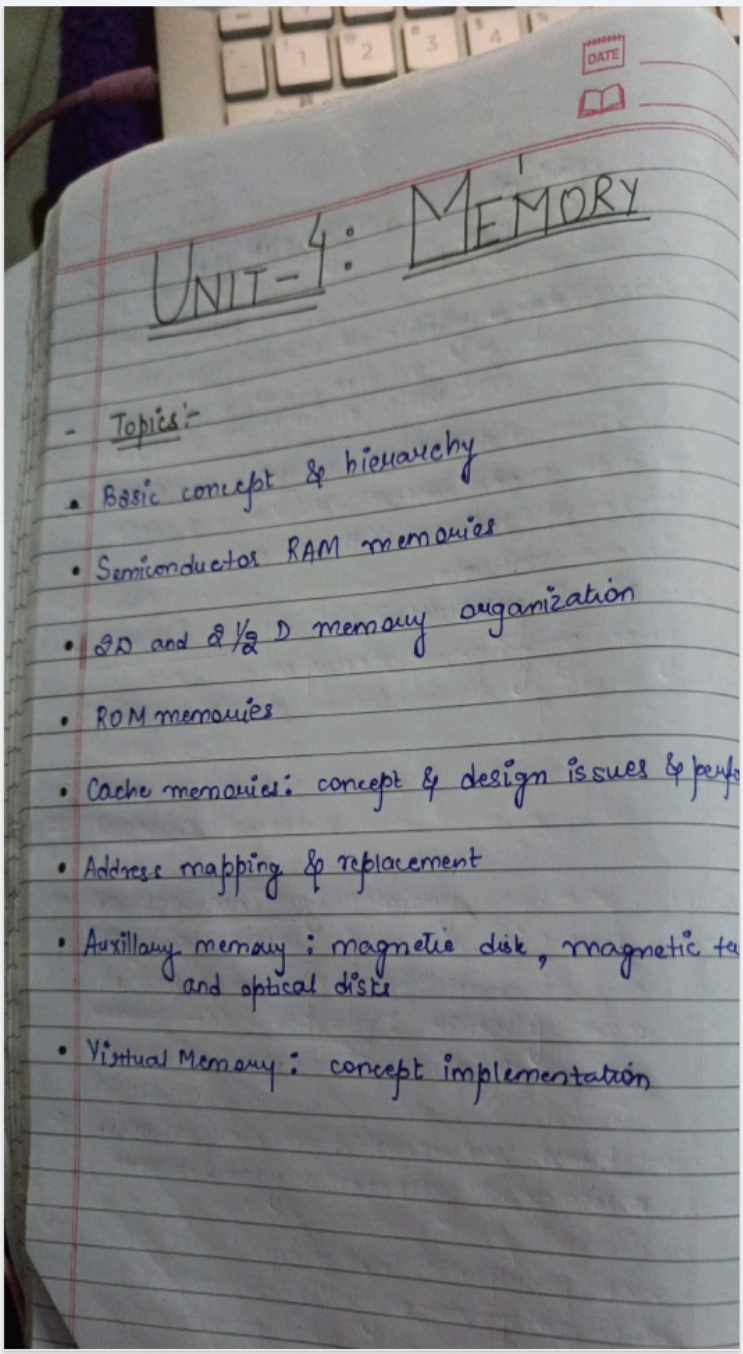
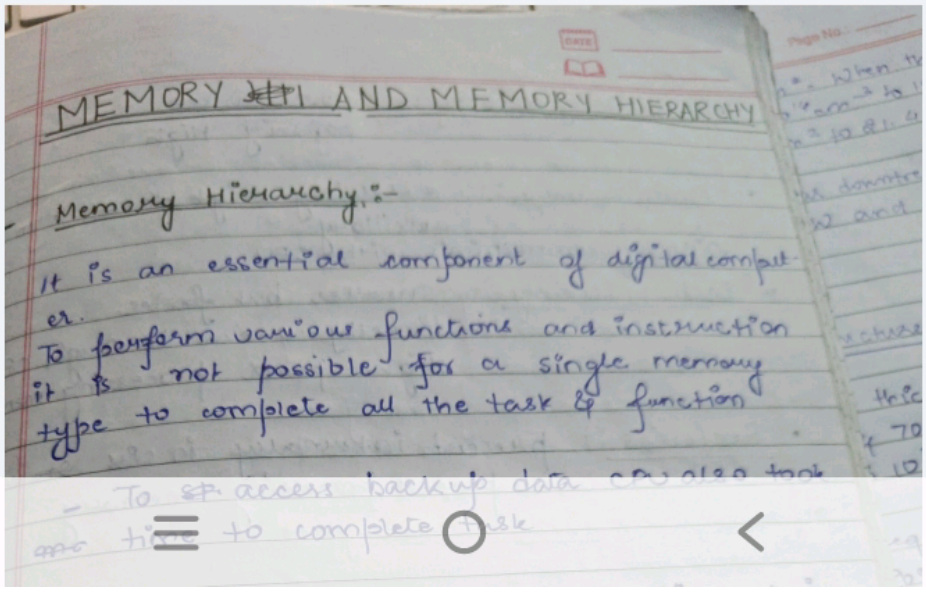


UNIT-4: MEMORY

- Topics:-
- Basic concept & hierarchy
- Semiconductor RAM memories
- 2D and 2½D memory organization
- ROM memories
- Cache memories: concept & design issues & perfo
- Address mapping & replacement
- Auxillary memory: magnetic disk, magnetic te and optical disks
- Virtual Memory: concept implementation



MEMORY ~~HI~~ AND MEMORY HIERARCHY

Memory Hierarchy:-

It is an essential component of digital comput er.
To perform various functions and instruction it is not possible for a single memory type to complete all the task & function

- Address mapping & replacement
- Auxillary memory : magnetic disk, magnetic ta and optical disk
- Virtual Memory : concept implementat

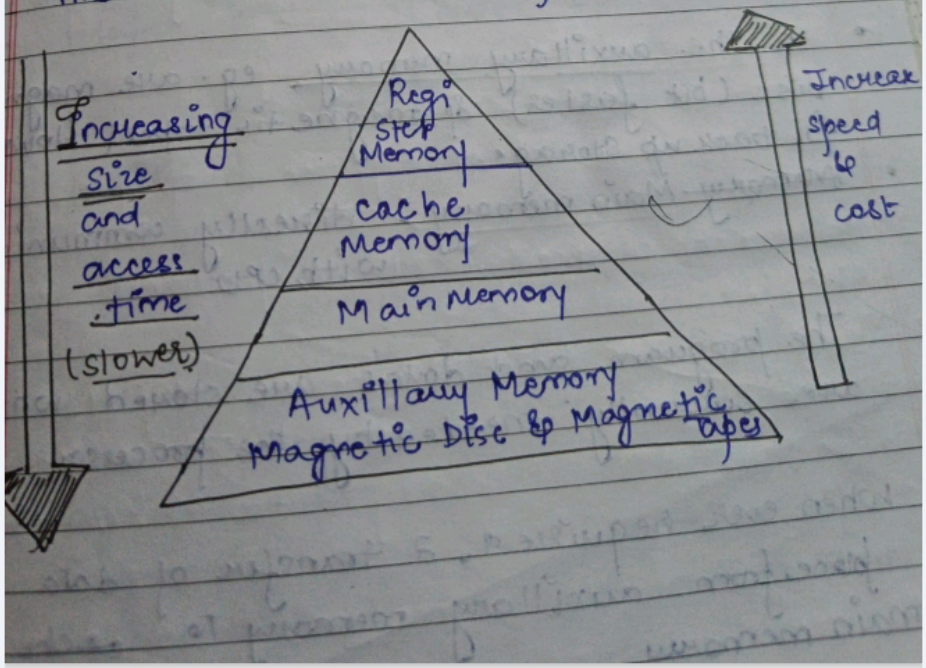
MEMORY ~~HI~~ AND MEMORY HIERARCHY

Memory Hierarchy :-

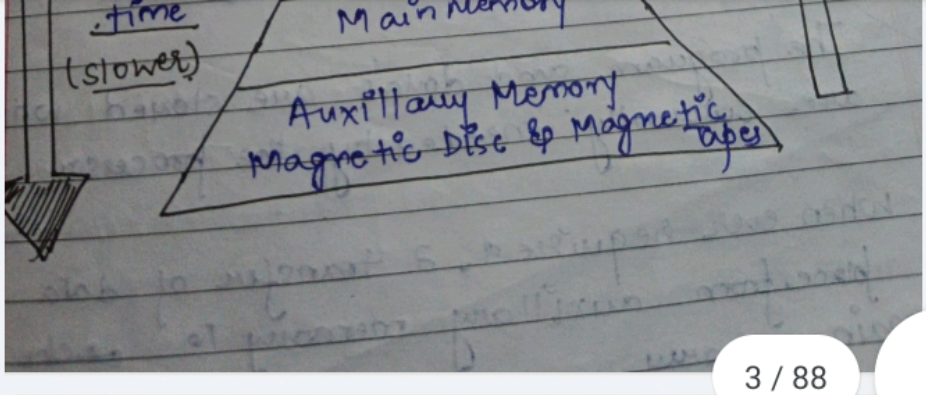
It is an essential component of digital comput er.
To perform various functions and instruction it is not possible for a single memory type to complete all the task & function

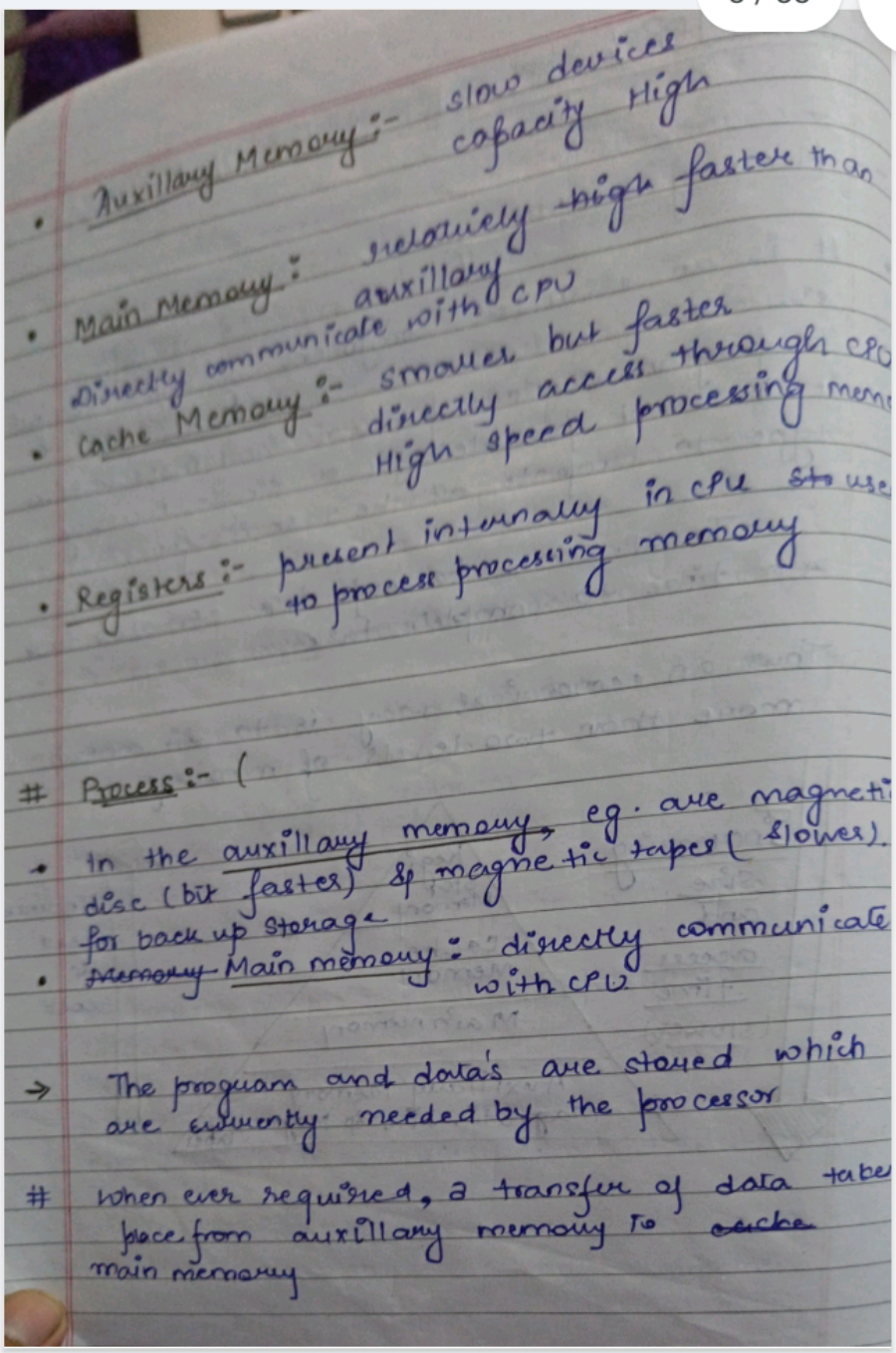
- To ~~sp.~~ access backup data CPU also took ~~and~~ time to complete task

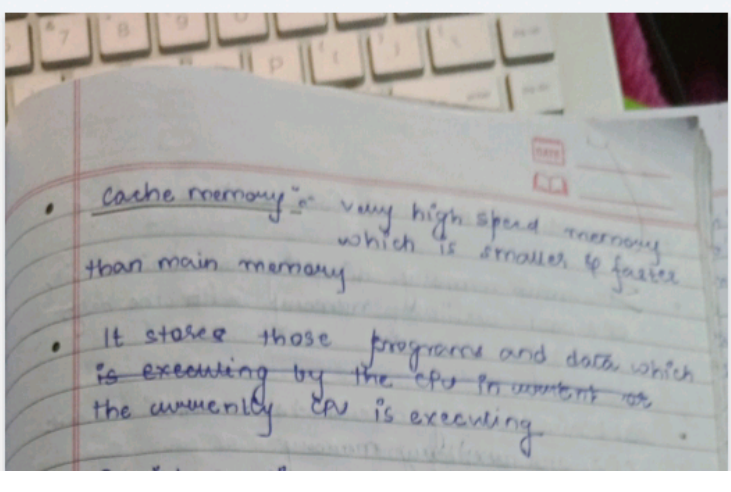
Thus an economical way is to ~~di~~ contain more than two levels of memory.



- Auxillary Memory :- slow devices
  capacity High
  relatively high faster than


time
(slower)
Main Memory
Auxiliary Memory
Magnetic Disc & Magnetic tapes


Auxillary Memory :- slow devices
capacity High
Main Memory :- relatively high faster than auxillary
Directly communicate with CPU
Cache Memory :- smaller but faster
directly access through CPU
High speed processing
Registers :- present internally in CPU
to process processing memory
Process :-
In the auxillary memory, eg. are magnetic disc (bit faster) & magnetic tapes (slower).
for back up storage
Main memory : directly communicate with CPU
→ The program and data's are stored which are currently needed by the processor
When ever required, a transfer of data take place from auxillary memory to main memory

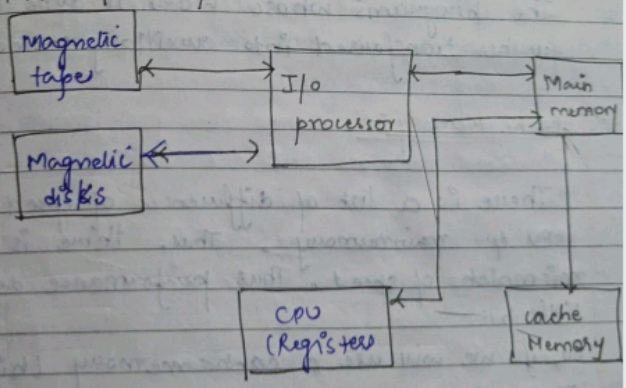

Cache memory :- very high speed memory which is smaller & faster than main memory
It stores those programs and data which is executing by the CPU
the currently CPU is executing

→ The program and data's are stored which are currently needed by the processor

When ever required, a transfer of data takes place from auxillary memory to ~~cache~~ main memory

- Cache memory :- very high speed memory which is smaller & faster than main memory
- It stores those programs and data which ~~is executing by the cpu in current~~ or the currently CPU is executing.
- Registers directly stored in CPU and consists of processing elements at the time of execution of program

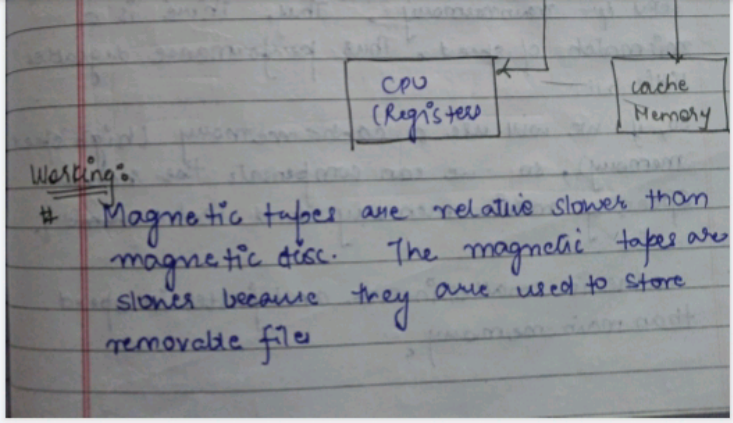
MEMORY HIERARCHY IN COMPUTER SYSTEM :-



Warning:

Magnetic tapes are relative slower than magnetic disc. The magnetic tapes are slower because they are used to store removable files

- Magnetic disks used for backup storage
- The main memory directly communicate with CPU and auxillary memory with the help of I/o processor.
- # Main memory needs to communicate with CPU and auxillary memory



Working:

Magnetic tapes are relative slower than magnetic disc. The magnetic tapes are slower because they are used to store removable file

- Magnetic disk used for backup storage
- The main memory directly communicate with CPU and auxillary memory with the help of I/O processor.
- # Main memory needs to communicate with CPU and auxillary memory
- CPU access program from main memory. If suppose the program (data) ~~for~~ is not present in main memory, then the program is transfered from auxillary carry to main mem

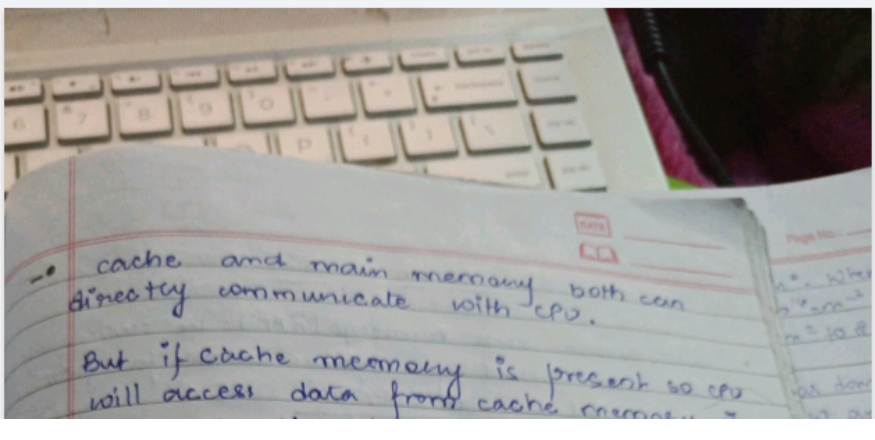
The programs whose work is done is now further transferred into auxillary memory

Cache Memory :-

There is a lot of diffurence of speed betwe CPU & mainmemory. Thus, there is a mismatch of speed. Thus performance degrade little bit

So, if we will use a cache memory (high spe memory), so we can compensate the between speed of main memory and cache memory.

- cache is smaller in size and faster in speed than main memory.



- cache and main memory both can directly communicate with CPU.

But if cache memory is present so CPU will access data from cache

• ~~Memory~~ Main memory with CPU

→ The program and data's are stored which are currently needed by the processor

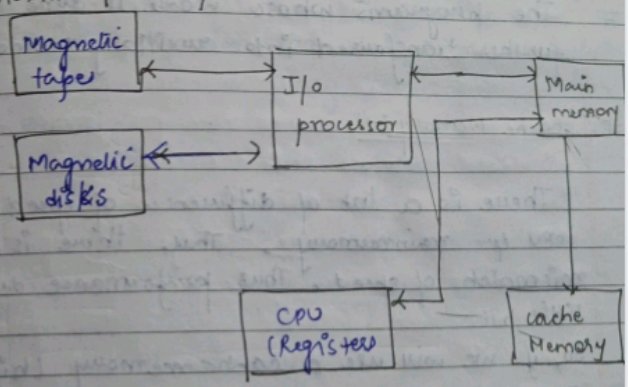
When ever required, a transfer of data takes place from auxillary memory to ~~cache~~ main memory

• Cache memory :- very high speed memory which is smaller & faster than main memory

• It stores those programs and data which ~~is executing by the CPU in current~~ the currently CPU is executing

• Registers directly stored in CPU and consists of processing elements at the time of execution of program

MEMORY HIERARCHY IN COMPUTER SYSTEM :-



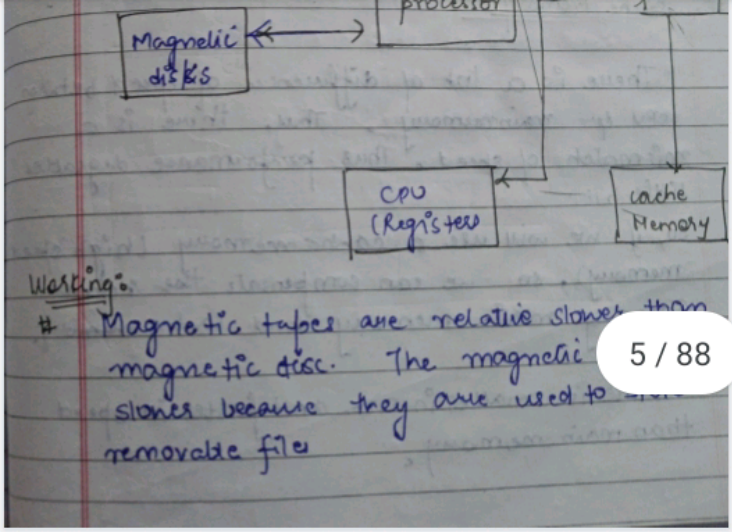
Working:

Magnetic tapes are relative slower than magnetic disc. The magnetic tapes are slower because they are used to store removable file

• Magnetic disks used for backup storage

• The main memory directly communicates with CPU and auxillary memory with the help of I/O processor

• # Main memory needs to communicate with



Working:

Magnetic tapes are relative slower than magnetic disc. The magnetic disc is faster than magnetic tapes because they are used to store removable file

- Magnetic disks used for backup storage
- The main memory directly communicate with CPU and auxillary memory with the help of I/O processor
- # Main memory needs to communicate with CPU and auxillary memory
- CPU access program from main memory. If suppose the program (data) is not present in main memory, then the program is transfered from auxillary carry to main memory

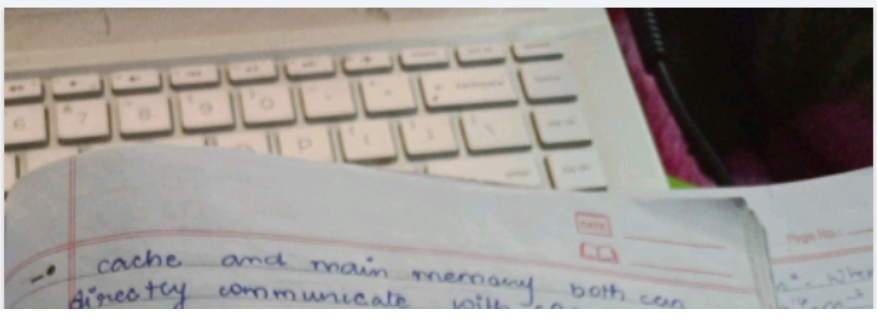
The programs whose work is done is now further transferred into auxillary memory

Cache Memory :-

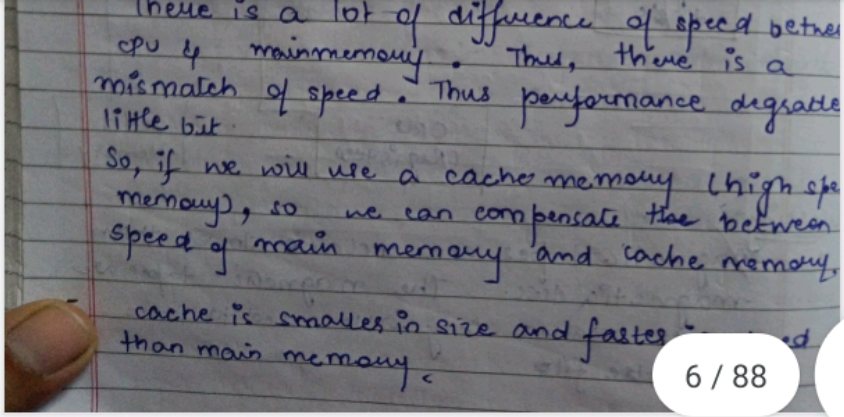
There is a lot of diffuence of speed between CPU & mainmemory. Thus, there is a mismatch of speed. Thus performance degrades little bit.

So, if we will use a cache memory (high speed memory), so we can compensate the between speed of main memory and cache memory.

- cache is smaller in size and faster in speed than main memory.

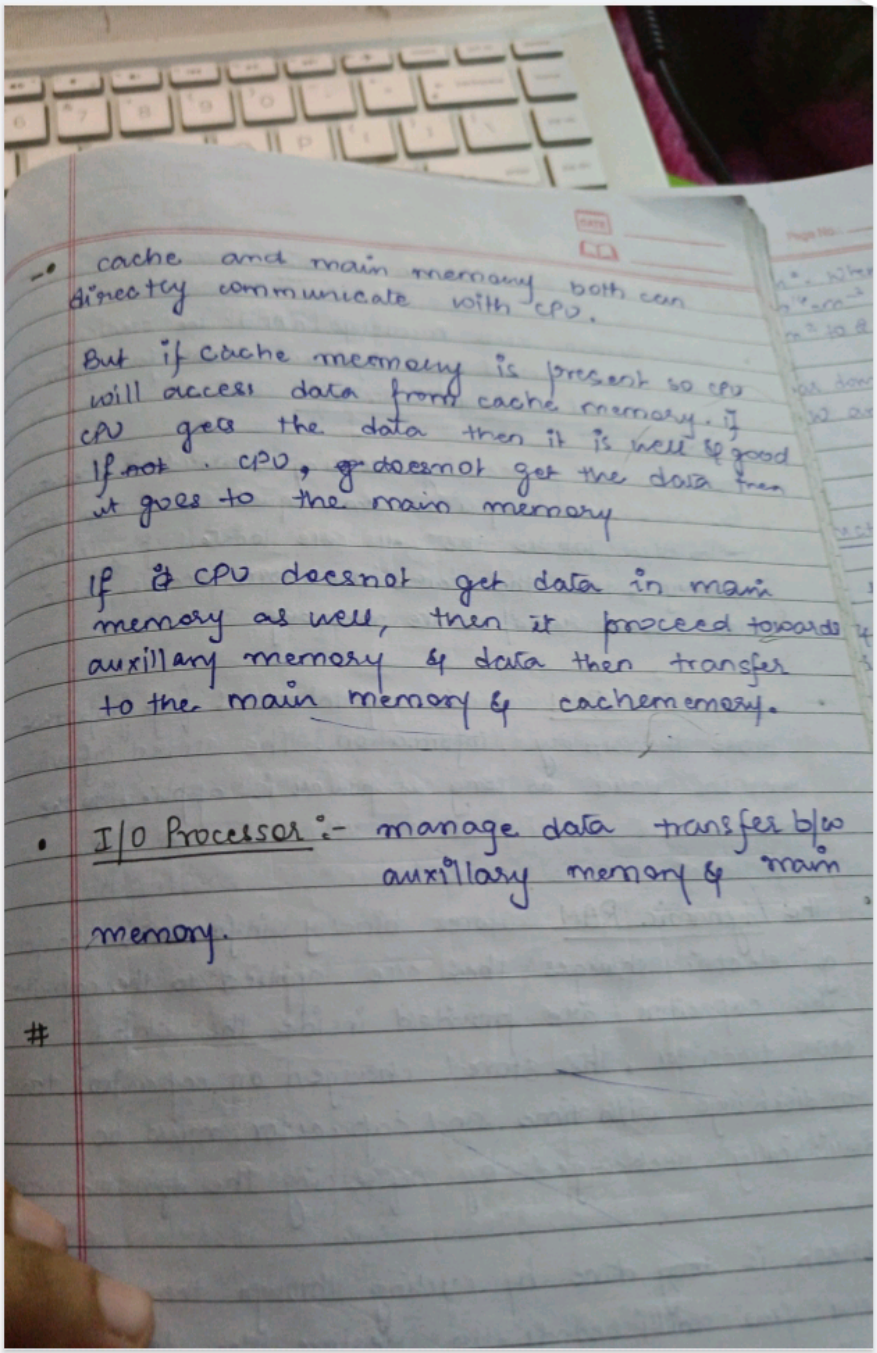


- cache and main memory both can directly communicate with


There is a lot of difference of speed betwe
CPU & mainmemory. Thus, there is a
mismatch of speed. Thus performance degrade
little bit.

So, if we will use a cache memory (high spe
memory), so we can compensate the between
speed of main memory and cache memory.

cache is smaller in size and faster
than main memory.

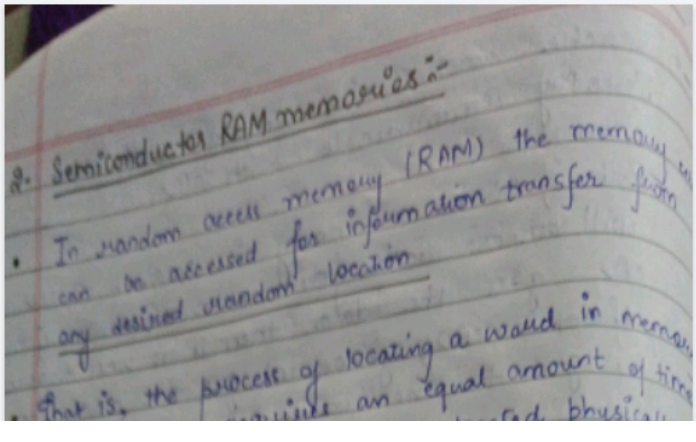

- cache and main memory both can directly communicate with CPU.

But if cache memory is present so CPU will access data from cache memory. If CPU gets the data then it is well & good If not, CPU, doesnot get the data from it goes to the main memory

If CPU doesnot get data in main memory as well, then it proceed towards auxillary memory & data then transfer to the main memory & cachememory.

- I/O Processor :- manage data transfer b/w auxillary memory & main memory.

#


2. Semiconductor RAM memories :-

- In random access memory (RAM) the memory can be accessed for information transfer from any desired random location.
- That is, the process of locating a word in memory

#

2. Semiconductor RAM memories :-

- In random access memory (RAM) the memory can be accessed for information transfer from any desired random location.
- That is, the process of locating a word in memory is same and requires an equal amount of time no matter where the cells are located physically in memory. Thus the name random access. It is also called read/write memory.
- The Static RAM consists of internal flip flop that store the binary information. The stored information remains valid as long as power is applied to the unit.
- The dynamic RAM stores binary information in form of electric charges that are applied to the capacitor. The capacitors are provided inside the chip by MOS transistor. The stored charged on capacitors tend to discharge with time and capacitor must be periodically recharged by refreshing the dynamic memory.
- Refresh is done by cycling through the words every few milliseconds to restore the decaying charge.
- The DRAM offer reduced power consumption & larger storage capacity in a single memory.

- One major application of SRAM is in implementing the cache memories. The SRAM are used for implementing main memory.

A RAM chip is better suited for communication with CPU. If it has one or more control inputs that select the chip only when needed.

- It also have bidirectional data bus that

The dynamic RAM stores [illegible] of electric charges that are applied to the capacitor. The capacitors are provided inside the chip by MOS transistor. The stored charged on capacitors tend to discharge with time and capacitor must be periodically recharged by refreshing the dynamic memory.

Refresh is done by cycling through the words every few milliseconds to restore the decaying charge.

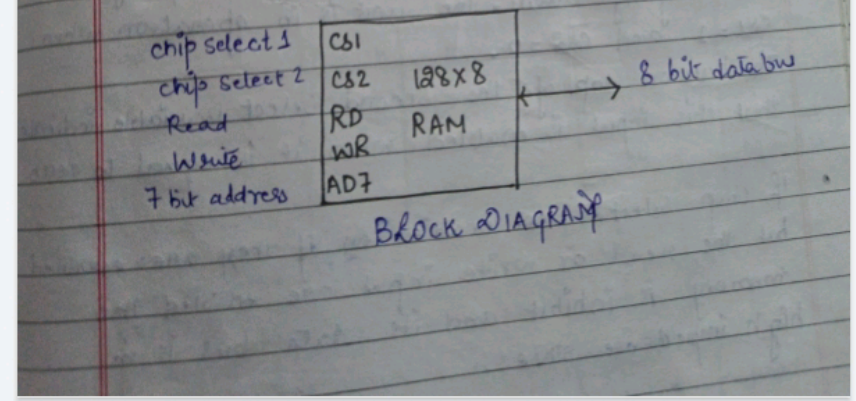
- The DRAM offer reduced power consumption & larger storage capacity in a single memory.

- One major application of SRAM is in implementing the cache memories. The DRAM are used for implementing main memory.

A RAM chip is better suited for communication with CPU. If it has one or more control inputs that select the chip only when needed.

- It also have bidirectional data bus that allows to transfer data either from memory to CPU during read operation or CPU to memory during write operation.

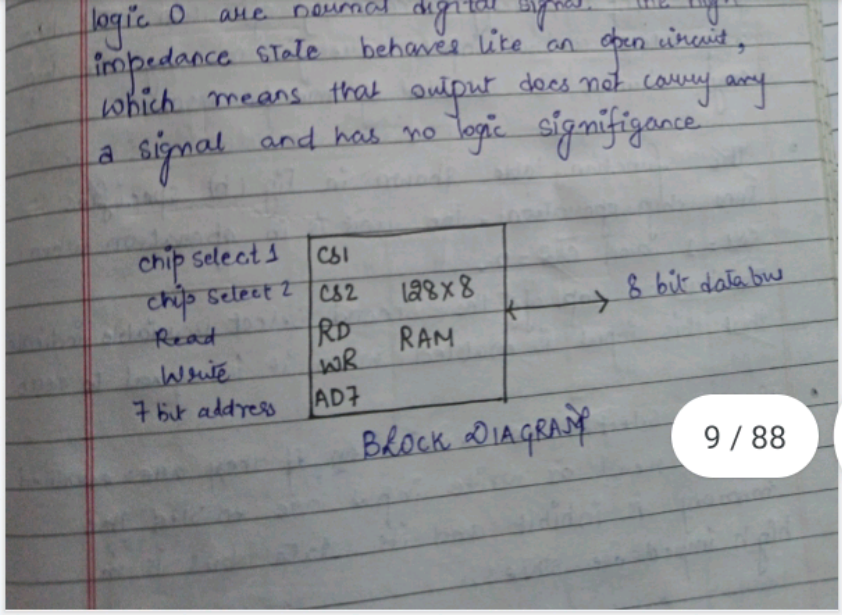
- A bidirectional bus can be constructed with three state bus buffer. In three state buffer, logic 1 & logic 0 are normal digital signal. The high impedance state behaves like an open circuit, which means that output does not convey any a signal and has no logic significance.



BLOCK DIAGRAM

CS1	$\overline{CS2}$	RD	WR	Memory function	Status of Data Bus
0	0	X	X	Inhibit	High impedance
0	1	X	X	Inhibit	High impedance
1	0	0	0	Inhibit	High impedance
1	0	0	1	Write	Input

logic 0 are normal digital signal. The high impedance state behaves like an open circuit, which means that output does not convey any a signal and has no logic significance



BLOCK DIAGRAM

CS1	$\overline{CS2}$	RD	WR	Memory function	State of Data bus
0	0	X	X	Inhibit	High impedance
0	1	X	X	Inhibit	High impedance
1	0	0	0	Inhibit	High impedance
1	0	0	1	Write	Input data to RAM
1	0	1	X	Read	Output data from RAM
1	1	X	X	Inhibit	High impedance

(b) Function Table

- In the block diagram of RAM chip, 128 words of eight bit (one byte) per word. This require 7 bit address and 8 bit bidirectional data bus. The read & write input specify the memory operation and two chip select control inputs are for enabling the chip only when it is selected by microprocessor

The function table shown in fig (b) specifies RAM chip operation. The unit is in operation when $CS1 = 1$ and $\overline{CS2} = 0$.

The bar on top of the second select variable indicate that this input is enabled when it is equal to zero.

If chip select is not enabled or if they are enabled but the read or write input are enabled the memory is inhibit and its data bus is in high impedance state.

- When $CS1 = 1$ and $CS2 = 0$, the memory can be placed in write or read mode. When WR input is enabled, the memory stores a byte from data bus into location specified by address input lines.

The function table shown in fig (b) specifies RAM chip operation. The unit is in operation when $CS1 = 1$ and $\overline{CS2} = 0$.

The bar on top of the second select variable indicate that this input is enabled when it is equal to zero.

If chip select is not enabled or if they are enabled but the read or write input are enabled the memory is inhibit and its data bus is in high impedance state.

10 / 88

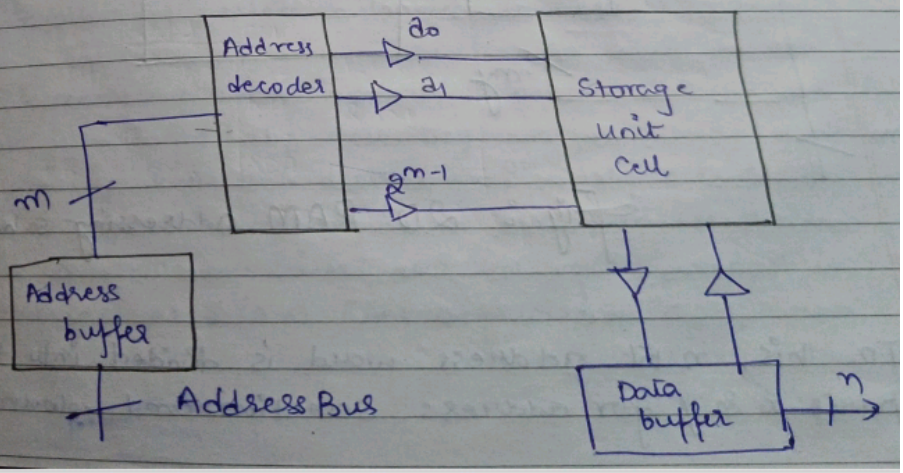
- When $CS1 = 1$ and $CS2 = 0$, the memory can be placed in write or read mode. When WR input is enabled, the memory stores a byte from data bus into location specified by address input lines.
- When RD input is enabled, the content of selected byte is placed in data bus

RAM ORGANIZATION :-

RAM memory can be organized by following three methods

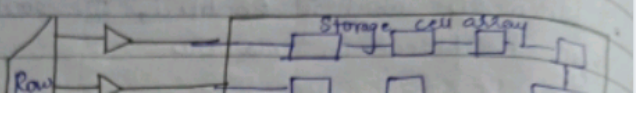
(1) One dimension RAM
(2) Two dimensional RAM
(3) 2½ dimensional RAM

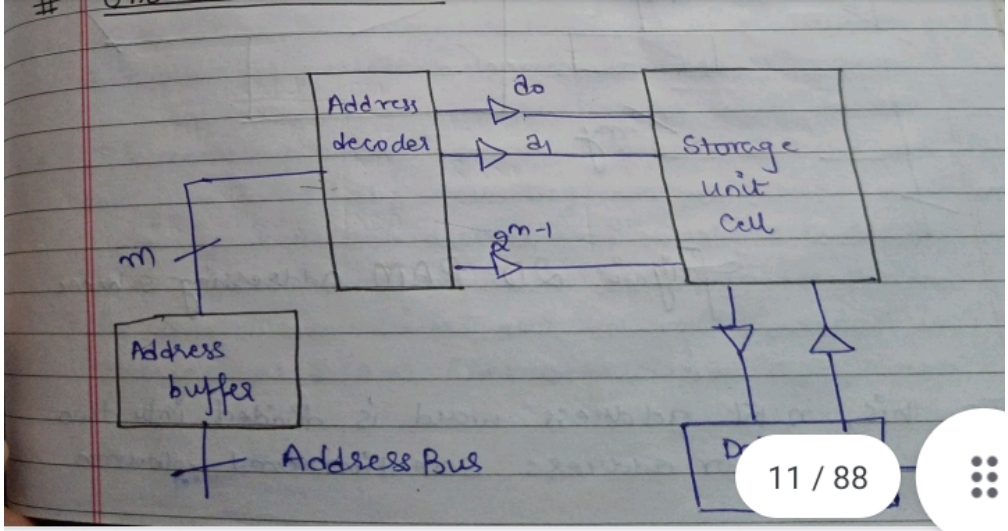
One Dimension RAM :-



It has a large number of addressable storage location (2^m), each address of which stores bit words. Now, the capacity of RAM is specified $2^m \times n$ (address line × number of bit per address)

(2) Two dimensional RAM :- 2D RAM (2D RAM) :





It has a large number of addressable storage location (2^m), each address of which stores bit words. Now, the capacity of RAM is specified $2^m \times n$ (address line × number of bit per address)

(2) Two dimensional RAM:- 2D RAM (2D RAM):

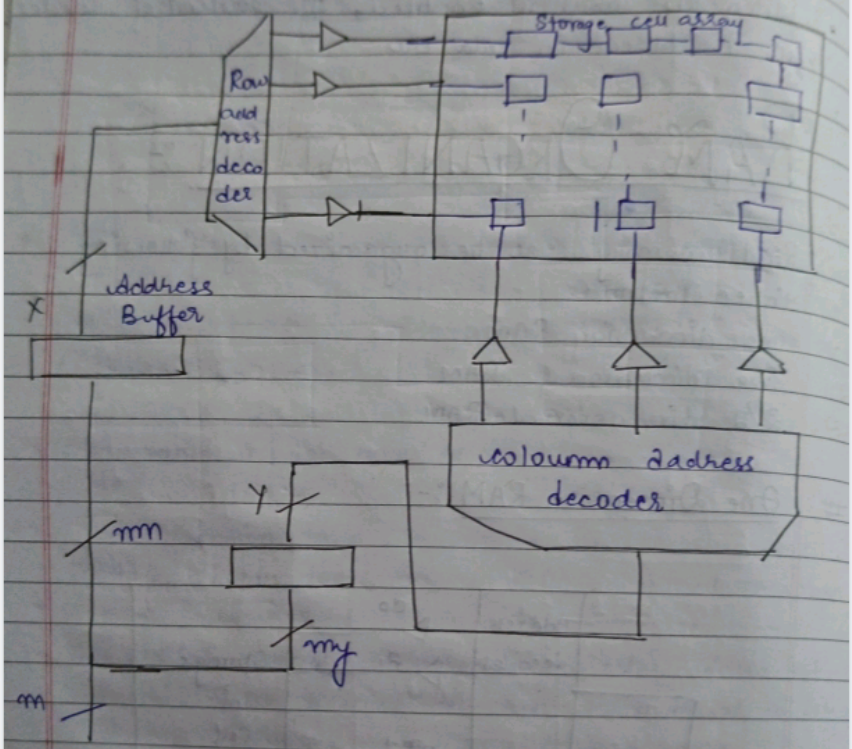
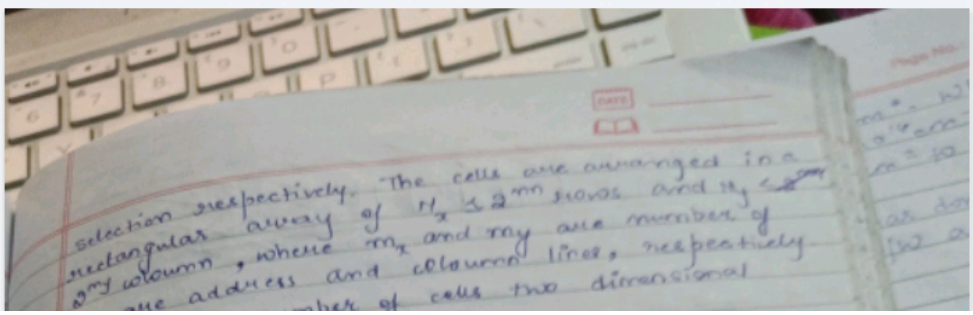


fig: 2D RAM addressing scheme

In this, n bit address word is divided into two parts x & y for address selection and colourm



selection respectively. The cells are arranged in a rectangular array of $m_x \times 2^{my}$ rows and 2^{my} colourm, where m_x and my are number of ...ue address and colourm lines, respectively ...mber of cells two dimensional

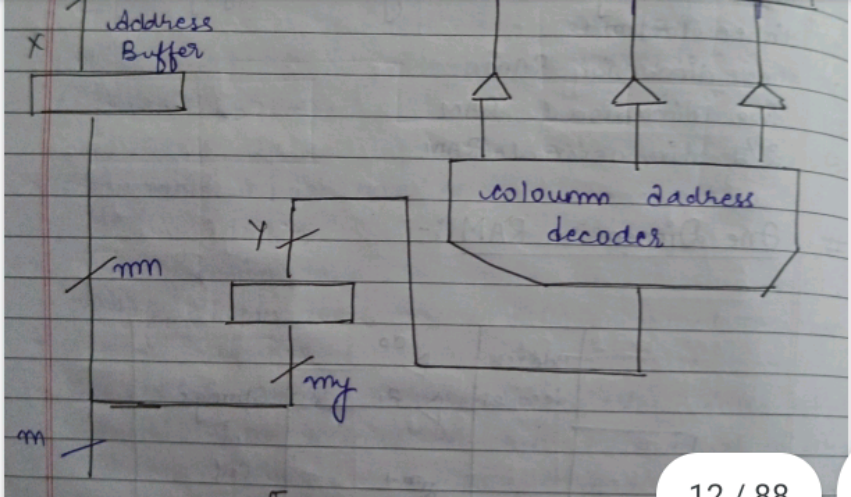


fig: 2D RAM address

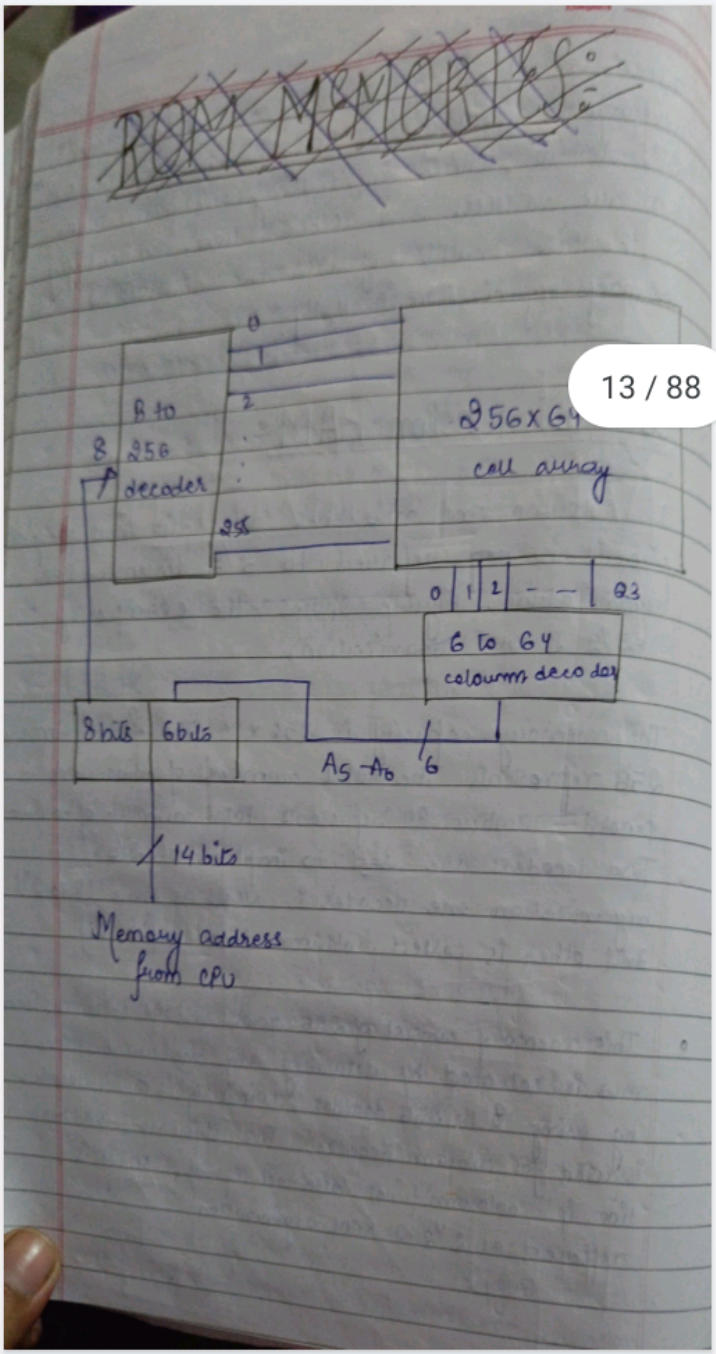
In this, n bit address word is divided into two parts x & y for address selection and colourm selection respectively. The cells are arranged in a rectangular array of $M_x = 2^{mn}$ rows and $M_y = 2^{my}$ colourm, where m_x and m_y are number of bit are address and colourm lines, respectively. So, the total number of cells two dimensional memory organization in $M_x N_y$.

(3) 2½ Dimension RAM:

If colourm and row lines are split into unequal size it is referred to 2½ dimenson RAM organization. Figure shows the example of 2½ D RAM organization.

The memory capacity is 256 x 64. The first number 256 represents the total number of rows and second number 64 represents total number of colourm. Two decoders are used to implement 2½D organization one decoder is called as row. (8 to 2^8) and other is called colourm decoder (6×2^6)

This memory consist of 256 rows and 64 colourm. Each row is selected by activating any one row at a time by using 8 to 256 decoder. Similarly 6 to 64 decoder is used for colourm decoder. For this case, address line & colourm lines are not equal, so it is refered as 2½ D RAM organization



13 / 88

ROM MEMORIES:-

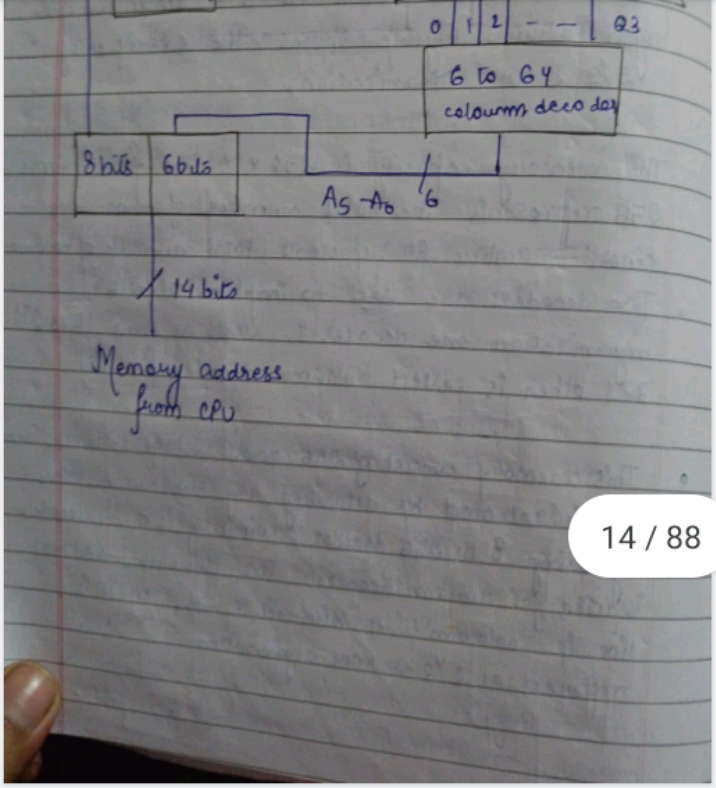
A ROM is essentially a memory device in which permanent binary information is stored. The binary information must be specified by designer.

A block diagram of ROM consists of n input & m output is shown in Figure 1. The

The input provides the address for memory & output gives the data bits of stored word that is selected by address.

NOTE: RAM doesnot have data inputs, because it does not have write operation

K input



ROM MEMORIES :-

A ROM is essentially a memory device in which permanent binary information is stored. The binary information must be specified by designer.

A block diagram of ROM consists of n input & n output is shown in Figure 1. The

The input provides the address for memory & output gives the data bits of stored word that is selected by address.

NOTE: RAM doesnot have data inputs, because it does not have write operation

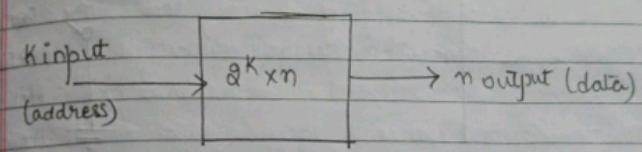


Fig 1: ROM block diagram.

Fig 2: shows a possible configuration of ROM cell. A logic value 0 is stored in the cell, if the transistor is connected to ground at point P, otherwise 1 is stored.

The bit line is connected to a register to a power supply. To read the state of cell, word line is activated.

(address)

Fig 1: ROM block diagram.

Fig 2: shows a possible configuration of ROM cell. A logic value 0 is stored in the cell, if the transistor is connected to ground at point P, otherwise 1 is stored.

The bit line is connected to a register to a power supply. To read the state of cell, word line is activated.

- Thus, the transistor switch is closed & voltage on bit line drops to near zero, if there is a connection between transistor and ground.
- If there is no connection to ground, the bit line remain high voltage, indicating a 1. Data are written into ROM when it is manufactured

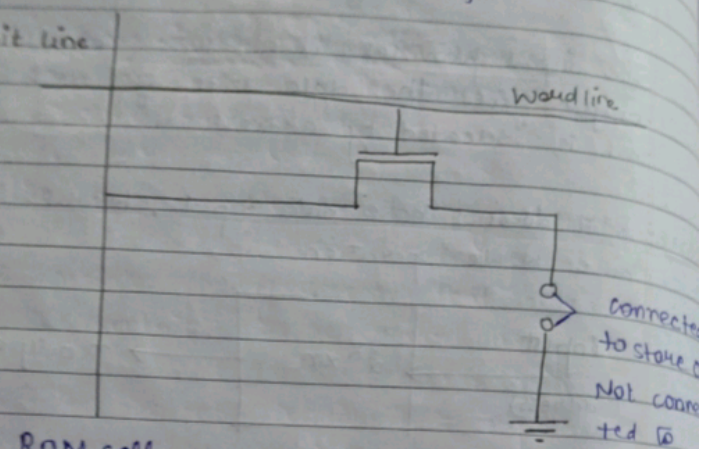


Fig 2: A ROM cell

ROM CHIP: Since a ROM can only read, the data bus can only be in output mode. The block diagram of a ROM chip is depicted in Fig 3. For the same size chip, it is possible to have more bits of ROM than of RAM, because internal binary cell of ROM occupy less space than in RAM.

- For this reason, the diagram specify a 512 byte ROM, while RAM has only 128 bytes.

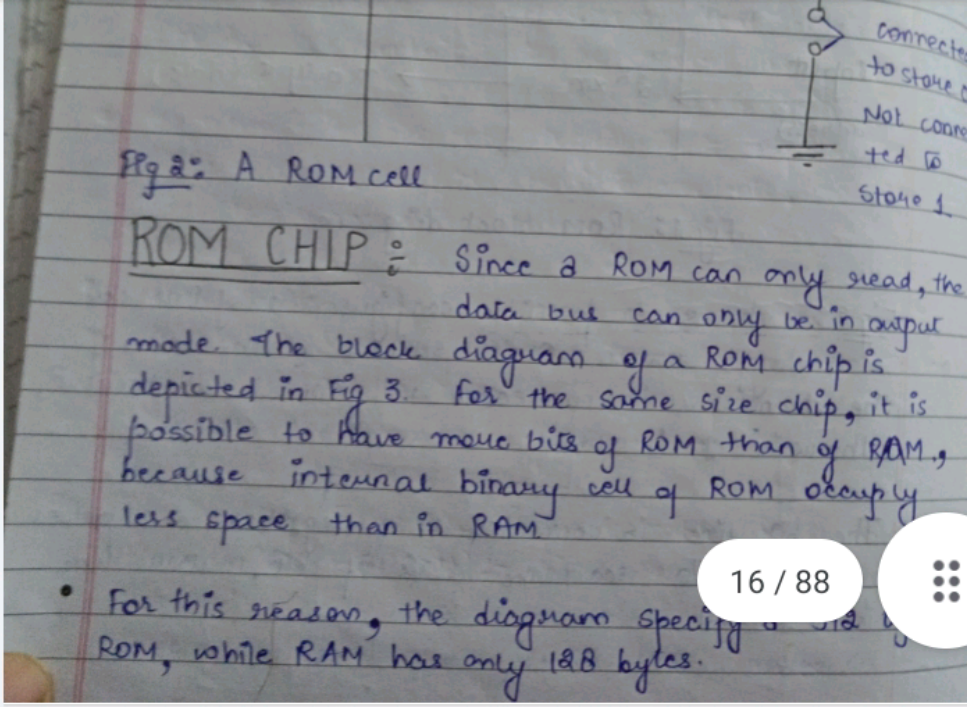
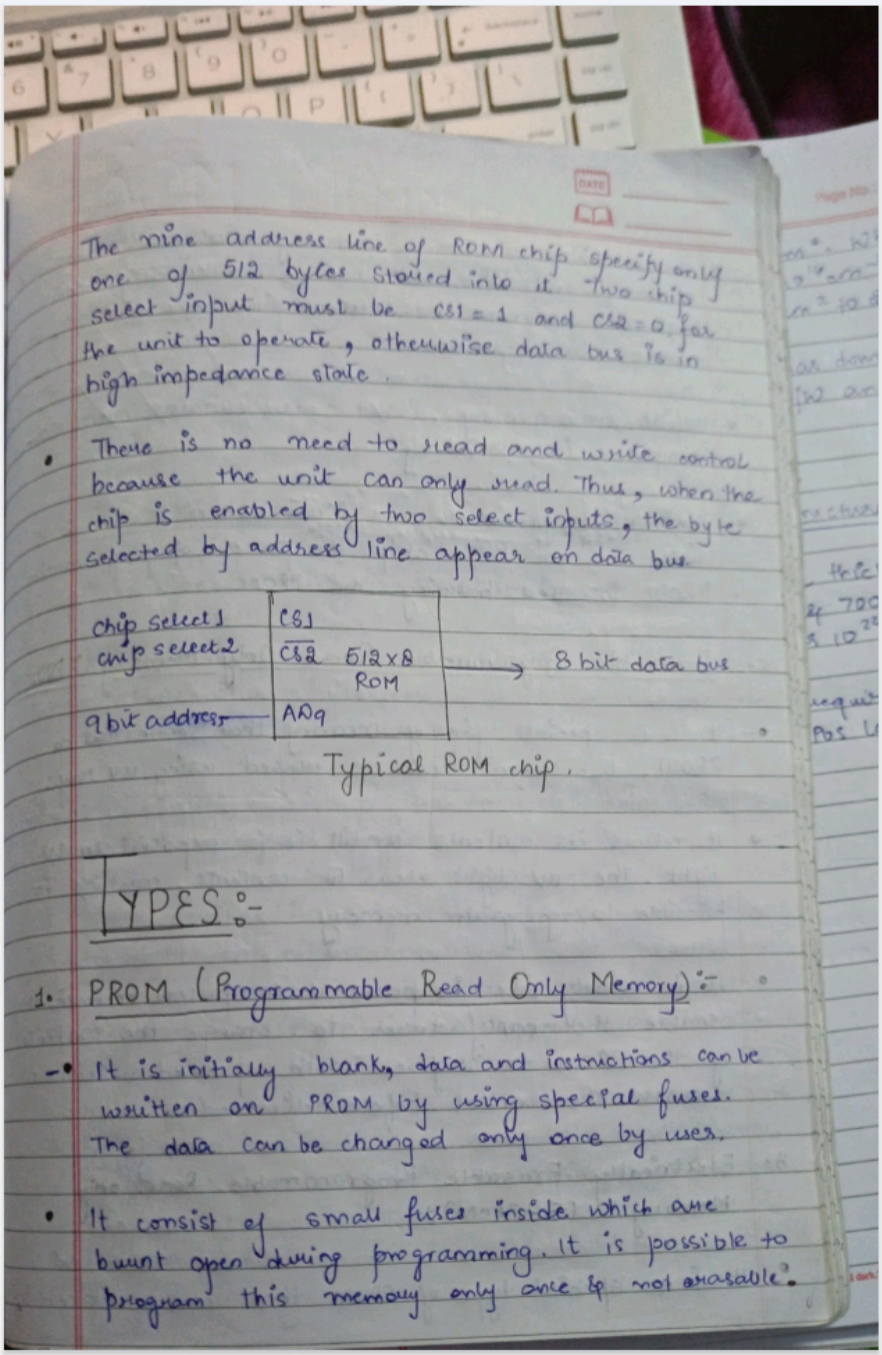


Fig 2: A ROM cell

ROM CHIP: Since a ROM can only read, the data bus can only be in output mode. The block diagram of a ROM chip is depicted in Fig 3. For the same size chip, it is possible to have more bits of ROM than of RAM, because internal binary cell of ROM occupy less space than in RAM

- For this reason, the diagram specify ... ROM, while RAM has only 128 bytes.



The nine address line of ROM chip specify any one of 512 bytes stored into it. Two chip select input must be $CS1 = 1$ and $\overline{CS2} = 0$ for the unit to operate, otherwise data bus is in high impedance state.

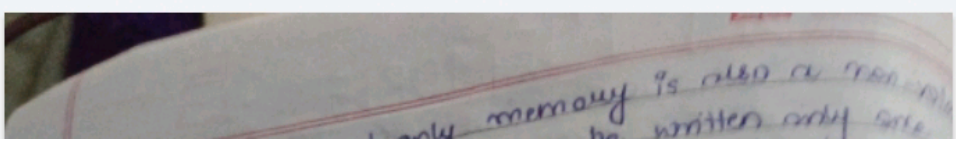
- There is no need to read and write control because the unit can only read. Thus, when the chip is enabled by two select inputs, the byte selected by address line appear on data bus.

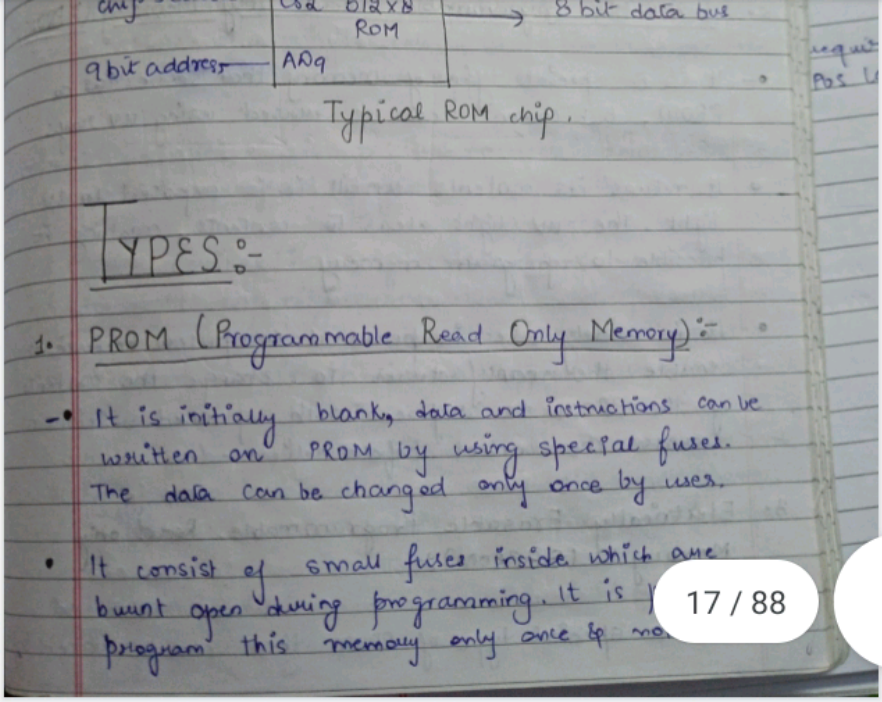
Typical ROM chip.

TYPES:-

1. PROM (Programmable Read Only Memory):-

- It is initially blank, data and instructions can be written on PROM by using special fuses. The data can be changed only once by user.
- It consist of small fuses inside which are burnt open during programming. It is possible to program this memory only once & not erasable.





Typical ROM chip.

TYPES :-

1. PROM (Programmable Read Only Memory) :-

- It is initially blank, data and instructions can be written on PROM by using special fuses. The data can be changed only once by user.
- It consist of small fuses inside which are burnt open during programming. It is [illegible] program this memory only once & no[illegible]

- Programmable read only memory is also a non-volatile memory on which data can be written only once. Once a program has been written onto a PROM, it remains there forever.
- Unlike main memory, PROM retain contains even when computer is turned off.
- A PROM programmer or a PROM burner is used to write data on PROM chip. The process of programming a PROM is called burning the PROM.

2. Erasable Programmable Read Only Memory (EPROM)

- It is a special type of memory that serves as a PROM, but content can be erased using UV rays.
- It retains its contents until it is exposed to UV light. The UV light clear to contents, making it possible to reprogram memory.
- It is widely used in personal computers because it enable the manufacturer to change the contents of the PROM to replace with updated versions or erase the content before delivered.

3. Electrically Erasable Programmable Read only Memory : (EEPROM) :-

It is a special type of PROM that can be erased by

PROM is

Erasable Programmable Read Only Memory (EPROM)

- It is a special type of memory that serves as a PROM, but content can be erased using UV rays
- It retains its contents until it is exposed to UV light. The UV light clear to contents, making it possible to reprogram memory
- It is widely used in personal computers because they enable the manufacturers to change the contents of the PROM to replace with updated versions or erase the content before delivered

3. Electrically Erasable Programmable Memory: (EEPROM):-

It is a special type of PROM that can be erased by

exposing it to an electrical charge.

- EEPROM retains its contents even when the power is turned off.
- Comparing with all other types of ROM, EEPROM is slower in performance.

4. FLASH MEMORY:- It is similar to EEPROM technology. A flash cell is based on a single transistor controlled by trapped charge.

- In a flash device, it is possible to read the contents of a single cell, but it is only possible to write an entire block of cells.
- In EEPROM it is possible to read and write the content of ~~signed~~ single cell.
- Flash devices have greater density, which leads to higher capacity and lower cost per bit
- Require a single power supply voltage and consume less power
- Application: Hand held computers, cell phones, digital cameras & MP3 music player
- Flash cards: Size: 8, 32 & 64 MB

Memory of the prog memory

to write an entire block of cells.

- In EEPROM it is possible to read and write the content of ~~signal~~ single cell.
- Flash devices have greater density, which leads to higher capacity and lower cost per bit
- Require a single power supply voltage and consume less power
- Application : Hand held computers, cell phones, digital cameras & MP3 music player
- Flash cards : Size : 8, 32 & 64 MB

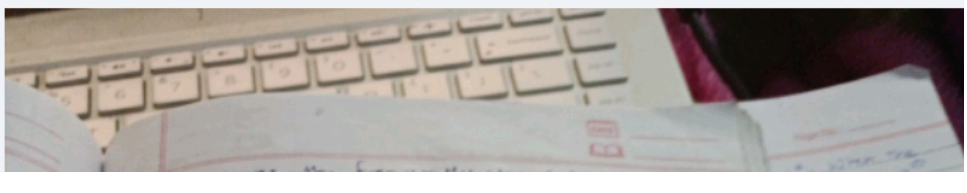
Cache Memory

If the active portions of the program are placed in a ~~fast~~ small memory, the average memory access time can be reduced, thus reducing the total execution time of program.

Cache Memory :-

1. Cache memory is a special very high speed memory.
2. It is costlier than main memory but economical than CPU registers.
3. It holds the frequently requested data & instruction so that they are immediately available to the CPU when needed.
4. Cache memory is an extremely fast memory that acts as a buffer between RAM & CPU.

Whenever you are executing a program or instructions, then some of data and instructions required frequently, those instructions and data should be availalable to the CPU whenever there is a requirement so that can be stored in the cache memory.



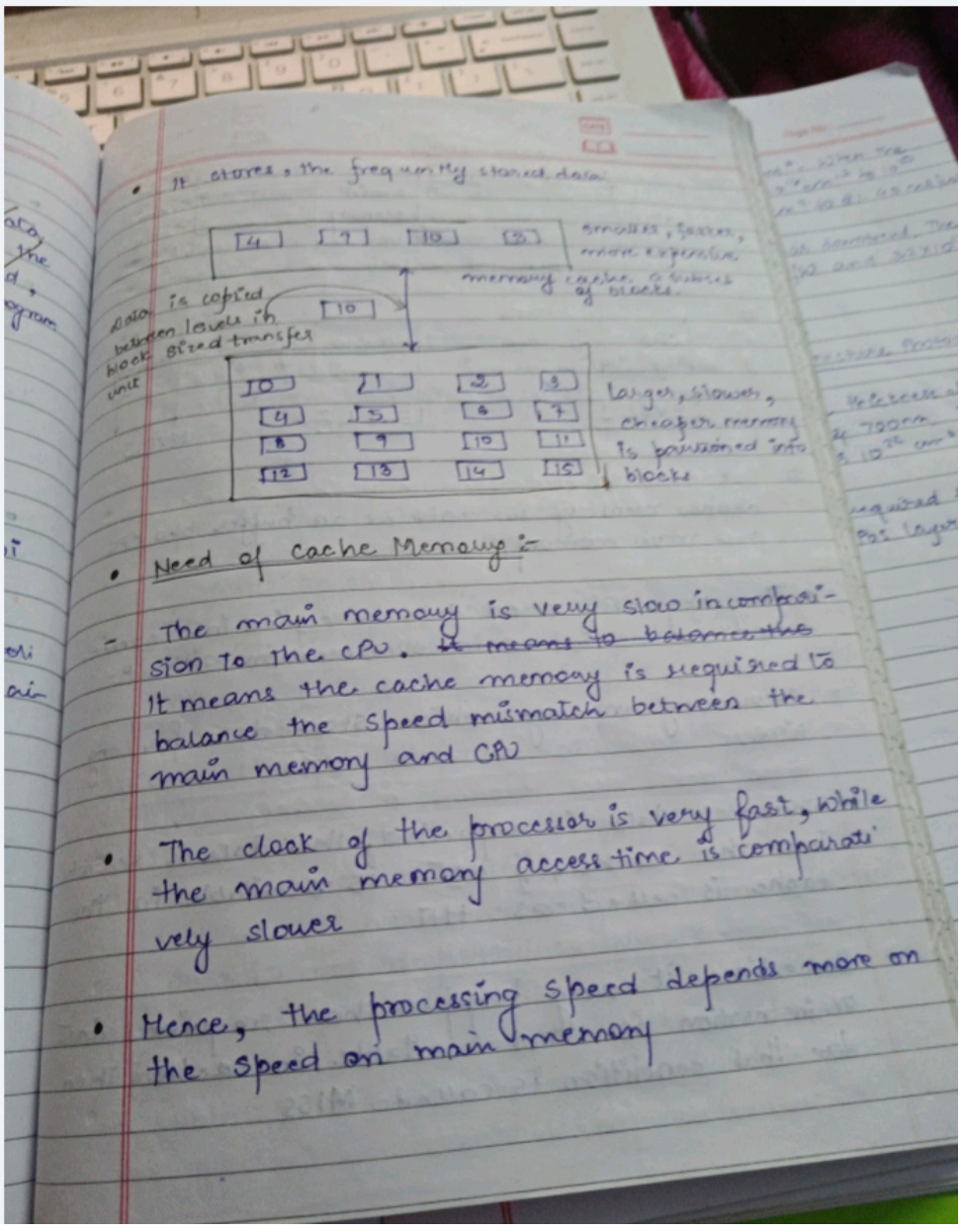
1. Cache memory is a special very high speed memory.

2. It is costlier than main memory but economical than CPU registers.

3. It holds the frequently requested data & instruction so that they are immediately available to the CPU when needed.

4. Cache memory is an extremely fast memory that acts as a buffer between RAM & CPU.

Whenever you are executing a program instructions, then some of data and instructions required frequently, those instructions and data should be availalable to the CPU whenever there is a requirement so that can be stored in the cache memory.



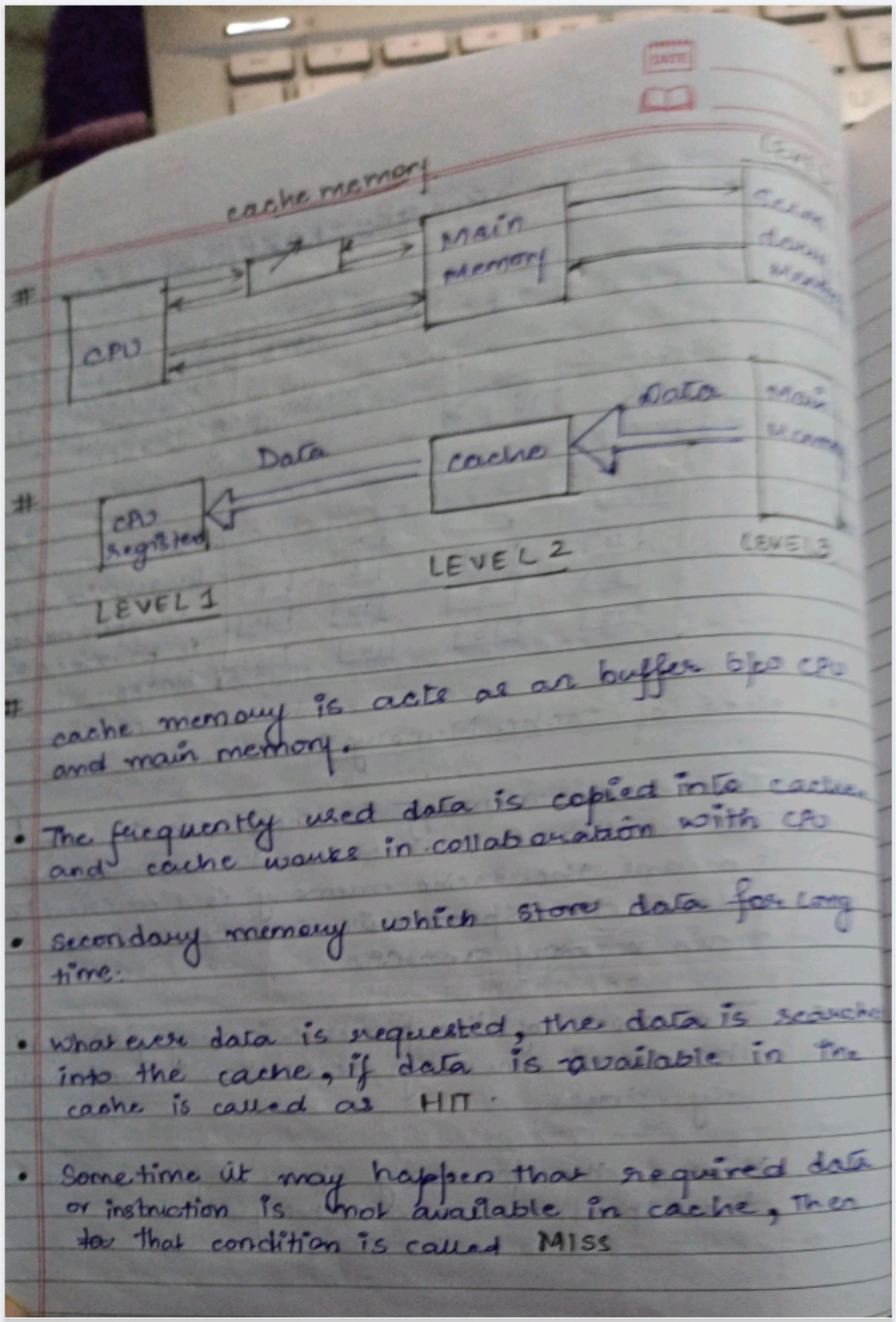
- Need of cache Memory :-

- The main memory is very slow in comparision to the CPU. ~~It means to balance the~~ It means the cache memory is required to balance the speed mismatch between the main memory and CPU

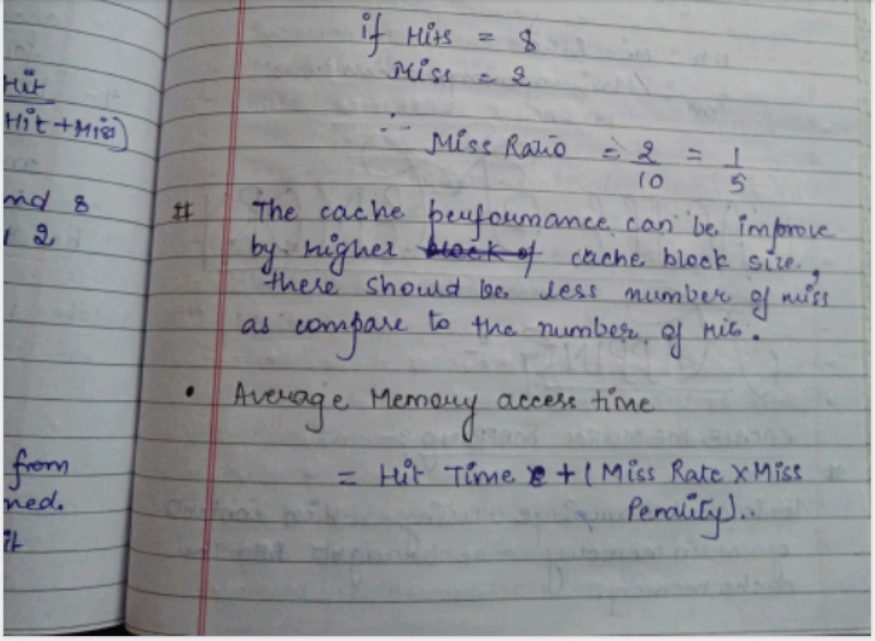
- The clock of the processor is very fast, while the main memory access time is comparatively slower

- Hence, the processing speed depends more on the speed on main memory

- Need of cache Memory :-
  - The main memory is very slow in comparision to the CPU. ~~It means to balance the~~ It means the cache memory is required to balance the speed mismatch between the main memory and CPU
  - The clock of the processor is very fast, while the main memory access time is comparatively slower
  - Hence, the processing speed depends more on the speed on main memory



- cache memory is acts as an buffer b/w CPU and main memory.
- The frequently used data is copied into cache and cache works in collaboration with CPU
- Secondary memory which store data for long time.
- Whatever data is requested, the data is searched into the cache, if data is available in the cache is called as HIT.
- Sometime it may happen that required data or instruction is not available in cache, Then for that condition is called MISS



if Hits = 8
Miss = 2

∴ Miss Ratio = $\frac{2}{10} = \frac{1}{5}$

The cache performance can be improve by higher ~~block of~~ cache block size. there should be less number of miss as compare to the number of hit.

- Average Memory access time

= Hit Time + (Miss Rate x Miss Penalty).

if Hits = 8
Miss = 2

∴ Miss Ratio = $\frac{2}{10} = \frac{1}{5}$

The cache performance can be improve by higher ~~block of~~ cache block size. there should be less number of miss as compare to the number of hits.

- Average Memory access time
  = Hit Time + (Miss Rate × Miss Penalty).

Average Memory access time = Hit Time + (Miss Rate × Miss Penalty)

$t_a = h \times t_c + (1-h) \times (t_c + t_m)$

$$t_a = t_c + (1-h)t_m$$

where

t_a = Average memory access time

h = hit ratio

t_c : cache memory access time

$1-h$: Miss Ratio

t_m : Main memory access time.

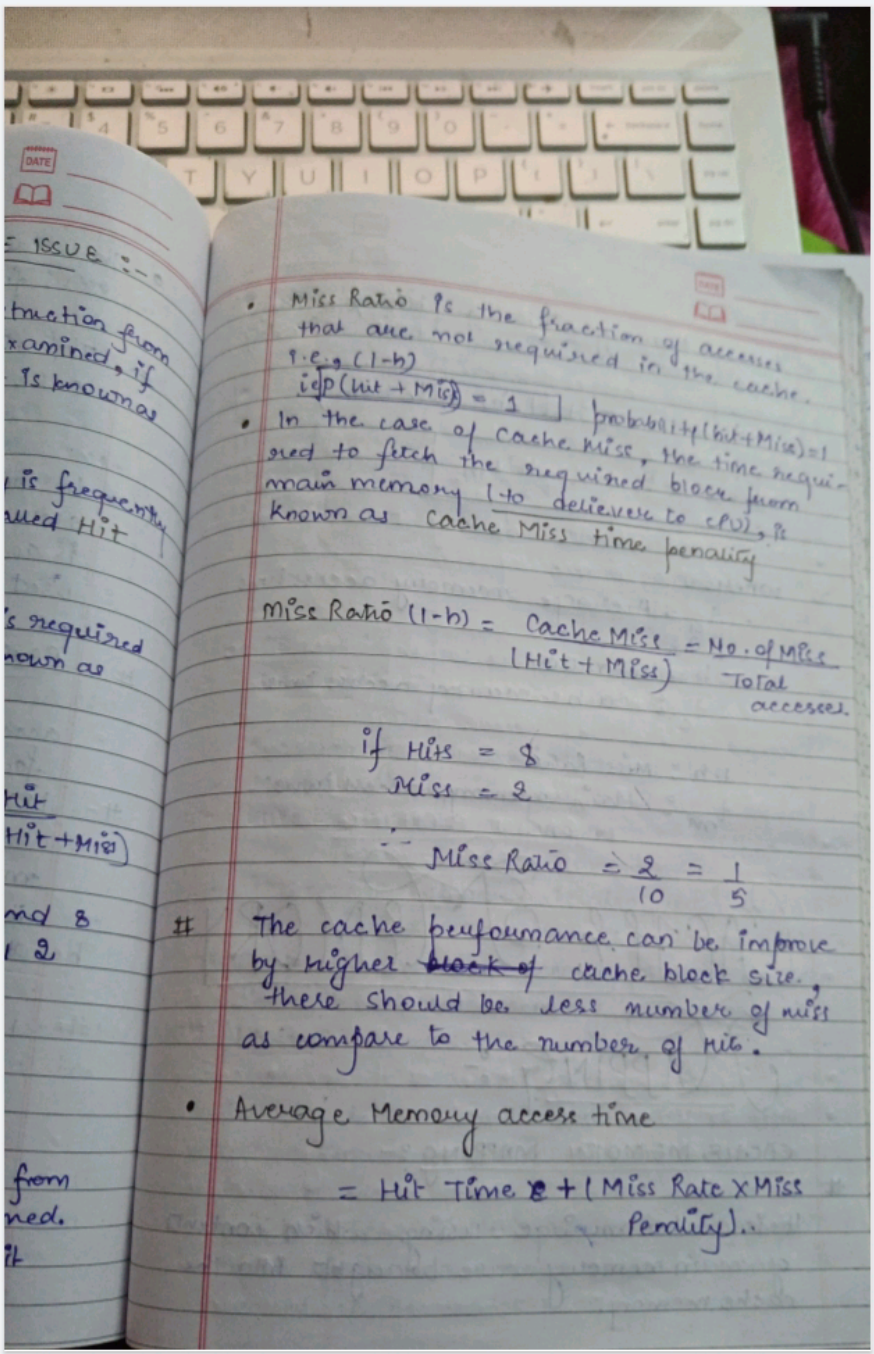
CACHE MEMORY MAPPING :-

CACHE MEMORY MAPPING :-

It is a technique using which contents of main memory are brought into the cache memory.

$= \frac{2}{10} = \frac{1}{5}$

- If the CPU needs the data / instruction from memory, then first cache is examined. If the word is not found in cache, it is known as Cache Miss.



- Miss Ratio is the fraction of accesses that are not required in the cache.
  i.e. $(1-h)$
  i.e. $P(\text{Hit} + \text{Miss}) = 1$ | probability of (Hit + Miss) = 1
- In the case of Cache Miss, the time required to fetch the required block from main memory (to delivered to CPU), is known as Cache Miss time penality

Miss Ratio $(1-h)$ = $\frac{\text{Cache Miss}}{(\text{Hit} + \text{Miss})} = \frac{\text{No. of Miss}}{\text{Total accesses}}$

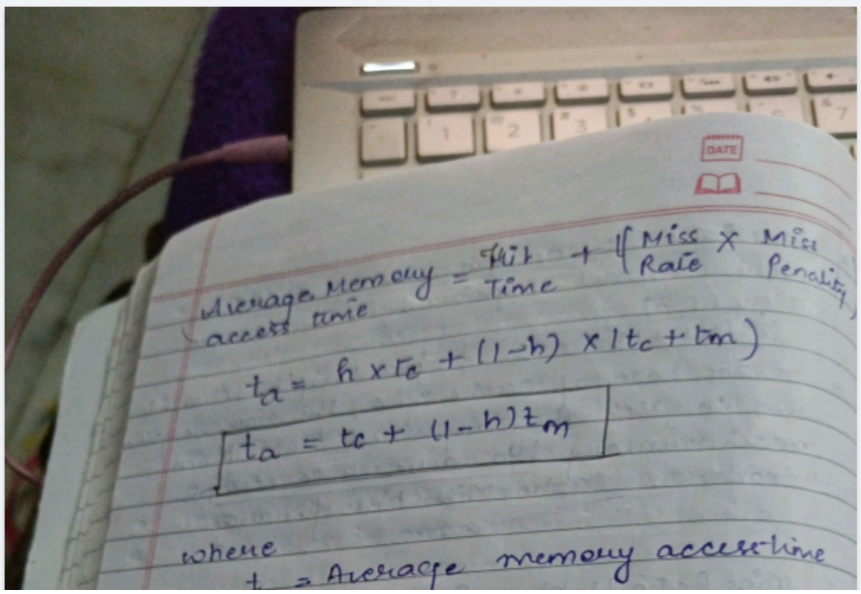
if Hits = 8
Miss = 2

$\therefore$ Miss Ratio $= \frac{2}{10} = \frac{1}{5}$

The cache performance can be improve by higher ~~block of~~ cache block size. There should be less number of miss as compare to the number of Hit.

- Average Memory access time

= Hit Time + (Miss Rate x Miss Penality).

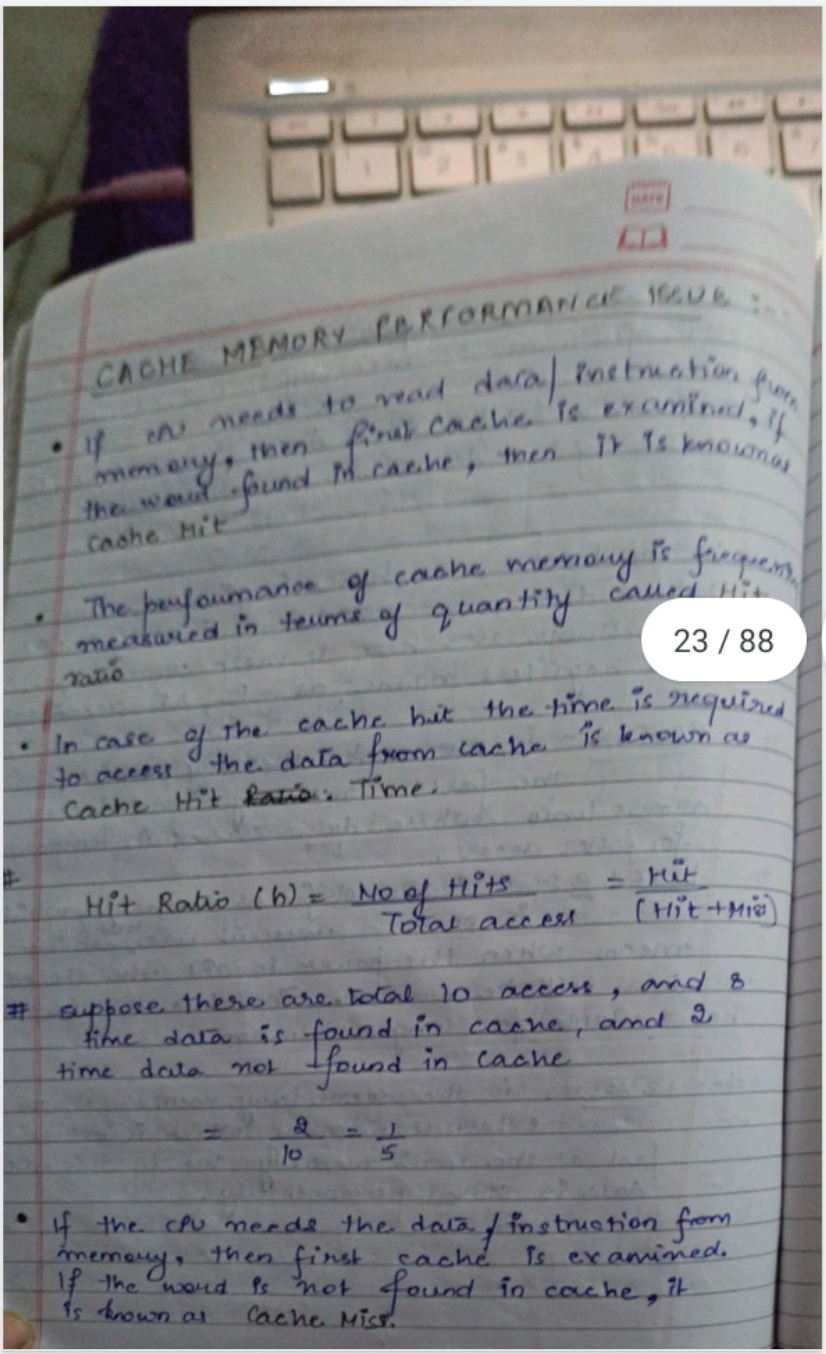


Average Memory access time = Hit Time + (Miss Rate x Miss Penality)

$t_a = h \times t_c + (1-h) \times (t_c + t_m)$

$$t_a = t_c + (1-h)t_m$$

where
t_a = Average memory access time



CACHE MEMORY PERFORMANCE ISSUE :-

- If CPU needs to read data/instruction from memory, then first cache is examined, if the word found in cache, then it is known as Cache Hit

- The perfoumance of cache memory is frequently measured in terms of quantity called Hit ratio

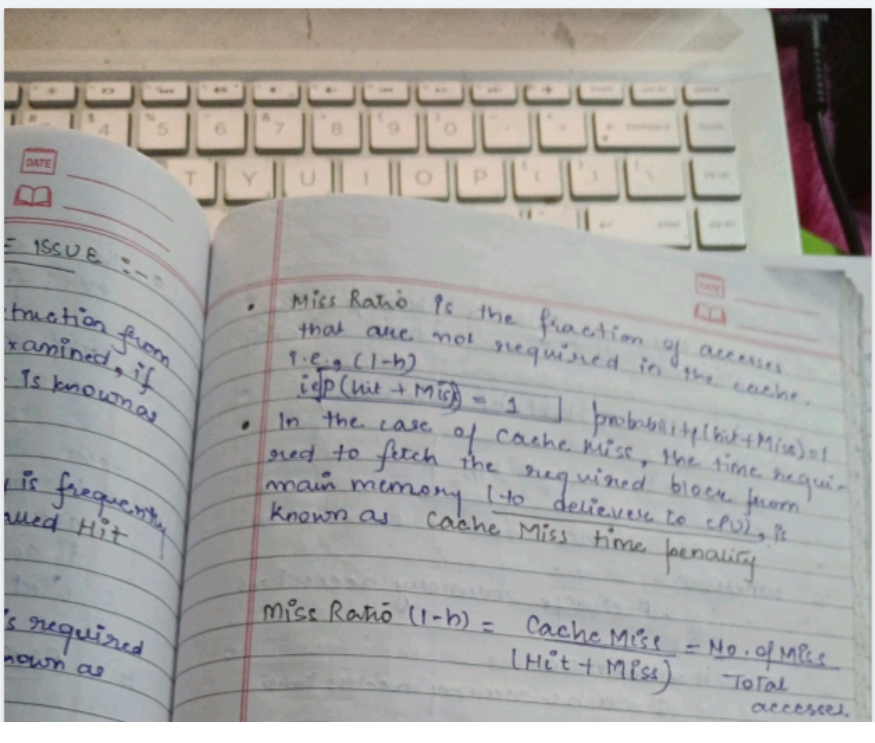
- In case of the cache hit the time is required to access the data from cache is known as Cache Hit ~~Ratio~~ Time.

Hit Ratio (h) = $\frac{\text{No of Hits}}{\text{Total access}} = \frac{\text{Hit}}{(\text{Hit}+\text{Miss})}$

suppose there are total 10 access, and 8 time data is found in cache, and 2 time data not found in cache

$= \frac{2}{10} = \frac{1}{5}$

- If the CPU needs the data/instruction from memory, then first cache is examined. If the word is not found in cache, it is known as Cache Miss.



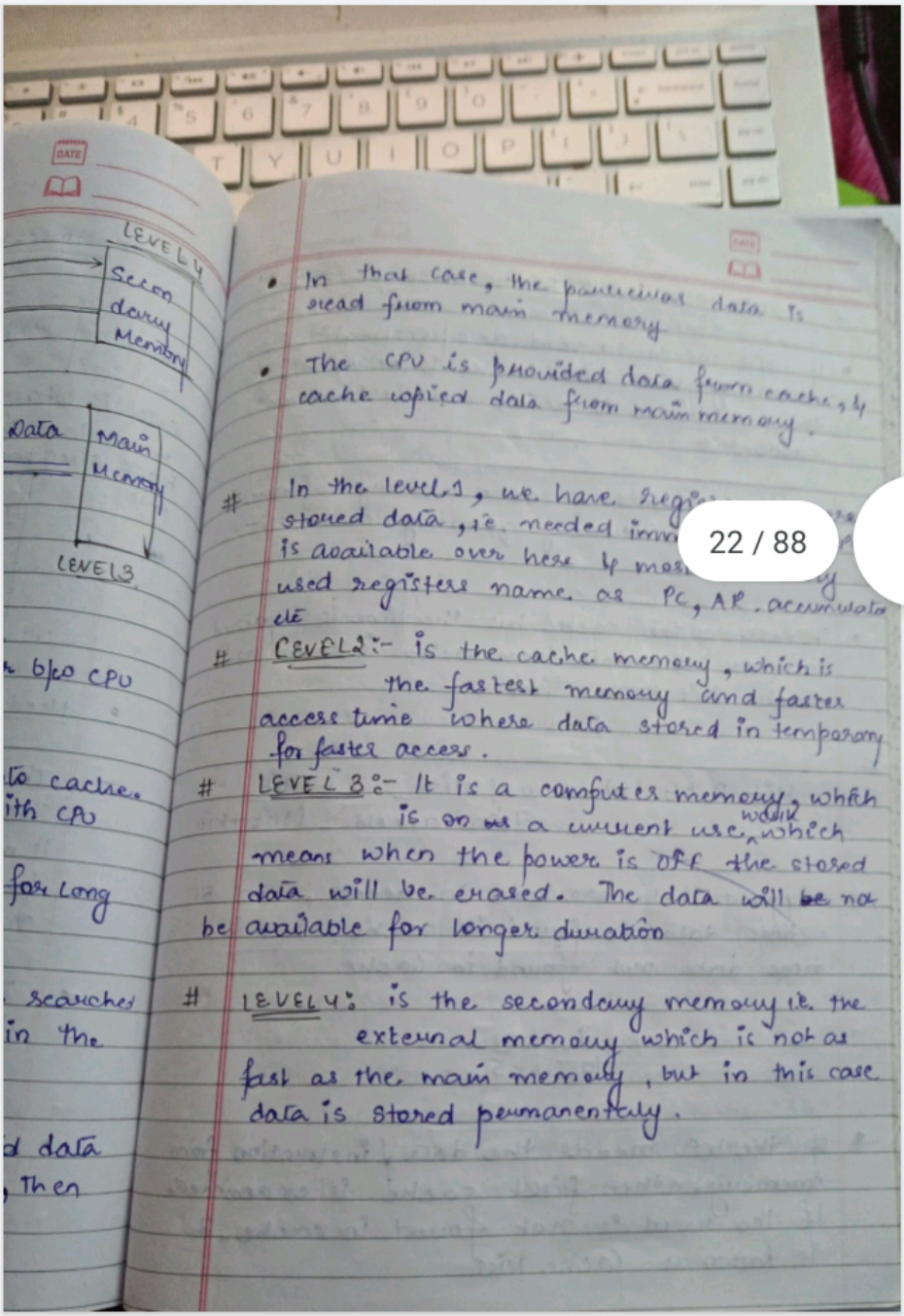
- Miss Ratio is the fraction of accesses that are not required in the cache. i.e. (1-h)

i.e. (Hit + Miss) = 1] probability (Hit+Miss) = 1

- In the case of Cache Miss, the time required to fetch the required block from main memory [to delievere to CPU], is known as Cache Miss time penality

Miss Ratio (1-h) = $\frac{\text{Cache Miss}}{(\text{Hit}+\text{Miss})} = \frac{\text{No. of Miss}}{\text{Total accesses}}$

- Sometime it may happen that required data or instruction is not available in cache, Then for that condition is called MISS



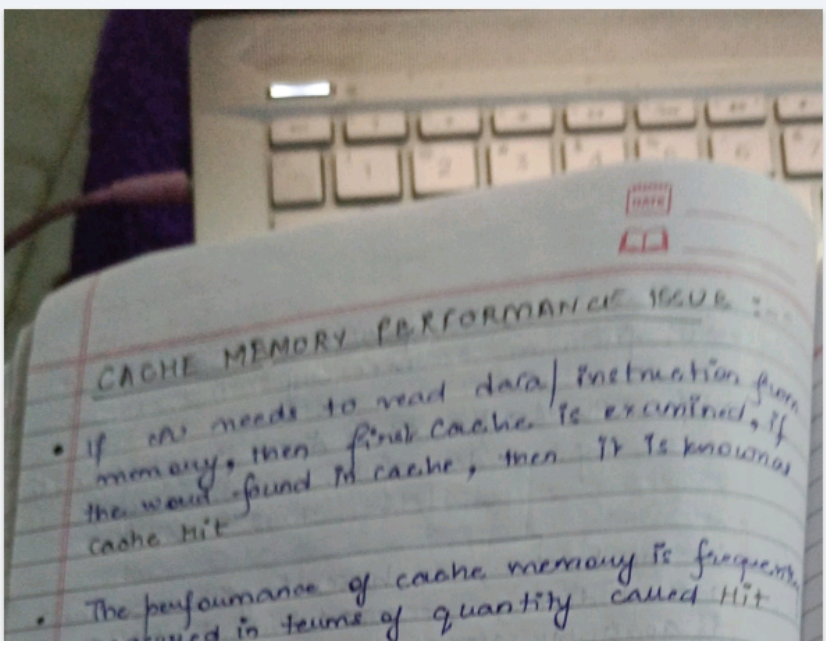
- In that case, the particular data is read from main memory
- The CPU is provided data from cache, & cache copied data from main memory.

In the level 1, we have registers stored data, ie. needed imm... is available over here & mos... used registers name as PC, AR, accumulator etc

LEVEL 2:- is the cache memory, which is the fastest memory and faster access time where data stored in temporary for faster access.

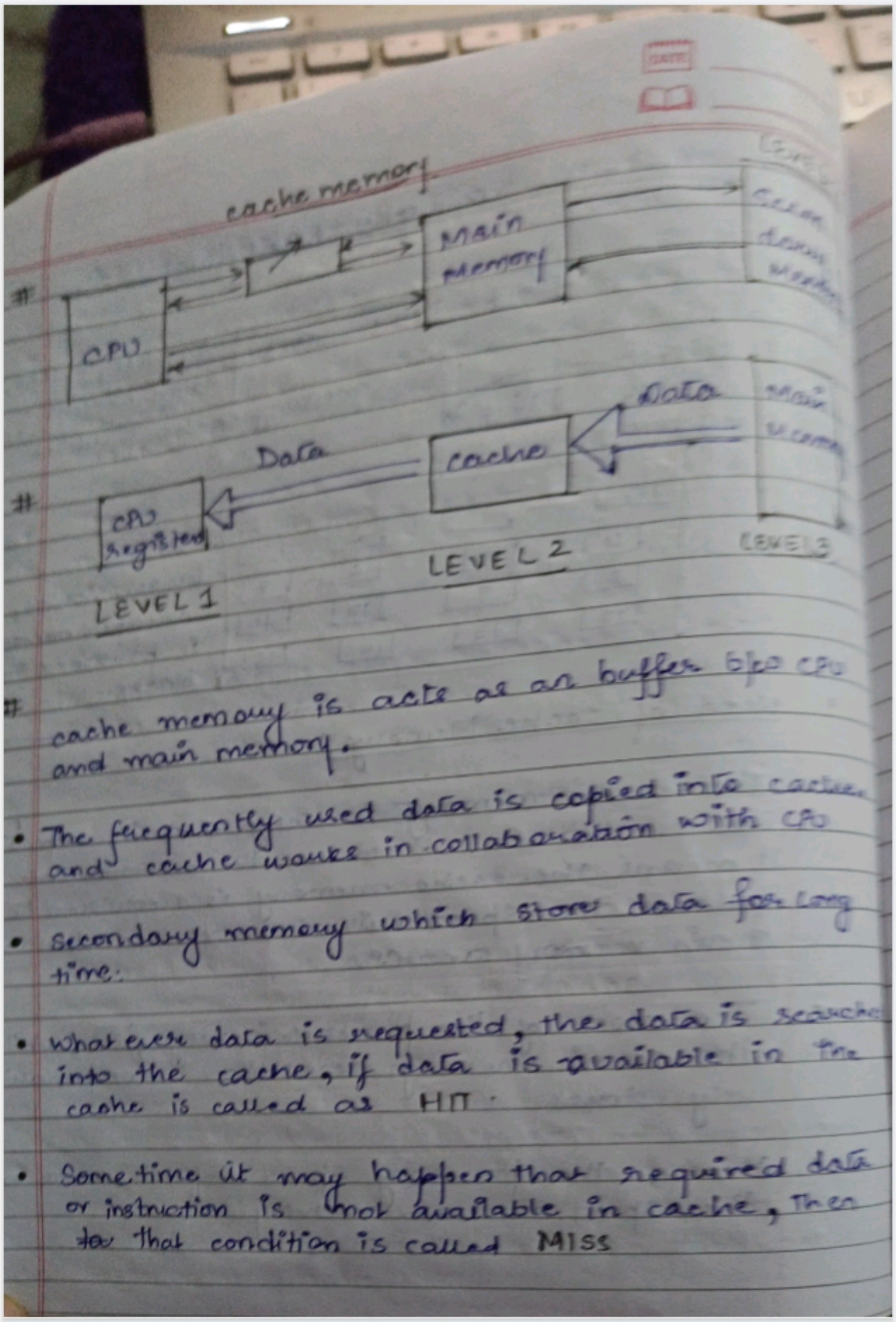
LEVEL 3:- It is a computer memory, which is on a current use, which means when the power is off the stored data will be erased. The data will be not be available for longer duration

LEVEL 4: is the secondary memory ie. the external memory which is not as fast as the main memory, but in this case data is stored permanently.



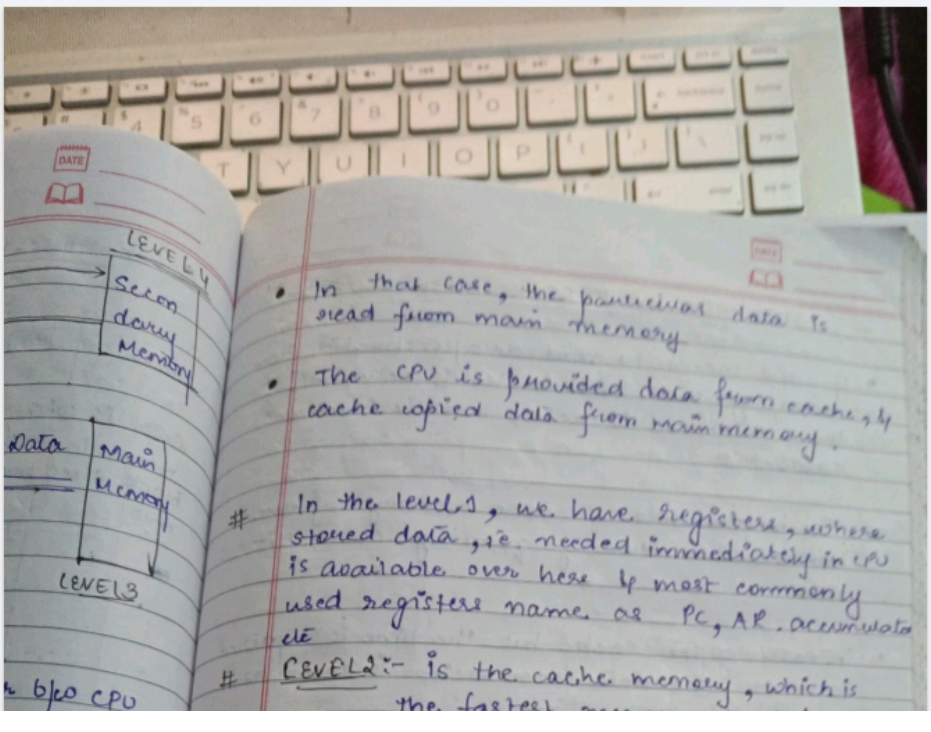
CACHE MEMORY PERFORMANCE ISSUE :

- If CPU needs to read data/instruction from memory, then first cache is examined, if the word found in cache, then it is known as cache Hit
- The performance of cache memory is frequently measured in terms of quantity called Hit



cache memory is acts as an buffer b/w CPU and main memory.

- The frequently used data is copied into cache and cache works in collaboration with CPU
- Secondary memory which store data for long time.
- whatever data is requested, the data is searched into the cache, if data is available in the cache is called as HIT.
- Sometime it may happen that required data or instruction is not available in cache, Then for that condition is called MISS



- In that case, the particular data is read from main memory
- The CPU is provided data from cache, & cache copied data from main memory.

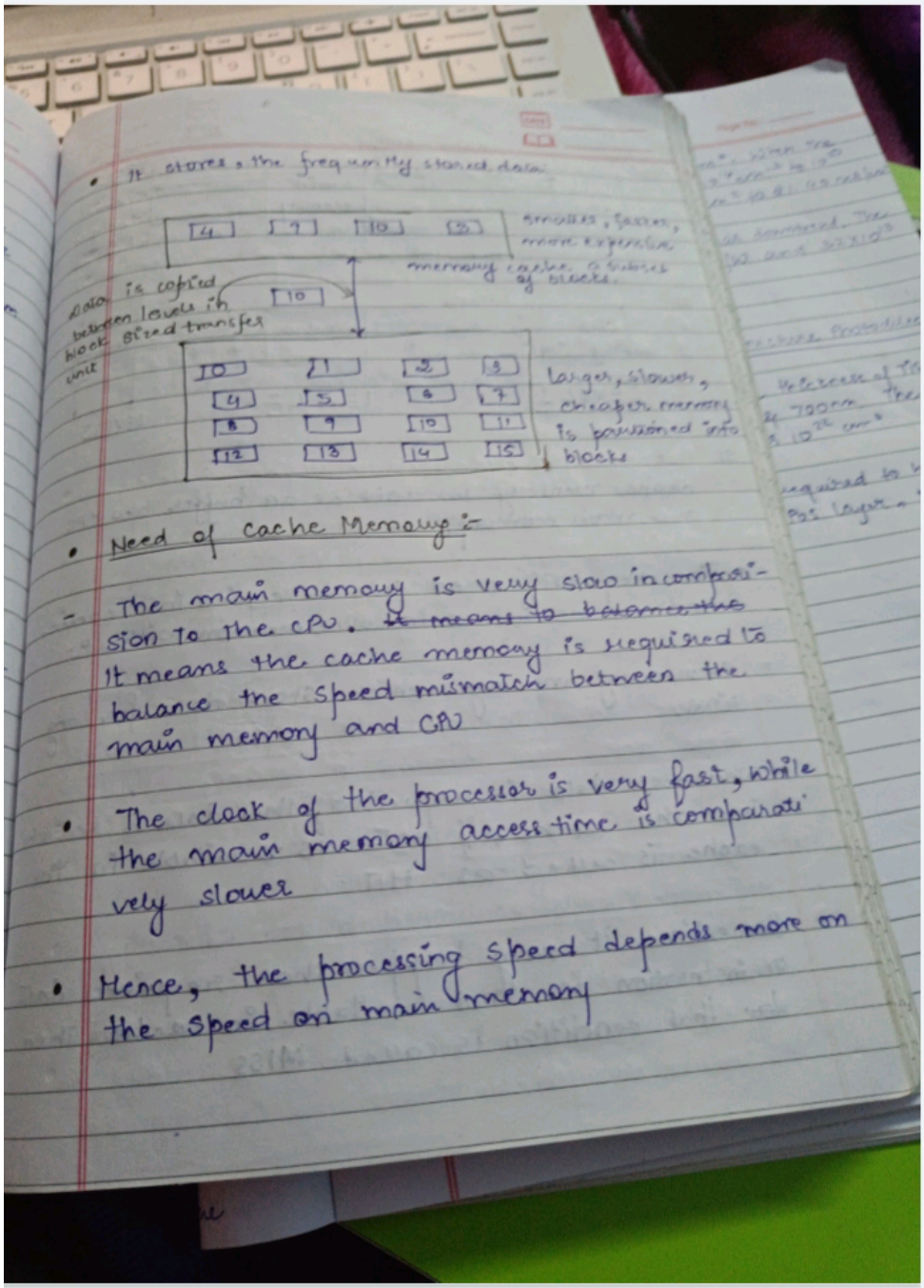
In the level 1, we have registers, where stored data, ie needed immediately in CPU is available over here & most commonly used registers name as PC, AR, accumulator etc

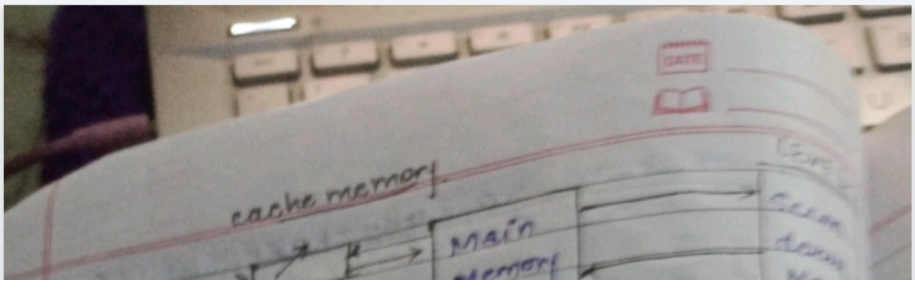
LEVEL 2:- is the cache memory, which is

-lable to the CPU when

4. Cache memory is an extremely fast memory that acts as a buffer between RAM & CPU.

Whenever you are executing a program or instructions, then some of data and instructions required frequently, those instructions and data should be available to the CPU whenever there is a requirement so that can be stored in the cache memory.





- Flash cards : Size : 8, 32 & 64 MB

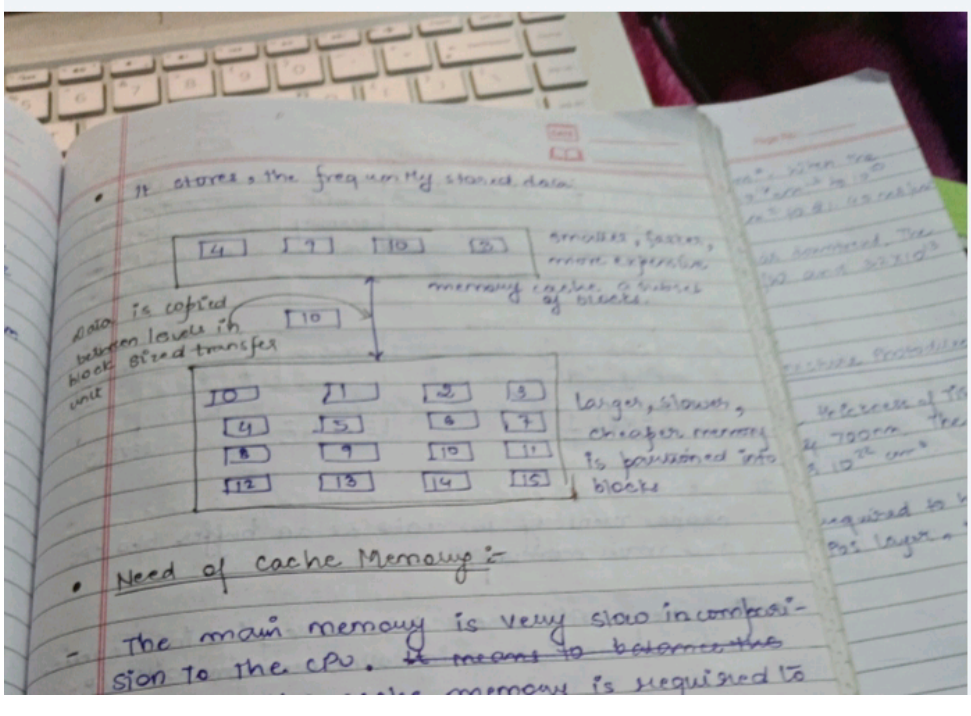
Cache Memory

If the active portions of the program are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of program.

Cache Memory :

1. Cache memory is a special very high speed memory.
2. It is costlier than main memory but economical than CPU registers.
3. It holds the frequently requested data & instruction so that they are immediately available to the CPU when needed.
4. Cache memory is an extremely fast memory that acts as a buffer between RAM & CPU.

Whenever you are executing a program or instructions, then some of data and instructions required frequently, those instructions and data should be availalable to the CPU whenever there is a requirement so that can be stored in the cache memory.



- Need of cache Memory :

- The main memory is very slow in comparision to the CPU. ~~It means to balance this~~ memory is required to

III Division & logic Operations -

Binary Subtraction

1 - 0 = 1
1 - 1 = 0
0 - 0 = 0
0 - 1 = 1 (with 1 borrow)

Ex- 1 2 0 1
- 0 1 1 1
0 1 1 0

Binary Division -

Ex - 11/3 11 = Dividend , 3 = Divisor 11 = 1011
3 = 011

0 1 1 } Quotient = 3
11 ⟌ 1 0 1 1
- 1 1 ↓
1 0 1
1 1
1 0 } remainder = 2

Ex - 27/7 27 : 11011 7 : 111
1 1 } Q = 3
111 ⟌ 1 1 0 1 1
1 1 1 ↓
1 1 0 1
1 1 1
1 1 0 } R = 6

Division is somewhat more complex than multiplication but is based on the same general principles. First, the bits of the dividend are examined from left to right, until the set of bit examined represents a number greater than or equal to the divisor, this is referred to as the division

being able to divide the number. Until this even occurs, 0s are placed in the quotient from left to right. When this event occurs, a 1 is placed in the quotient & the divisor is subtracted from the partial dividend. The result is referred to as a partial remainder.

The division follows a cyclic pattern. At each cycle, additional bits from the dividend are appended to the partial remainder until the result is greater than or equal to the divisor. As before, the divisor is subtracted from this number to produce a new partial remainder. The process continues until all the bits of the dividend are exhausted.

00011101 ← Quotient = 29
Divisor → 1011 ⟌ 100100011 ← Dividend = 291

2:18 13.79 KB/s 69%

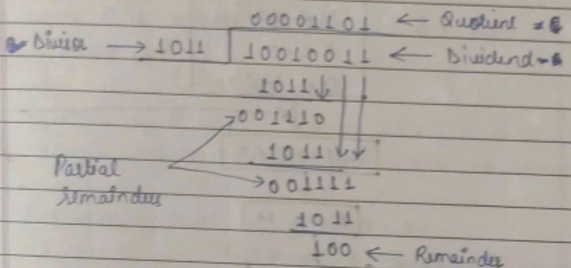
Ex - 27/7   27 : 11011   7 : 111

```
           11    } Q = 3
111 ) 11011
      111↓
      1101
      111
       110  } R = 6
```


Division is somewhat more complex than m[...] but is based on the same general principle. First, the bits of the dividend are examined from left to right, until the set of bit examined represents a number greater than or equal to the divisor, this is referred to as the divisor

being able to divide the number. Until this event occurs, 0s are placed in the quotient from left to right. When the event occurs, a 1 is placed in the quotient & the divisor is subtracted from the partial dividend. The result is referred to as a partial remainder.

The division follows a cyclic pattern. At each cycle, additional bits from the dividend are appended to the partial remainder until the result is greater than or equal to the divisor. As before, the divisor is subtracted from this number to produce a new partial remainder. The process continues until all the bits of the dividend are exhausted.



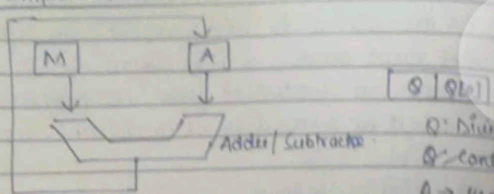
Division of Unsigned Binary Integer.

A machine algorithm that corresponds to the long division process.

Step - The A & Q registers together are shifted to the left 1 bit. M is subtracted from A to determine whether A divides the partial remainder. If it does, then Q_0 gets a 1 bit. Otherwise, Q_0 gets a 0 bit & M must be added back to A to restore the previous value. The count is then decremented & the process continues for n steps.

RESTORING DIVISION ALGORITHM

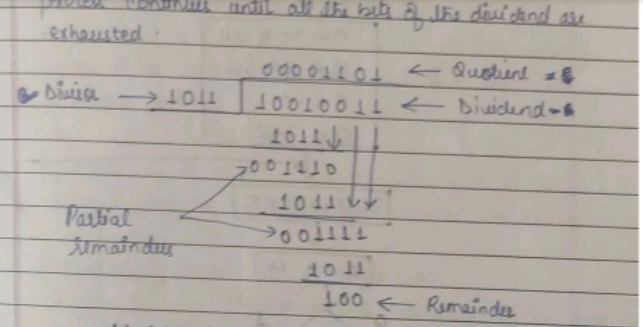
Hardware Implementation.



Q: Dividend [...]
Q: contain [...]
A → remainder
M → divisor [...]

Flow Chart   Restoring Algorithm

exhausted.



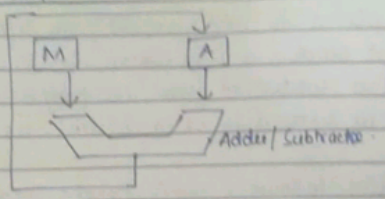
Division of Unsigned Binary Integer.

A machine algorithm that corresponds to the long process.

Step - The A & Q registers together are shifted to the left 1 bit. M is subtracted from A to determine whether A divides the partial remainder. If it does, then Q gets a 1 bit. Otherwise, Q gets a 0 bit & M must be added back to A to restore the previous value. The count is then decremented & the process continues for n steps.

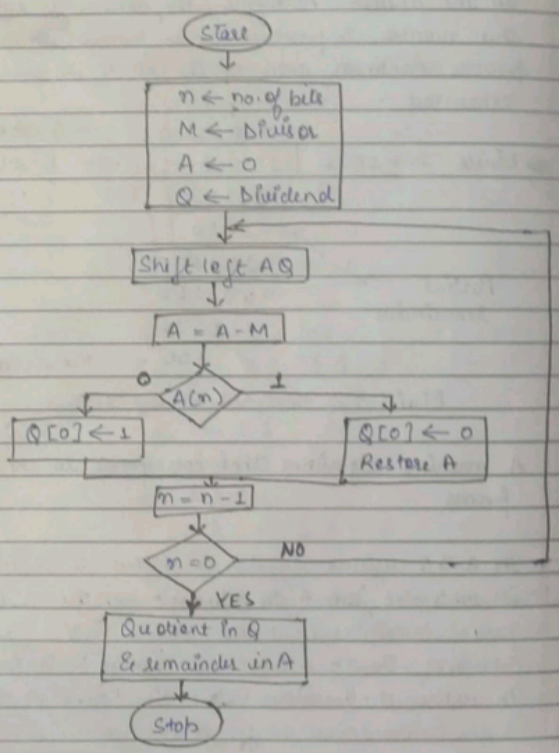
RESTORING DIVISION ALGORITHM

Hardware Implementation.



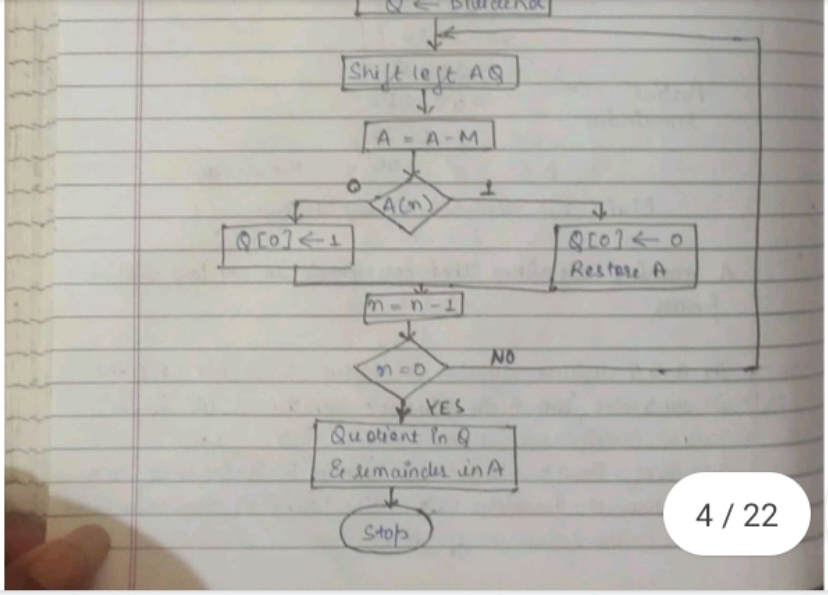
Q: Dividend
Q: Contain
A → remainder
M → divider

Flow Chart Restoring Algorithm



Step1: First the registers are initialized with corresponding value (Q = dividend)

M = divisor, A = 0, n = no. of bits in dividend (Q).



Step1: First the registers are initialized with corresponding value (Q = dividend)

M = divisor, A = 0, n = no. of bits in dividend (Q).

bits in A & M will be n+1, if bit in Q is n

Step2: Content of A & Q is shifted left

Step3: Content of register M is subtracted from A & stored in A

A = A - M { A + (M̄ + 1) }

↳ 2's complement of M

Step4: Then MSB bit of A (first bit of A) is checked if it is 1 then LSB of Q (last bit of Q) is set to 0 & value of register A is restored i.e the value of A before the subtraction with M

If MSB of A is 0, then LSB of Q is set to 1 & value of register A will be value after the subtraction of M

Step5: The value of counter n is decremented

Step6: If n = 0 ; final result Q → Quotient & A ← remainder

n ≠ 0 ; repeat step 2 to 6, until n = 0

Example: Dividend = 11 = 1011 = Q n = 4

Divisor = 3 = 00011 = M → n+1 bits (5)

M̄ + 1 = 11100 + 1 = 11101

n	M	A	Q	Action/Operation
4(100)	00011	00000	1011	Initial
		00001	0110 ?	shl AQ
		11101		Sub M
		1 11110	0110	as A(n)=1, Q[0]=0
3(011)		00001	0110	Restore A
		00010	110 ?	shl AQ
		11101		Sub M
		1 11111	1100	Q[0] ← 0
2(010)		00010	1100	Restore A
		00101	100 ?	shl AQ
		11101		Sub M
1(001)	discard carry	1 00010	1001	Q[0] ← 1
		00101	001 ?	shl AQ

A = A - M ⇒ A + (M+1)
↳ 2's complement of M

Step4: Then MSB bit of A (first bit of A) is checked if it is 1 then LSB of Q (last bit of Q) is set to 0 & value of register A is restored i.e the value of A before the subtraction with M

If MSB of A is 0 ; then LSB of Q is set to 1 & value of register A will be value after the subtraction of M

Step5: The value of counter n is decremented

Step6: If n = 0 ; final result Q → Quotient & A ← remainder
n ≠ 0 ; repeat step 2 to 6, until n=0

Example: Dividend = 11 = 1011 = Q n = 4
Divisor = 3 = 00011 = M → n+1 bits (5)
M+1 = 11100+1 = 11101

n	M	A	Q	A
4(100)	00011	00000	1011	Initial
		00001	0110	Shl AQ
		11101		Sub M
		11110	0110	as A(n)=1, Q[0]=0
3(011)		00001	0110	Restore A

5 / 22

Date: / /
Page:

		00010	110?	Shl AQ
		11101		Sub M
		11111	1100	Q[0]←0
2(010)		00010	1100	Restore A
		00101	100?	Shl AQ
		11101		SubM
1(001)	discard carry	00010	1001	Q[0]←1
		00101	001?	Shl AQ
		11101		Sub M
0(000)	discard carry	00010	0011	Q[0]←1

Remainder (2) | Quotient (3)

NON RESTORING DIVISION ALGORITHM Ex Q = 1011, n=4
M = 00011

n	M	A	Q	Action/Operation
4(100)	00011	00000	1011	Initial
		00001	011?	Shl AQ
		11101		As A[n]=0, A←A-M
3(011)		00010	0110	As A[n]=1, Q[0]←0
		00100	110?	Shl AQ
		00011		As A[n]=1, A←A-M
2(010)		00111	1100	As A[n]=1, Q[0]←0
		01111	100?	Shl AQ
		00011		As A[n]=1, A←A-M
1(001)		00010	1001	As A[n]=0, Q[0]←1
		00101	001?	Shl AQ
		11101		As A[n]=0, A←A+M
0(000)		00010	0011	As A[n]=0, Q[0]←1

Remainder (2) | Quotient (3)

NON RESTORING DIVISION ALGORITHM

Start

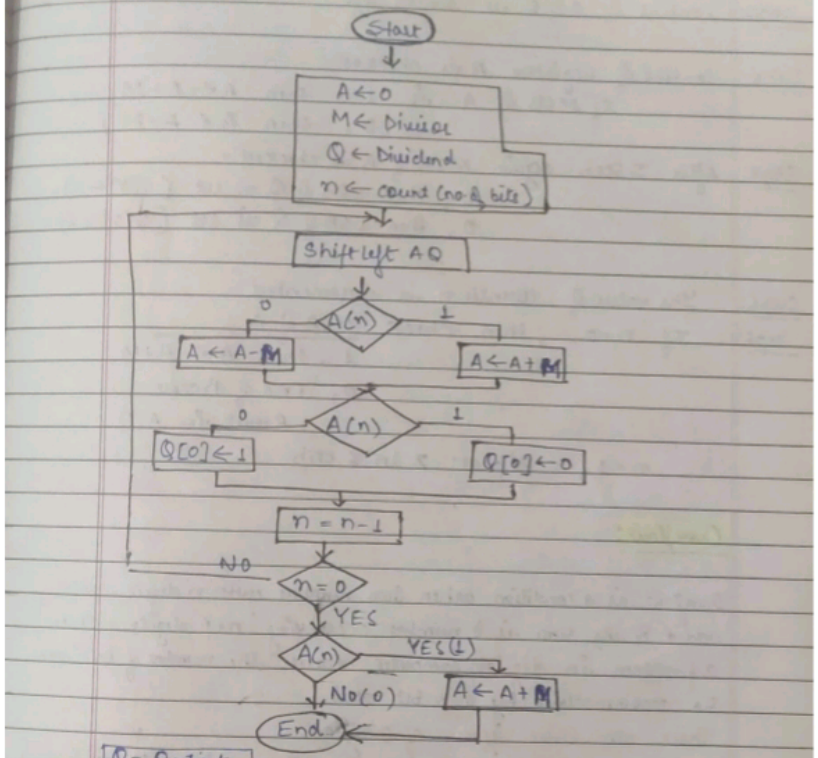
NON RESTORING DIVISION ALGORITHM Ex Q = 1011, M = 0011

n = 4

n	M	A	Q	Action / Operation
4 (100)	00011	00000	1011	Initial
		00001	011?	Shl AQ
		11101		As A[n]=0, A←A−M
3 (011)		11110	0110	As A[n]=1, Q[0]←0
		11100	110?	Shl AQ
		00011		As A[n]=1, A←A−M
2 (010)		00111	1100	As A[n]=1, Q[0]←0
		01110	100?	Shl AQ
		00011		As A[n]=1, A←A−M
1 (001)		00010	1001	As A[n]=0, Q[0]←1
		00101	001?	Shl AQ
		11101		As A[n]=0, A←A+M
0 (000)		00010	0011	As A[n]=0, Q[0]←1

Remainder (2) — Quotient (3)

NON RESTORING DIVISION ALGORITHM



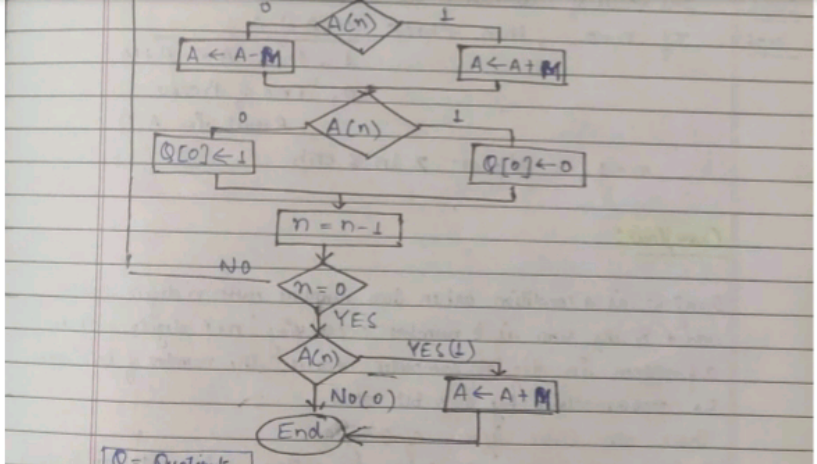
Q = Quotient
A = Remainder

STEPS :-

Step 1: First the register are initialized with Q = dividend, M = divisor, A=0, n = no. of bits in (Q) dividend. bits in A & M will be n+1 if bit in Q is n.

Step 2: Content of A & Q is shifted left.

Step 3: Content of register A is checked.
If MSB of A is 1, then A ← A+M
0, then A ← A−M



Q = Quotient
A = Remainder

STEPS :-

Step 1: First the register are initialized with Q = dividend, M = divisor, A = 0, n = no. of bits in (Q) dividend. bits in A & M will be n+1 if bit in Q is n.

Step 2: Content of A & Q is shifted left.

Step 3: Content of register A is checked.
If MSB of A is 1, then A ← A+M
0, then A ← A-M

Step 4: After 3 step again MSB of A is checked -
1, then LSB of Q is set (Q[0] ← 0)
0, then LSB of Q is set (Q[0] ← 1)

Step 5: The value of counter n is decremented.

Step 6: If n = 0, then check MSB of A
1, then A ← A+M
0, End of Process
Result in A, Q

n = 1, Repeat 2 to 6 step.

Overflow :

Overflow is a condition when two numbers with n digits are added & the sum is a number occupying n+1 digits. It is a problem in digital computer because the number of bits can be accommodated by a n-bit word.

There are three types of overflow -

1. Positive overflow - It refers to integer representations & the number that is larger than that can be represented in given no.
2. Exponent Overflow - It refers to floating point representation & to a positive number (exponent) that exceeds the max. possible exponent value.
3. Significant Overflow - It occur when the add^n of two significant no of the same sign result in a carry out of the most significant bit.

Steps: The value of counter n is decremented.

Steps: If n=0, then check MSB of A

1, then A ← A+M

0, End of Process

Result in A, Q

n=1, Repeat 2 to 6 step.

Overflow:

Overflow is a condition when two numbers with n digits are added & the sum is a number occupying n+1 digits. It is a problem in digital computer because the number of bits can be accommodated by a n-bit word.

There are three types of overflow -

1. Positive overflow - It refers to integer representations & to a number that is larger than that can be represented in given no. of bits.
2. Exponent Overflow - It refers to floating point representations & to a positive number (exponent) that exceeds the max. possible exponent value.
3. Significant Overflow - It occurs when the add" of two significant no of the same sign result in a carry out of the most significant bit.

8 / 22

Date:- / /

Page:-

Logic Microoperations - It specify binary operations for strings of bit stored in registers. These operations consider each bit of the register separately & treat them as binary variables.

Ex P: $R_1 \leftarrow R_1 \oplus R_2$

Operation means for P=1, perform $R_1 \oplus R_2$ operation & store it in R_1.

Let $R_1 = 1010$

$R_2 = 1100$

0110 — Content of R_1 if P=1

Here, bit by bit EXOR operation is performed.

Ex-OR operation
$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

Special symbols - $\vee$ → OR microoperation (logical OR)

$\wedge$ → AND microoperation (logical AND)

Ex: P: $R_1 \leftarrow R_2 + R_3$, $R_4 \leftarrow R_5 \vee R_6$

↳ Add operation ↳ OR operation.

List of Logic Microoperations -

There are 16 different logic operations that can be performed with two binary variables. They can be determined from all possible truth tables obtained with two binary variables. In this table, each of the 16 columns F_0 through F_{15} represents a truth table of one possible Boolean function for the two variables x & y.

Date:- / /

Page:-

F_0 F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8 F_9 F_{10} F_{11} F_{12} F_{13} F_{14} F_{15}

Overflow is a condition when two numbers with n digits are added & the sum is a number occupying n+1 digits. It is a problem in digital computer because the number of bits can be accommodated by a n-bit word.

There are three types of overflow -

1. Positive overflow - It refers to integer representations & the number that is larger than that can be represented in given no. of bits.
2. Exponent Overflow - It refers to floating point representations & to a positive number (exponent) that exceeds the max. possible exponent value.
3. Significant overflow - It occurs when the add^n of two significant no of the same sign result in a carry out of the most significant bit.

Date:- / /
Page:-

Logic Microoperations - It specify binary operations for strings of bit stored in registers. These operations consider each bit of the register separately & treat them as binary variables.

Ex P: $R_1 \leftarrow R_1 \oplus R_2$

Operation means for P=1, perform $R_1 \oplus R_2$ operation & store it in R_1.

Let $R_1 = 1010$
$R_2 = 1100$
0110 — content of R_1 if P=1

Here, bit by bit EXOR operation is performed.

Ex-OR operation
$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

Special symbols - $\vee$ → OR microoperation (logical OR)
$\wedge$ → AND microoperation (logical AND)

Ex: P : $R_1 \leftarrow R_2 + R_3$, $R_4 \leftarrow R_5 \vee R_6$

$R_2 + R_3$ → Add operation
$R_5 \vee R_6$ → OR operation.

List of Logic Microoperations -

There are 16 different logic operations that can be performed with two binary variables. They can be determined from all possible truth tables obtained with two binary variables. In this table, each of the 16 columns F_0 through F_{15} represents a truth table of one possible Boolean function for the two variables x & y.

Date:- / /
Page:-

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

The 16 logic microoperation are derived from these functions by replacing variable x by the binary content of register A & variable y by the binary content of register B.

Let $R_1 = 1010$
$R_2 = 1100$
0110 — content of R_1 if $P=1$

Here, bit by bit EXOR operation is performed.

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

Special symbols - $\vee \rightarrow$ OR microoperation (logical OR)
$\wedge \rightarrow$ AND microoperation (logical AND)

Ex: P : $R_1 \leftarrow R_2 + R_3$, $R_4 \leftarrow R_5 \vee R_6$
($+$: Add operation; $\vee$: OR operation.)

List of Logic Microoperations -

There are 16 different logic operations that can be performed with two binary variables. They can be determined from all possible truth tables obtained with two binary variables. In this table, each of the 16 columns F_0 through F_{15} represents a truth table of one possible Boolean function for the two variables x & y.

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

The 16 logic microoperation are derived from these functions by replacing variable x by the binary content of register A & variable y by the binary content of register B.

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = x\bar{y}$	$F \leftarrow A \wedge \bar{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = \bar{x}y$	$F \leftarrow \bar{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Ex NOR
$F_{10} = \bar{y}$	$F \leftarrow \bar{B}$	Complement B
$F_{11} = x + \bar{y}$	$F \leftarrow A \vee \bar{B}$	
$F_{12} = \bar{x}$	$F \leftarrow \bar{A}$	Complement A
$F_{13} = \bar{x} + y$	$F \leftarrow \bar{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

The 16 logic microoperation are derived from these functions by replacing variable x by the binary content of register A & variable y by the binary content of register B.

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = x\bar{y}$	$F \leftarrow A \wedge \bar{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = \bar{x}y$	$F \leftarrow \bar{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Ex NOR
$F_{10} = \bar{y}$	$F \leftarrow \bar{B}$	Complement B
$F_{11} = x + \bar{y}$	$F \leftarrow A \vee \bar{B}$	
$F_{12} = \bar{x}$	$F \leftarrow \bar{A}$	Complement A
$F_{13} = \bar{x} + y$	$F \leftarrow \bar{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

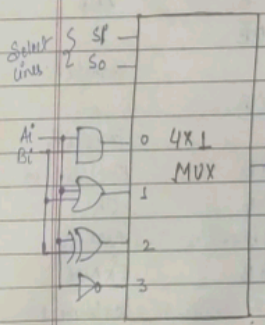
Date
Page:-

Hardware implementation -

The hardware implementation of logic microoperations requires logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function.

Figure shows one stage of a circuit that generates the four basic logic microoperations. It consists of four gates & a multiplexer. The diagram shows one typical stage with subscript i. For a logic circuit with n bits, the diagram must be repeated n times $i = 0, 1, 2, \ldots, n-1$. The selection variables are applied to all stages.

Figure: One stage of logic circuit



S_1 S_0	Output	Operation
0 0	$E = A \wedge B$	AND
0 1	$E = A \vee B$	OR
1 0	$E = A \oplus B$	XOR
1 1	$E = \bar{A}$	Complement

(a) Logic diagram (b) Functional Table

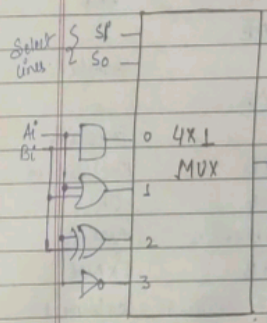
as function of bits of registers A & B

Applications of Logic Microoperations - Shows that how the bits of Register A [designated A] are manipulated by logic microoperations.

Selective Set operation: The selective set operation sets to 1 in register A when there are corresponding 1's in register B. It doesn't affect bit positions that have 0's in B.

basic logic microoperations. It consists of four gates & a multiplexer. The diagram shows one typical stage with subscript i. For a logic circuit with n bits, the diagram must be repeated n times $i = 0, 1, 2, \ldots, n-1$. The selection variables are applied to all stages.

Figure: One stage of logic circuit



S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	complement

(a) logic diagram (b) Functional Table

as function of bits of register

Applications of Logic Microoperations - Shows how the bits of Register A [designated A] are manipulated by logic microoperations.

1. Selective Set operation: The selective set operation sets to 1 in register A where there are corresponding 1's in register B. It doesn't affect bit positions that have 0's in B.

Date: / /
Page:

Ex 1010 A before
1100 B (logic operand)
1110 A after

Result shows it can be obtained from the logic OR operation

2. Selective complement - It complements bits in A where there are corresponding 1's in B. It doesn't affect bit positions that have 0's in B.

Ex 1010 A before
1100 B (logic operand)
0110 A after

Result shows it can be obtained from the logic EXOR operations.

3. Selective clear - It clears to 0 the bits in A only where there are corresponding 1's in B. It doesn't affect bit position that have 0's in B.

Ex 1010 A before
1100 B (logic operand)
0010 A after

Result shows it can be obtained from $A\overline{B}$ logic operation. $[A \wedge \overline{B}]$

4. Mask operations - It is similar to the selective clear except the bits of A are cleared only where there is corresponding 0's in B.

Ex 1010 A before
1100 B (logic operand)
1000

Result shows it can be obtained from logic AND operation

Date: / /
Page:

Ex 1010 A before
1100 B(logic operand)
1000

Result shows it can be obtained from logic AND operation.

Date:- / /
Page:-

6. Insert operation - It inserts a new value into a group of bits. This is done by first masking the bits & then ORing them with the required value.

The mask operation — AND microoperation
Insert operation — OR microoperation.

Ex - We have insert 1001 instead of 0010.

ⓐ first Mask

0110 1010 A before
0000 1111 B(mask) [Clear for 0's in B]
0000 1010 A after masking

ⓑ Insert New Value -

0000 1010 A before
1001 0000 B(insert) [ORing with required value]
1001 1010 A after insert

7. Clear operation - It compare the word in A & B, produce an all 0's result if the two numbers are equal. This is achieved by EXOR operation.

Ex : 1010 A before
1010 B
0000 A ← A ⊕ B.

12 / 22

Date:- / /
Page:-

IV Floating Point Arithmetic Operations -

Floating point number consist of two parts :
ⓐ a mantissa 'm'
ⓑ an exponent 'e'

We represent - $\pm m \times r^{e}$ ← exponent; r → radix; m → magnitude

	Ex - Number	Mantissa	Radix/Base	Exponent
1.	3×10^{6}	3	10	6
2.	110×2^{8}	110	2	8
3.	1.2345	12345	10	-4

ⓐ Mantissa may be a fraction or an integer.
ⓑ A floating point is normalised if the most significant digit of the mantissa is non-zero.
ⓒ A zero can't be normalised because it doesn't have non-zero digit. It is represented in floating point by all 0's in the mantissa & exponent.

achieved by EXOR operation

Ex : 1010 A before
1010 B
0000 A ← A ⊕ B.

Date:- / /
Page:-

IV Floating Point Arithmetic Operations -

Floating point number consist of two parts :
(a) a mantissa 'm'
(b) an exponent 'e'

We represent - $\pm m \times r^{e}$ (e ← exponent, r → radix, m ← magnitude)

	Ex - Number	Mantissa	Radix/Base	Exponent
1.	3×10^{6}	3	10	6
2.	110×2^{8}	110	2	8
3.	1.2345	12345	10	-4

(a) Mantissa may be a fraction or an integer.
(b) A floating point is normalised if the most significant digit of the mantissa is non-zero.
(c) A zero can't be normalised because it doesn't have non-zero digit. It is represented in floating point by all 0's in the mantissa & exponent.

The four possible operations performed as follows -
1. Addition
2. Subtraction
3. Multiplication
4. Division.

13 / 22

Addition & Subtraction -

1. ~~Addition~~ Arithmetic operation with floating point no. is more complicated than with fixed point numbers & their execution takes longer & requires more complex hardware.

Date:- / /
Page:-

2. The addition / subtraction between two numbers requires an alignment of the radix point since the exponent parts must be made equal before adding or subtracting the mantissa.

Considers the two numbers

$m_1 \times r^{e_1}$
$m_2 \times r^{e_2}$

$e_1 = e_2$ is must to perform addition or subtraction.

Addition & Subtraction Rule -

1. Choose the number with the smaller exponent & shift its mantissa right a number of steps equal to the difference in exponents (now both number exponents must be equal)

mantissa & exponent.

The four possible operations performed as follows -

1. Addition
2. Subtraction
3. Multiplication
4. Division.

Addition & Subtraction -

1. ~~Addition~~ Arithmetic operation with floating point no. is more complicated than with fixed point numbers & their execution takes longer & requires more complex hardware.

Date:- / /
Page:-

2. The addition / subtraction between two numbers requires an alignment of the radix point since the exponent parts must be made equal before adding or subtracting the mantissa.

Consider the two numbers

$m_1 \times r^{e_1}$

$m_2 \times r^{e_2}$

$e_1 = e_2$ is must to perform addition or subtraction

Addition & Subtraction Rule -

1. Choose the number with the smaller exponent & shift its mantissa right a number of steps equal to the difference in exponents (now both number exponents must be equal)
2. Set the exponent of the result equal to the exponent
3. Perform addition / subtraction on the mantissa & determine the sign of result.
4. Normalise the resulting value, if necessary.

Example : (i) A+B (ii) A-B

A = 225 B = 30

$A = 225 = 1110\,0001 = 1.1100001 \times 2^7$

$B = 30 = 1110 = 1.110 \times 2^3$

Number B has smaller exponent with different 4. Hence its mantissa is shifted right by 4 bits

$B = 1.110 \times 2^3$

$= 0.0001110 \times 2^7$

(Shift by 4 bits)

Date:- / /
Page:-

for addition -

$A = 1.1100001 \times 2^{111}$ (7 (binary value))

$+ B = 0.0001110 \times 2^{111}$

1.1101111×2^{111} ← Answer

for Subtraction -

$A = 1.1100001 \times 2^{111}$

$- B = 0.0001110 \times 2^{111}$

1.1010011×2^{111} ← Answer

Multiplication Rule -

1. Add the exponents & subtract bias (127, in case of single precision number & 1023 in case of double precision numbers)

A = 225 B = 30

A = 225 = 11100001 = 1.1100001 × 2^7

B = 30 = 1110 = 1.110 × 2^3

Number B has smaller exponent with different 4. Hence its mantissa is shifted right by 4 bits

B = 1.110 × 2^3

= 0.0001110 × 2^7

Shift by 4 bits

Date: / /
Page:

for addition - A = 1.1100001 × 2^{111} (7 (binary value))

+ B = 0.0001110 × 2^{111}

1.1101111 × 2^{111} ∴ Answer

for Subtraction - A = 1.1100001 × 2^{111}

− B = 0.0001110 × 2^{111}

1.1010011 × 2^{111} ∴ Answer

Multiplication Rules -

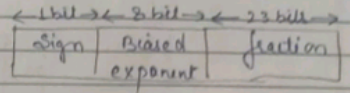
1. Add the exponents & subtract bias (127, in case of single precision number & 1023 in case of double precision numbers)
2. Multiply the mantissas & determine the sign of result.
3. Normalize the result.

Division Rules -

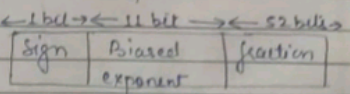
1. Subtract the exponents & add bias (127 in case of single precision no. & 1023 in case of double precision no.)
2. Divide the mantissas & determine the sign of result.
3. Normalize the result.

PS

Single Precision :- 32 bit format



Double Precision :- 64 bit format



15 / 22

Date: / /
Page:

V ARITHMETIC LOGIC UNIT -

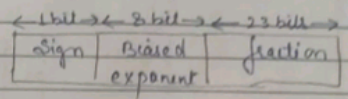
Instead of having individual registers performing the microoperations directly, computer system employ a number of storage register connected to a common operational unit called arithmetic logic unit abbreviated ALU.

To perform microoperations, the contents of specified register are placed in the inputs of the common ALU. The ALU performs an operation & result of operation is then transferred to a destination register.

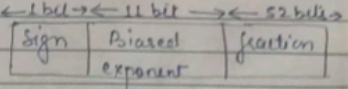
The ALU is combinational circuit so that the entire register transfer operation from the source register through ALU

Ps:-

Single Precision :- 32 bit format



Double Precision :- 64 bit format



Date:- / /
Page:-

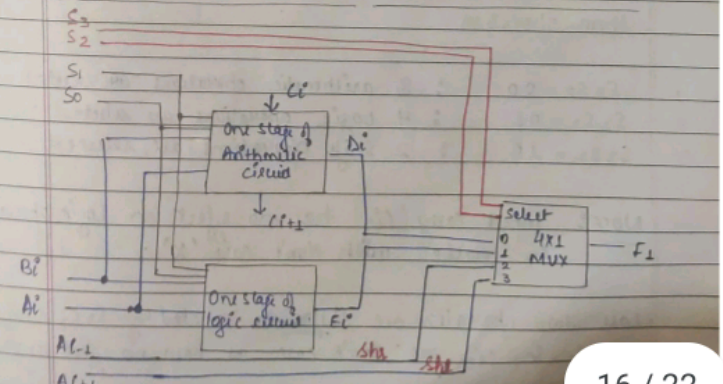
V ARITHMETIC LOGIC UNIT -

Instead of having individual registers performing the microoperations directly, computer system employ a number of storage register connected to a common operational unit called arithmetic logic unit, abbreviated ALU.

To perform microoperations, the contents of specified registers are placed in the inputs of the common ALU. The ALU performs an operation & result of operation is then transferred to a destination register.

The ALU is combinational circuit so that the entire register transfer operation from the source register through ALU & into the destination register can be performed during one clock pulse period. All the arithmetic, logic & shift circuits are combined into one ALU with common selection variables.

ONE STAGE OF ARITHMETIC LOGIC SHIFT UNIT -



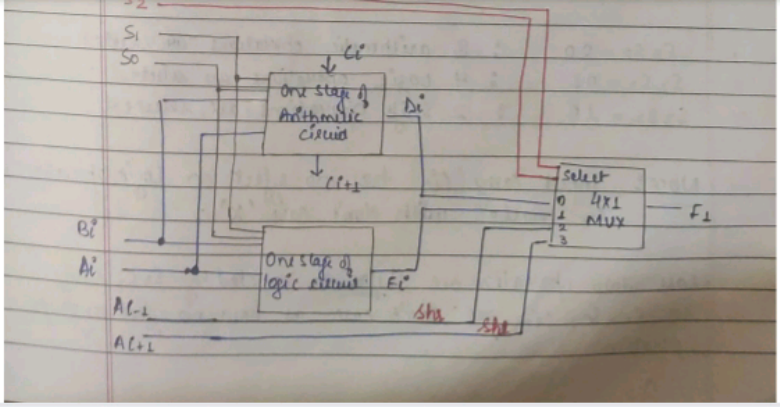
16 / 22

Date:- / /
Page:-

'i' designate typical stage

(a) Inputs Ai & Bi are applied to Arithmetic & logic unit.
(b) A micro operation is selected with inputs S1 & S0
(c) A 4x1 MUX at the output chooses b/w an arithmetic output in Di & logic output in Ei
(d) The data in the MUX are selected with inputs S3 & S2
(e) The other two data inputs to MUX are
Ai-1 : Shift right operation
Ai+1 : Shift left operation.

This circuit only shows 1 stage. for n-bit ALU same circuit should be repeated 'n' times.



Date:- / /
Page:-

'i' designate typical stage

(a) Inputs A_i & B_i are applied to Arithmetic & logic unit
(b) A micro operation is selected with inputs S_1 & S_0
(c) A 4x1 MUX at the output chooses b/w an arithmetic output in D_i & logic output in E_i
(d) The data in the MUX are selected with inputs S_3 & S_2
(e) The other two data inputs to MUX are
A_{i-1} : Shift right operation
A_{i+1} : Shift left operation.

This circuit only shows 1 stage. for n-bit ALU same circuit should be repeated 'n' times.

(f) The o/p carry C_{i+1} must of arithmetic stage must be connected to the input carry C_i of the next stage in sequence.
The input carry of first stage is C_{in}.

To perform 8 arithmetic, 4 logic & 2 shift operations we use 5 variable S_3, S_2, S_1, S_0 & C_{in} to select any of these operation.

1. $S_3 S_2 = 00$: 8 arithmetic operations are selected.
2. $S_3 S_2 = 01$: 4 Logic operations are selected.
3. $S_3 S_2 = 10$: 2 Shift operations are selected.

Note: Input carry 'Cin' has no effect on logic operation so masked with don't care 'x'.

Last two operation are shift so selected by $S_3 S_2$ only S_0 S_1, S_0, C_{in} are don't care as have no effect on shift operation.

Date:- / /
Page:-

Function Table for Arithmetic logic Shift Unit -

Operation Select					Operation	Function
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtract-
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	(x) don't care	$F = A \wedge B$	AND

2. $S_3S_2 = 01$: 4 Logic operations are selected.
3. $S_3S_2 = 10$: 2 Shift operations are selected.

Note: Input carry 'Cin' has no effect on Logic operation so masked with don't care 'x'.

Last two operation are shift so selected by S_3S_2 only so S_1, S_0, Cin are don't care as have no effect on shift operation.

Date:- / /
Page:-

Function Table for Arithmetic logic Shift Unit -

Operation Select					Operation	Function
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borr
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtract-
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	X (don't care)	$F = A \wedge B$	AND
0	1	0	1	X	$F = A \vee B$	OR
0	1	1	0	X	$F = A \oplus B$	XOR
0	1	1	1	X	$F = \overline{A}$	Complement A
1	0	X	X	X	$F = shr\ A$	Shift right A into F
1	1	X	X	X	$F = shl\ A$	Shift Left A into F

Date:- / /
Page:-

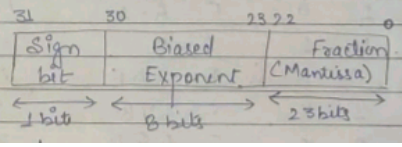
IEEE 754

VII IEEE STANDARD FOR FLOATI...

The standards for representing floating point numbers in 32 bits & 64 bits have been developed by the Institute of Electrical & Electronics Engineering [IEEE]

It represent number in Single Precision (32 bit) & double Precision (64 bit) forma

SINGLE PRECISION (32 bit) FORMAT -



1	0	×	×	×	F = shr A	Shift right A into F
1	1	×	×	×	F = shl A	Shift left A into F

Date:- / /
Page:-

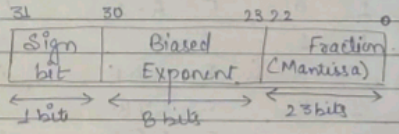
IEEE 754

VII IEEE STANDARD FOR FLOATING POINT NUMBERS

The standards for representing floating point numbers in 32 bits & 64 bits have been developed by the Institute of Electrical & Electronics Engineering [IEEE]

It represent number in Single Precision (32 bit) & double Precision (64 bit) format

SINGLE PRECISION (32 bit) FORMAT-



Stores sign (0 → +ve, 1 → -ve)

Contain exponent/radix value in 8 bits

Store mantissa in 23 bits representation

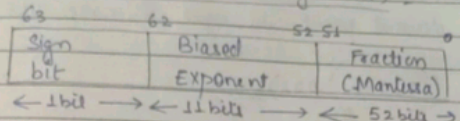
(a) 8 bit exponent ranges from 0 to 255 for normal values. Thus the actual exponent range is -126 to 127 obtained above value by adding 127 to actual range.

(b) The value 0 & 255 are used to indicate the floating point values of 0 & infinite respectively.

Date:-
Page:-

19 / 22

DOUBLE PRECISION (64 bit FORMAT)



(a) 11 bit exponent ranges from 0 to 2047 for normal values. Thus the actual range is -1022 to 1023.

Note- Consider biased exponent

Stores sign (0 → +ve, 1 → -ve)

contain exponent/radix value in 8 bits

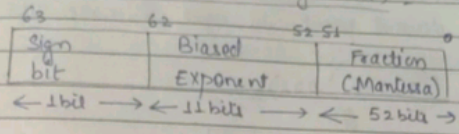
store mantissa in 23 bits representation

(a) 8 bit exponent ranges from 0 to 255 for normal value. Thus the actual exponent range is -126 to 127 obtained above value by adding 127 to actual range

(b) The value 0 & 255 are used to indicate the floating point values of 0 & infinite respectively.

Date:- / /
Page:-

DOUBLE PRECISION (64 bit FORMAT)



(a) 11 bit exponent ranges from 0 to 2047 for normal values. Thus the actual range is -1022 to 1023.

Note- Consider biased exponent in binary format.

Example -

1 Represent $(85.125)_{10}$ in Single Precision & double Precision formats.

Steps: $(85.125)_{10}$ Convert to binary.

2	85	
2	42	1
2	21	0
2	10	1
2	5	0
2	2	1
2	1	0
	0	1

$(85)_{10} = (1010101)_2$

$(0.125)_{10} \longrightarrow (\quad)_2$

$0.125 \times 2 = 0.250$	0	$(001)_2$
$0.25 \times 2 = 0.500$	0	
$0.5 \times 2 = 1.0$	1	

$(85.125)_{10} \longrightarrow (1010101.001)_2$

Date:-

Step2 Normalize the number.

Shift decimal point to right & make MSB bit 1 after

$(1010101.001)_2$

$\Downarrow$

1.01010101001×2^6

Step3 Single Precision format

Compare

1 Represent $(85.125)_{10}$ in Single Precision & double Precision formats.

Step1: $(85.125)_{10}$ Convert to binary.

2	85	
2	42	1
2	21	0
2	10	1
2	5	0
2	2	1
2	1	0
	0	1

$(85)_{10} = (1010101)_2$

$(0.125)_{10} \longrightarrow (\quad)_2$

$0.125 \times 2 = 0.250$	0	$(001)_2$
$0.25 \times 2 = 0.500$	0	
$0.5 \times 2 = 1.0$	1	

$(85.125)_{10} \longrightarrow (1010101.001)_2$

Step2 Normalize the number.

Shift decimal point to right & make MSB bit 1 after bit

$(1010101.001)_2$

$\Downarrow$

1.01010101×2^6

Step3 Single Precision format → Compare

$1.M\ 2^{e-127}$

Compare step 2 & step 3 equation.

$M = 01010101$

$e-127 = 6$

$e = (133)_{10} = (10000101)_2$

2	133	
2	66	1
2	33	0
2	16	1
2	8	0
2	4	0
2	2	0
2	1	0
	0	1

fill format of Single Precision.

31	30 ... 23	22 ... 0
0	10000101	01010100100000...0

upto 23 for leftover bit put zeros

Step4 Double Precision format

$1.M\ 2^{e-1023}$ compare with Step2 eq^n.

$M = 01010101$

$e-1023 = 6$

$e = 1029 = (10000000101)_2$

fill format of Double Precision

63	62 ... 52	51 ... 0
0	10000000101	0101010010...0

Ques $(1259.125)_{10}$

$1 \cdot M \; 2^{e-127}$

Compare step 2 & step 3 equation.

$M = 01010\,1001$

$e - 127 = 6$

$e = (133)_{10} = (10000101)_2$

2 | 133
2 | 66 | 1
2 | 33 | 0
2 | 16 | 1
2 | 8 | 0
2 | 4 | 0
2 | 2 | 0
2 | 1 | 0
0 | 1

fill format of Single Precision.

31	30 ... 23	22 ... 0
0	10000101	01010100100...0

(upto 23 for left over bit put zero.)

Step4 Double Precision format

$1 \cdot M \; 2^{e-1023}$ compare with step 2 eq^n.

$M = 01010\,1001$

$e - 1023 = 6$

$e = 1029 = (10000000101)_2$

Date:- / /
Page:-

fill format of Double Precision

63	62 ... 52	51 ... 0
0	10000000101	01010100100...0

Ques: $(1259.125)_{10}$

Step1 $(1259)_{10} = (10011101011)_2$

$(0.125)_{10} = (001)_2$

$(1259.125)_{10} = (10011101011.001)_2$

Step2 Normalize the no.

$1.0011101011001 \times 2^{10}$

Step3 Single Precision format → $1 \cdot M \times 2^{e-127}$

$M = 00111010\,11001$

$e - 127 = 10$

$e = (137)_{10} = (10001001)_2$

31	30 ... 23	22 ... 0
0	10001001	00111010110010...0

Step4 Double Precision format → $1 \cdot M \times 2^{e-1023}$

$M = 00111010\,11001$

$e - 1023 = 10$

$e = (1033)_{10} = (10000001001)_2$

63	62 ... 52	51 ... 0
0	10000001001	00111010110010...0