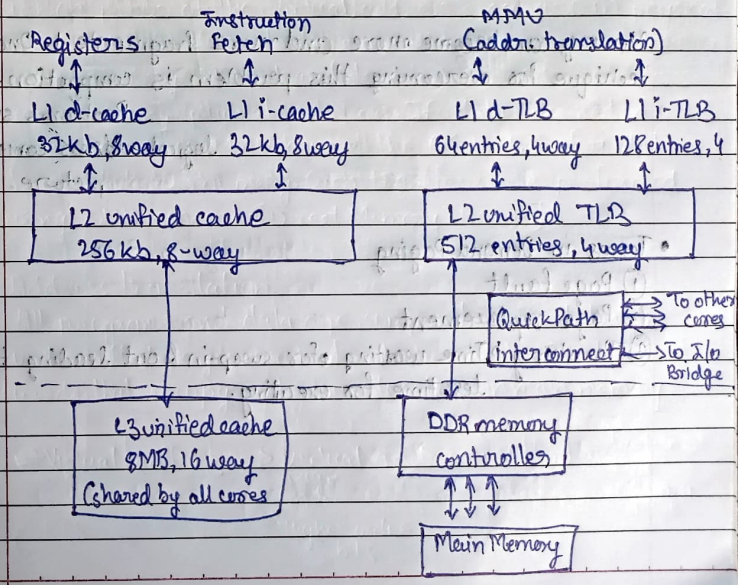




- Translation Lookaside Buffer.

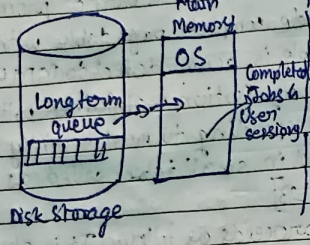
MMU is normally implemented as part of processor chip, it's impossible to include complete page table within MMU. So small portion of table is accomodated to MMU, rest is kept in main memory. Portion maintained within MMU consist of entries corresponding to most recently accessed pages. They are stored in small table called as Translation Lookaside Buffer [TLB]

Address translation proceeds as follows. It is given with a virtual address, MMU looks in TLB for referenced page. If page table entry found in TLB, the physical address is obtained immediately. If there is a miss in TLB, then required entry is obtained from page table in main memory and TLB is updated.

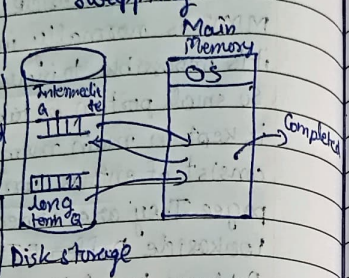


- Page Fault

- Simple Job Scheduling



Swapping



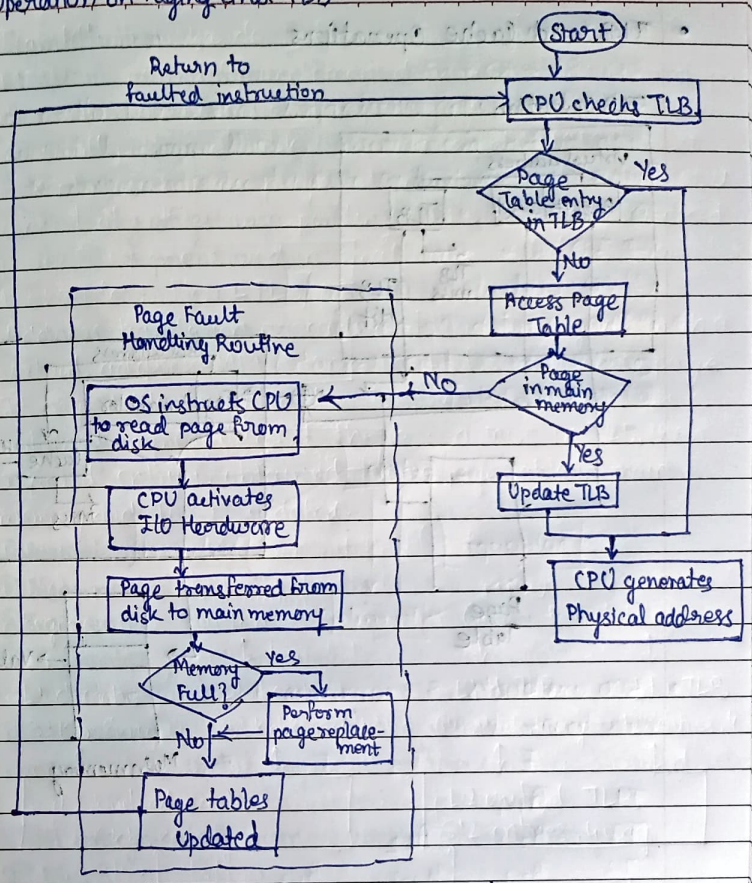
- Partitioning

Can be of equal size or unequal size partitions.
More efficient approach is to use variable-size partitions.
When a process in brought into memory, it is placed in the smallest partition that will hold it.

Memory can become more and more fragmented. One techique for overcoming this problem is compaction.
Problems so logical address came into picture.

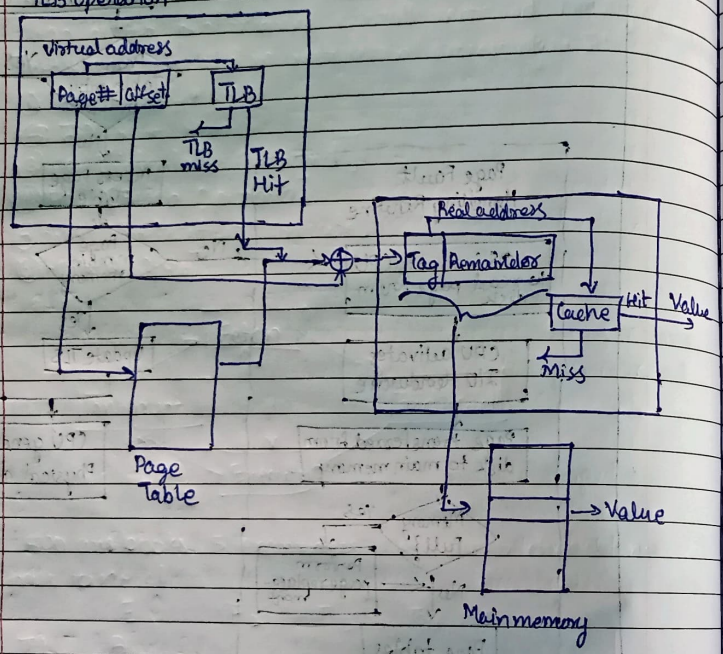
- Terms in Demand Paging : Stalling Pg. 324
  1. Page fault
  2. Page replacement
  3. Thrashing → Time wasting for swap-in & out leading to less time for executing.

- Operation on Paging and TLB



- TLB and Cache Operations

TLB Operation



- Segmentation

It allows programmers to view memory as consisting of multiple address spaces or segments. Segments are of variable size.

Advantages:

1. Handling of growing data structures. OS will expand or shrink segment as needed.
2. Independently altered and recompiled.
3. Sharing among processes
4. Protection.

Intel x86 memory management

x86 includes hardware for both segmentation and paging.

4 distinct views of memory:-

1. Unsegmented unpaged memory
2. Unsegmented paged memory
3. Segmented unpaged memory
4. Segmented paged memory.

When segmentation is used, each virtual address consists of a 16-bit segment reference and 32-bit offset.
2 bits for protection mechanism

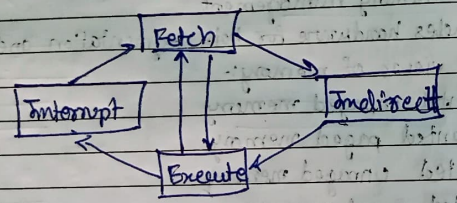
With unsegmented memory, user's virtual memory is $2^{32} = 4GB$.

With segmented memory, virtual memory $= 2^{46} = 64$ TBs.

The Physical address space employs a 32-bit address for a maximum of 4GB.

- Pipelining

Instructions are overlapped one one another to leap better performance overall improved throughput.
The stages involved in instruction processing:
① IF
② ID
③ CO [Calculate Operand]
④ FO [Fetch Operand]
⑤ IE [Instruction Execute]
⑥ WR [Write]



- 500 ans books, 5 examiners. 1 examiner checks one and pass to next

	E_1	E_2	E_3	E_4	E_5
T1	A_{1-1}				
T2	A_{2-1}	A_{1-2}			
T3	A_{3-1}	A_{2-2}	A_{1-3}		
T4	A_{4-1}	A_{3-2}	A_{2-3}	A_{1-4}	
T5	A_{5-1}	A_{4-2}	A_{3-3}	A_{2-4}	A_{1-5}

5 min to ques each.

$T = n \times S$ → avg clock cycle, → clock rate

Without pipeling = $5 \times 5 \times 500 = 12500$

with the pipeling = $T_p = T + (n-1) * \left(\frac{n}{k}\right) = 5 * 5 + 5 * 499$
$= 2520$

Calculate speed up = $\frac{\text{old}}{\text{new}}$
$= \frac{nT}{T + (n-1)\left(\frac{n}{k}\right)}$

	1	2	3	4	5	6	7	8
I1	IF	ID	IE	M	WR			
I2		IF	ID	IE	M	WR		
I3			IF	ID	IE	M	WR	
⋮								

- Dependencies

① Flow dependencies
② Anti dependency (WAR)
③ Output dependence (WAW)
④ I/O dependence
⑤ Unknown dependence

- Data forwarding.

- Pipeline Hazards
① Resource Hazards
② Data Hazards [RAW] [WAR] [WAW]
③ Control Hazards

- RAW — Working on old value
  I1: R1 ← R2 * R3
  I2: R4 ← R1 * R3

- WAR — I2 wrote before due to stall in I1
  I1: $R_1 \leftarrow R_2 * R_3$
  I2: $R_2 \leftarrow R_4 + R_5$

- WAW
  I1: R3 ← R1 * R2
  I2: R3 ← R4 + R5

NUMERICALS

① CPI = T exec × f / No. of instr.
CPU execution time = Instr. count of prgm. × CPI × clock cycle time

a)
Compiler A : Count = 1.0E9 = 1.0×10^9
execution time = 1.1 s

CPI of A = $\frac{1.1}{10^9 \times 10^{-9}}$ = 1.1

Compiler B : Count = 1.2E9 = 1.2×10^9
exec. time = 1.5 s

CPI of B = $\frac{1.5}{1.2 \times 10^9 \times 10^{-9}}$ = 1.25

b)
Exec (A) = Exec (B)
$1.0 \times 10^9 \times 1.1 \times C_1 = 1.2 \times 10^9 \times 1.25 \times C_2$
$\frac{C_1}{C_2} = \frac{15}{11} = 1.37$

c) New instr. = 6.0E8 = 6.0×10^8 Avg CPI = 1.1
$\frac{\text{Exec(A)}}{\text{Exec(New)}} = \frac{1.0 \times 10^9 \times 1.1 \times 10^{-9}}{6.0 \times 10^8 \times 1.1 \times 10^{-9}} = 1.67$

New instr.
$\frac{\text{Exec(B)}}{\text{Exec(New)}} = \frac{1.2 \times 10^9 \times 1.25 \times 10^{-9}}{6.0 \times 10^8 \times 1.1 \times 10^{-9}} = 2.27$

UNIT-2

- Performance.

X is n times faster than Y

Execution Time of Y is n times as long as it is on X.

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution}_Y}{\text{Execution}_X} = n$$

Improve performance → decrease execution time

- $\text{Clock rate} = \frac{1}{\text{Clock period}}$

- CPU Execution = CPU clock cycles × clock period

$$= \frac{\text{CPU clock cycles}}{\text{Clock rate}} = \frac{\text{Instr. in Prgm.} \times \text{avg. clock per instr.}}{\text{clock rate}}.$$

Example Program runs for 10 secs on A 2GHz clock.
We want B to run in 6 secs., for it B requires 1.2 times as many clock rates.

→ $CPU_A = \frac{\text{cycles}}{2\times10^9} \Rightarrow 10 = \frac{\text{clock cycles}}{2\times10^9}$

∴ clock cycles of A = 20×10^9

$CPU_B = \frac{1.2\times20\times10^9}{\text{clock rate}}$

∴ $6 = \frac{1.2\times20\times10^9}{\text{clock rate}} \Rightarrow \text{clock rate} = \frac{1.2\times20\times10^9}{6}$

$= 4\text{ GHz}$

CPI | Page No. | Date / /

CPU clock cycles = Instr. in Prgm × Avg. clock per instr.

Ex-2 Computer A : cycle time = 250 ps
CPI = 2

Computer B : cycle time = 500 ps
CPI = 1.2

CPU clock cycles (A) = I × 2
(B) = I × 1.2

CPU time (A) = CPU clocks × clock cycle time
= I × 2 × 250 ps
= 500 I ps.

— " — (B) = 1.2 × I × 500 ps
= 600 I ps.

CPU Performance A = Exec (B) / Exec (A) = 600 I ps / 500 I ps = 1.2
— " — B

A faster than B.

Ex-3

	A	B	C
CPI	1	2	3

Code seq	Instr. count A	B	C
1	2	1	2
2	4	1	1

→ ~~CPU Time~~ Code seq 1 = clock cycles / CPI = clock cycles / 2+1+2 = clock cycles / 5

= Instr. in program × CPI / 5

= (2×1) + (1×2) + (3×2) / 5 = 2+2+6 / 5 = 2

~~CPU Time~~ Seq 2 = 4+2+3 / 6 = 9 / 6 = 1.5
CPI

Instruction per second = clock rate / CPI

DOMS | Page No. | Date / /

CPU Execution time = Clock cycles / Clock rate = clock cycles × clock time.

clock cycles = CPI × instr. in a program.

CPU Exec = CPI × instr. in prgm / Clock rate.

- Java Prgm runs = 15 sec.
  Eold = 15 sec.
  Enew = 0.6 CPI increases by 1.1

- P1, P2, P3 executing same instructions
  clock rate (P1) = 3 GHz CPI (P1) = 1.5
  P2 ⇒ 2.5 GHz CPI = 1.0
  P3 ⇒ 4.0 GHz CPI = 2.2

→ Instruction per second = clock rate / CPI

P1 : 3×10^9 / 1.5 = 2×10^9 instr. per sec. 1.5 / 3 = 0.5

P2 : 2.5×10^9 / 1.0 = 2.5×10^9 CPU exec = CPI / rate = 0.6

P3 : 4×10^9 / 2.2 = 1.82×10^9 = 0.55

∴ P2 have highest performance.

If program execute in 10sec.

→ CPU execution time = $\frac{\text{CPU clock cycles}}{\text{clock rate}}$

∴ CPU clock cycles = CPU execution × clock rate

P1 clock cycles = $10 \times 3 \times 10^9 = 3 \times 10^{10}$

P2 $= 10 \times 2.5 \times 10^9 = 2.5 \times 10^{10}$

P3 $= 4 \times 10^{10}$

No. of instruction?

~~CPU execution time~~

clock cycles = CPI × instr. in prgm

P1: 3GHz clock rate CPI = 1.5

P2: 2.5GHz, 1.0 P3: 4GHz CPI = 2.2

instructions per second = clockrate / CPI

P1 = $3 \times 10^9 / 1.5 = 2 \times 10^9$

P2 = $2.5 \times 10^9 / 1 = 2.5 \times 10^9$ → Fastest

P3 = $4 \times 10^9 / 2.2 = 1.82 \times 10^9$

Execution = 10 sec. no. of instruction = ?

CPU time = $\frac{\text{clock cycles}}{\text{clockrate}}$ ∴ Clock cycles = CPU × clockrate

P1 = $10 \times 3 \times 10^9 = 3 \times 10^{10}$ | P2 = 2.5×10^{10} | P3 = 4×10^{10}

Instr. ⇒ Clock cycles = CPI × No. of inst.

∴ No. of instru. = Clock cycle / CPI

P1 = $3 \times 10^9 \times 10 / 1.5 = 2 \times 10^{10}$

P2 = 2.5×10^{10}

P3 = 1.82×10^{10}

① CPU execution time = $\frac{\text{clock cycles}}{\text{clock rate}}$ = $\frac{\text{clock cycles}}{1/\text{clock period}}$

② clock cycles = CPI × no. of instruction.

③ Instruction per second = $\frac{\text{clock rate}}{\text{CPI}}$

CPU Exec. Time = $\frac{\text{No. of instr.} \times \text{CPI}}{\text{Clock rate}}$

Exec time × 0.7 = $\frac{N \times \text{CPI} \times 0.2}{\text{new clock rate}}$

new clock rate = $\frac{N \times \text{CPI} \times 1.2}{0.7 \times \text{Exec Time}}$ = $\frac{\text{clock cycles} \times 1.2}{0.7 \times \text{Exec. Time}}$

$= \frac{1.2}{0.7} \times$ clockrate

$= 1.71 \times$ clockrate

• MIPS = $\frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$ = $\frac{\text{Instr. count}}{\left(\frac{\text{clock cycles}}{\text{clockrate}}\right) \times 10^6}$

$= \frac{\text{Instr. count}}{\left(\frac{\text{CPI} \times \text{instr. count}}{\text{clockrate}}\right) \times 10^6}$

$= \frac{\text{clockrate}}{\text{CPI} \times 10^6}$

Q) 2×10^6 instructions, 400 MHz
4 ports and each seperate CPI and it %

I	CPI	%
A	1	60
B	2	18
C	4	12
D	8	10

$$\text{Avg CPI} = \frac{1\times60 + 2\times18 + 4\times12 + 8\times10}{100} = 2.24$$

$$\text{MIPS} = \frac{\text{Instr. count}}{\text{Exec time}\times10^6} = \frac{N}{\left(\frac{\text{clockcycles}}{\text{clockrate}}\right)\times10^6}$$

$$= \frac{N}{\frac{N\times CPI\times10^6}{\text{clockrate}}} = \frac{\text{clockrate}}{CPI\times10^6}$$

$$= \frac{400\times10^6}{2.24\times10^6} = 178.57$$

Q) P1 ⇒ 3GHz 1.5 CPI
P2 ⇒ 2.5 GHz 1.0 CPI
P3 ⇒ 4.0 GHz 2.2 CPI
• Execution time = 10
Find no of cycles, no. of instruction.
→ E time = $\frac{\text{clock cycles}}{\text{clockrate}}$
⇒ Clock cycles = E time × rate
P1 = 10×3 GHz = 3×10^{10} Hz
P2 = 2.5×10^{10}
P3 = 4×10^{10}

Execution time = 10
Clock cycles = CPI × no. of instructions
∴ No. of instructions = $\frac{\text{Clock cycles}}{CPI}$

$$P1 = \frac{3\times10^{10}}{1.5} = 2\times10^{10}$$

$$P2 = 2.5\times10^{10} \qquad P3 = \frac{4\times10^{10}}{2.2} = 1.82\times10^{10}$$

- Big-Endian and little-Endian Assignment

Little-Endian Assignment.
[LSB] placed first LSB → MSB
Big-Endian
[MSB] placed first MSB → LSB

- RISC and CISC instruction set.
RISC → Reduced instruction set computers
CISC → Complex

- RISC
① Each instruction should occupy exactly one word in memory.
② Pipelined manner processing
③ Activities are overlapped and total execution time is reduced.
④ Reduces complexity

- CISC
① Makes a more complex instruction that may span more than 1 word.
② More complicated.
③ Used prior to RISC approach.

Date / /

- Load dest, src.
  Load processor reg, memory loc

① C=A+B
→ Load R_2, A
Load R_3, B
ADD R_4, R_2, R_3
~~Load~~ Store R_4, C

Program counter - Holds address of next instruction.

② branch_if_[R2]>0

③ Load operation on memory operands, to load number into register we use
MOV R1, #NUM

Program to add n numbers

```
        Load    R2,N
        Clear   R3          IDK, what this is?
        MOVE    R4,#NUM1
LOOP    LOAD    R5,(R4)
        ADD     R3,R3,R5
        ADD     R4,R4,#4
        SUBTRACT R2,R2,#1
        Branch_if_[R2]>0 loop
        store   R3,sum
```

Page No.
Date / /

- CISC instruction set
  C ← [A] + [B]
  ⇒ Move C,B
  Add C,A

⇒
```
        MOVE    R2,N
        CLEAR   R3
        MOVE    R4,NUM#1
LOOP:   ADD     R3,(R4)+
        SUB     R2,#1
        Branch>0 loop
        MOVE    SUM,R3
```

- Z=K+B*C

③ set instruction
MUL B,B,C
ADD B,B,K

② — " — MUL B,C
ADD K,B
MOV Z,K

① LD C
MUL B
ADD K
ST Z

⓪
PUSH K
PUSH B
PUSH C
MUL
ADD
POP Z

C
B
K

Program Status Words.
① Sign
② Zero
③ Carry
④ Equal
⑤ Overflow
⑥ Interrupt
⑦ Supervisor

11 B=
12 C=1100

64kb main memory
2-way set association, 256 sets and block size = 16 bytes

*6	*8	4

18

$64kb = 2^8 \times 2^{10} = 2^{18}$

$256/2 = 128 = 2^7$

0C50 = to 2C5FH = 064

Parallel Processing

1. SISD [Single Instruction stream → Single Data stream]
2. SIMD [— " — Mutiple — " —]
3. MIMD [Mutiple — " — Mutiple — " —]
4. MISD [Mutiple — " — Single — " —]

Instruction stream → Memory
Data stream → Processor

- RAID [Redundant Array of Independent Disk]

RAID 0 - Stripping

A B C D E F G H

Advantage = Speed.

Disk1	Disk2
A	B
C	D
E	F
G	H

RAID 1 → Full tolerant. Have copy.

A B C D

Disk1	Disk2
A	A
B	B
C	C
D	D

RAID 2 [Redundancy Through Hamming Code]
Disks are synchronized
Very small strip offer single byte/word.
Error correction is calculated across corresponding bit on disks.
Multiple parity disk store hamming code corrections in corresponding to lots of redundancy.
Expensive, not commercially available.

RAID 3 [~~Block level~~ Bit interleaving Parity]
Usage of 1 parity bit for strips. So 1st disk ~~is from~~ 4 disk its parity is calculated and parity bits is kept at 5th Hard disk.

RAID 4 [Block level Parity]
1) Strip size is very large.
2) First parity for each block in same index in each HD is calculated seperately
3) After this we have 4 parity bits which are saved in 5th HD with index as a strip/block.

RAID 5 [Block level distributed Parity]
1) Like RAID4, parity splits across all disks
2) Round Robin Allocation for parity strip.
3) Avoid RAID4 bottleneck at parity disk, commonly used in network services.

RAID 6
1) Two parity calculations
2) Stores seperate blocks on different disks.
3) User requirement are n+2 disks.
4) High data availability.

RAID 5

A B C D E F G H I J K L

A	B	C	Parity
D	E	Parity	F
G	Parity	H	J
Parity	J	K	L

RAID 10 → 1+0

RAID 6

A B C D E F G H I J K L

A	B	Parity	Parity
C	Parity	Parity	D
Parity	Parity	E	F
Parity	G	H	I
J	K	L	Parity

Date / /

HDD Numericals

① Time to read = seek time + rotational delay + data transfer time

② seek time = time to travel move disk arm to required track.

③ rotational delay = revolution time $\times \frac{1}{2}$

④ Data transfer = $\frac{1}{r} \times \frac{b}{n}$

r = revolution in 1 sec n = length of track
b = bytes to be transferred.

Q1) 2000 sectors with 500 sector per track
512 bytes per track, rotational speed = 15000 rpm
avg. seek time = 4 msec

Rotational speed = $\frac{15000 \text{ rpm}}{60}$ = 250 rps

time = seek time + rotational delay + dat

$= 4ms + \frac{60 \times 4}{15000} + \left(\frac{4 \times 2000}{500}\right)$

$= 4 + 4 \times 0.5 + 4 \times 4ms$

$= 4 + 2 + 16$

$= 22ms$

Date / /

Q2) 8 surfaces, 512 tracks on each
64 sector per track, each of size = 1kb
access time = 1.5
3600 rpm.
Successive cylinders can be read without head movement.

→ Each cylinder = 8 tracks. each of 64kb as 64 sector in each of size 1kb.

Total memory = 8 × 64 = 512KB
To store 5MB, we will need 10 cylinders.

seek time = 8 rotational delay = $\frac{60}{2 \times 3600}$ = 8.3 ms

Time taken to read cylinder

Access time = $8ms + 8.33ms + 8 \times \frac{60}{3600}$

$+ 9 \times (1.5 \times 8.33 + 8.33ms)$

$= 16.33 + 133.33 + 9 \times 145.825$

$= 16.33 + 133.33 + 1312.425$

$= 1462.085$

Burst rate = revolution per sec × sector per rev. × bytes per sector

Data transfer date = no. of rotation per second * track capacity * no. of surfaces.