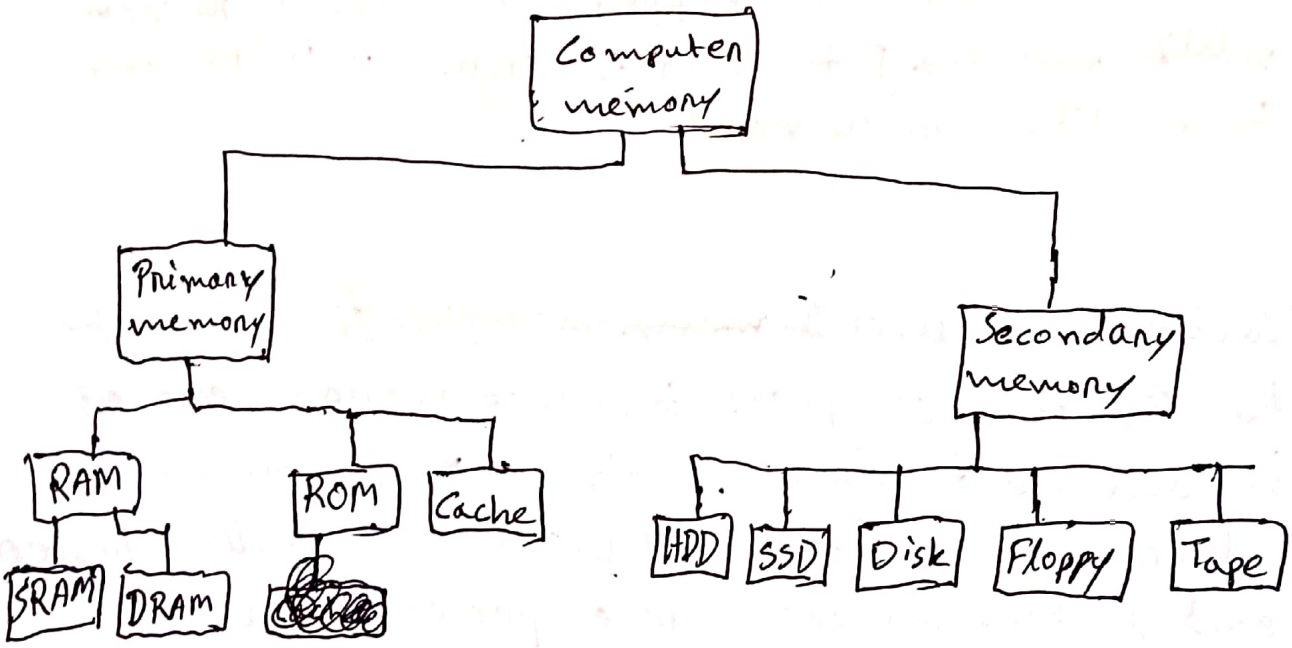
Name: Rashik Rahman
ID: 17201012

Answer to the Q. No. 1

A memory is just like a human brain. In computer sceince memory is considered as a storage space where data to be processed and instructions required for processing are stored. Memory types are given below.



RAM: RAM stands for Random Access Memory. As the name suggests it can be accessed in any random order. Read, write operation can be done on it and it is a volatile memory. Data is fetched from secondary memory and is stored in RAM for processing. After process result is given to RAM and RAM saves it to secondary memory. There are two types of RAM, SRAM and DRAM

DRAM: DRAM stands for Dynamic RAM. DRAM is a type of ~~memory~~ random access ~~memory~~ semiconductor memory that stores each bit of data in a memory cell consisting of a tiny capasitor and a transistor.

SRAM: SRAM stands for Static RAM. It is a type of memory that uses latching circuitry to store each bit. ~~It is~~ As it is a type of RAM so it is also a volatile memory.

ROM: ROM stands for Read Only Memory. That is only read operation can happen on it. It is ~~No~~ non volatile memory. Data that is stored is ROM can't be modified or changed.

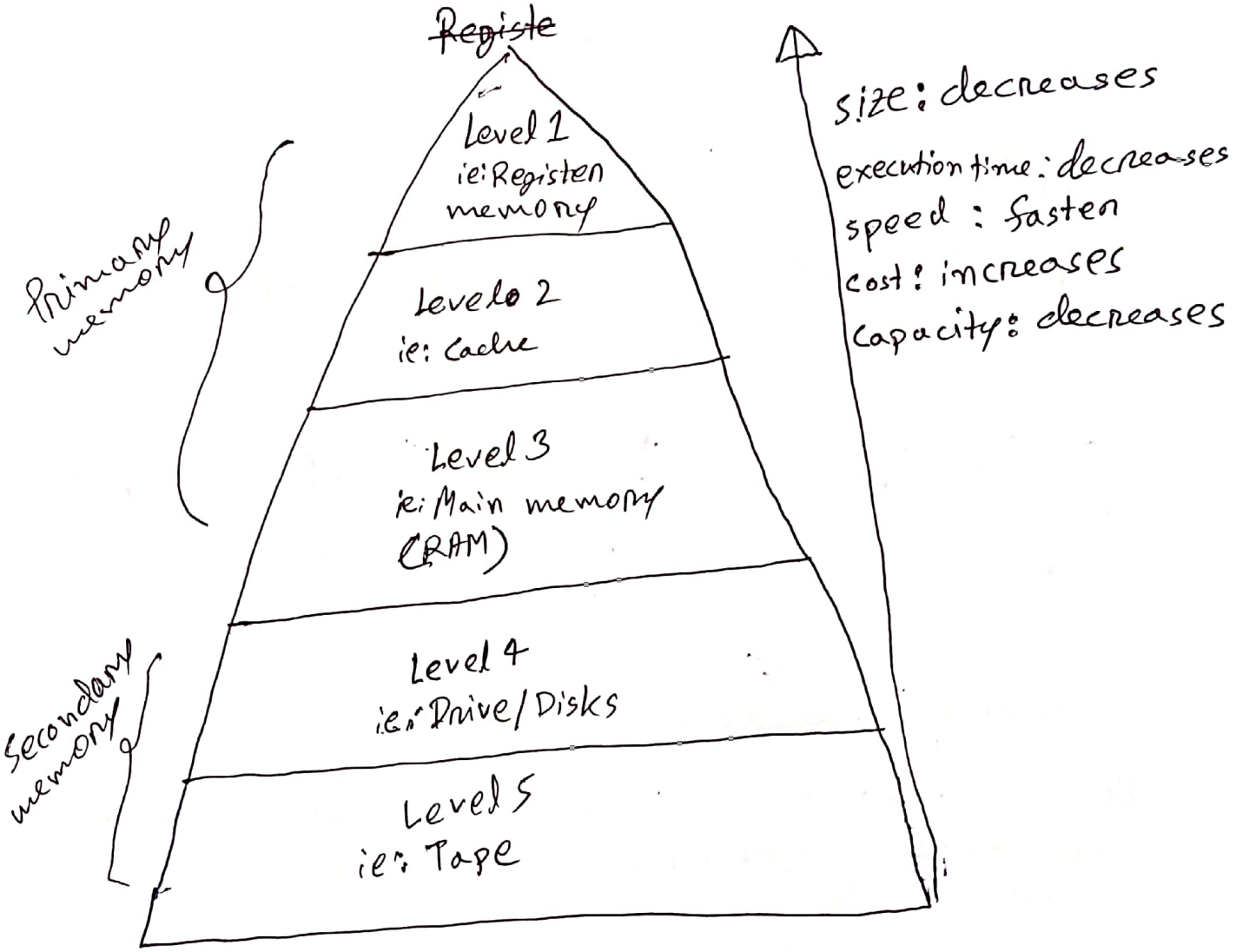
Cache: Cache is a level ~~D memory~~ memory. It is used by the CPU of a computer to reduce average cost ~~of~~ to access data from main memory or in other words from RAM. Cache is smaller, faster and located ~~to~~ closer to a processor core.

~~HDD: Hard Hard Disk Drive~~

HDD/SSD/Disk/Tape: Hard Disk Drive, Solid ~~St~~ State Drive, compact Disk, Floppy ~~to~~ tape these all are secondary memory that is used to store data permanently. Data can also be changed ~~or~~ modify or delete in these memories.

Answer to the Q. No. 2

Memory hierarchy drawn below:

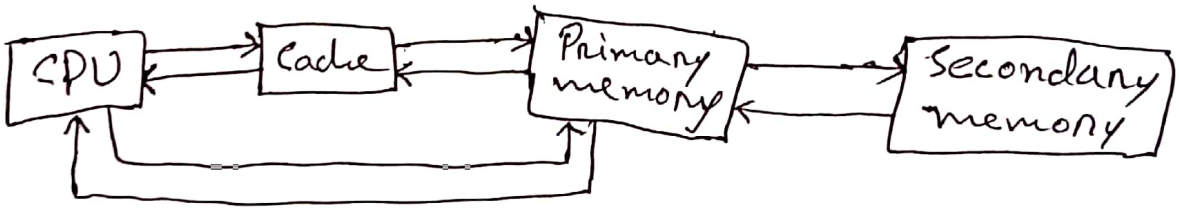


In computer architecture the memory hierarchy separates computer storage into a hierarchy based on response time. Since the response time, complexity and capacity are related the levels may also be distinguished by their performance and controlling technologies. From the fig we can see that

when we goto upper level from lower level the size on form factor decreases, execution time decrease thus increases speed, and capacity ~~is~~ also ~~are~~ decreases.

Answer to the Q. NO. 3

Steps of cache memory are given below.



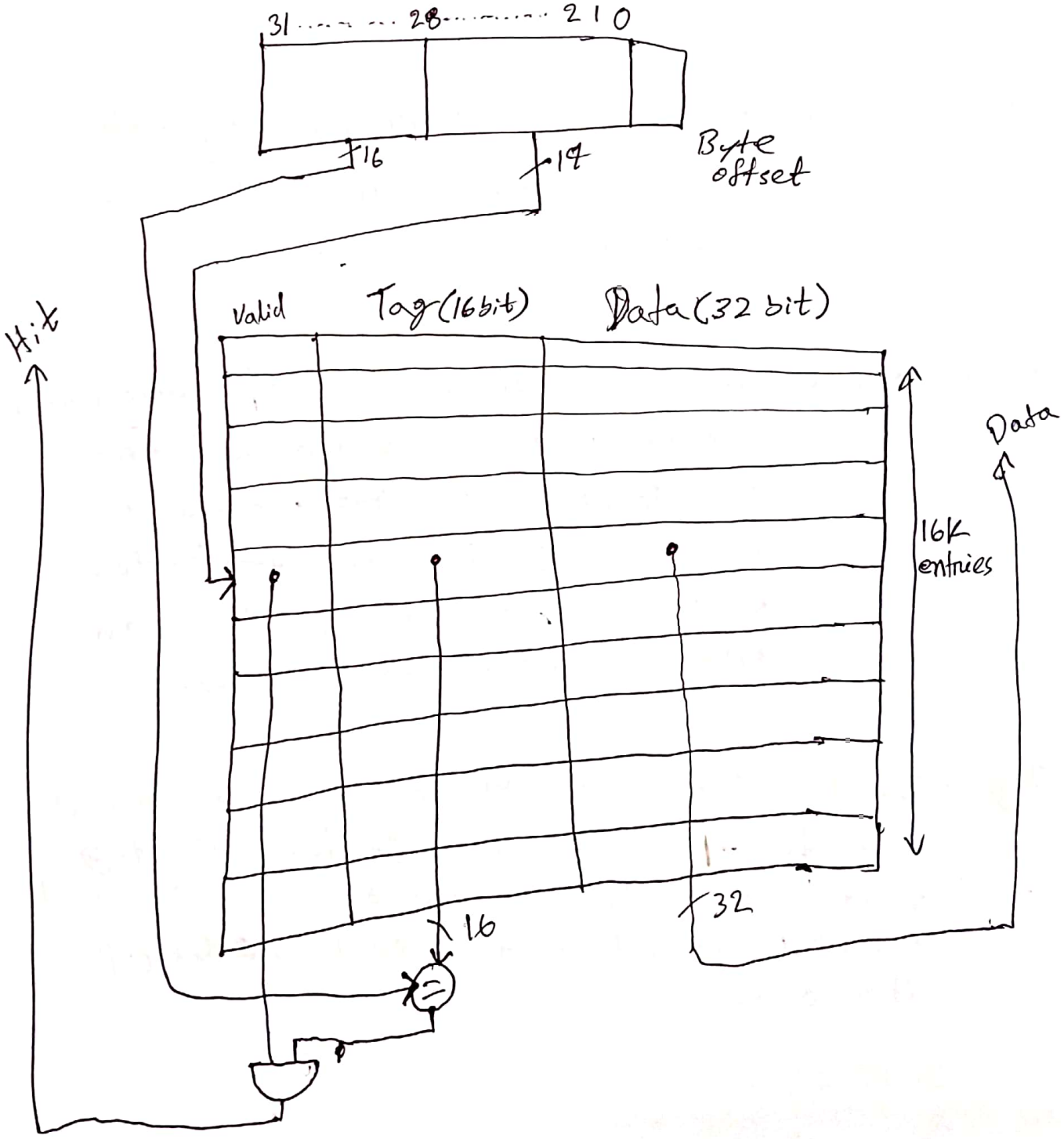
→ A request is made by the CPU to cache.

→ ~~If~~ If the data is ~~o~~ found is in cache then it is ~~retrieved f~~ returned to the CPU. This is called hit.

→ If the data isn't in cache ~~this is~~, then it is ~~retried~~ returned from ~~to~~ lower level memory like main memory or secondary memory. This is called miss. Data flows from lower to upper level by ~~two cofe~~ moving up ~~two~~ between two cojecutive memory level at a time.

Answer to the Q. NO. 4

Hardware logic of cache hit on miss drawn below:



Answen to the Q. NO. 5

Defination of the following terms:

→Cache: It is a hardware cache used by the central processing unit of a computen to reduce the avenage cost to access data from the main memony.

→Memony hienanchy: The memony hienanchy separates computen stonage into a hienanchy based on response time.

→Direct map cache: A cache where the cache location for a given address is detenmined from the middle address bits. If the cache line is 2^n the the bottom n address bits connespond to an offset within a cache entry.

→Tag: The tag field of the CPU address is compared with the tag of the line. If the two tag match then hit occurs and the desined wond is found in the cache. Otherwise it's a miss.

→ Hit time: Hit time is the time it takes to access a memory location in the cache.

→ Miss rate: ~~Mis~~ Miss rate is the ratio of total miss and total request.

→ Block: A memory block is a ~~conte~~ ~~cont~~ contiguous chunk of memory.

→ Hit: When CPU request for data. If the data is in cache then it is is hit.

→ Miss: When CPU requests for data. If the isn't in cache and has to be retrieved from ~~lower~~ lower level memory then it is a miss.

Answer to the Q. No. 6

Given,

Cache size = 32 words

Main memory size = 512 words

Block size = 4 words.

Thus

$\therefore$ Memory blocks = $\frac{512}{4}$ = 128 blocks

$\therefore$ Cache lines = $\frac{32}{4}$ = 8 blocks.

$\therefore$ Total bits required for physical address = $\log_2$ (no. of memory size)

$= \log_2 (512)$

$= 9$ bits.

We know that physical address is divided into 3 sections, idex, tag and block offset.

$\therefore$ Index = $\log_2$ (no. of cache lines)

$= \log_2(8)$

$= 3$

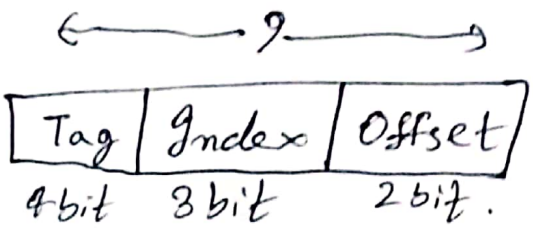
offset = $\log_2$ (Block size)

$= \log_2(4)$

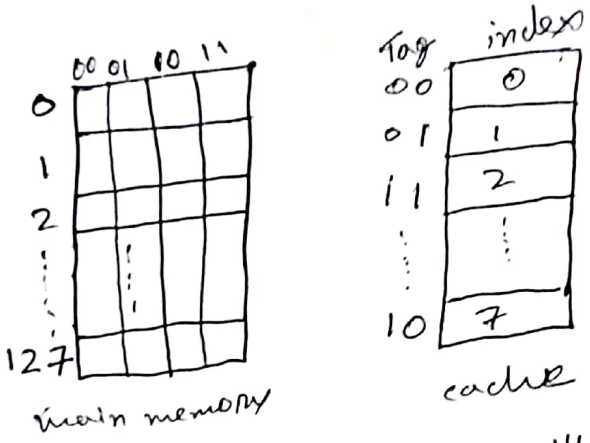
$= 2$

∴ Tag bit = 9-3-2 = 4

So,



Direct mapping cache config:



In case of ~~data~~ miss data will flow from memory to cache. ~~Each~~ So to ~~know~~ if it is ~~hit~~ or miss. we need to know cache config. If the requested tag's ~~word~~ index ~~matches~~ matches with the index bits of the physical address then its a hit otherwise its a miss. For miss we need to map data from memory to cache. Let assume we move block 12 to cache. For direct method we mod 12 with total cache block so we do 12%8 = 4. So block 12 ~~will~~ ~~go~~ of memory will goto block 4 of cache.

Answer to the Q. No. 7

Four questions to design cache are given below:

→ Where can a block be placed in upper level? (Block placement → mapping function)

→ How is a block found if it is in the upper level? (Block identification)

→ Which block should be replaced on a miss? (Block replacement)

→ ~~Which~~ What happens on a write? (Write strategy).

Answer to the Q. No. 8

Sequence: 0, 8, 0, 6, 6, 8, 7, 11

(i) Direct method:

Requested memory address	Hit/Miss	Blocks			
		0	1	2	3
0	miss	mem[0]			
8	miss	mem[8]			
0	miss	mem[0]			
6	miss			mem[6]	
6	hit			mem[6]	
8	miss	mem[8]			
7	miss				mem[7]
11	miss				mem[11]

(ii) 2 way set associative:

$$Set = \frac{4}{2} = 2$$

Now we mod block address by 2.

requested memory address	hit/ miss	Blocks			
		set 0		set 1	
		0	1	2	3
0	miss	mem[0]			
8	miss		mem[8]		
0	~~hit~~ hit	mem[0]			
6	miss		mem[6]		
6	hit		mem[6]		
8	miss	mem[8]			
7	miss			mem[7]	
11	miss				mem[11]

(iii) Full associative:

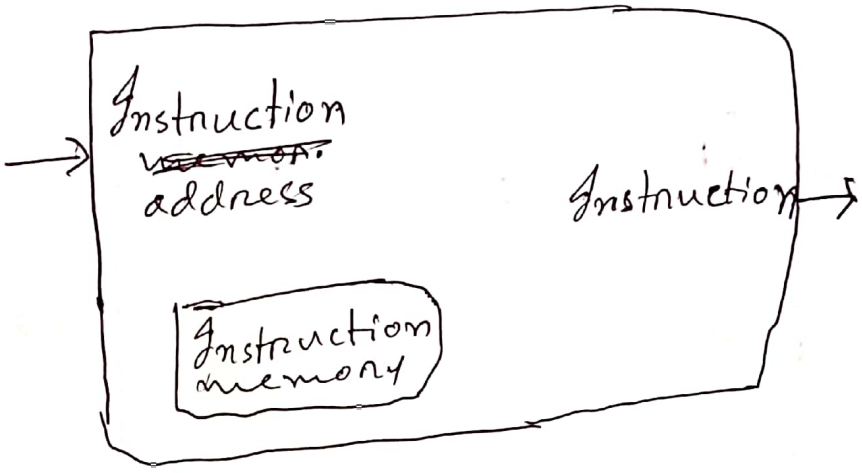
~~set = 4/4 = 1 ; we mod block address by 1~~

requested memory address	hit/ miss	Blocks			
		0	1	2	3
0	miss	mem[0]			
8	miss		mem[8]		
0	hit	mem[0]			
6	miss			mem[6]	
6	hit			mem[6]	
8	hit		mem[8]		
7	miss				mem[7]
11	miss	mem[11]			

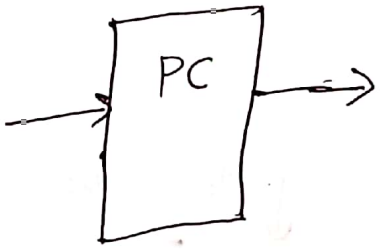
Answer to the Q. No. 9

All the functionals units of processor are drawn below:

(i) Instruction memory:

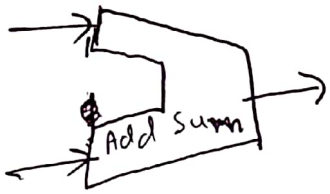


(ii) Program counter:

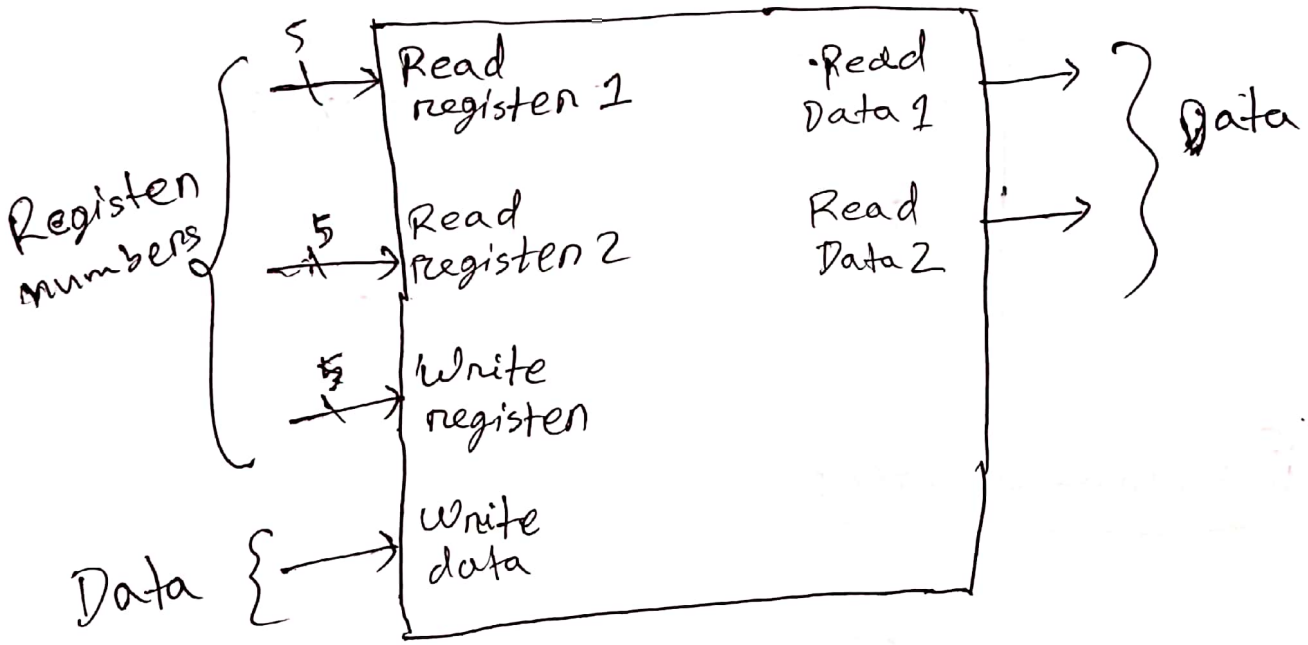


~~(ii) Adder~~

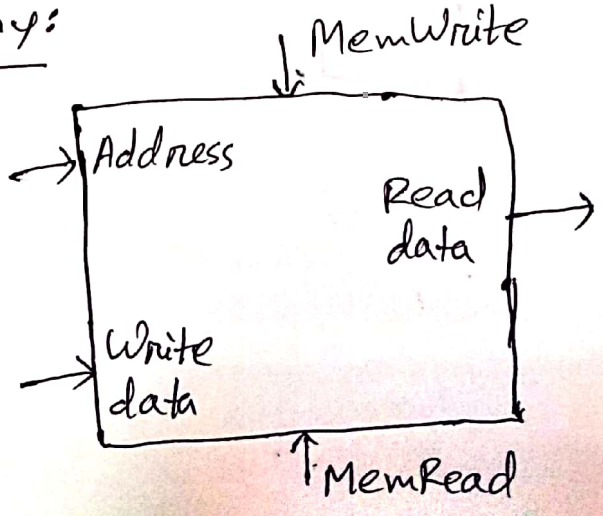
(iii) Adder:



(iv) Register file:

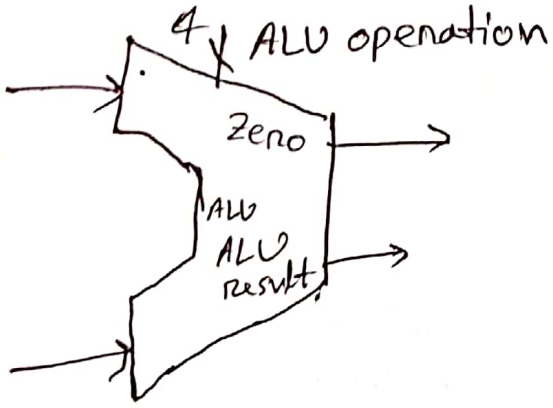


(v) Data memory:



(vi) Sign extend:

(vii) ALU



Answer to the Q. No. 10

R[i] = R[i+2] - Y

here,
i = 12

So,
R[12] = R[14] - Y

Mips instructions,

Considering,
R → $S0 (16)
Y → $S1 (17)
temp → $t0 (8)

(i) lw $t0, 56($S0)

(ii) sub $t0, $t0, $S1

(iii) sw $t0, 48($S0)

(i) ~~For the 1st~~

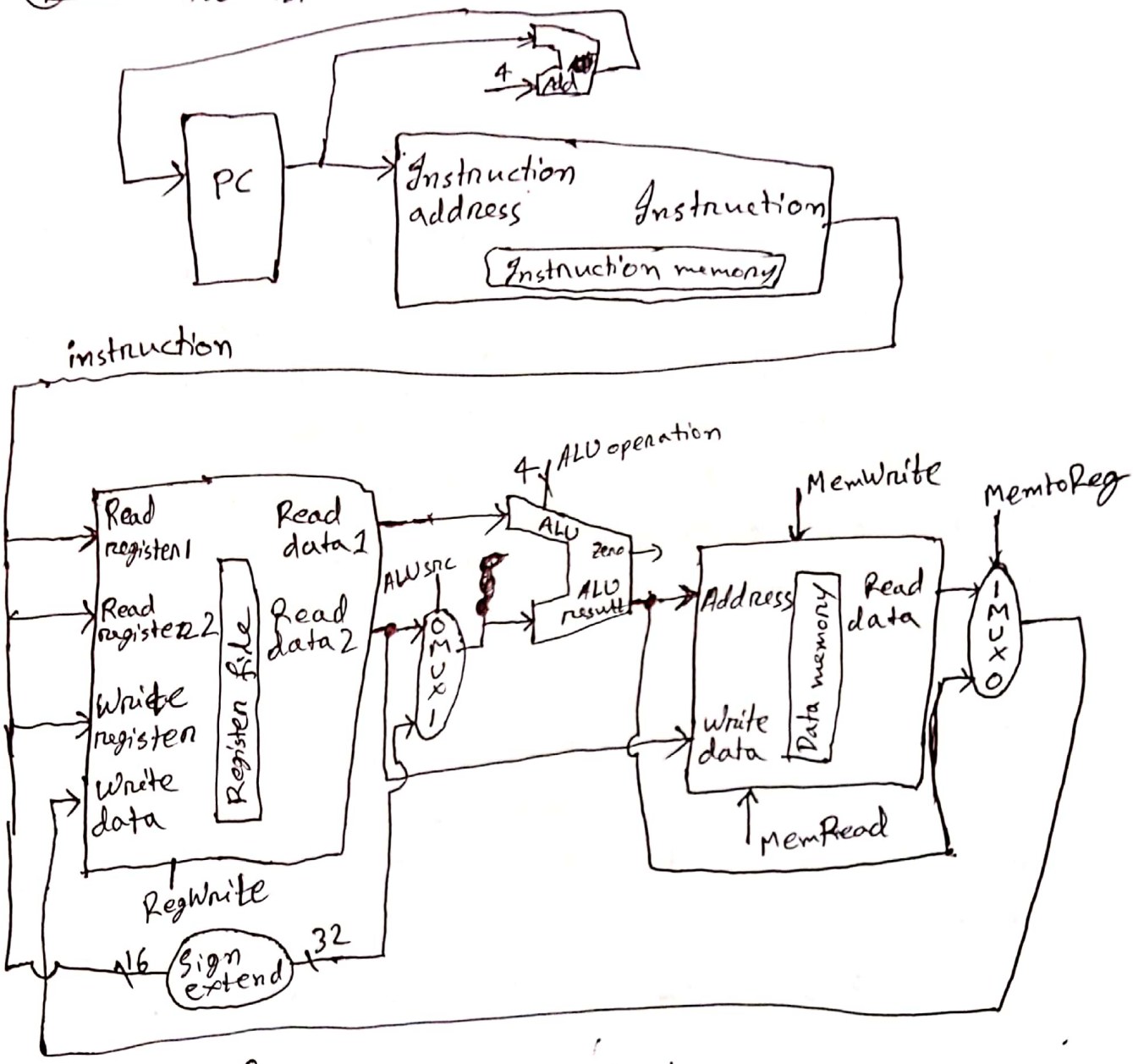


fig: Datapath organization.

Instruction address is placed on the instruction memory from program counter. As each instruction is of 4 words, so every time PC passes an address to instruction memory it increaments by 4 by using an adder. Instruction memory retrieves and passes the instruction, corresponding to that instruction address.

(i) For the 1st lw instruction

As it is an Itype instruction so the number of $s0 that is 16 would be passed to read register 1 and the content of $s0 that is the base address would come out of read data 1. The ALUsrc MUX is SET to indicate it's an Itype instruction. ~~offset~~ 16 bit offset would be passed to sign extend and will be converted to 32 bit offset. Now this ~~of~~ offset and base address would be passed to ALU and we'll get the physical address from ALU result. This physical address is passed to the address port of data memory. Now MemRead and MemtoReg are SET so data memory will fetch the data corresponding to that physical address and pass it to write data in register file & RegWrite is SET and the number of $t0 that is 8 is passed to write register. This is how data is fetched from memory and ~~stor~~ loaded in register.

(ii) For the 2nd sub instruction:

The 2nd instruction is Rtype so the ALUsrc is CLEAR. Now we pass the number of the register $t0 that is 8 to Read register 1 and the number of the register $s1 that is 17 to Read register 2. The content of these two register are passed to ALU then sub operation happen. The result is passed to MUX. MemtoReg is CLEAR so the result is passed to write data and ~~the~~ RegWrite is SET so the number of $t0 is passed to write register. Thus the result is stored is the register $t0.

(iii) For the 3rd instruction (sw):

This sw instruction is an Itype instruction so ALUsrc is SET. The number of $So that is 16 is passed to Read register 2 and the 16 bit offset is converted to 32 bit offset via sign extend. Then the content of $So that is the base address and the offset is passed to ALU. ALU returns the physical address and passes it to Data memory. The number of $to that is 8 is passed to Read register 2 and the content of $to that is the result comes out of Read data 2 and is passed to write data section of Data memory. MemWrite & MemtoReg are SET so the result is stored at that calculated physical address. This is how save instruction work.