Computer Architecture & Organization Previous year -

Question

Q.) Draw a flow chart of instruction cycle of computer and explain.

Register Involved in each Instruction Cycle :-

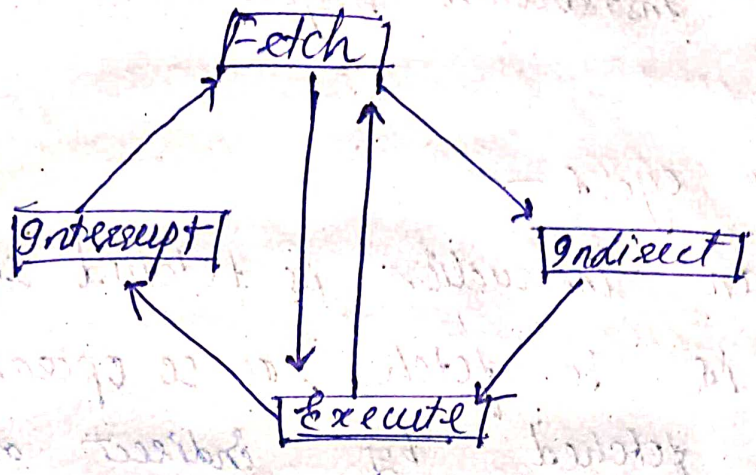
1) memory address register (MAR) - It specifies the address in memory for a read or write operation. It is connected to the address lines of the system bus.

2) memory Buffer Register (MBR) : → It is connected to the data lines of the system bus. It contains value to be stored in memory.

3) Program counter - Holds the address of the next instruction to be fetched.

4) Instruction Register - Holds last instruction fetched.

The instruction cycle –



→ Fetch → Indirect → Execute → Interrupt

Fetch Cycle –

A simple fetch cycle consist of three steps and four micro-operation –

we can write it as

MAR ← PC
MBR ← memory
PC ← (PC) + 1
IR ← (MBR)

step 1) The address in the program counter is moved to memory address register

step 2) – Here the control unit issues a Read command on the control bus, and the result appears on the data bus and is then copied to memory buffer register (MBR)

Step 3 – The content of MBR is moved to the Instruction register.

Indirect cycle –

Once an Instruction is fetched the next step is to fetch source operand.

It is fetched by indirect addressing

note → Register based operands need not to be fetched

Execute cycle – ~~It is perform~~ For a machine with N different opcoded there are N different sequence of micro-operations that can occur.

For example – ADD R, X

Here content of location X to register R is added.

corresponding micro-operations —

MAR ← IR (address)

MBR ← memory

R ← R + (MBR)

(operation by ALU)

Interrupt Cycle :-

At the completion of the execute cycle, a test is made to determine whether any enabled interrupt has occured or not. If any enabled interrupt has occured then interrupt cycle occured.

Question

what is the main idea behind addressing modes? Give a brief account of different kinds of addressing modes used in a system citing examples.

We need addressing modes because it provided flexibility to programmer to access data as per convenience or requirement of his design different addressing modes further help by making code simple and efficient

Definition - Addressing mode is the way of addressing a memory location in instruction. The method of specifying source of operand and output of result in an instruction is known as addressing mode.

Types of Addressing modes -

1) Register Addressing mode
2) Direct Addressing mode
3) Register Indirect Addressing mode
4) Immediate Addressing mode
5) Index Addressing mode.

1) Register Addressing mode -

Here the name of register is given in the instruction where the data to be read or result to be stored.

example ADD A, R5

2) Direct Addressing mode -

The address of data to be read is directly given in the instruction, for storing result the address given in instruction is used to store result.

3) Register Indirect Addressing mode -

Here we provide a register in the instruction in which the address of other register is to be stored or which points to other register where data is stored or to be stored

example - MOV A, R0 (it will move data to the accumulator from register whose address is stored in R0).

4) Immediate Addressing mode →

The data immediately follows the instruction, This means that data to be used is already given in the instruction itself.

5) Index Addressing mode →

This addressing mode is used for reading lookup tables in Program memory.

The address of the exact location of the table is formed by adding accumulator Data to the base pointer.

Question –

condition flag? & its types –

There are six conditional flag –

1) The parity flag (PF)
2) The Zero flag (ZF)
3) The sign flag (SF)
4) Auxiliary Flag (AF)
5) The carry Flag (CF)
6) The overflow Flag (OF)

conditional flag – It is a condition where after execution of operation the result either sets or resets the register.

(1). Parity Flag (PF) –

If the number of 1's is even in the O/P stored in the accumulator then it is set otherwise it is reset for the odd.

11 → set
01 → reset

(2) The Zero Flag – If the result stored in an accumulator is zero then this flip-flop is set otherwise it is reset.

(3) The Sign Flag (SF) –

If the D7 of the result is 1 then sign flag is set otherwise reset.

The number on the D7 decided the sign of the number.

If D7 is 1: the number is negative.
If D7 is 0: the number is positive.

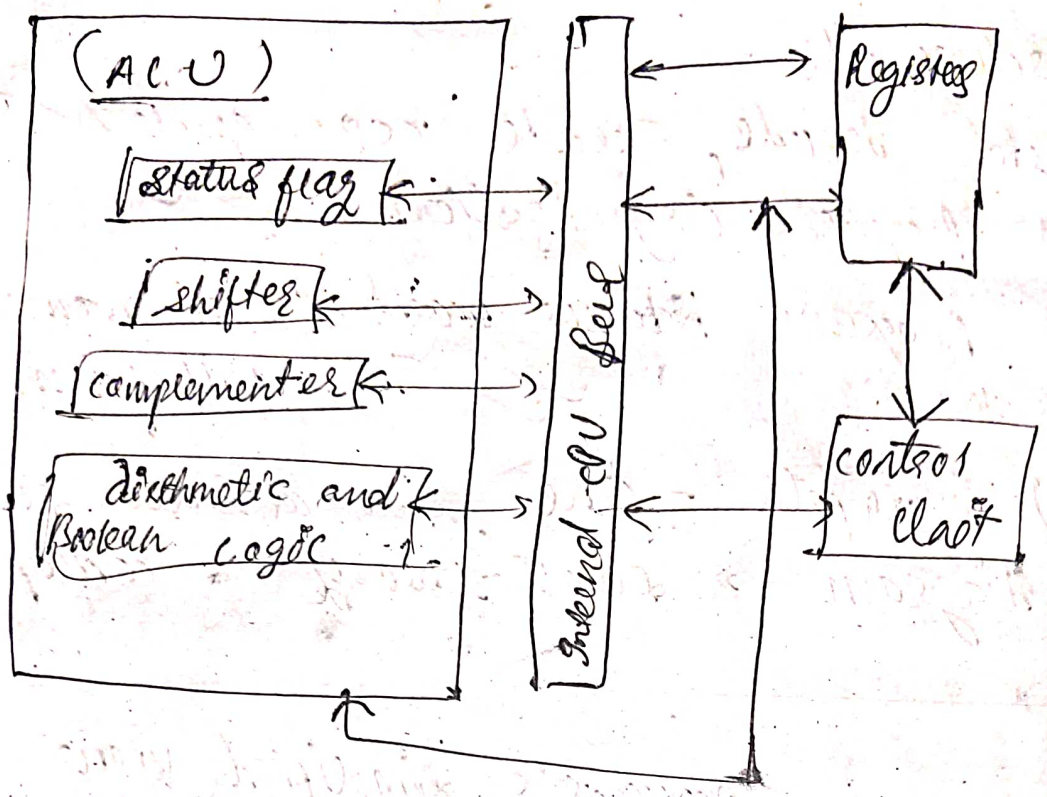
4) The Auxiliary Flag – If any carry goes from D3 to D4 in the output then it is set otherwise it is reset.

5) The Carry Flag (CF) – If the result stored in an accumulator generated a carry in its final output then it is set otherwise it is reset. //

6) The overflow flag -

If a carry goes from D_6 to D_7 then OF is set otherwise it is reset.

(Internal Structure of CPU)



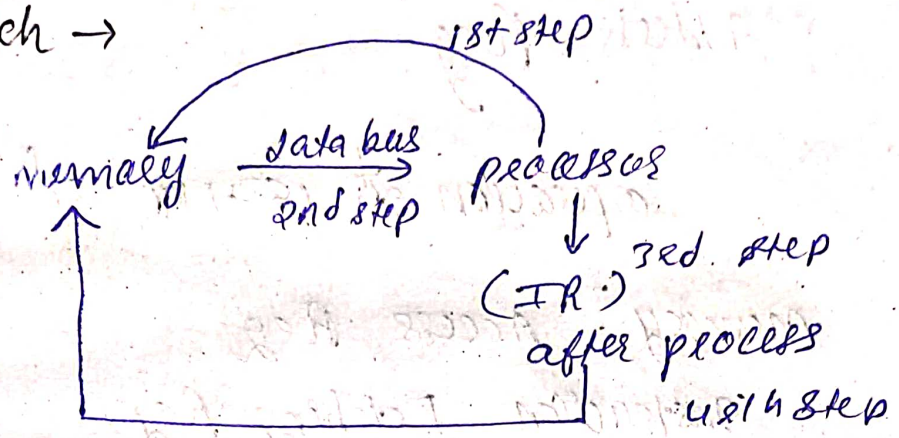
(Major components of processor)

1) ALU — Performs computation & processing of data
2) CU → controls instructions & operations in ALU
3) Registers — used as Internal memory
4) System Bus — Acting as a pathway btw processor, memory and I/O module.

CPU Cycle (Von Neumann)

Fetch → Decode → Execute
cycle ko called CPU cycle.

Fetch →

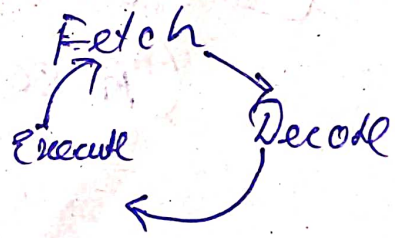


Decode →

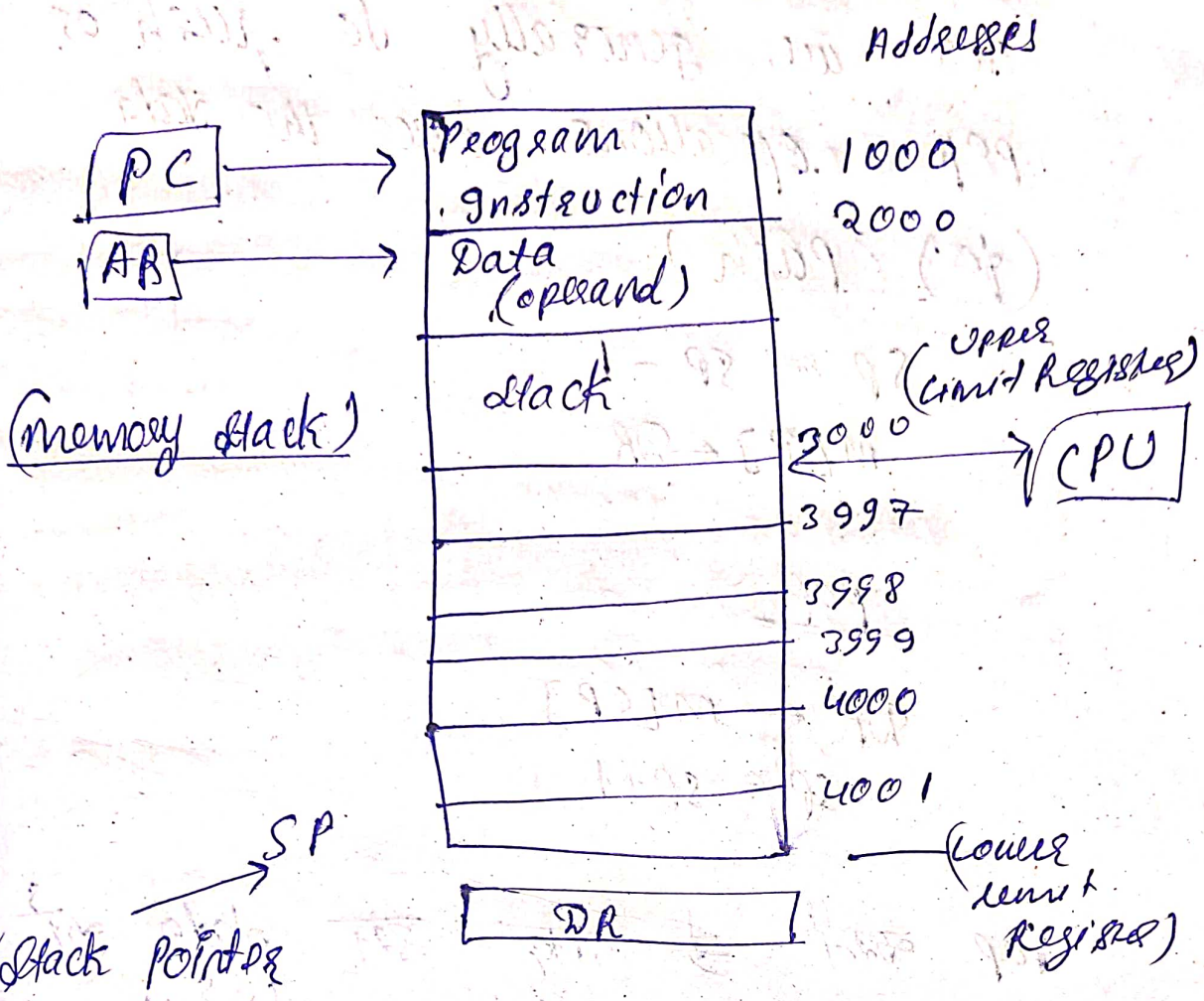
Instruction Register
↓
Decoder
↓
then process it by processor

Execute — the processor execute the instruction.

Diagram →



(Memory stack Organization)



1 Here we are using RAM as a stack, & we use stack pointer to fetch the address from where we have to (read) gain data & ~~to~~ to the location where we have to write a data

3) PC में next instruction लेगे then उसे decode करेगे उसके (AR) मतलब address register में रखेगे अब during execution में AR की मदद से direct operand पर operation perform करना है और अब execute ||

Now comes stack after fetching the operand

Here we generally do push or pop operations over the data

(pop) (push)

$SP \leftarrow SP - 1$

$M[SP] \leftarrow DR$

pop

$DR \leftarrow M[SP]$

$SP \leftarrow SP + 1$

Pop करने के बाद जो data है Data register में आ जाता है और उस operation में data bus बच जाती है।

Reverse Polish Notation →

Infix notation (A+B)

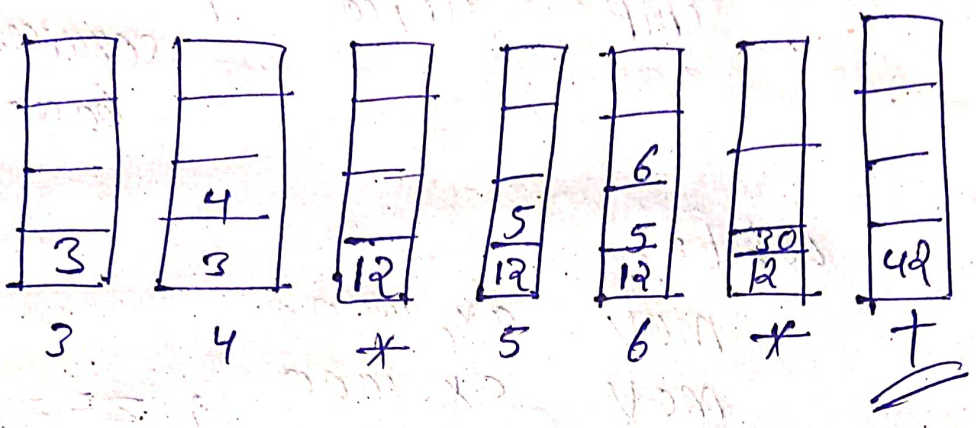
Prefix or Polish notation (+AB)

Postfix or reverse polish (AB+)

example —

Using Reverse Polish or Postfix Notation

$(3*4)+(5*6) \Rightarrow 34*56*+$



(CISC VS RISC)

	CISC	RISC
①	Emphasis on hardware	Emphasis on software
②	It consists of multiple Instructions set and formats	It consist Instructions of same set with few formats
③	It has less registers	Uses more registers
④	More addressing modes	Less addressing modes
⑤	Here pipelining is difficult as circuits are complex	Here it is easy

(Previous year)

Create a comparison instruction using zero and carry flag.

```
Label 1:
        mov     al, 5
        cmp     al, 10      ; CF = 0
                            requires a
                            borrow

Label 2:
        mov     ax, 1000
        mov     cx, 1000    ; ZF = 1
        cmp     cx, ax      (cx = ax)
```

(Shift & Rotate operations)

1. SHR → shift Right
2. SAR → Shift Arithmetic Right
3. SHC → Shift Left
4. SAC → Shift Arithmetic Left
5.

(Rotate)

1. Rotate Left
2. Rotate carry Left
3. Rotate Right
4. Rotate carry Right

SHR → Shift Right

The SHR instruction simply shifts the mentioned bits in register to the right side one by one by inserting same number of zeroes from left end. the right most bit that is being shifted in carry flag.

(SHR AX, ?)

SAR → Shift Arithmetic Right →

The SAR instruction shifts bits to right side one by one but instead of inserting zeroes from left end, the MSB is restored. The right most bit is shifted in carry Flag.

SAR AX, ?

(SHL SAL) has same working as (SHR, SAR)

(ROL) (Rotate Left)

It ~~stores~~ rotates the mentioned bits in the register to the left side one by one such that leftmost bit that is being rotated is again stored as the rightmost bit in the register and also stored in carry Flag

ROL AX, 4

Rotate carry left

It rotates bits in register to left side one by one such that left most bit that is being rotated stores in carry flag and that bit is moved as (LSB) in the register.

1
least significant Bit

RCL CH, 1

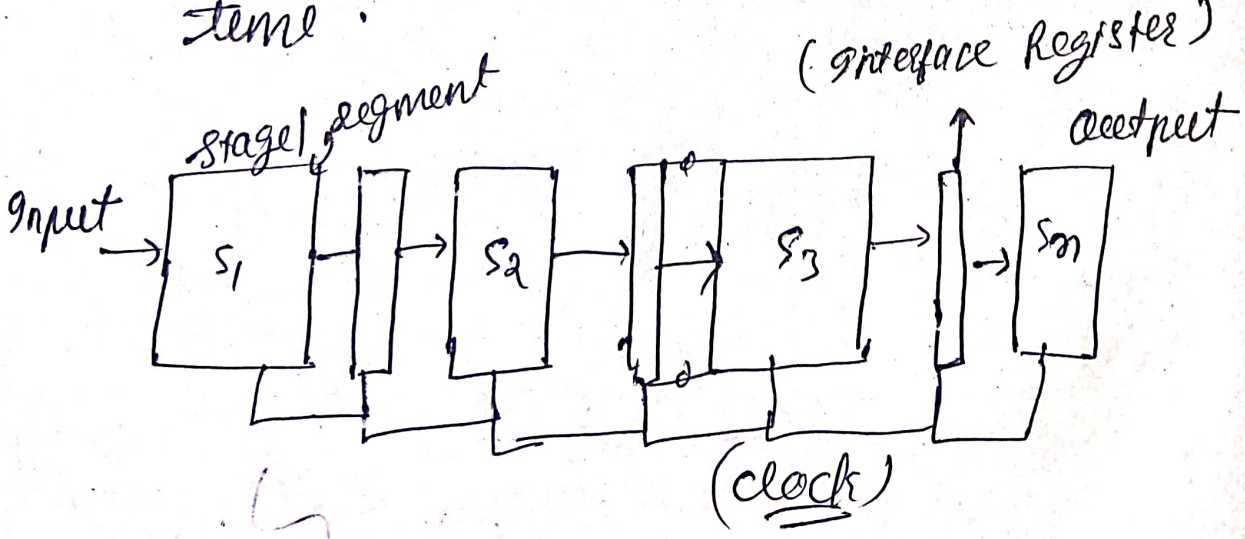
Pipelining →

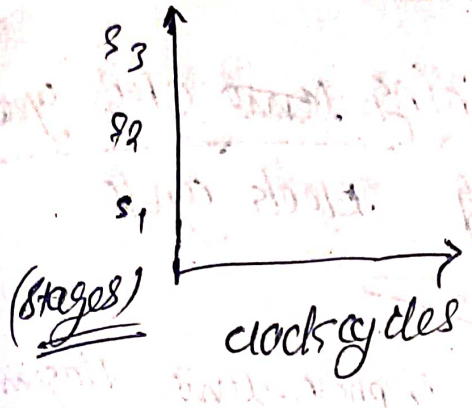
→ It is a process of arrangement of hardware elements of CPU in such a way that it's overall performance is increased.

→ Simultaneous execution of more than one instruction take place in pipelined processor.

→ In pipelining multiple instructions are overlapped in execution.

with the help of space - time diagram we represent how a process gets executed at real time.





S1	I1	I2	I3	I4	I5
S2		I1	I2	I3	I4
S3			I1	I2	I3
S4				I1	I2
S5					I1

If we have k stages and then first instruction that will get executed in (time = k) and rest will take place in (n-1) where (n) is the number of instruction

$$S + (S-1) \Rightarrow$$ (9 clock cycles are required)

The basic aim of pipelining is CPI ≈ 1 mean (clock per instruction)

Speedup = $\frac{\text{non pipelining clock cycle}}{\text{pipelining clock cycle}}$

Utilization = $\frac{\text{(used space as blocks)}}{\text{(total space as blocks of space)}}$

(in phase - time diagram)

(in phase time diagram)

Throughput = $\frac{\text{number of instructions executed}}{\text{total time taken}}$

Stage Delay in Pipelining.

(question)

A 4 stage pipeline has stage delays as 150, 120, 160 and 140 ns registers are used between stages and have a delay of 5ns each. Assuming const. clock rate, the total time taken to process 1000 data items on this pipeline will be — ?

① 120.4 usec ② 160.5

③ 165.5 usec ④ 590 usec

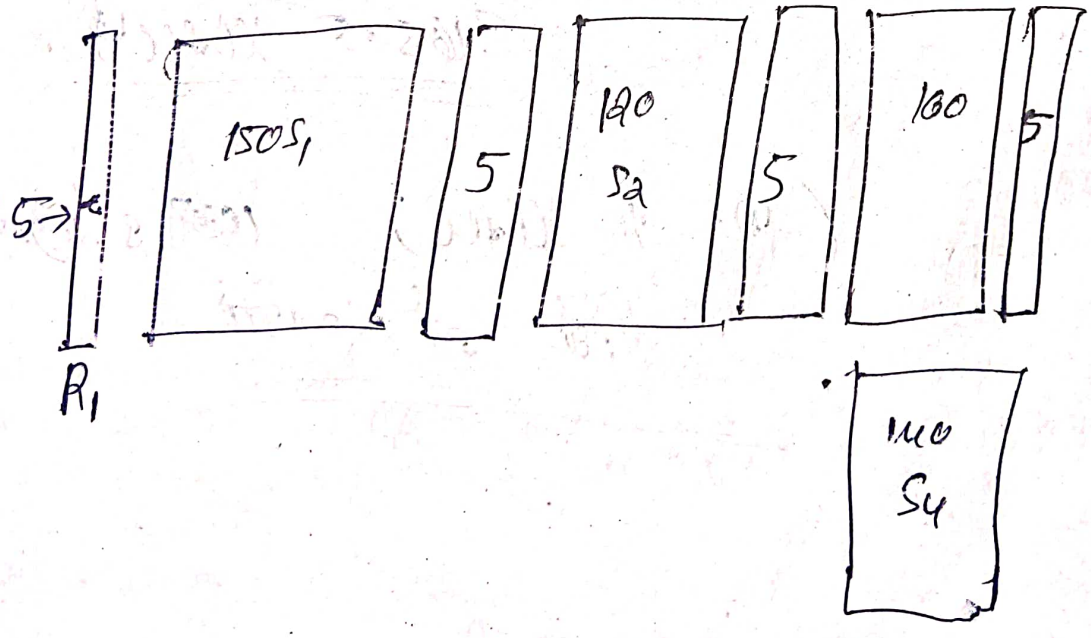
Syllabus

1) Memory Interfacing → memo Hierarchy
   → 2 level, 3 level
   → cache mapping
   → Secondary memo

2) Machine Instruction & addressing modes.

3) Pipelining → Synchronous
   → Asynchronous
   → Instruction pipe

Diagram →



For first insteuction will cross all 4 stages i.e 1×4

and time devote in each stages is 165 (ns)

$1 \times 4 \times 165$

Now as 999 insteuctions will came in every clock pulse and time taken will be 165 (ns)

so can be written as

$= 999 \times 1 \times 165$

combining $= 1 \times 4 \times 165 + 999 \times 1 \times 165$

$= 165495$ ns $\times 10^{-9}$

$(165.5$ usec$)$.

(ns to usec) 1000 s divide

पोईंट एक लगाओ

consider

a non pipelined processor with a clock rate of 2.5 gigahertz and avg. cycles / instruction of four

- The same processor is upgraded in a pipelined processor with five staged, but due to internal pipeline delay, the clock speed is reduced to 2 gigahertz. Assume that there is no stall in pipeline. The speedup achieved in pipeline processor is ____ ?

(a) 3.2
(b) 3.0
(c) 2.2
(d) 2.0

(So(n) without pipeline

$$\text{Speedup} = \frac{T_{wp}}{T_p}$$

↑ with pipeline

Time = 1/freq

$$T_{wp} = 4 \times \frac{1}{2.5 \times 10^9} \text{ sec}$$

freq = 2.5 GHz

$$T = \frac{1}{2.5}$$

$$T_p = \frac{1}{2 \times 10^9} \text{ sec}$$

$$= \frac{4 \times \frac{1}{2.5 \times 10^9} \text{ sec}}{\frac{1}{2 \times 10^9} \text{ sec}}$$

$$= \frac{4}{2.5} \times 2 = \frac{8}{2.5} = (3.2)$$

Ans

(***) (Hazards in Pipelining)

In Pipelining concept each instruction should get executed in 1 CPI which is practically very difficult because following point creates hazard in it.

Note

5 stages in pipelining

1. Instruction fetch/Decode
2. Operand fetch
3. Execute
4. memory
5. write Back.

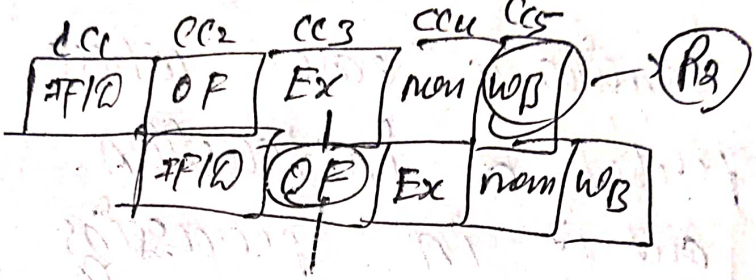
Q Hazards

① Data Hazard

(i) Read after write

example $R_2 \leftarrow R_2 + R_3$

$R_5 \leftarrow R_2 + R_4$



R_2 accessing before the final value.

(ii) write after Read

(iii) Write After write

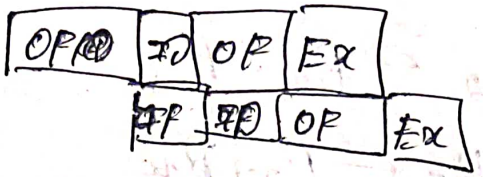
② Structural Hazard

When multiple instruction fetch the same resource then structure hazard occur.

③ control Hazards

* occurs because of branch condition.

example



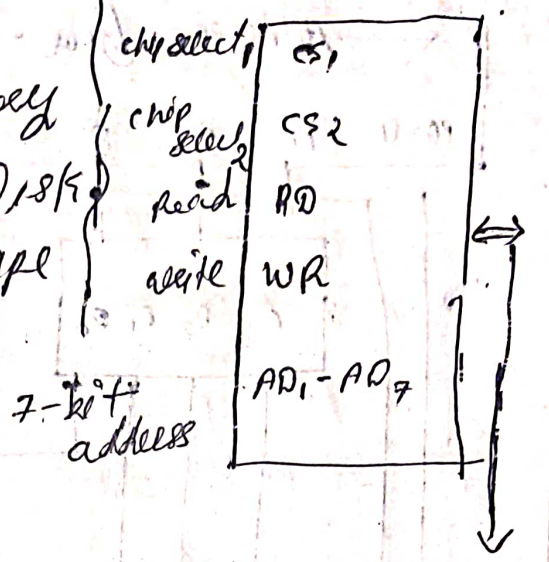
consider a case where you have fetched a instruction from PC and it in processing stage at the same time you have fetched another instruction from PC which is also in queue of execution but after executing the first instruction which was branch instruction it gives a address which the processor have to execute immediately but there is already a process present in queue which creates a control hazards

(Memory Hierarchy)

(1)

Register
cache
Main memory
magnetic ~~memory~~ Disk
magnetic ~~memory~~ tape

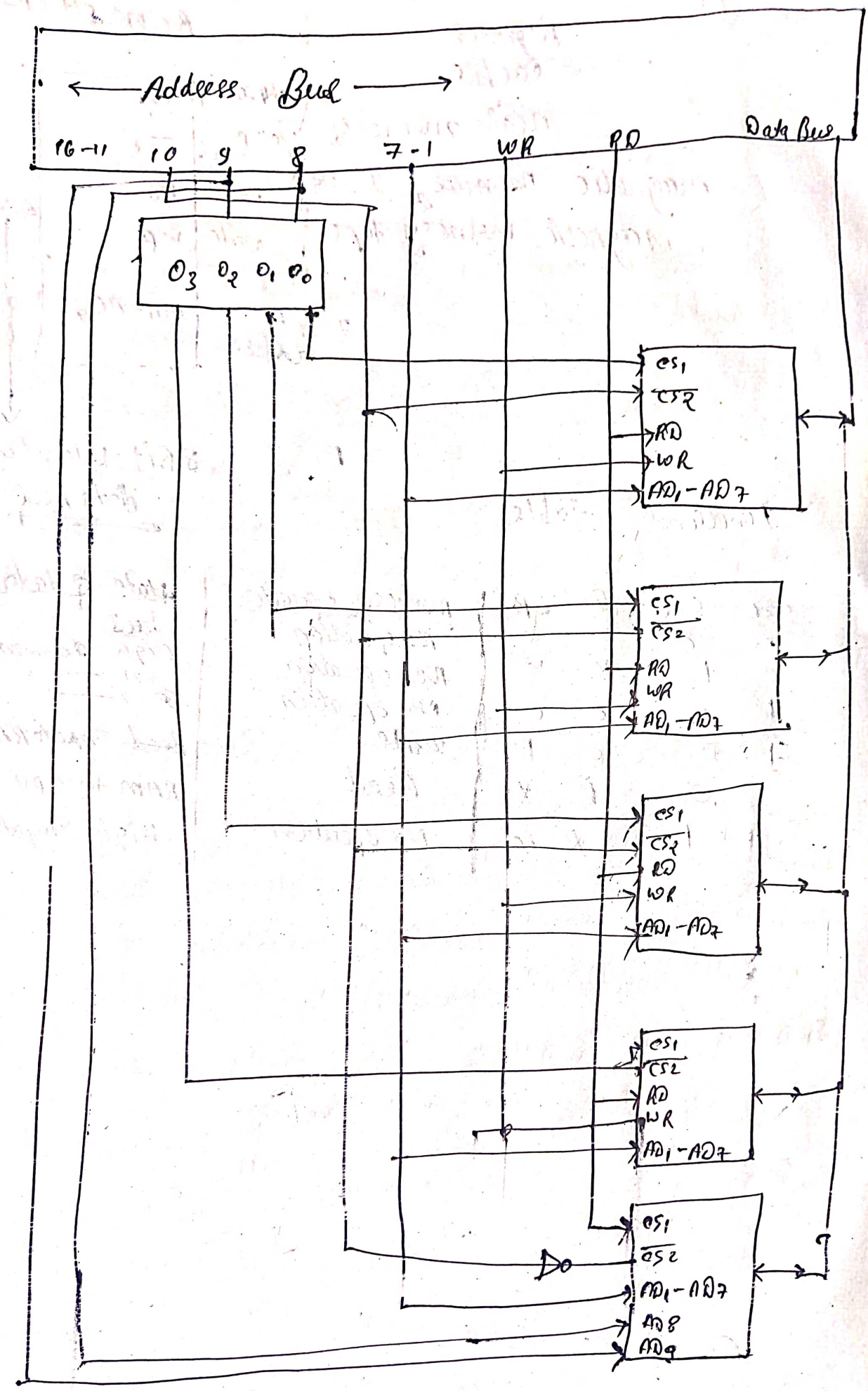
RAM CHIPS



Functional Table

CS1	CS2	RD	WR	Memory Operation	State of data bus
0	0	X	X	No operation	High Impedance
0	1	X	X	No operation	" "
1	0	0	0	No operation	" "
1	0	0	1	Write	Input to RAM
1	0	1	X	Read	RAM to CPU
1	1	X	X	No operation	High Impedance

Memory connection to CPU



(Cache - Mapping Function)

This mapping function is used to transfer the block from main memory to cache memory.

mapping functions maps particular block of main memory to particular block of cache memory.

(chapter - 7)

1) Memory Hierarchy ✓
2) Memory mapping ✓
3) SRAM vs DRAM ✓
4) cache memory ✓
5) cache memory function ✓
6) Virtual memory ✓
7) Paging ✓
8) Last example in numerical
9) pipelining ✓
10) RISC V/s CISC ✓

(cache) ka memory

Definition

It is a extremely fast memory that acts as a buffer between RAM and CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

It reduces average time to access data from main memory.

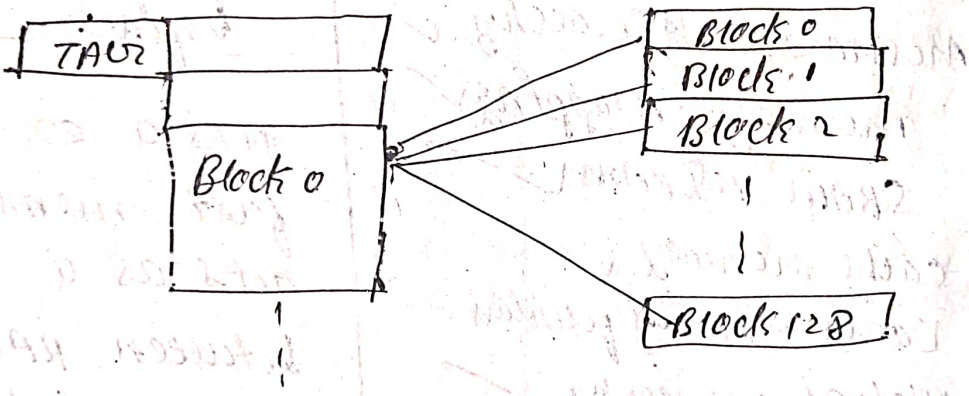
Cache memory Functions

Three different mapping functions are available -

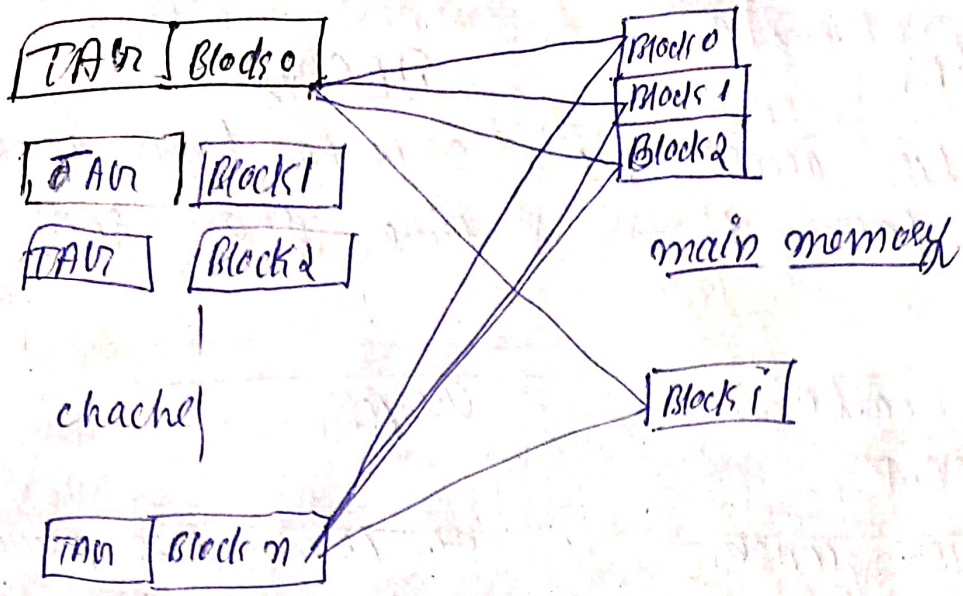
1) Direct mapping -

A particular block of main memory can be brought to a particular block of cache memory. So it is not flexible.

Direct mapping diagram

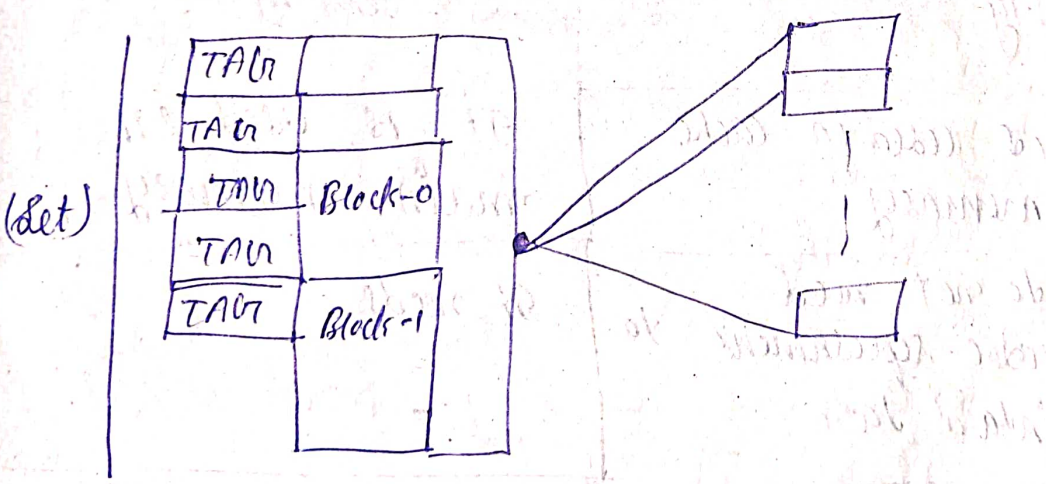


2) Associative mapping - In this function any block of main memory can potentially reside in any cache block position. This is flexible method.



Block set associative mapping –

In this method cache blocks are grouped into sets and then mapping allows a block of main memory to reside in any block of a specific set.



SRAM	DRAM
1) It has less access time so it is faster than DRAM	It is has higher access time than SRAM
2) It is costlier than DRAM	Cheaper
3) It required constant power supply which means they consume more power	It takes less power as the informations are is stored in capacitor
4) It had complex circuit design and less storage capacity	It has simpler circuits and also have a large storage compared to SRAM
5) It has low packging density	It has high packging Density
6) It is used in cache memory	It is used in main memory
7) It do not need periodic refreshment to maintain data	It needs

A dynamic cell is simples and smaller than a static memory cell thus a DRAM is more dense.

(Virtual memory)

Virtual memory is a memory management technique where secondary memory can be used as if it were a part of the main memory. (Virtual memory used both hardware and software to enable a computer to compensate physical memory shortage

virtual memory frees up RAM by swapping data that had not been used recently over to a storage device)

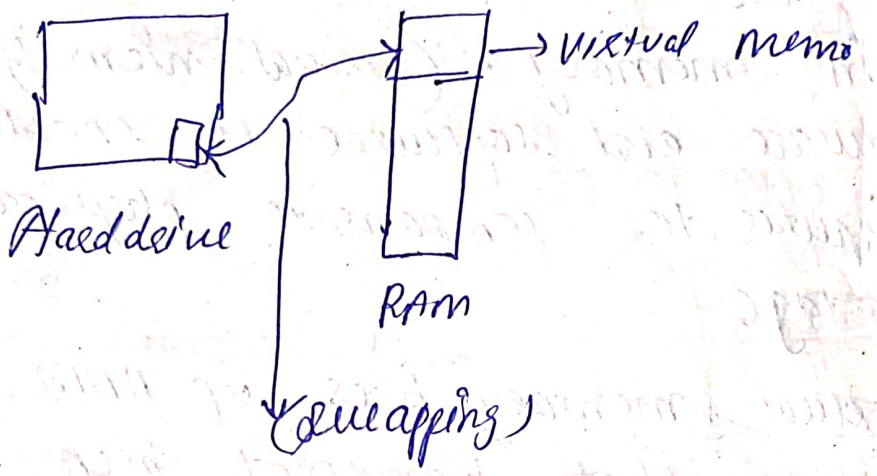
→ (working)

Usage—

With virtual memory a system can load pages or multiple programs running at the same time enabling each one to operate as if it has a more space without having to purchase more RAM.

Detailed working — When a computer uses up its available RAM, pages not in use are transferred to the hard drive using a swap file. A swap file is a space on the hard drive to be used as the virtual memory extension for computer's RAM. When a the swap file is needed it

is sent back to RAM using a process called page swaping



(Benefits of virtual memory)

1) It can handle twice as many addressed as main memory

2) It increases security because of memory isolation

3) Allocating memory is relatively inexpensive

4) It does not need external fragmentation.

5) Data can be moved in ease when used.

What are the limitations of using virtual memory

1) Data must be mapped between virtual and physical memory which required extra hardware support for address translations which slow down computer.

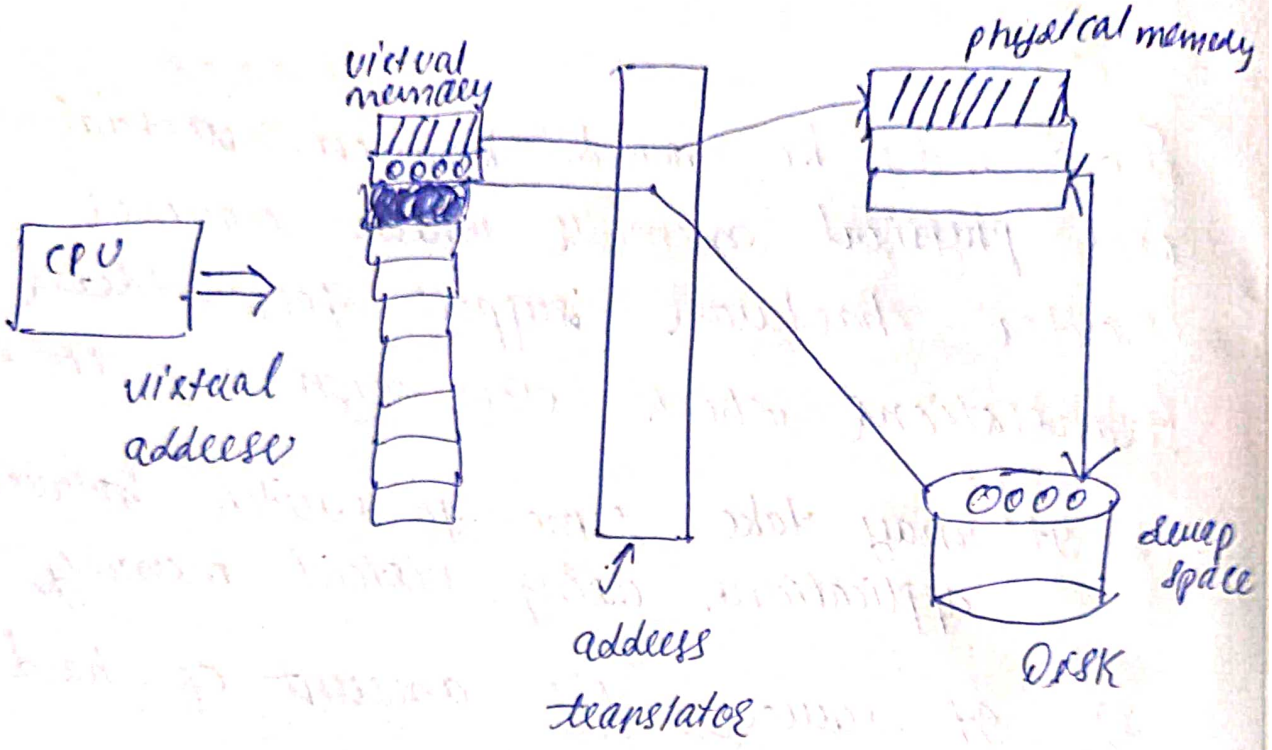
2) It may take time to switch between applications using virtual memory

3) It reduced the amount of hard drive space

4) Thrashing can occur if there is not enough RAM.

Thrashing - when the page fault and swapping happend very frequently at a higher rate then operating system has to spend more time swapping these pages this phenomenon is called thrashing.

Virtual memory Diagram



////// → addresses available in physical memo

oooo → addresses available in swap space

▨▨▨ → addresses available nowhere

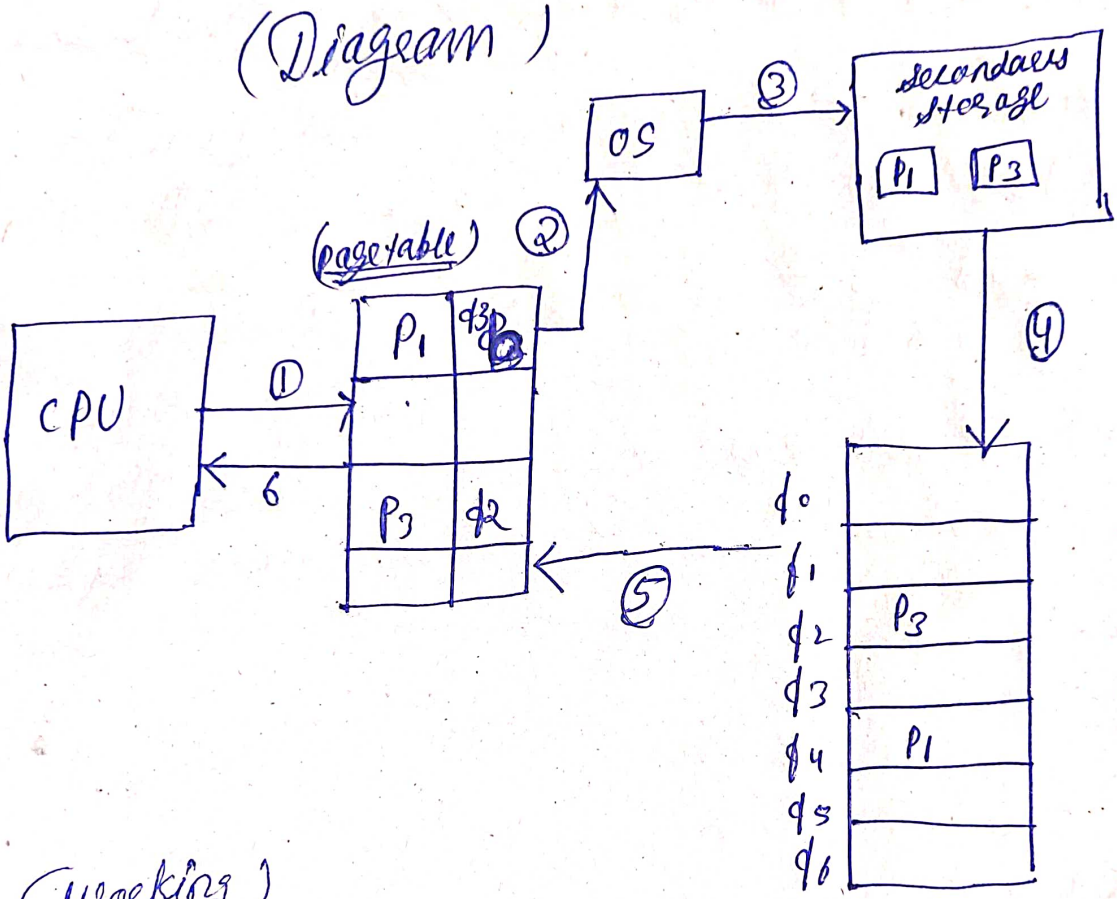
(Paging)

Paging is a storage mechanism used to retrieve processes from secondary storage into the main memory in form of pages.

The main idea is to divide each process in form of pages.

when a program needs a page, it is available in the main memory as the OS copies a certain number of pages from your storage device to main memory.

(Diagram)



(Working)

① 1st step — CPU demands pages of processes

② 2nd step — page table demands pages of processes from OS

3rd step — OS takes process from secondary storage

4th step — OS puts process from secondary storage to main memory

5th step — from main memory pages go to page table

6th step from page table to CPU.