Pipelining

- Need of Pipelining

→ CPU perfamance must be high is today's requirement.

How it is possible?

1) Circuit may be change and install the fastest circuit in CPU.

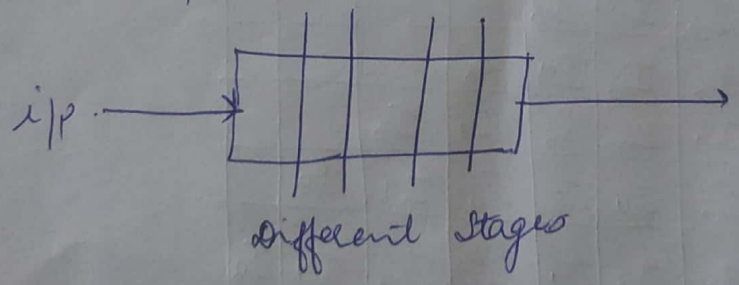
2) Hardware is arrange in such a way that multiple instru execute at a time.

If i3 → i5 → i7 cost ↑ ses

so from cost pt of view, which method is used ~~if~~ such that cost will not ↑se and existing hardware will change to ↑se the perfomance. for this purpose, pipelining is used

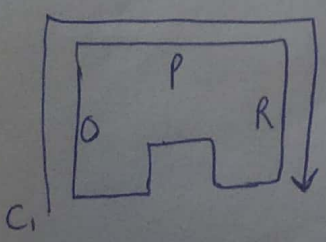
Pipelining help to not to change the ckt, ckt must be designed as a pipe.

eg



Every stage depends on each other.

Deine Theory



$C_1 \rightarrow$	O	P	R					
$C_2 \rightarrow$	-	O	P	R				
$C_3 \rightarrow$	-	-	O	P	R			

$\frac{5}{3} = 1.6$

$C_1 \rightarrow$	O	P	R					
$C_2 \rightarrow$	-	-	-	O	P	R		
$C_3 \rightarrow$	-	-	-	-	-	-	O	P	R

$\frac{9}{3} = 3$

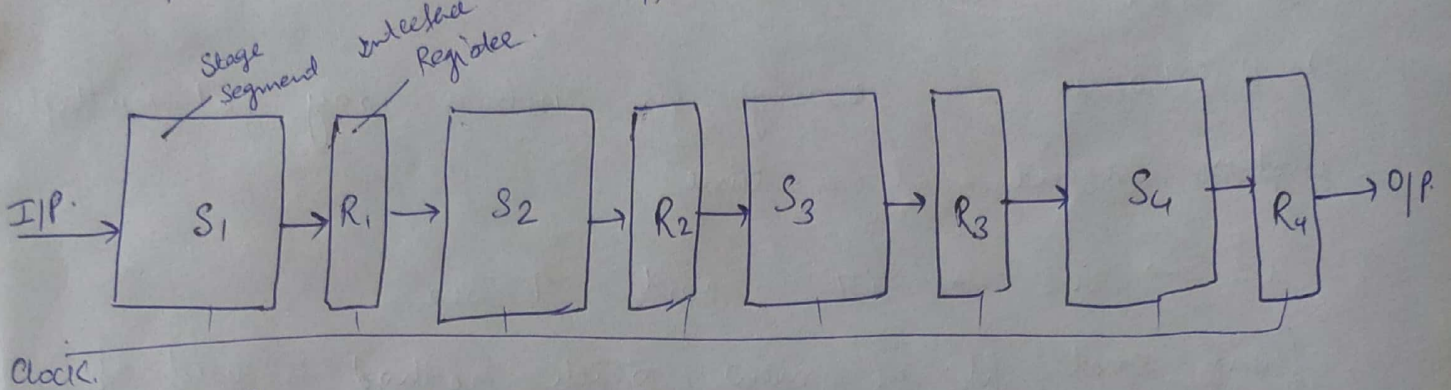
4.

• Definition of Pipelining

→ Pipelining is a process of arrangement of hardware elements of CPU such that its overall performance is ↑ sed.

→ Simultaneous execution of more than one instruction takes places in pipelined processor.

→ Multiple instrn are overlapped in execution.



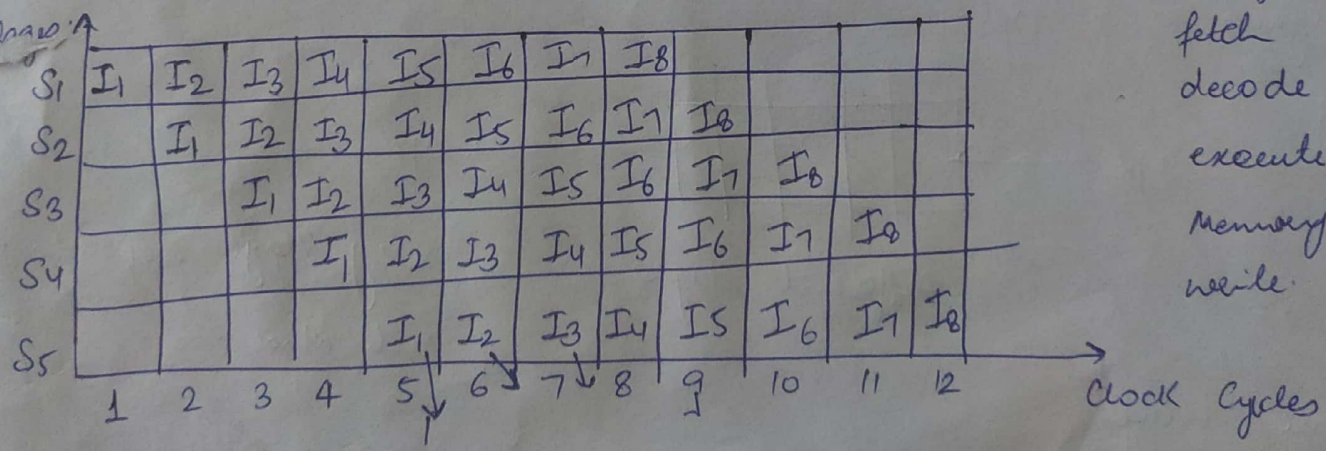
four Segment pipeline.

Space time diagram:

Instn Execution in pipeline architecture.

P = 8 I $I_1 - I_8$

5 Stage
fetch
decode
execute
Memory
write.



1 Instrn — 5

8 — 5 × 8 = 40 clock cycles

Total Instn = n.

No. of stages = K.

First Instn will take = 1 X K

Rest (n-1)

Total No. of Clock Cycle required = K + (n-1)

5 + (8-1) = 12 ✓

Clock Per Instn

CPI ≅ 1 AIM of Piplened.

eg 1000 Instn.

5 + (1000 - 1) = 1004 ≅ 1

tn tp → clock cycle time

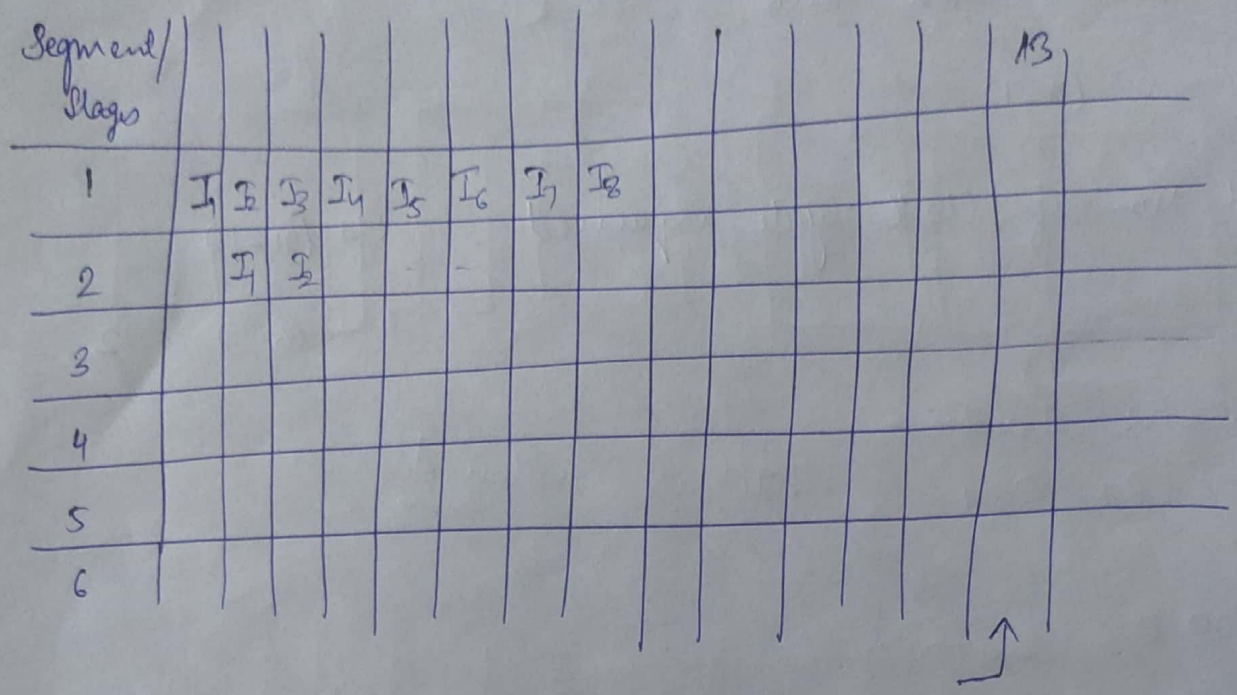
Speed up = $\frac{NP}{P} = \frac{n\ tn}{(K+n-1)\ tp}$.

$= \frac{40}{12} = 3$ ✓

Effeciency = $\frac{\text{Used}}{\text{Total Box.}}$

$= \frac{40}{60} = \frac{2}{3}$. ✓

Q Draw a space time diagram for six-segment pipeline showing the time it takes to process eight task.



$K + n - 1 = 6 + 8 - 1 = 13$ cycles

Q A non-pipelined system takes 50 ns to process a task. The same task process in six-segment pipeline with clock cycle of 10 ns. Determine speed up ratio of pipeline for 100 tasks

$t_n = 50$ ns, $K = 6$, $t_p = 10$ ns, $n = 100$

$$S = \frac{n\, t_n}{(K+n-1)\, t_p} = \frac{100 \times 50}{(6+100-1)\, 10} = 4.76.$$

$$S_{max} = \frac{t_n}{t_p} = \frac{50}{10} = 5$$

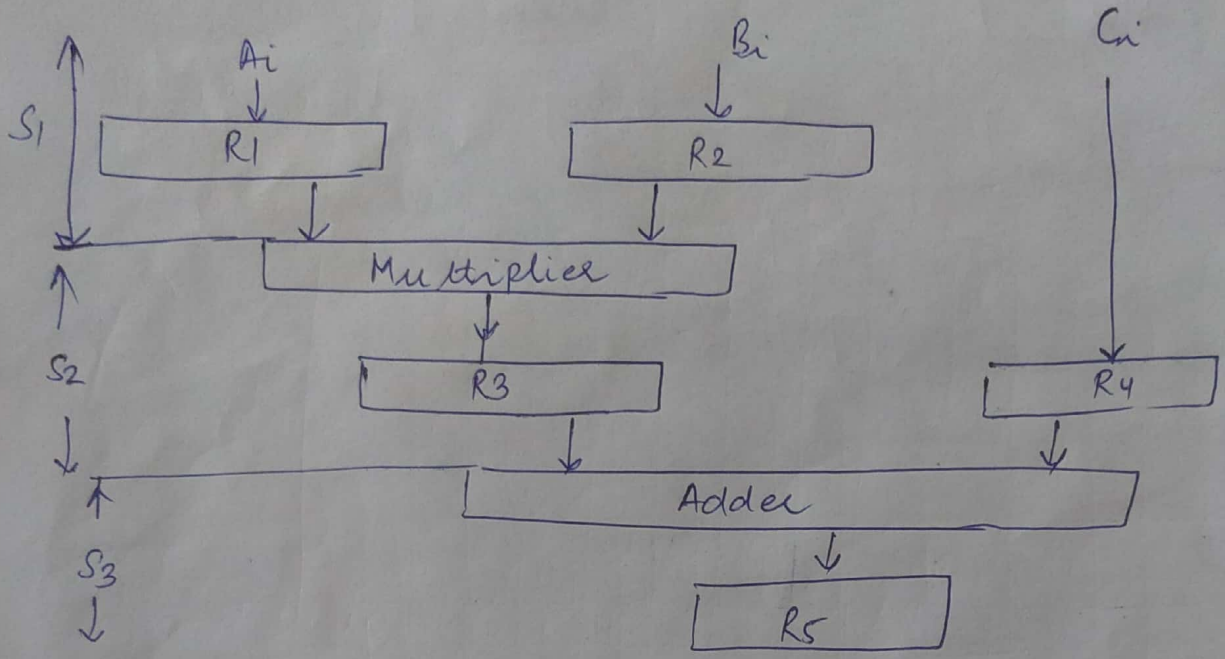
eg of Pipeline

$A_i * B_i + C_i$, $i = 1, \dots 7$

$R_1 \leftarrow A_i$, $R_2 \leftarrow B_i$

$R_3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$

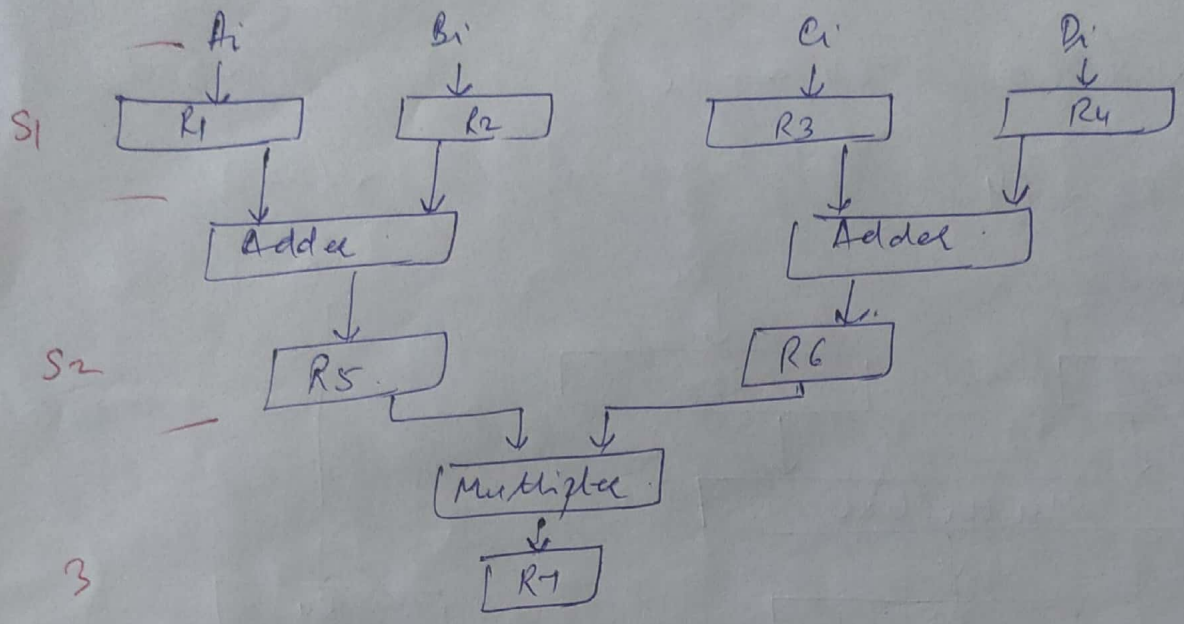
$R5 \leftarrow R3 + R4$



Clock Pulse	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1			
2	A_2	B_2	$A_1 * B_1$	C_1	
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8			$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9					

Q In certain specific computation it is necessary to perform arithmetic operation $(A_i + B_i) * (C_i + D_i)$

Specify the pipeline configuration to carry the task.



Arithmetic Pipeline

- To speed up the performance.

eg: Two floating pt numbers.

Steps to follow

1. Compare the exponent
2. Align the mantisa
3. Add or subtract
4. Normalise the result

eg: $X = 0.9504 \times 10^{3}$

$Y = 0.8200 \times 10^{2}$

After Alignment

$X = 0.9504 \times 10^{3}$

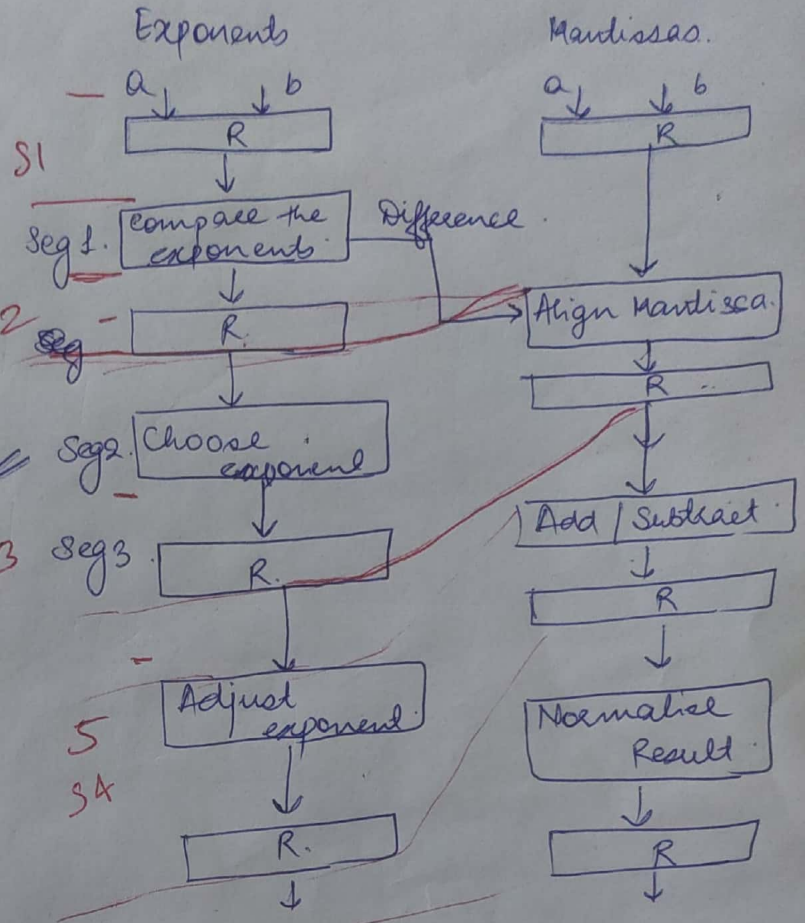
$Y = 0.08200 \times 10^{3}$

$Z = X + Y$

$= 1.0324 \times 10^{3}$

Normalise the result.

$= 0.10324 \times 10^{4}$



6 Pipeline for floating pt adder & subtraction

Flynn's Classification:-

(Parallel Processing in CO)

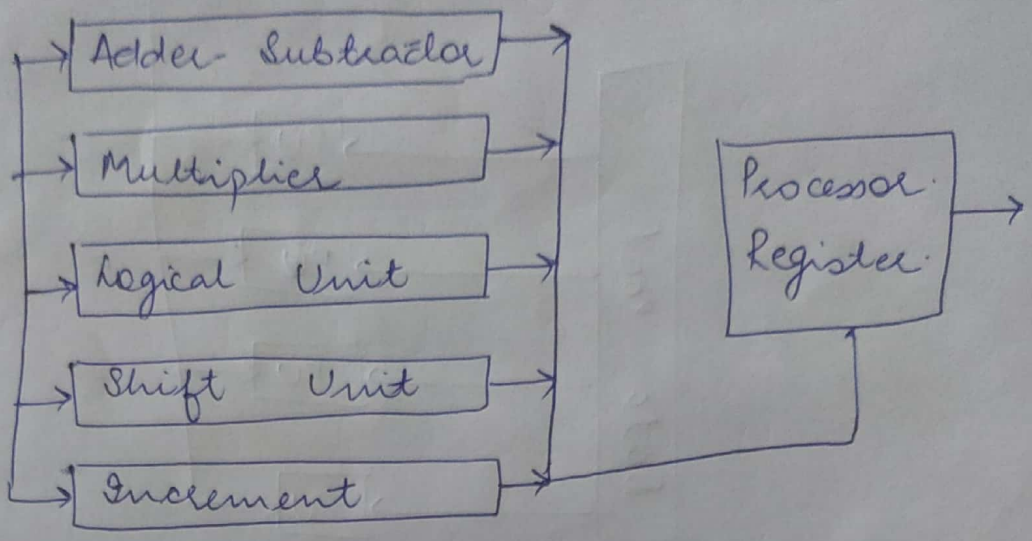


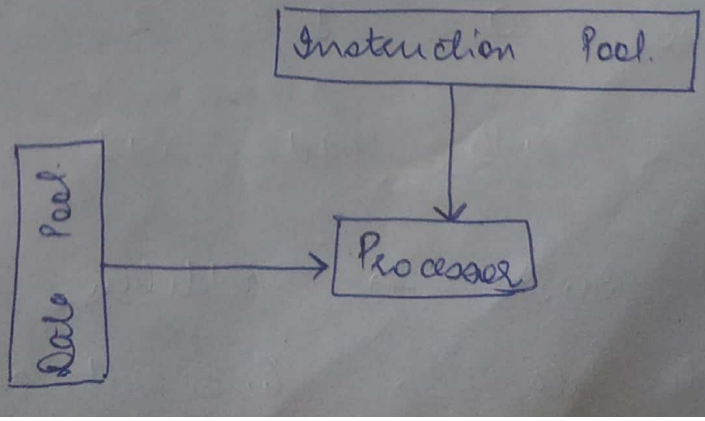
Fig: Processor with Multiple Unit (ALU)

- Michal Flynn
- Instruction & Data Stream

Classification

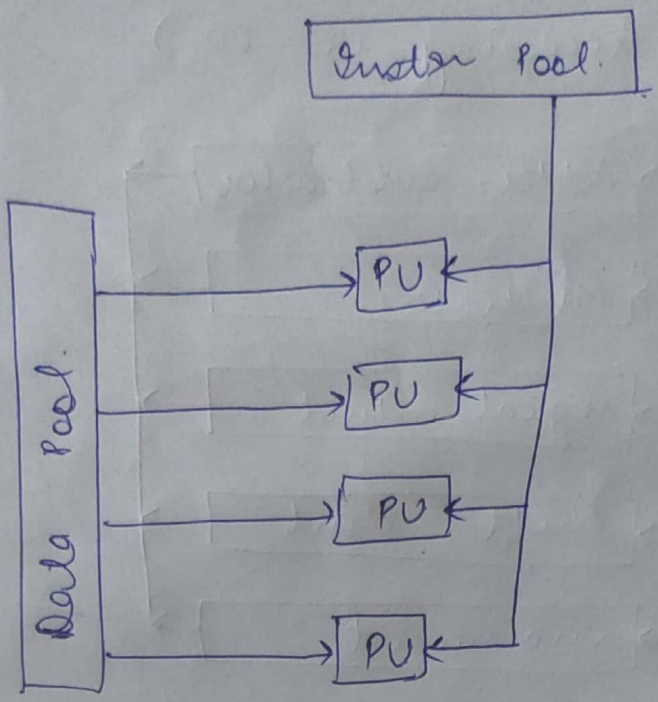
1. SISD — Single Instruction Single Data
2. SIMD — Multiple
3. MISD
4. MIMD

SISD



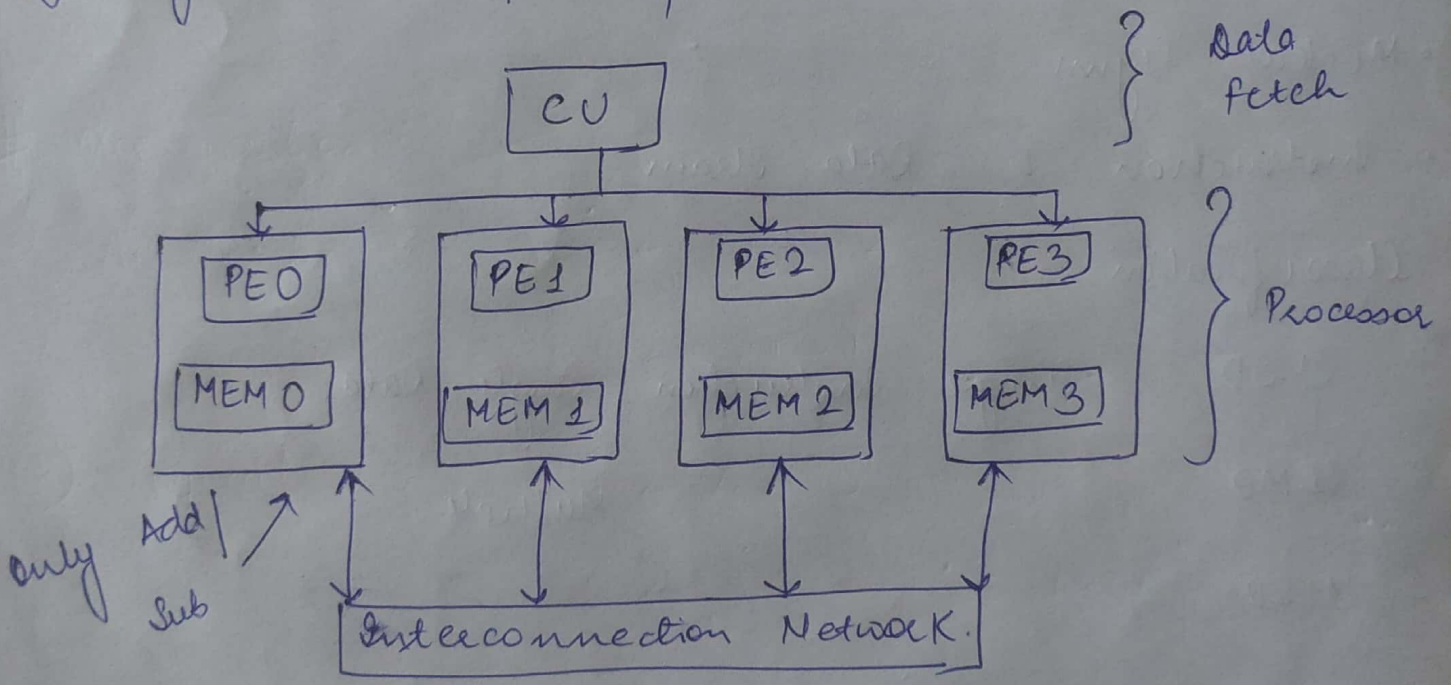
IBM
I^st Comp.

2. SIMD Architecture



• Lock Stepped.

eg Synchronous / Multiple Processor



• Processor 0 1 2 3

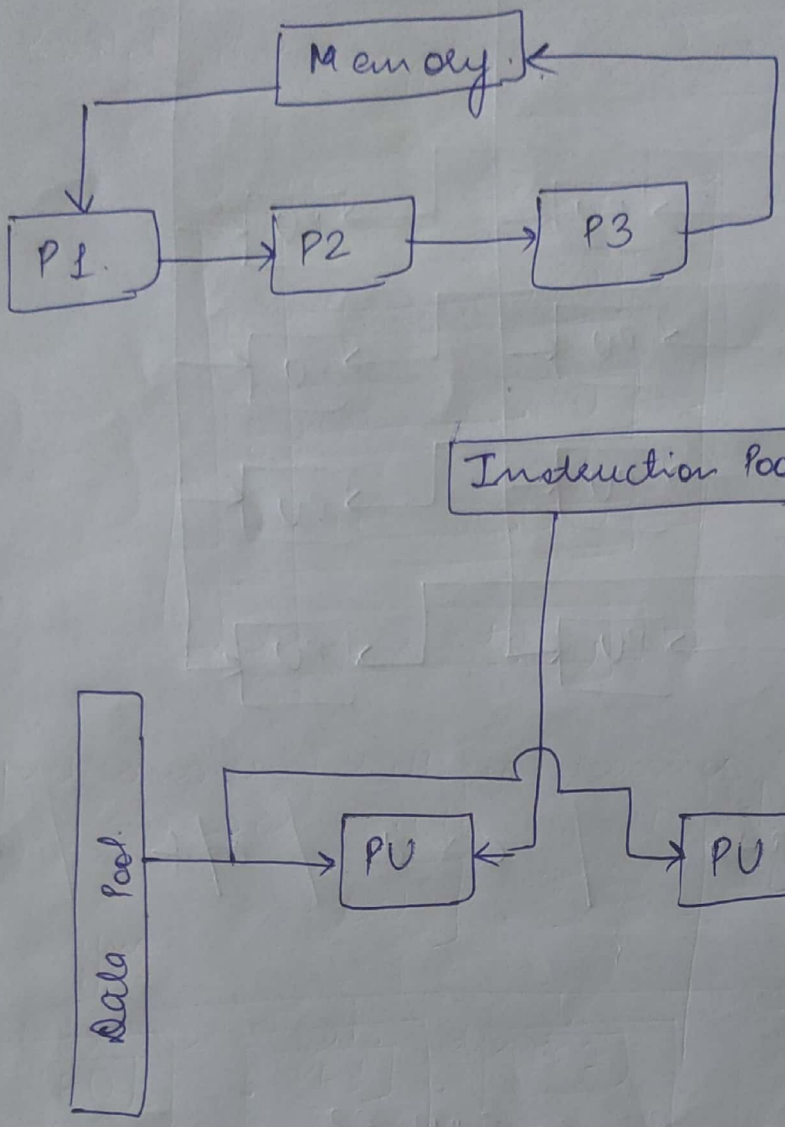
a+b c+d e+f g+h.

Single Instr → Addition

Multiple data → a, b, c, d, e, f, g, h

eg
ILLIAC-IV
→ 1952
→ I^st parallel computer

3. MISD Architecture



- Rarely used for Specified Applns

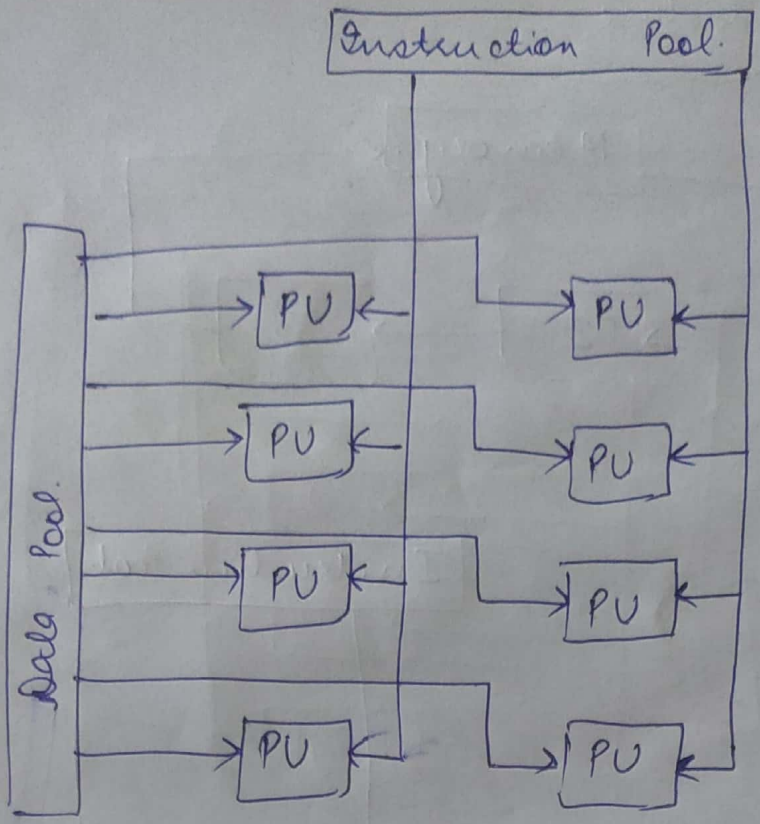
4. MIMD

- On Parallel Computer in real sense
- All Multiprocessor system fall under this class

eg Burrough's Comp.

→ 1962

→ 60 programs complied at a time.



Processor → a+b / c*d / e/f / m-n

Micro operation

① Arithmetic — R3 ← R#4 R2

R1 ← R1 − 1

② Logical — F ← A ∧ B AND.

F ← A Transfer A

③ Shift — R ← shl R shift left register R

R ← shr R — right " "

④ Logical shift — shl logical shift left

shr " " right

⑤ Circular Shift

cil circular shift left

cir " " right

Application

- Manipulate bits of register

eg Register A holds the 8 bit binary 1 1 0 1 1 0 0 1. Determine the B operation and logical microoper to be performed in order to change value in A to 0 1 1 0 1 1 0 1, 1 1 1 1 1 1 0 1.

- Selective Set

→ Bit pattern in B is used to set certain bits in A (Insert)

```
  1 1 0 0
  1 0 1 0
-----------
  1 1 1 0
```

A ← A + B (OR opn)

- Selective Complement

→ Bit pattern B is used to complement bits in A

```
  1 1 0 0
  1 0 1 0
-----------
  0 1 1 0
```

$A\bar{B} + \bar{A}B$

A ← A ⊕ B (XOR)

- Selective Clear

→ Bit pattern in B is used to clear certain bits in A.

```
  1 1 0 0
  1 0 1 0      A ← A · B̄
-----------
  0 1 0 0
```

- Mask operation

→ Bits of A are cleared only if there are corresponding 0's in B.

```
  1 1 0 0
  1 0 1 0   (A ← A · B)
-----------   AND Opn
  1 0 0 0
```

eg A register contain eight bit 0 1 1 0 1 0 1 0. Replace the left most bits by the value 1 0 0 1.

Soln:

```
  0 1 1 0  1 0 1 0      A
  0 0 0 0  1 1 1 1      B [AND becoz clear]
-----------------
  0 0 0 0  1 0 1 0
```

Insert new value

```
  0 0 0 0 1 0 1 0    A before
  1 0 0 1 0 0 0 0    (B Insert)
-------------------
  1 0 0 1 1 0 1 0    A + B.
```

Solⁿ (i)

```
  1 1 0 1 1 0 0 1    A (Given)
  1 0 1 1 0 1 0 0    XOR.
-------------------
  0 1 1 0 1 1 0 1
```

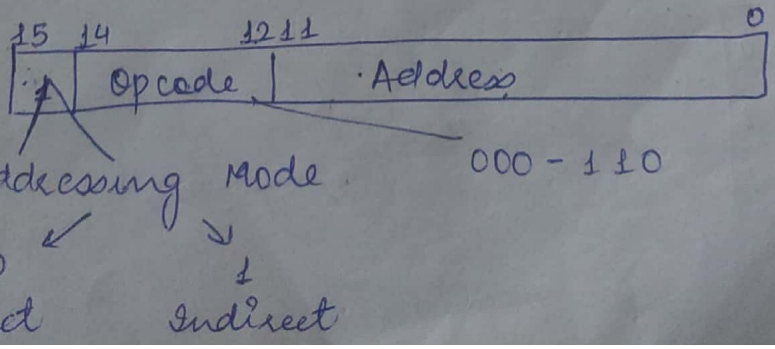
(ii)

```
  1 1 0 1 1 0 0 1
  0 0 1 0 0 1 0 0    XOR
-------------------
  1 1 1 1 1 1 0 1
```

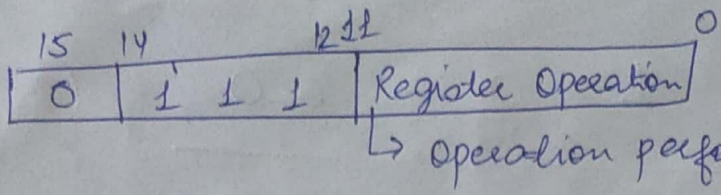
Computer Instructions :-

- The basic computer has three instruction code format.

(a) Memory- Reference Instruction

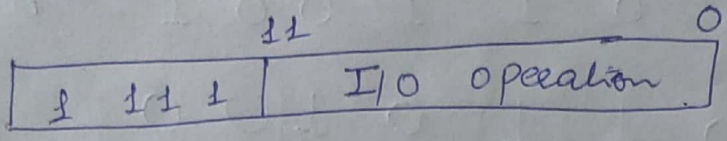


2. Register Reference



Reg Ref { 1 1 1 Opcode
0 Addressing mode

3. I/O Instruction



Symbol	Hexadecimal Code I=0	I=1	Description
AND	0XXX	8XXX	AND Memory Word to AC
ADD	1XXX	9XXX	Add " " "
LDA	2XXX	10XXX	Load " " "
STA	3XXX	11XXX	Store content of AC in Memory
BUN	4XXX	12XXX	Branch Unconditionally
BSA	5XXX	13XXX	Branch & Save Return Address
ISZ	6XXX	14XXX	Increment & Skip if zero

CLA	7800	Clear Accumulator
CLE	7400	" E
CMA	7200	Complement AC
CME	7100	" E
CIR	7080	Circulate Right
CIL	7040	" Left
INC	7020	Increment AC
SPA	7010	Skip Next Instr if AC is +ve
SNA	7008	" " " " " " -ve
SZA	7004	" " " " " " zero
SZE	7002	" " " " " E is zero
HLT	7001	Halt Computer

INP	F800	I/P to AC
OUT	F400	O/P ———
SKI	F200	Skip on I/P flag
SKO	F100	" " O/P flag
ION	F080	Interrupt ON
IOf	F040	" OFF

Instruction Cycle.

- A program is stored in the memory unit of the computer consist of sequence of instructions
- The program is executed in the computer by going through a cycle for each instruction.
- Each instruction cycle is subdivided into sequence of phases. as following

1) Fetch an instruction from memory.
   (Retrieving the information)
2) Decode the instruction
   (Which operation to be performed)
3) Read the effective address from memory.
   (If the instruction has indirect address
4) Execute the instruction

Upon completion of step 4, control goes back to step 1 to fetch, decode and execute the next instruction.

This process continues until a HALT instruction encountered.

PC - contains the address of next instruction to be executed

SC - If apply a clock pluse on sequence counter we get time T0

Step 1 → Start the Program SC ← 0 (Sequence Counter)

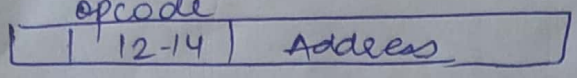
Step 2 → Initial time T_0 : AR ← PC (Program Counter placed in Address Register)

Step 3 → T_1 : IR ← M[AR], PC ← PC + 1 } fetch data from memory

(Instn is readed from memory) (IR → Instruction Register)

(Whatever the instn, decode it)

Step 4 → T_2 : decode opcode IR(12-14) } Decode.



AR ← IR (0-11)

I ← IR (15)

~~Steps~~

Whatever the data in the Memory is decoded in this step

Step 5. T_3 : decision taking cycle → MR / RR / IO

→ Indirect execution at T_4

direct " " T_3

Step 6. T_4

Program → Memory Unit

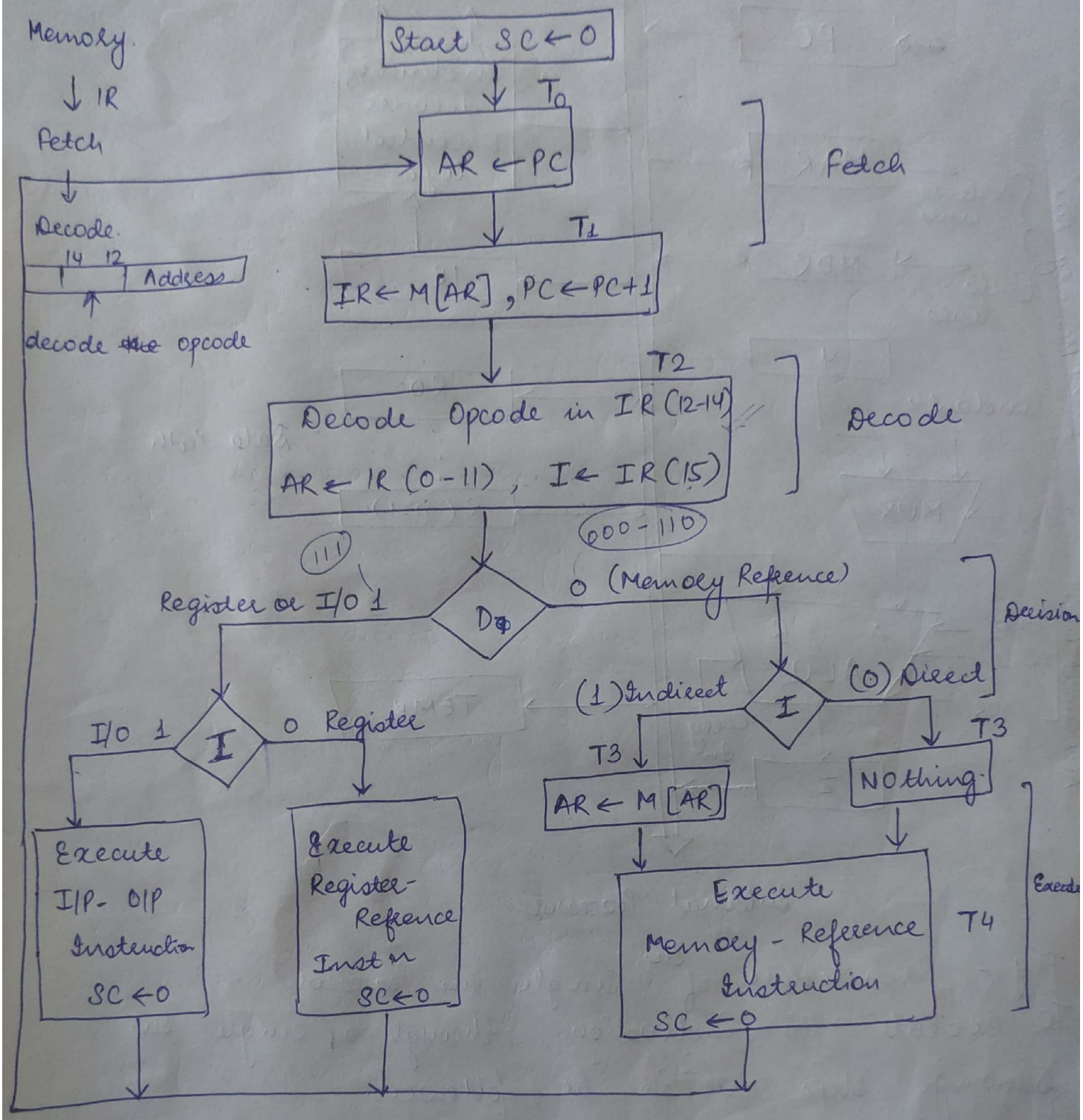
↓

Sequence of Instructions → Instruction Cycle.

Fetch | Decode | Decision | Execute.

T0T1	T2		
Fetch	Decode	EA	Execute Decision

Flow Chart of Instruction Cycle.



Execution of Complete Instruction

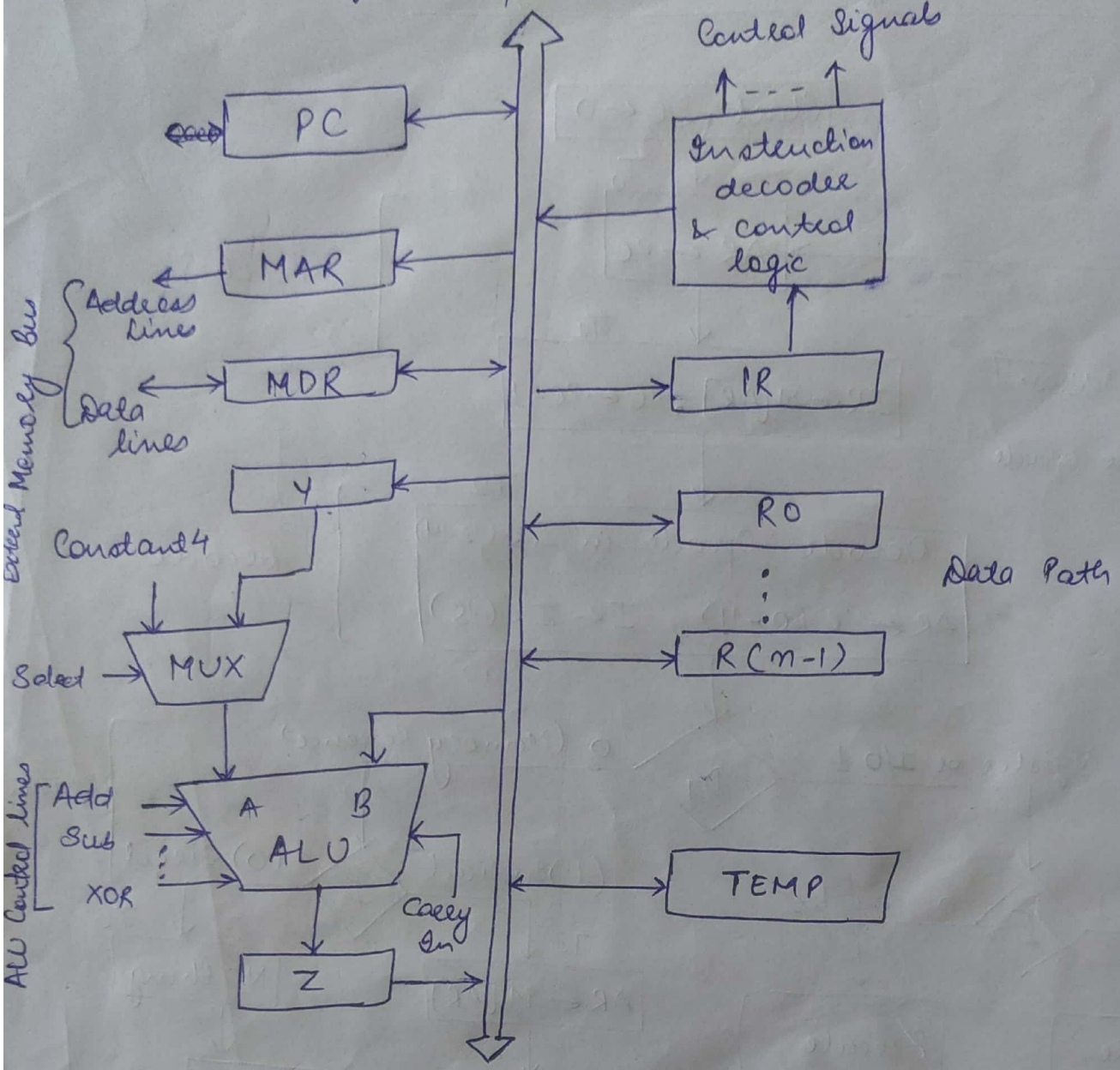


Fig:- Architecture of Single Bus Processor

To execute the instruction, through a single bus following steps are to be followed.

(1) Fetch the data

In figure

① R_0 to R_{n-1} are general purpose registers

② IR - Instruction / Indexed Register
It holds the instruction currently being executed

③ Instruction decoder - tells about the instruction to be performed

for fetching :- (i) AR ← PC

④ Y, Z & TEMP are temporary registers used by processor during the execution of some instruction.

⑤ PC : contains the address of next instruction to be executed.

⑥ MAR - Memory Address Register

⑦ MDR - All the data stored in Memory Data Register

eg ADD (R3), R1

↑ Indirect Add ↑ Direct Add

R3 → 1008

R1 → 10

Memory

Address	Content
1000	Add(R3),R1
1008	20

PC 1000

R3 1008

R1 10

IR Add (R3), R1

Steps 1) Fetch the instruction from memory

2) Fetch the first operand from memory.

3) Perform addition operation

4) Load the result into R1.

Control Sequence

1) PC_{out} , MAR_{in} , Read , Select 4 , Add , Z_{in}

2) Z_{out} , PC_{in} , WMFC

(Wait for memory function to complete)

3) MDR_{out} , IR_{in}

Fetching instruction in IR

4) $R3_{out}$, MAR_{in} , Read

5) $R1_{out}$, Y_{in} , WMFC

6) MDR_{out} , Select Y, Add , Z_{in}

7) Z_{out} , R_{in} , End.

Mux → 4 Select

A = 4

B = 1000

Z = 1004

PC → 1004 Instr

MDR_{out} → ADD(R3) , R1

eg Sequence of operation

Add content of register R1 to those of R2 and store result in R3.

1. R_1 out , Y_{in}

2. R_2 out , Select Y , Add , Z_{in}

3. Z_{out} , R_3 in

fetching a word from memory.

eg MOV (R1) , R2

→ MAR ← [R1]

→ Start Read operation on memory bus.

→ WMFC

→ Load MDR from memory bus

→ R2 ← [MDR]

Control Sequence

1. R_1 out , MAR in , Read

2. MDR in , WMFC

3. MDR_{out} , R2 in

Q Add the immediate number (NUM) to register (R1)

CS

(1) PC_{out}, MAR_{in}, Read, Select 4, Add, Z_{in}

(2) Z_{out}, PC_{in}, WMFC

(3) MDR_{out}, IR_{in}

(4) $IR_{NUM\ out}$, Y_{in}

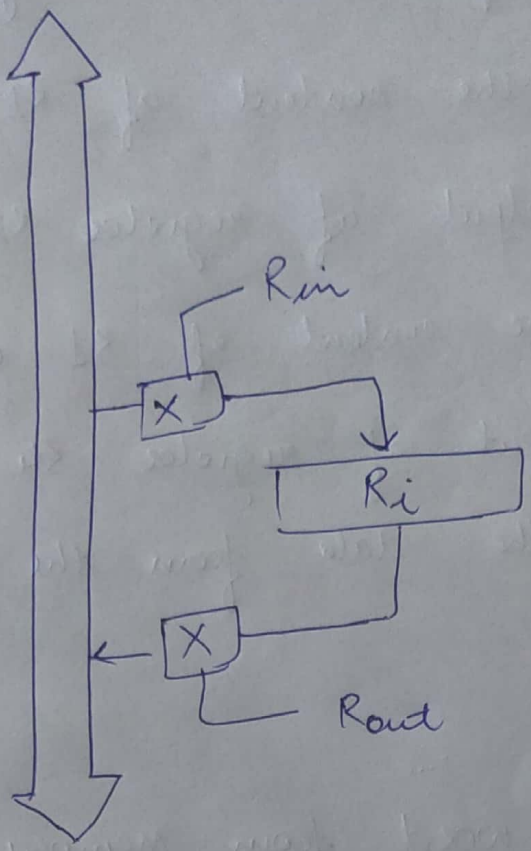
(5) R_1 out, Select Y, Add, Z_{in}

(6) ~~Add~~ Z_{out}, $R_{1\ in}$, End

Q Add the content of memory location NUM to register R_1.

CS.

Register Transfer



- I/P & O/P gates for register Ri are controlled by signals R_{in} & R_{out}.
- If R_{in} is set to 1, data available on common bus are loaded into R_i.
- If R_{out} is set to 1, content of register are placed on bus
- If R_{out} is set to 0, bus can be used for transferring data from other registers

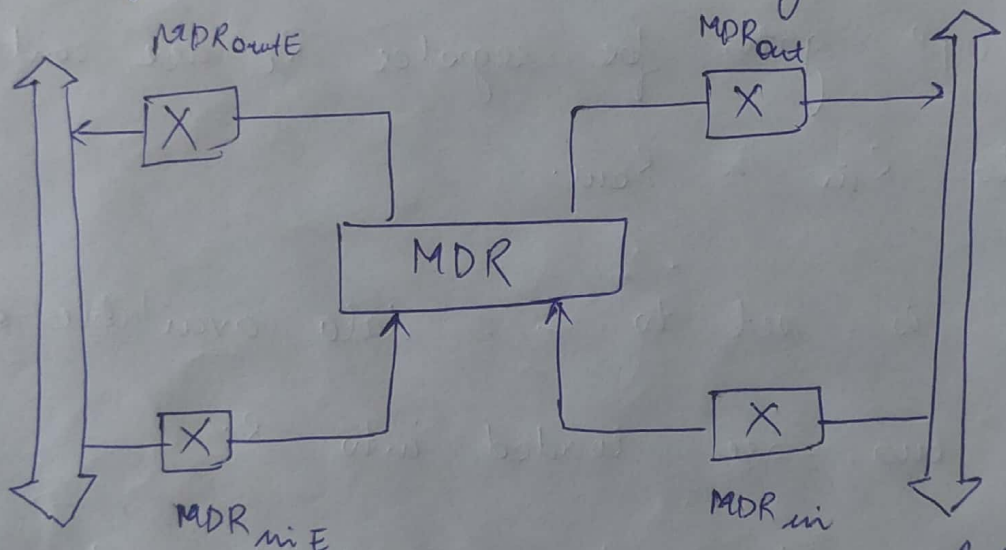
Data transfer b/w two registers.

→ Transfer the content of R1 to R4.

1) Enable output of register R1 by setting $R_{1out} = 1$. This place the content of R1 on the processor bus.

2) Enable input of register R4 by setting $R_{4in} = 1$. This loads the data from the processor bus into register R4.

fetching a word from memory.



Memory Hierarchy

- The memory unit is an essential component in any digital computer since it is needed for storing programs and data.
- Not all the accumulated information is needed by the processor at the same time. Therefore, it is more economical to use low-cost storage devices to serve as backup for storing the information that is not currently used by the CPU.
- The main memory unit the communicates directly with CPU is called main memory. Only programs and data currently needed by processor reside in main memory.
- Devices that provide back up storage are called auxillary ~~storage~~ memory.

→ eg Magnetic disk / hard disk (TB)
Magnetic Tape / Tape drives (PB/EB)

→ AM are used for storing programs, large data files, and other back up information

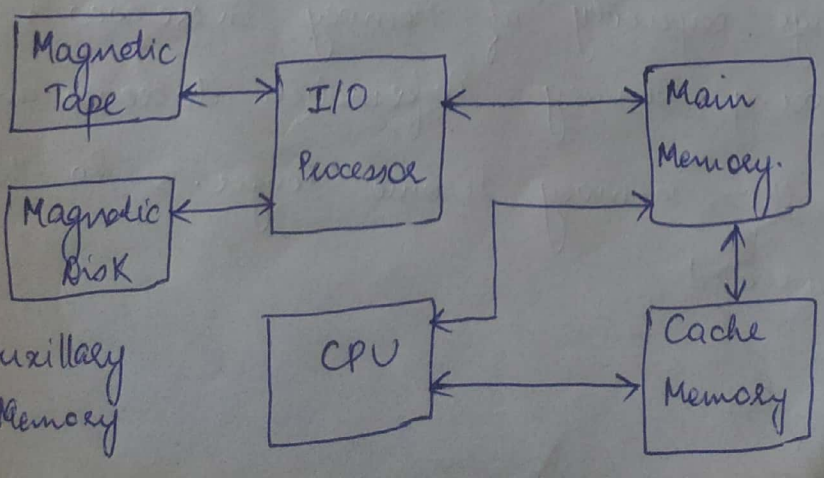


Fig:- Memory Hierarchy in Computer

The total memory capacity of computer can be visualized as hierarchy of components

→ Magnetic Tapes - used to store removable files.

→ Magnetic Disk - used as Backup storage.

→ Main memory occupies a central position to communicate directly with CPU and auxillary memory.

When programs in main memory are not needed by the CPU, they are transfered to auxillary memory., to provide space for currently used program and data.

Cache Memory-

A special very-high memory called a cache is used to increase the speed of processing by making current programs and data available at the CPU at a rapid rate.

The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations.

Note:- As the storage capacity of memory increases, the cost per bit for storing binary information decreases and the access time of the memory become longer. So the memory hierarchy is needed.

Auxillary Memory - large storage capacity, relatively inexpensive
- low access speed compared to main memory

Cache Memory - small, relatively expensive,
- very high access speed

24
53

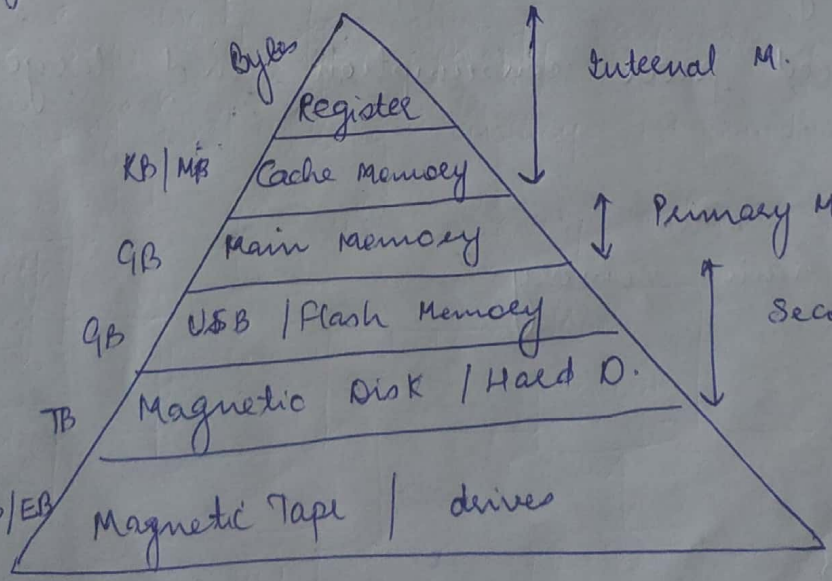
Therefore as the memory access speed increases, its cost increases.

Multiprogramming

Many operating system are designed to enable the CPU to process number of independent programs concurrently. This is known as multiprogramming.

Memory Management System

The part of computer system that supervise the flow of information b/w auxillary memory & main memory is known as memory management system.



1000 Tera byte = Petabyte / PB/EB Exa byte

1EB = 1000 PetaByte
1PB = 1000 Tera Byte

Main Memory (RAM)

- Central storage in computer system.
- Relatively large and fast memory.
- Used to store programs and data during computer operation.
- Integrated circuit RAM chips are available in two possible operating modes, static and dynamic.

→ Static RAM -

* Consist of internal flip-flop that store the binary information.
* Stored information remain valid as long as power is applied to the unit.
* Used in cache memory.

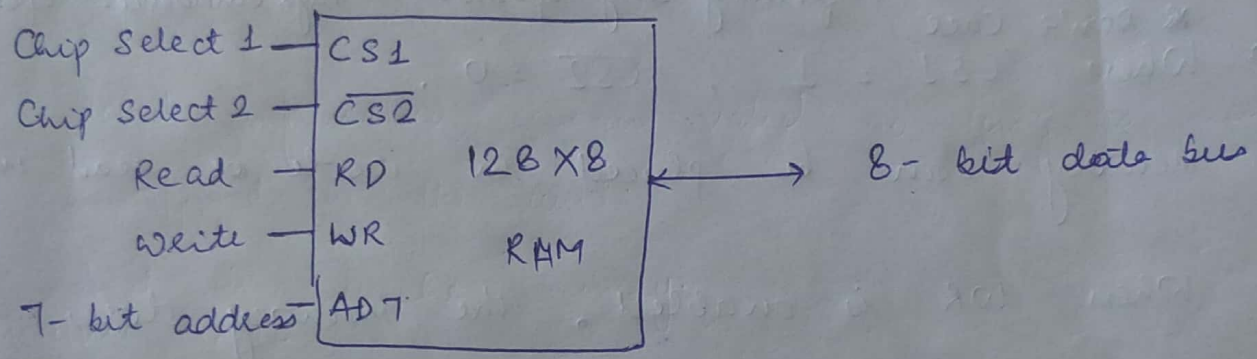
→ Dynamic RAM -

* Stores the binary information in the form of electric charges that are applied to capacitors.
* offers reduced power consumption and larger storage capacity.
* Used in main memory.

RAM & ROM Chips.

* Read - data transfer b/w main memory to CPU.
* Write - data transfer b/w CPU to main memory.

- RAM chip is better suited for communication with the CPU if it has one or more inputs that select the chip only when needed.
- RAM has bidirection data bus that allows the data transfer either from memory to CPU i.e read operation or CPU to memory i.e write operation
- Block diagram of RAM chip



→ The capacity of the memory is 128 words of eight bits (one byte) per word.

→ This requires 7- bit address and 8- bit bidirectional data bus.

→ The read and write input specify the memory operation and the two chip select (CS) control input are used for enabling the chip only when selected by processor.

* 2 Chip Select → $2^2 = 4$ RAM can be selected

- Functional Table of RAM

CS1	$\overline{CS2}$	RD	WR	Function	State of Bus
0	0	x	x	Inhibit	High Impedence
0	1	x	x	Inhibit	"
1	0	0	0	Inhibit	"
1	0	0	1	Write	Input data to RAM
1	0	1	0	Read	Output data from RAM
1	1	x	x	Inhibit	High Impedence

* Inhibit - Not performing any operation * High Impedence - Open Circuit
* Don't Care

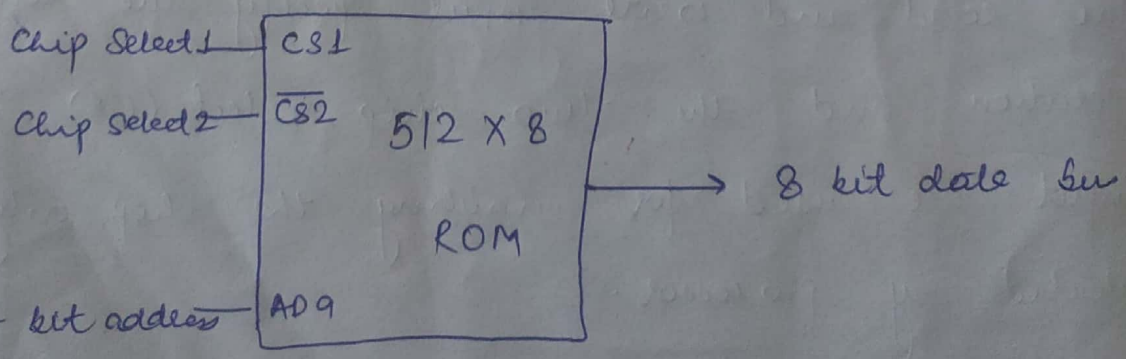
→ When CS1 = 1 , $\overline{CS2}$ = 0 ,

Memory can be placed in write or read mode.

→ When WR is enabled, the memory stores a byte from data into location

→ When RD is enabled, the content of selected bus is placed into the data bus.

- Block Diagram of ROM



→ Since ROM can only read, data bus can only be in output mode.

Note :- For the same-size chip, it is possible to have more bits of ROM than of RAM, because the internal binary cells of ROM occupy less space than RAM. For this reason, ROM specifies 512 byte, while RAM has only 128 bytes.

→ 9 address lines specifies 512 byte.

→ $CS1 = 1$ and $\overline{CS2} = 0$ for operation

Otherwise data bus is in high impedence state.

Memory Address Map.

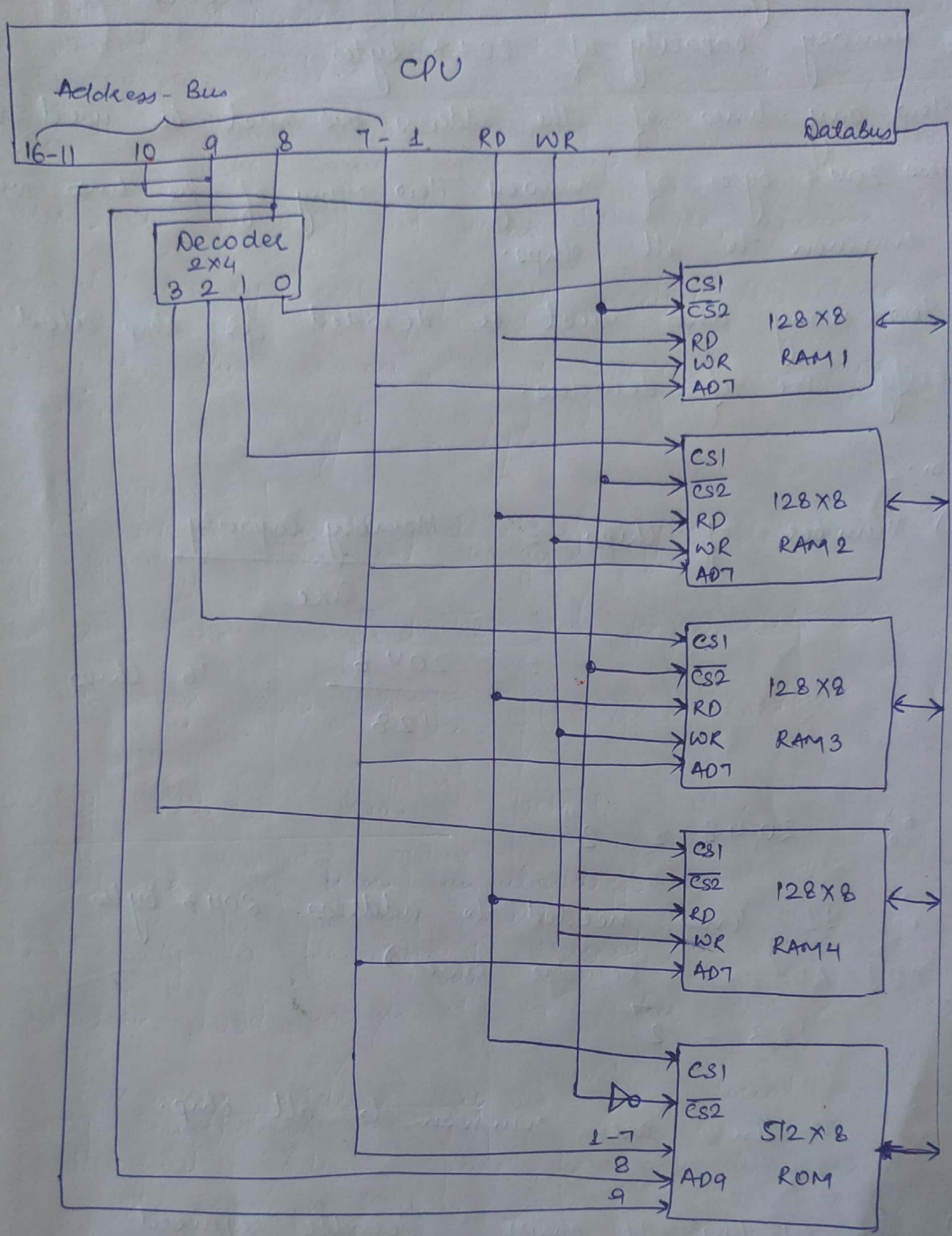
- The designer of the computer system must calculate the ammount of memory required for particular application and assign it to either RAM or ROM.
- The Memory address map is a pictorial representation of assigned address space for each chip in the system.
- eg 512 bytes RAM & 512 bytes ROM.

Component	Hexadecimal Address	Address Bus 10 9 8 7 6 5 4 3 2 1
RAM 1	0000-007F	0 0 0 x x x x x x x
RAM 2	0080-00FF	0 0 1 x x x x x x x
RAM 3	0100-017F	0 1 0 x x x x x x x
RAM 4	0180-01FF	0 1 1 x x x x x x x
ROM	0200-03FF	1 x x x x x x x x x x

- The x's are assigned to low-order bus lines.
- When line 10 is 0, CPU selects RAM
  When lines is 1, it selects ROM.
- There are 16 lines in address bus. The table shows only 10 because in this example other 6 are not used.

Memory Connection to CPU

Note:- The example shows gives an indication of interconnection complexity that can exist between Memory chips and CPU.



- 2 X 4 Decoder

Input		Output
0	0	RAM 1
0	1	RAM 2
1	0	RAM 3
1	1	RAM 4

Q a) How many 128 X 8 RAM chips are needed to provide a memory capacity of 2048 bytes.

b) How any lines of the address bus must be used to access 2048 bytes of memory. How many of the lines will be common to all chips.

c) How many lines must be decoded for chip select? Specify size of decoders.

Solⁿ:-

a) Number of chips = $\frac{\text{Memory Capacity}}{\text{Size}}$

$= \frac{2048}{128} = 16$ chips

b) ∵ $2048 = 2^{11}$.

∴ 11 lines needed to address 2048 bytes.

(AD1 - AD7, + Decoder lines)

$128 = 2^7$

∴ 7 lines are common to all chip.

c) For selecting 16 chips, decoder required is

4 X 16 decoder

Q for 4096 bytes of RAM and 4096 bytes of ROM. List the memory address map and indicate decoder size.

Solⁿ:-

Number of chips = $\frac{4096}{128}$

= 32 RAM chips

= $\frac{4096}{512}$

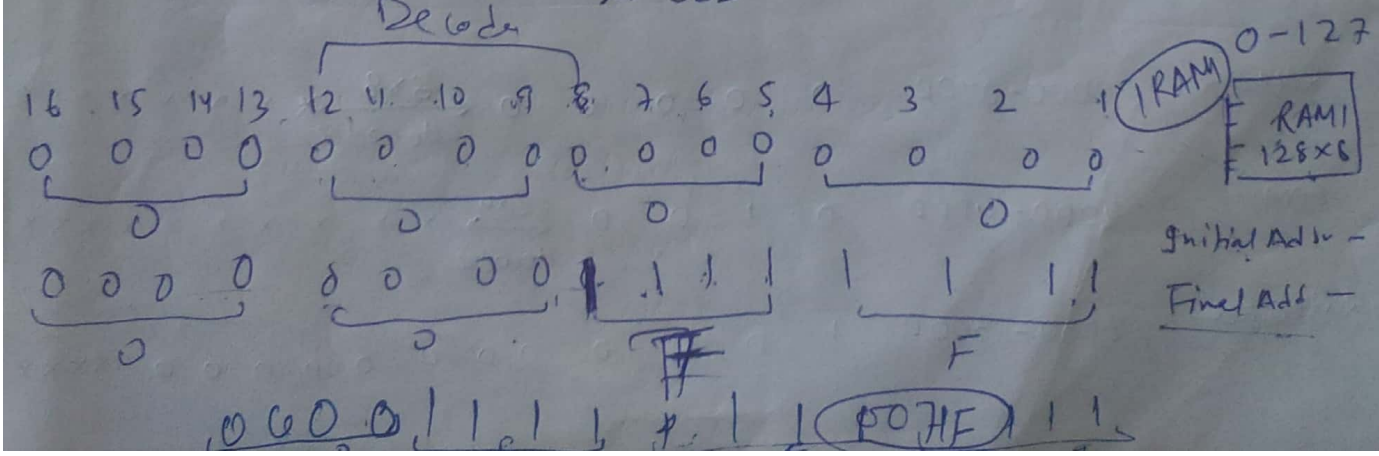
= 8 ROM chips.

Also, $4096 = 2^{12}$

12 common address lines & 1 to select between RAM & ROM.

Component	Address	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
RAM	0000-0FFF	0	0	0	0	← 5×32 decoder →											
ROM	1000-1FFF	0	0	0	1	← 3×8 decoder →											

to $\overline{CS2}$



8) A computer employs RAM chips of 256 X 8 and ROM chips of 1024 X 8. The system needs 2K bytes of RAM, 4K bytes of ROM and four interface units each with four registers. A memory mapped I/O configuration used. The two highest order bits of the address bus are assigned 00 for RAM, 01 for ROM and 10 for interface registers.

a) How many RAM & ROM chips are needed.

b) Draw a memory address map for the system.

c) Give the address range in hexadecimal for RAM, ROM & interface.

Solv:- a) RAM chips = $\frac{2048}{256}$ = 8 chips

ROM chips = $\frac{4096}{1024}$ = 4 chips.

b) $2048 = 2^{11}$, $256 = 2^8$

$4096 = 2^{12}$, $1024 = 2^{10}$

Interface 4 X 4 = 16 Registers = 2^4

c)

Component	Address	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
RAM	0000-07FF	0	0	0	0	0	← 3X8 decoder →		x	x	x	x	x	x	x	x	x
ROM	4000-4FFF	0	1	0	0	← 2X4 decoder →			x	x	x	x	x	x	x	x	x
Interface	8000-800F	1	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x

Cache Memory

Locality of Reference - Analysis of large number of typical programs has shown that the references to memory at any given interval of time tend to be confined with in a few localized areas in memory. This phenomena is refer as locality of reference.

Cache Memory - If the active portion of the program and data are placed in fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a fast small memory is refered as cache memory.

Hit Ratio - The performance of cache memory is frequently measured in terms of quantity called hit ratio

- When the CPU refers to memory and finds the word in cache, it is said to produce a hit
- If the word is not found in cache, it is in main memory and it counts as a miss.

The ratio of the number of hits divided by the total CPU refences to memory is the hit ratio.

Types of Mapping

- The transformation of data from main memory to cache memory is refered as mapping process.
- There types of mapping.
  1) Associative Mapping
  2) Direct Mapping
  3) Set - Associative Mapping

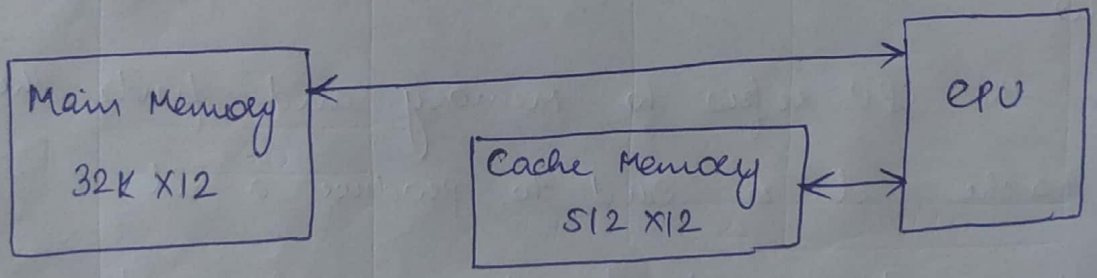
1) Associative Mapping :-

To understand mapping procedure, consider an example

→ The main memory can store 32 K words of 12 bits each.

→ The cache is capable of storing 512 of these words at any given time.

$32\ K = 2^5 \cdot 2^{10} \rightarrow 2^{15}$



→ For every word stored in cache, there is a duplicate copy in main memory.

→ The CPU communicates with both memories. It sends 15 bit address to cache.

Hit - CPU accepts data from cache.

Miss - CPU read the word from main memory.

1) Associative Mapping

CPU Address (15 bits)

↓

Argument Register

Address	Data
01000	3450
02777	6710
22345	1234

$5 \times 3 = 15$ → Address
$4 \times 3 = 12$ bit data

→ It stores both address and content of memory word.

→ The address value of 15 bit is shown as five-digit octal number and its corresponding 12 bit word as four digit octal number.

→ A CPU address of 15 bit is placed in the argument register and associate memory is search for matching process.

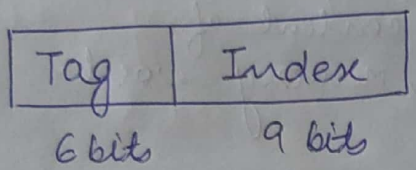
If the address is found, the 12 bit data is read and sent to CPU.

If no match occur, the main memory is accessed for the word.

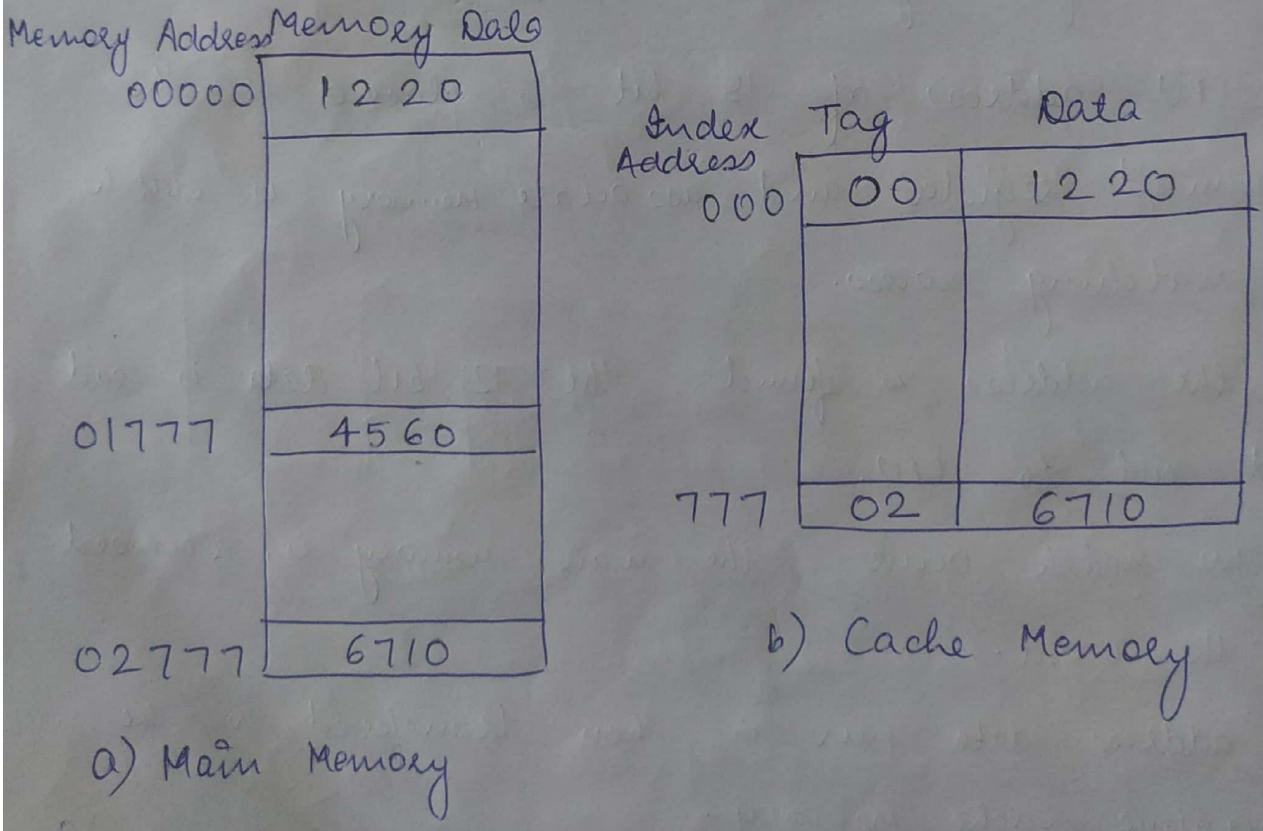
The address data pair is then transfered to the associative cache memory.

2) Direct Mapping

- Associative mapping is expensive compared to RAM because of added logic associated with each cell.
- The CPU address of 15 bits is divided into two fields. The nine least significant bits constitute the index field and the remaining six bits form tag field.



- The internal organization of the words in cache memory is as shown



a) Main Memory

b) Cache Memory

Fig:- Direct Mapping Cache Organization

- Each word in cache consist of data word and its associated tag.
- When a new word is associated with cache, the tag bit are stored along the data bits.
- When the CPU generates memory request, the index field is used for the address to access the cache.
- The tag field of CPU address is compared with the tag in the word read from cache. If there is no match, there is a miss and required word is read from main memory.
- Disadvantage- Hit ratio drops, if two or more address have same index but different tags are accessed repeatly.

3) Set Associative Mapping

It is an improvement over direct mapping organization in that each word of cache can store two or more words of memory under the same index address.

Each data word is stored together with its tag and the number of tag data items in one word of cache is said to form a set.

Index	Tag	Data	Tag	Data
000	01	3450	02	5670
777	02	6710	00	2340

Fig :- Two way Set - Associative mapping cache

- Each index address refers to two data words and their associated tags.
- Each tag require six bits and each data word has 12 bits, so the

word length = 2 (6 + 12)

= 36 bits

- ~~Thus~~, An index address can accomdate 512 words. Thus size of cache memory is 512 X 36
- It can accommodate 1024 words of main memory since each word of cache contains two data words.
- In fig, the word stored at addressess 01000, & 02000 of main memory are stored in cache memory at index 000.

- When the CPU generates a memory request, the index value of the address is used to access the cache.
- The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs.
- The hit ratio will improve as the set size increases because more words with the same index but different tags can reside in a cache.

Note :- When a miss occurs in set associative cache, it is necessary to replace one of the tag data items with a new value.

Q) A two way associate cache memory uses block of four words. The cache can accommodate a total of 2048 words from main memory. The main memory size is 128 K × 32. Formulate all pertinent information required to construct the cache memory. What is the size of cache memory.

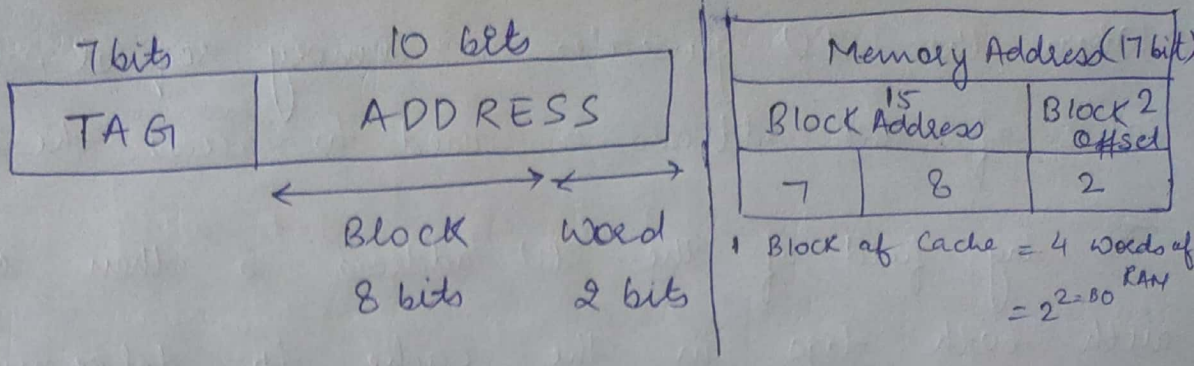
Soln :- ∴ Size of Main Memory = 128 K

∴ 128 K = 2^{17}

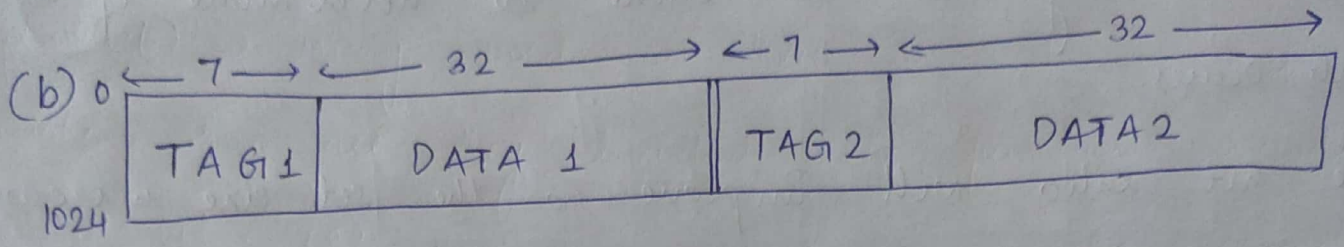
2048 words from main memory (

i.e 2048/2 = 1024 words of cache

(a)



(b)



Size of Cache Memory = $1024 \times 2\ (7+32)$

$= 1024 \times 78$

Q) The access time of a cache memory is 100 ns and that of main memory is 1000 ns. It is estimated that 80% of the memory request are for read and 20% for write. The hit ratio for read access only is 0.9. A write through procedure is used.

a) What is the average access time of the system considering only memory read cycles. b) What is the average access time of the system for both read & write requests.

c) What is the hit ratio considering the write request also

Solv.-

(a) Average Access time for Cache to read = 0.9 × 100

= 90 ns

Average Access time for cach & Memory to write = 0.1 × (100 + 1000)

= 110 ns

Total time taken = 90 + 110

= 200 ns

(b) Average access time for read = 0.8 × 200

= 160

Average access time for write = 0.2 × 1000

= 200

Total time taken = 160 + 200

= 360 ns

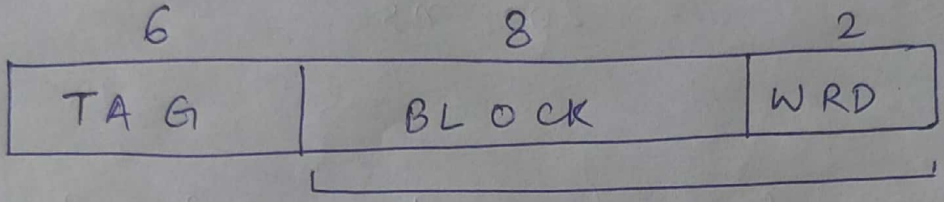
(c) Hit ratio = 0.8 × 0.9

= 0.72

Q A computer has memory unit of 64K X 16 and cache memory of 1K words. The cache uses direct mapping with block size of four words (a) How many bits are there in the tag, index, block and words fields of the address format (b) How many words are there in each word of cache, how are they divided into functions. Include a valid bit. (c) How many blocks can cache accomodate:

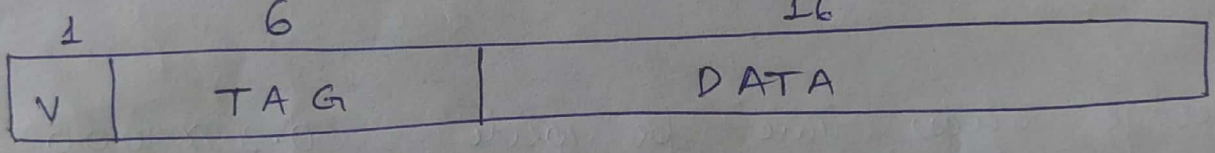
Soln:

(a)



Index = 10 Bit Cache Address

(b)



23 Bits in each word of cach

(c) 2^8 = 256 block of 4 words each.

$1K \cong \frac{1024}{4} = 256$ Block

Virtual Memory.

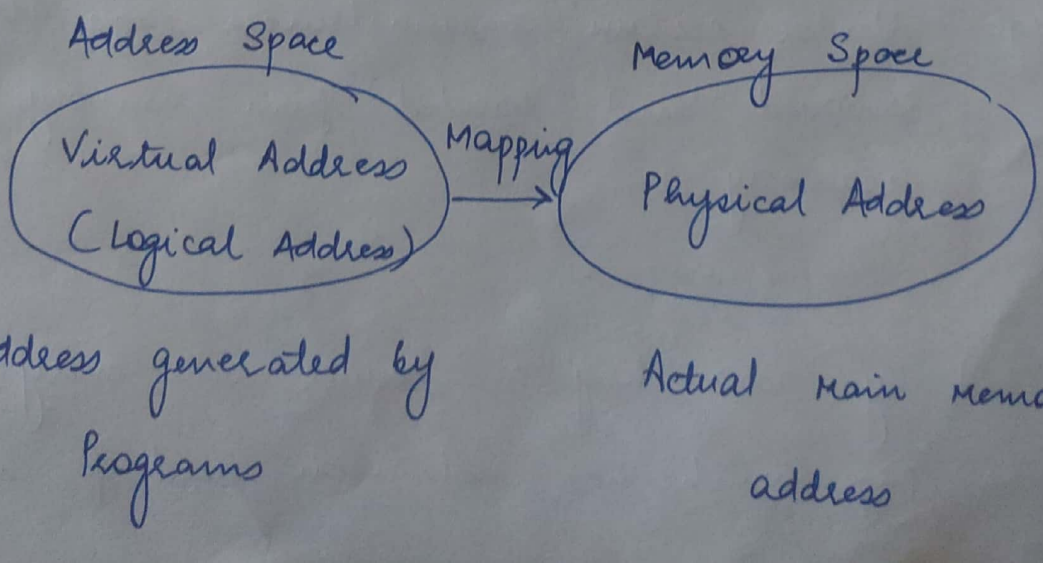
- It is a concept used in some larger computer systems that permit the user to construct programs as though a large memory space were available.
- A virtual memory system provides a mechanism for translating programs generated addresses into correct main memory locations.

→ **Address Space**

An address used by programmer will be called virtual address, and the set of such addresses the address space.

→ **Memory Space**

An address in main memory is called location or physical address. The set of such locations is called memory space.



→ Page Replacement:

When a page fault occurs in a virtual memory system, it signifies that the page referenced by the CPU is not in the main memory.

If main memory is full, it would be necessary to remove a page from a memory block to make room for new page.

The most common replacement algorithms are:

1) FIFO (First In First Out)
2) LRU (Least Recently Used)
3) OPR (Optimal Page Replacement)

FIFO

- It selects for replacement the page that has been in memory for long time.
- Each time a page is loaded into memory, the identification number is pushed into FIFO stack.

Advantage- Easy to Implement.

Disadvantage- Under certain circumstances pages are removed and loaded from memory too frequently.

LRU

- Implemented by associating a counter with every page
- Replace the page which has not been used for longest period of time.

OPT

- Optimal Replacement
- Lowest page fault rate of all algorithms.

Eg: Consider three memory frames.

Refrence String

First In First Out (FIFO)

7 0 1 2 0 3 0 4 2 3 0 3 1 2 0

f3			1	1	1	~~1~~	0	0	~~0~~	3	3	3	~~3~~	2	2
f2		0	0	0	~~0~~	3	3	~~3~~	2	2	2	~~2~~	1	1	1
f1	7	7	~~7~~	2	2	2	~~2~~	4	4	~~4~~	0	0	0	0	0
	*	*	*	*	Hit	*	*	*	*	*	*	Hit	*	*	Hit

Main Memory frame

Page Hit = 3, Page Miss = 12, Hit Ratio = $\frac{\text{No. of Hits}}{\text{Total Refences}}$

Miss Ratio = $\frac{\text{No. of Miss}}{\text{Total Refence}}$

$= \frac{12}{15} \times 100 = 80\%$

$= \frac{3}{15} \times 100 = 20\%$

Optimal Page Replacement (OPR)

→ Replace the Page which is not used in longed dimension of time in future.

7 0 1 2 0 3 0 4 2 3 0 3 1 2 0

		1	1	[1]	3	3	3	3	3	3	[3]	1	1	1
	0	0	0	0	0	[0]	4	4	[4]	0	0	0	0	0
7	7	[7]	2	2	2	2	2	2	2	2	2	2	2	2
*	*	*	*	Hit	*	Hit	*	Hit	Hit	*	Hit	*	Hit	Hit

Hit = 7

Miss = 8

Hit Ratio = $\frac{7}{15} \times 100$

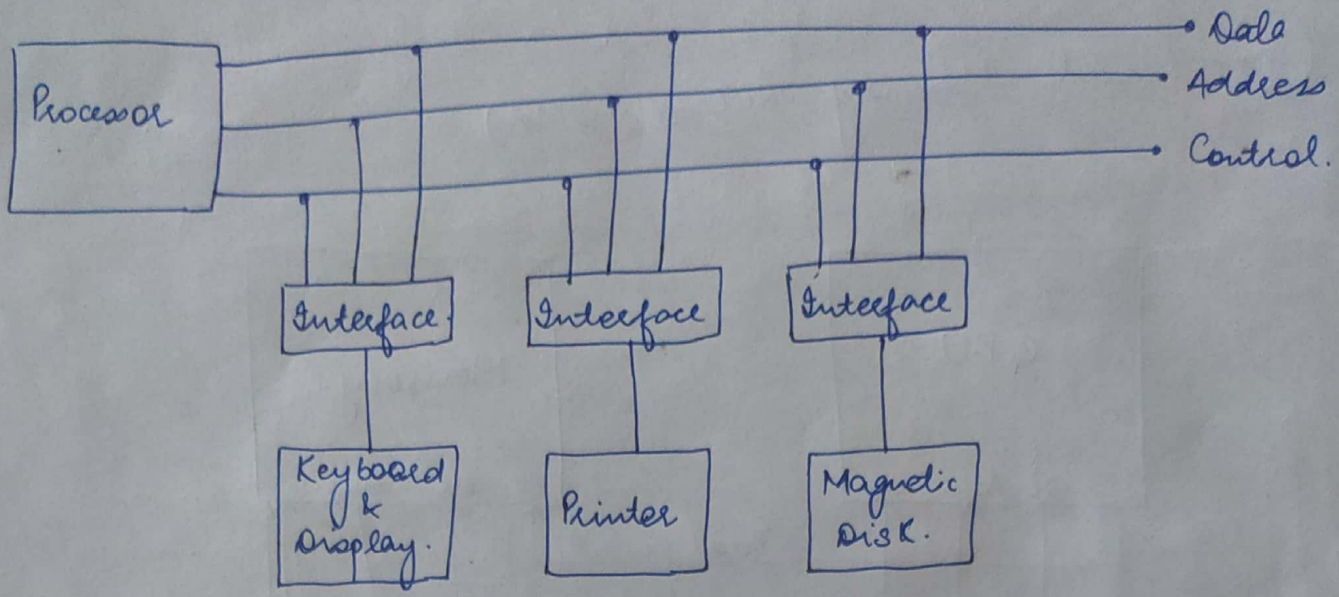
Miss Ratio = $\frac{8}{15} \times 100$

Least Recently Used (LRU)

7	0	1	2	0	3	0	4	2	3	0	3	1	2	0	→ Replace the page that is least recently used.
		1	1	[1]	3	3	[3]	2	2	2	[2]	1	1	1	
	0	0	0	0	0	0	0	[0]	3	3	3	3	[3]	0	
7	7	[7]	2	2	2	[2]	4	4	[4]	0	0	[0]	2	2	
*	*	*	*	Hit	*	Hit	*	*	*	*	Hit	*	*	*	

- If the page demanded, is not present in the main memory is refer as page fault.
- Then information is fetch from hard disk (virtual memory) and placed in main memory.
- If the frames in main memory are full, then replace the page (via page replacement algorithm)
- If the demand page is already placed in main memory it is refer as Hit.

Input - Output Interface.



Need of Interface

1) Speed — Million of instructions in secs — Slow

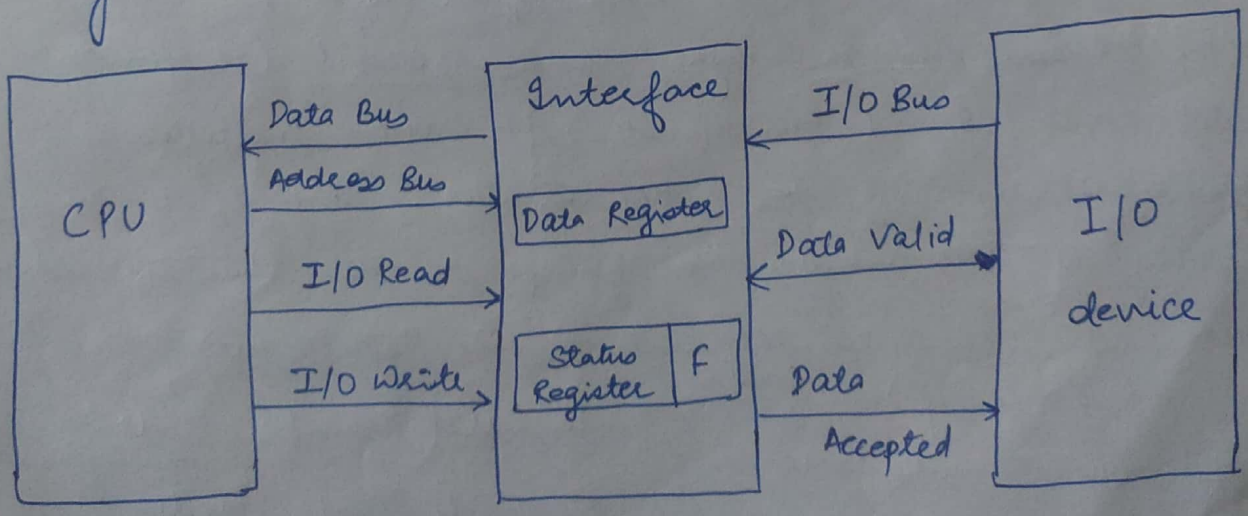
2) Signal — Electromagnetic

3) Format — 32 byte, 64 byte

4)

Input - Output Transfer

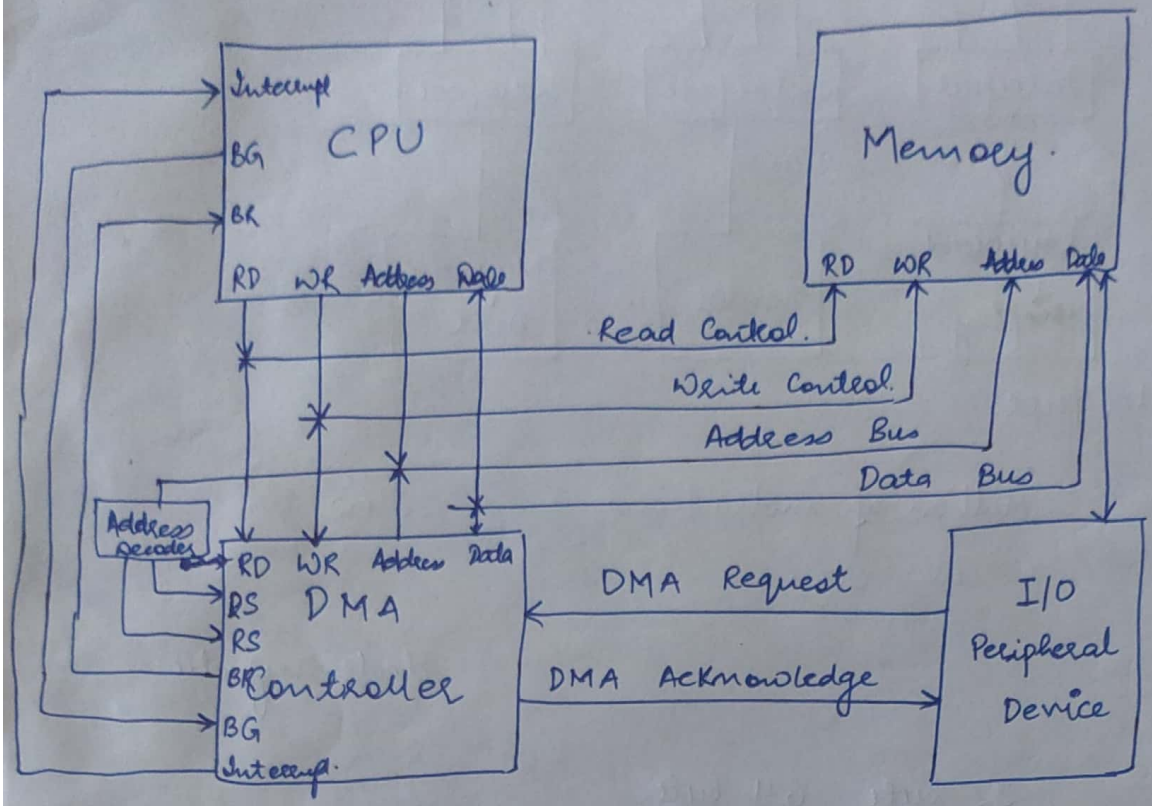
(1) Programmed I/O



F = Flag Bit

(2) Interrupt I/O Initiated

(3) Direct Memory Access (DMA)



→ DMA transfer the data b/w I/O to Memory w/o using CPU

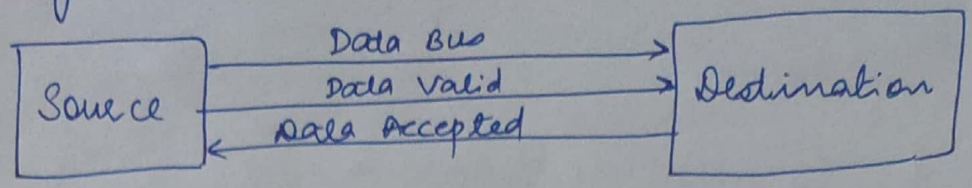
→ When I/O device is ready to send data, it sends DMA request (i.e ready to transfer the data)
then DMA Controller sends BR (Bus Request to CPU), CPU stands RD, WR, Address, Data in high impedence circuit (ie open circuit)
Now CPU sends BG (Bus Grant) to DMA Controller.
Now bus is handovered to the DMA controller. Now it is the responsibility of DMA controller to transfer the data b/w I/O and Memory
Now DMA send DMA Acknowledgement to the I/O device.
(i.e DMA is ready)

BG = 1 Buses with the DMA
= 0 Bus with CPU

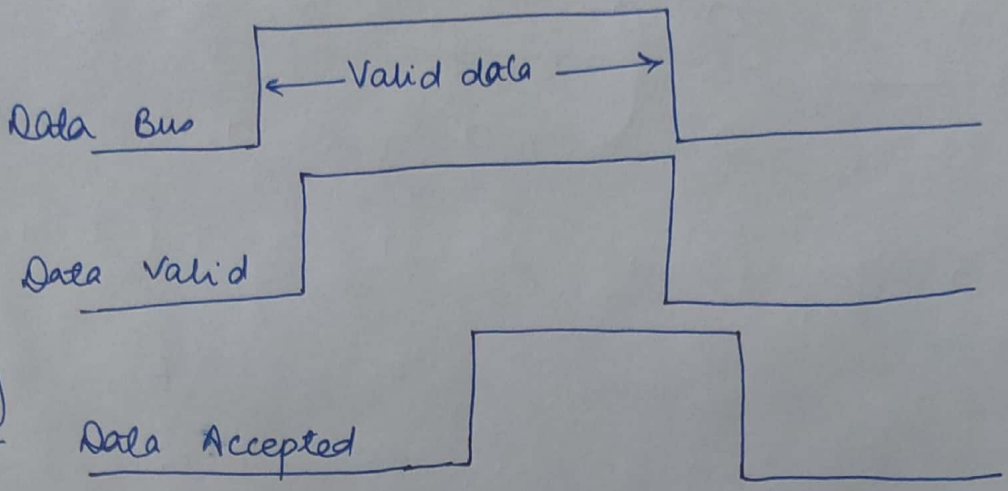
1. Burst Transfer Mode.
Allowed at a time (10)
2. Cycle Seals Mode

Handshaking.

Block Diagram



Timing Diagram



Sequence

1. Place data on bus
2. Enable data valid.
3. Accept data from bus.
4. Enable data accepted.
5. Disable Data Valid
6. Invalidate data on bus
7. Disable data accepted.
8. Ready to accept data.