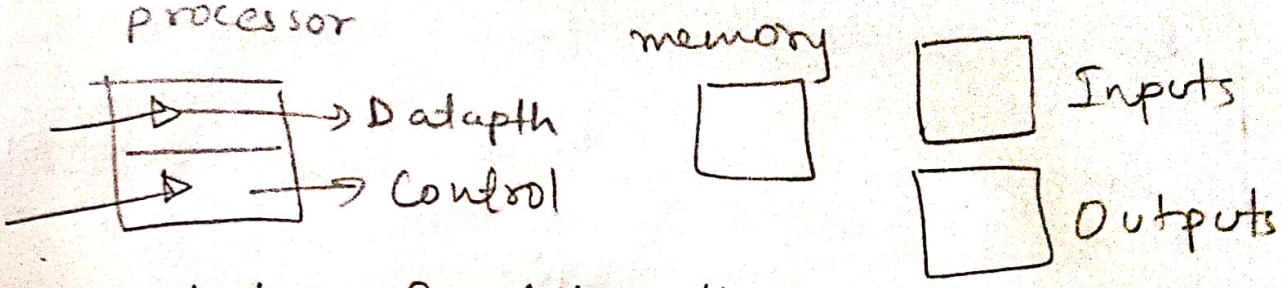
18101089 After Mid

Datapath:

Computer basic components



Flow of data is datapath.

→ flow from one register to another, or memory, or which step should come after which step is datapath

→ transmission of data is datapath.

Instruction have 2 parts:

1) Fetch cycle 2) Execution Cycle.

→ In a particular instruction set arch, every ins has same FC.

→ The steps for FC is same for all instructions.

→ For eg:- In MIPS, FC for all ins is same.

→ EC will be different

→ EC depends on types of instructions.

In MIPS, 3 types of instructions:

1) I-type (register + memory) +I

2) R-type (all register involved)

3) J-type. (Register or immediate constant value) I

1810089

→ Due to different categories, the EC depends on type of ins.

Processor:
Datapath contains functional units of processor.

SAP arch, unit called Control Word present 12-bit CW. (Only command to control)

Datapath: 1) Holds data. (12 bit)
2) Operate on those data.

To hold and perform ops, we need hardware elements / data store elements.

For FC, what elements do we need?

1) Instruction memory.

Ins mem structure given.
Input - Ins address Output → address content

→ address will come from Program counter.
→ Instruction memory is a memory element.
→ Ins mem is a memory that holds ins address.

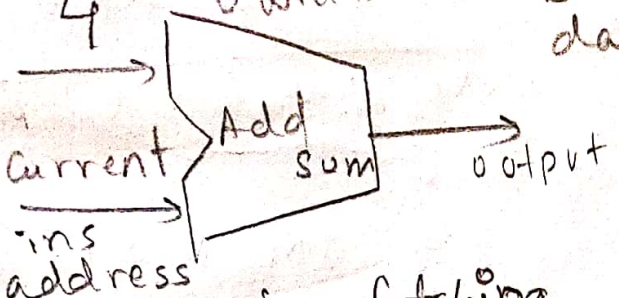
2) Program Counter: When you write a program, program lines are instructions. They are stored in PC. PC points the ins being executed currently. PC sends current ins add to IM.

18101089

PC does 2 things : 1) Holds current ins
2) After execution of current ins in that clock cycle, ins is incremented to send next ins using Adder.

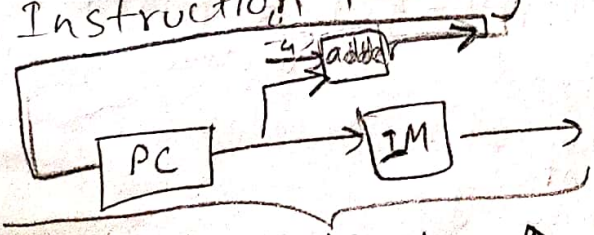
→ J-type ins breaks the sequence of PC. It jumps to another line instead of running program ins sequencially.

⇒ For FC, common elements are: Ins mem, PC and adder.



because in MIPS, for every data length you need 4 bytes.

→ Instruction fetching



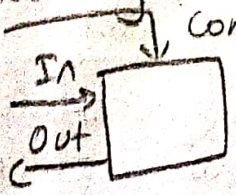
It is the hardware element / datapath for FC. And it is same for all ins.

* Same for all instruc ↑

* Instruction decoding: determining fields within the instruction.

* Ins execution : Perform op indicated by ins.

After decoding, ALU gets command.



which func to be executed
now, it'll be implemented thru CU.

There are steps of Datapath, explain by yourself using ins given.

Datapath has 2 elements : data hold — ①
operand on that data — ②

EU is ALU.

how will the register fields in ~~other~~ ins format be involved with add operation?

ALU reqd for all kinds of ~~operation~~ ins : address calc, data calc. In func selection of ALU, add & func command will be given. Then ALU performs calculations.

Ins execution : Register R/W / Mem R/W, func execution in ALU.

FC (Datapath) is complete now.

EC depends on type of ins.

We have seen 3 elements : PC, IM, adder.

New element : Register file for R-formad ins.

2 source registers read and values sent to ALU, then result in register write.
from register file element.

→ Write data is a command.

Reg write is a part of CU.

control ~~CU~~ bit set to 1, output data should be write to write register

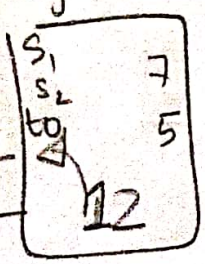
RD1 - content of S_0
RD2 content of S_1

Write data — content of dest. register.
when write, moved to write register.

A = B + C

B = 7
C = 5

A → to
B → S_1
C → S_2



content of S_1
content of S_2
content of to

After calcu, data kept in Write Data. When C bit=1, output goes to WR (write register) otherwise not.

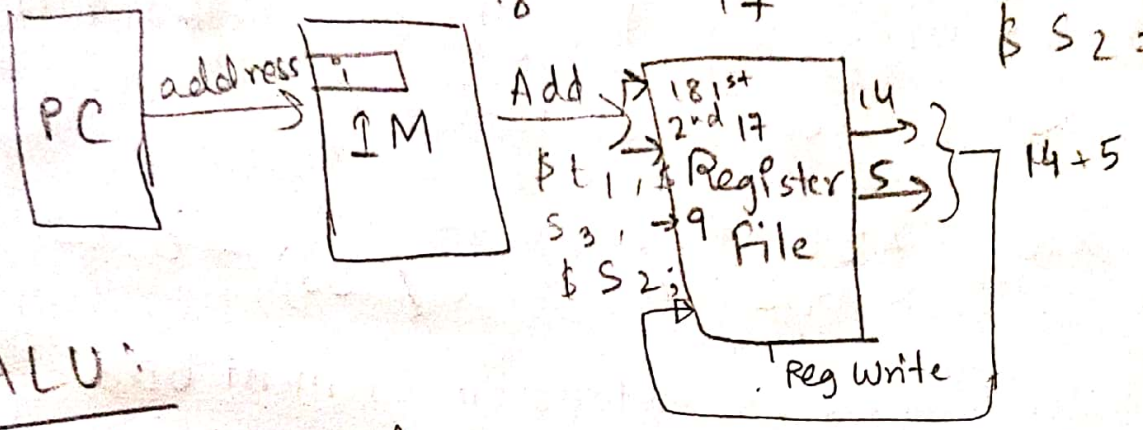
C bit → control bit

① Add t_1 ; s_3 ; s_2 ;

↳ 9 ↓ 18 ↓ 17

$s_3 = 14$

$s_2 = 5$



ALU:

4 bit wide operation selector value.

$2^4 = 16$ operations in ALU.

Zero is zero flag.

status flag → zero flag.

ZF = 1 result is zero. — from ALU

R-type 5 elements : PC, IM, RF, ALU, Adder

Read and Decode Insuc^n ⇐ IM.

which register no. is passed to RR1, RR2, WR.

I-format Ins: Lecture 05

→ base register, source/destination.

lw → dest, base (read)

sw → source, base (write)

Datapath for memory read/write:

Datapath consists of processor's functional units.

rs – source register (base register)

rt – source/destination

lw $t0 , 32($s0) → rs

rt

1) for sign extending, sign extension element used.

→ convert 16 bit to 32 bits

op	rs	rt	address
6	5	5	16 bits → 32 bits (convert)

16 bit value : (2)

sign extension, msb

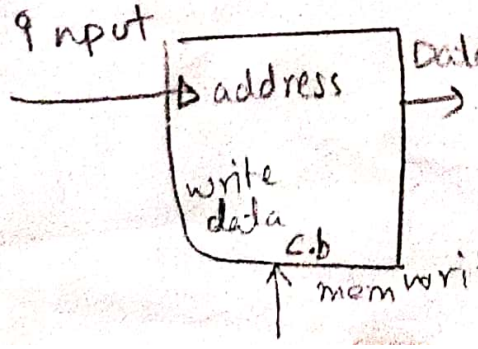
If 0, 16 0s will be copied on left

If 1, 16 1s will be copied on left.

6th element – sign extension element.

Datapath for load word (lw):-

7th element – Data memory on which address to perform memory read/write.



(rs) base register + immediate value → mem address

In ALU, rs + imm = physical address, using PA we get address, from there we read data and is sent to WriteData.

II. lw $t0, 32($s1);

Datapath:

As in R-type, 8 steps were described. Here, the steps are not described. We have to describe ourselves.

① PC provides ins address, ins address sent to IM

② From IM, ins decoded and sent to Register File.

Instruction is lw $t0, 32($s1);

③ In RF, rd, rs. ~~Sent to~~ $s1 to rs, 32 is 16 bit value, 32 sent to SE (SignExt) converted from 16 → 32 bits. RD1 and 32 bit value to ALU. (base reg value and offset value). ALU gives PA, ~~sen~~ address sent DM (Data Mem).

④ In DM, the data or content in that address is read. The content is sent to Write data of RF (Reg File) when RegWrite is 1. In rd, to is passed after calculation.

⑤ Last clk cycle, Adder adds 4 bytes to current ins address and it is passed to PC.

Datapath for lw:

lw $t0, 32($s1);

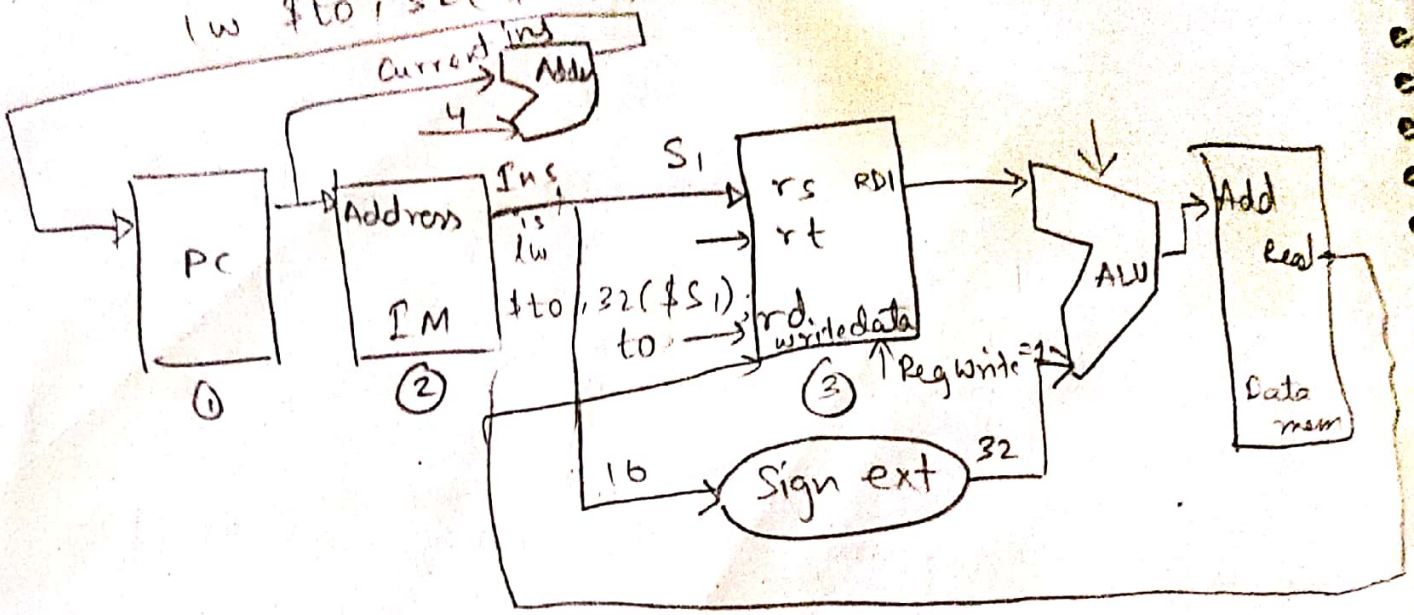


Fig: DP for lw

DM has 2 control units: MemW / MemR
We should set control bit for preferred op.

Store Word:

sw $rt, immed ($rs);

$rt's contents will be stored in that mem's specific add.

→ Everything will be same as lw till add calcu.
→ From RF, RD2 to DM, write data.
Here MemWrite should be active.

R-I type DP:

2 MUXs added – 2×1 MUX

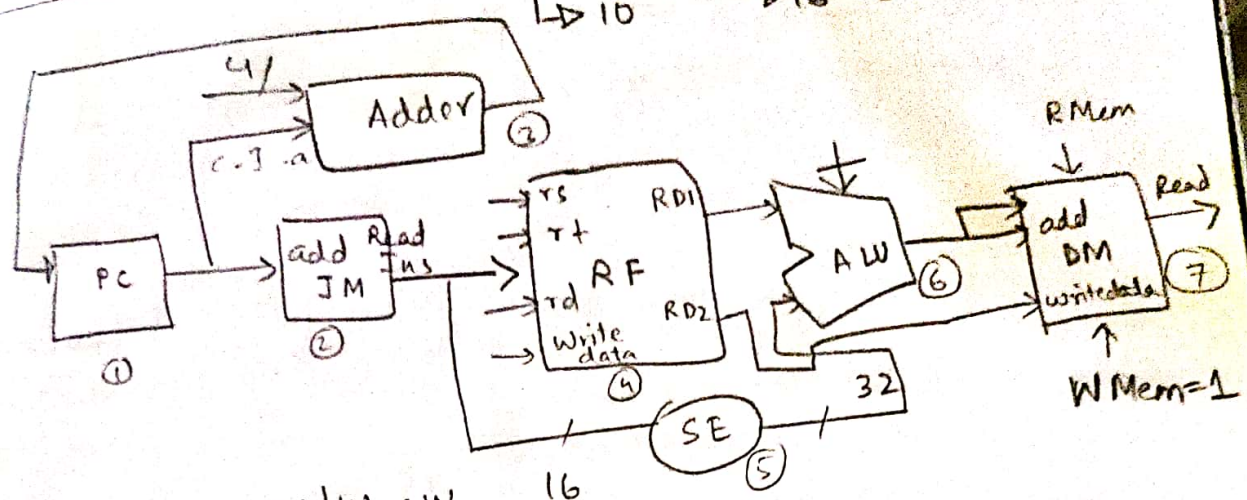
1st mux → I_0, I_1 till IM same (FC)

Type of ins selection:

I_0 → 0 (R-type) RD2
I_1 → 1 (I-type) SE

DP for sw:- sw $t2, 100($s0);

↳ 10 rt ↳ 16 rs



I-type → lw, sw

elements – 7 total

R + I DP:-

1st MUX selects type of Ins.

1 → base + SE value → PA (memory address)

0 → RD2 only → ALU ~~does~~ R-type ins. calcu according to

2nd MUX ALUSrc – 0/1

1 → Mem to Reg (I-type) # write for I/R type?

0 → R-type

→ last MUX going to R-type result.

Branching instructions:-

DP for BEQ

BEQ falls in I-type

Since in PC sequence is broken, because of breaking the value of sign ext. is used to calcu

branch
↑ target address using add sum.
→ Another element used here for shift.
→ Datapath = Processor's functional Unit
→ Branch will not normally be given in exam.

→ Jump target address of instruction.
→ In Jump, address will be changed, and sequence will be broken. Because of that, 2 bit left shift is used and address is calculated and sent to PC.

Single Cycle Control :-

→ CU is present in ALU, Register File, MUX (control func is controlled), memory read/write control regarding function is considered.

ALU Control lines :-

SAP 1 has control wires which defines in which block cycle what should happen.
→ ALU has function table, func table values are changed to perform diff operations.
→ Rest should reviewed by ourselves
→ CU is added in single cycle control, to decide which bit should go where.

Datapath ends here.

18101089

Quiz - 3

Answer to Question 1 a:

```
R[i] = R[i+55] + Y;
```

∴ i = 89

```
R[89] = R[144] + Y;
 ↳ $0      ↳ $0    ↳ $1
```

MIPS instructions

(i) lw $t0, 576($S0);

(ii) Add $t0, $t0, $S1;

(iii) s.w $t0, 356($S0);

Let $S0 = 16, S1 = 17, t0 = 8.

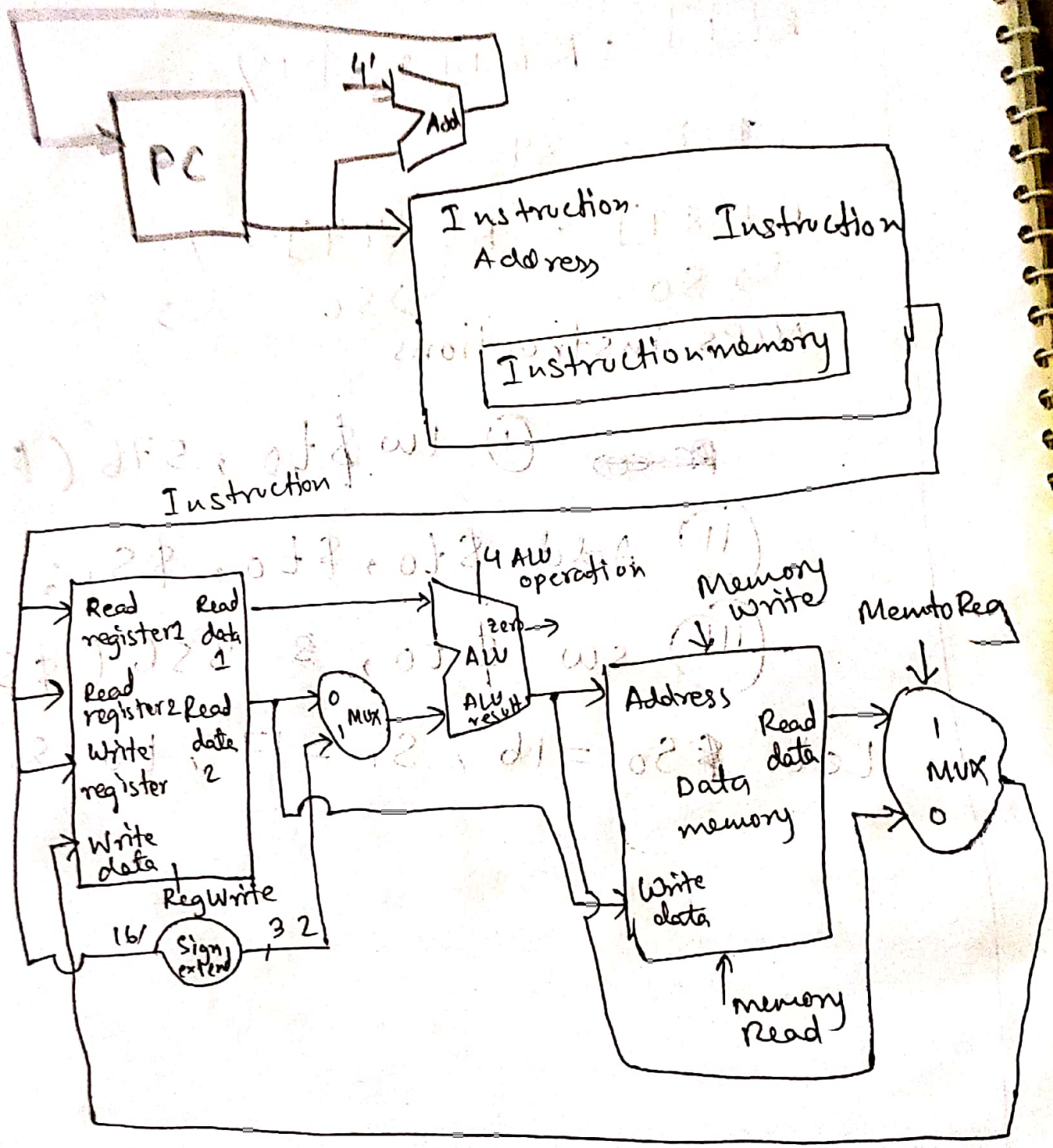


Fig: Datapath organisation

Instruction address is placed on the instruction memory from program counter. As each instruction is of 4 words, each time PC passes an address to instruction memory and it increments by 4 bytes on adder. Istruction memory retrieves and passes the instruction corresponding to the instruction address.

(1) lw instruction:
As it is an Itype instruction, so the number of $s0 ~~that~~ is 16, it would be passed to read register 1 of Register File. The content of s0 that is the base address would come out of read data 1. The ALUsnc MUX is set to indicate it's an I type instruction. 16 bit offset would be passed to sign extend and will be converted to 32 bit offset. Now this offset and base address will be passed to ALU and we'll get the physical address from ALU result. The physical address from address of data memory. Now MemRead and MemtoReg are set so data memory will fetch the data

corresponding to physical address and pass it to write data in register file. RegWrite is SET and the number 'to' is 8 which is passed to write register. This is how data is fetched from memory and loaded in register.

(11) Add operation:

This instruction is R-type, so the ALUsrc is CLEAR. Now, we pass the number of the register to, that is 8 to Read register 1 and number of register 5, that is 17, to read register 2. is passed

The content of those two registers are passed to ALU for add operation to occur. The ALU result is passed to MUX. MemtoReg is CLEAR so the result is passed to write data and RegWrite is SET so the number of to is passed to write register. The result is stored in register to.

(iii) sw operation:

This is I-type instruction so ALUsrc is SET. The number of So that is 16 is passed to Read register 2 of register file and 16 bit offset is converted to 32 bit offset via sign extend element. The content of So that is the base address and offset is passed to ALU, ALU ~~result~~ returns physical address and passes to Data memory. The number of to (8) is passed to read register 2 and content of to, that is the result comes out of read data 2 and is passed to write data section of data memory. MemWrite and MemtoReg are set so the result is stored at that calculated physical address.

Answer to Question 1b :-

There are 5 basic components in MIPS single cycle data path. These are given below:

1) Instruction memory
2) Program counter
3) Adder
4) Register file
5) ALU

~~Program~~ Register File : A register file is a collection of readable and writeable registers.

ALU : It performs the operation indicated by instruction. It takes two operands and result is calculed in ALU which is the output value.

Program Counter :

*focus: R-type and I-type lw, sw, combined DP for I and R-type. Branching less important.

Example of DP: so → s_1

A[10] = Y + A[12];

Draw and explain DP for MIPS Instruction set.

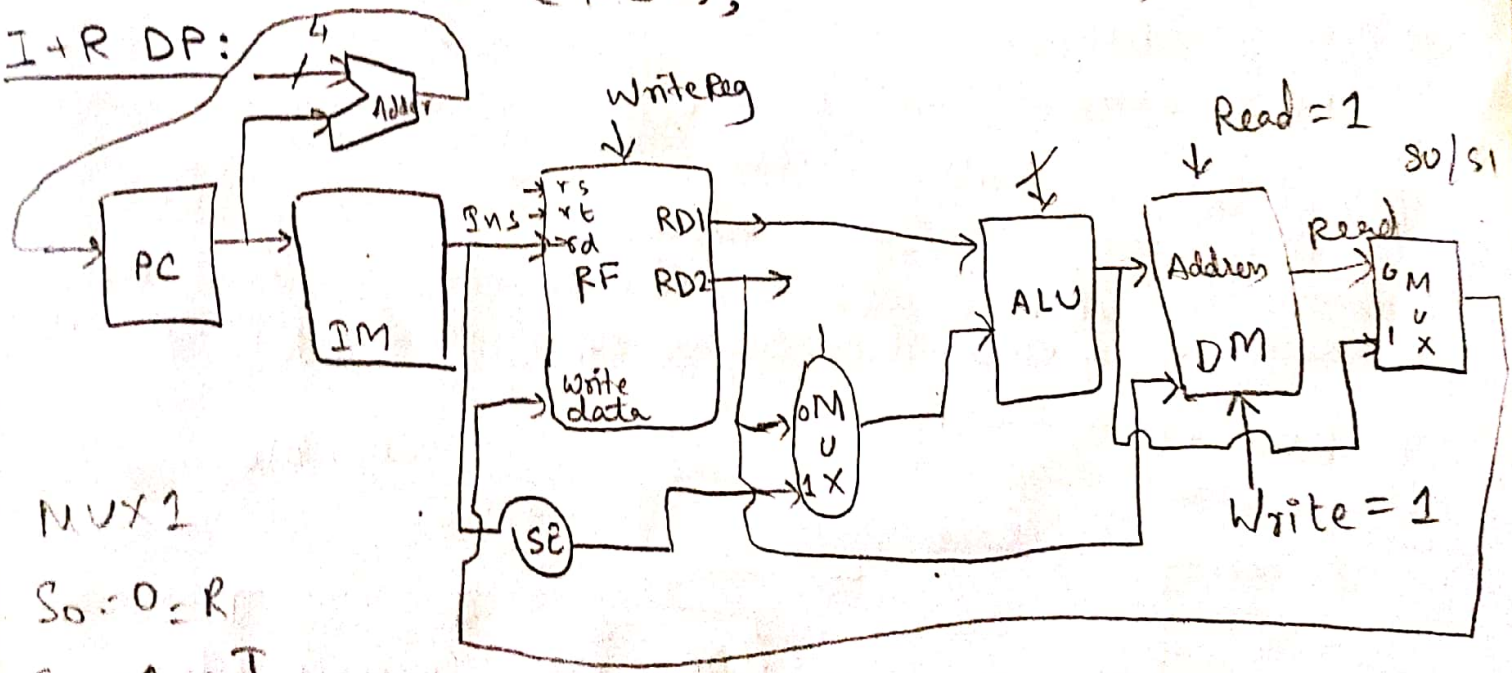
1) Required elements: (R and I)

i) PC ii) IM iii) Adder iv) Register File
v) ALU vi) Sign extend vii) Data Memory
viii) Multiplexer (2:1)

2) Assembly code: lw $t0, 48($s0);
Add $t0, $s0, $t0;
sw $t0, 40($s0);

$s0 = 16, t0 = 8

I+R DP:



MUX 1

S_0 : 0 : R

S_1 : 1 : I

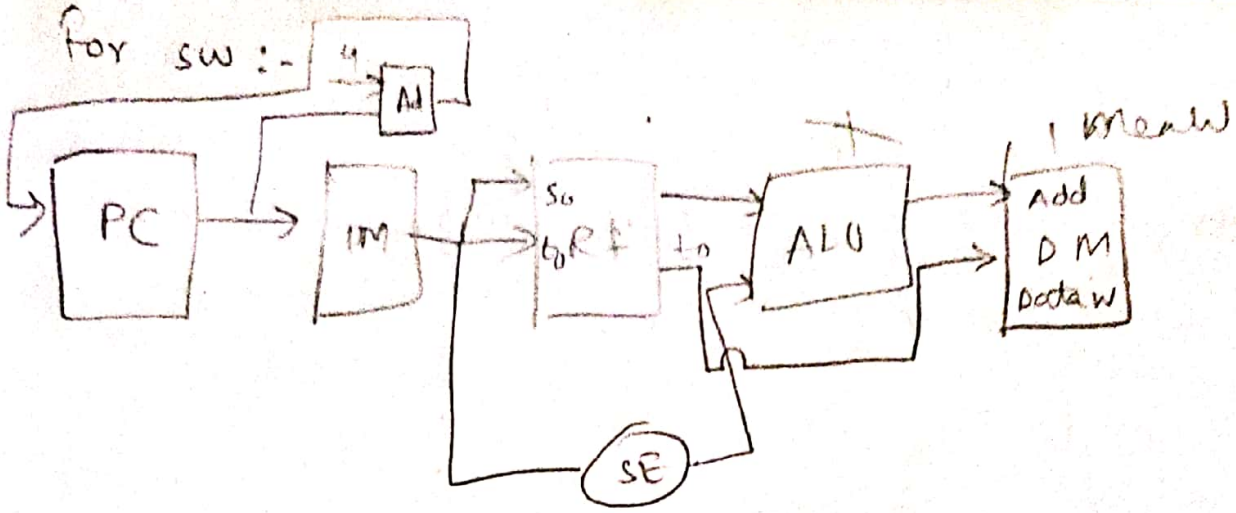
S_0 → read (M to R) = I

S_1 → write = ~~R to~~ M ALU result to register = R

MUX2

Operador - Adder, MUX, ALU

2nd MUX no need for sw.



Chapter - Memory

2 types
- Primary
- Secondary

→ Memory hierarchy
→ locality
→ Design pattern
→ Memory mapping funcs

Mem Read/Write

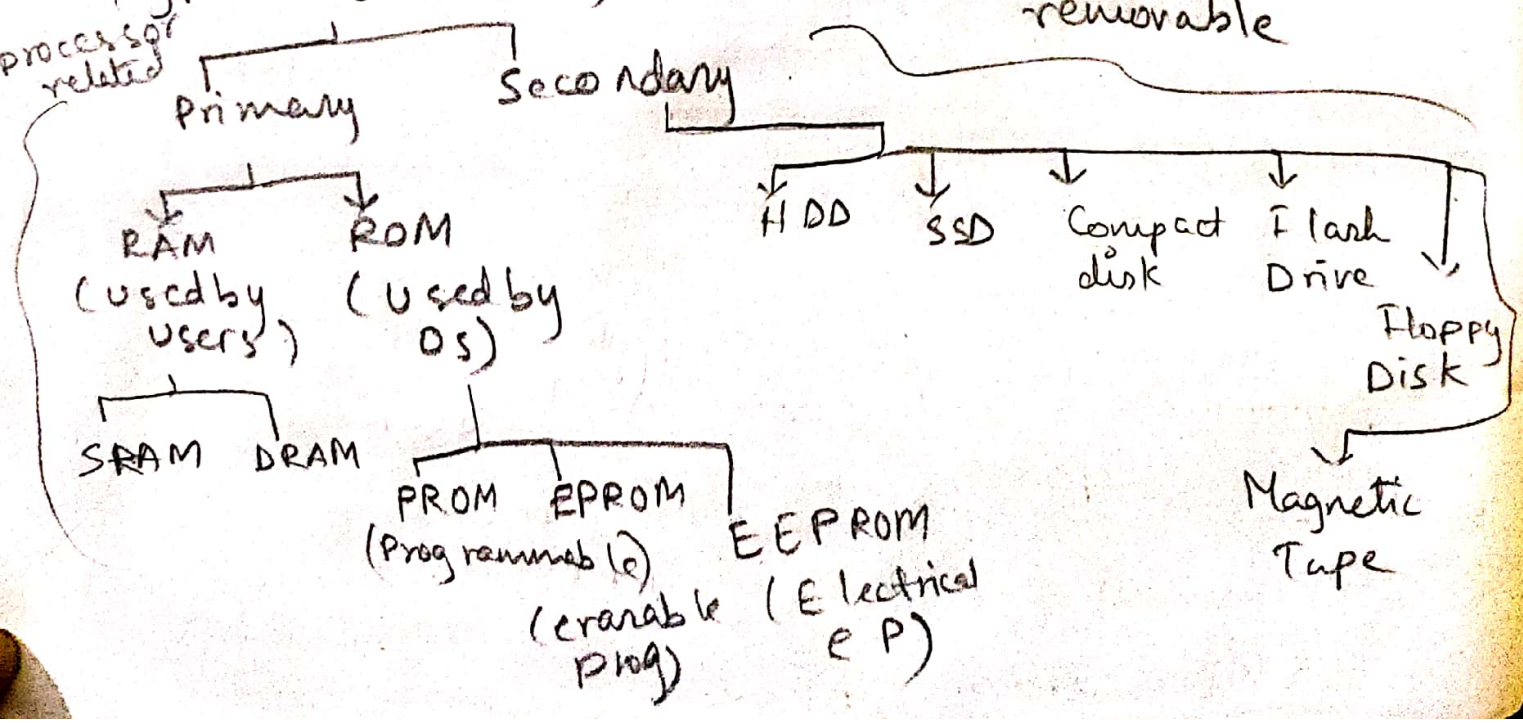
#Memory : (Design)

Reading table and bookshelf → Cache and memory
2 memory levels

Types of memory :
1) Primary 2) Secondary

Memory is storage area in which programs can find their running data and it contains data used by programs.

Types of memory



Dynamic changes but static faster (static used in cache)
DRAM slower than SRAM, used in main memory
SRAM → F-F constructed so faster.
DRAM → Capacitor used to ready chip, so slower.

PROM → can be programmed to a particular range by user (only for resource allocation, not for user programming).

EPROM → has range, how many times you can erase and P depending on version

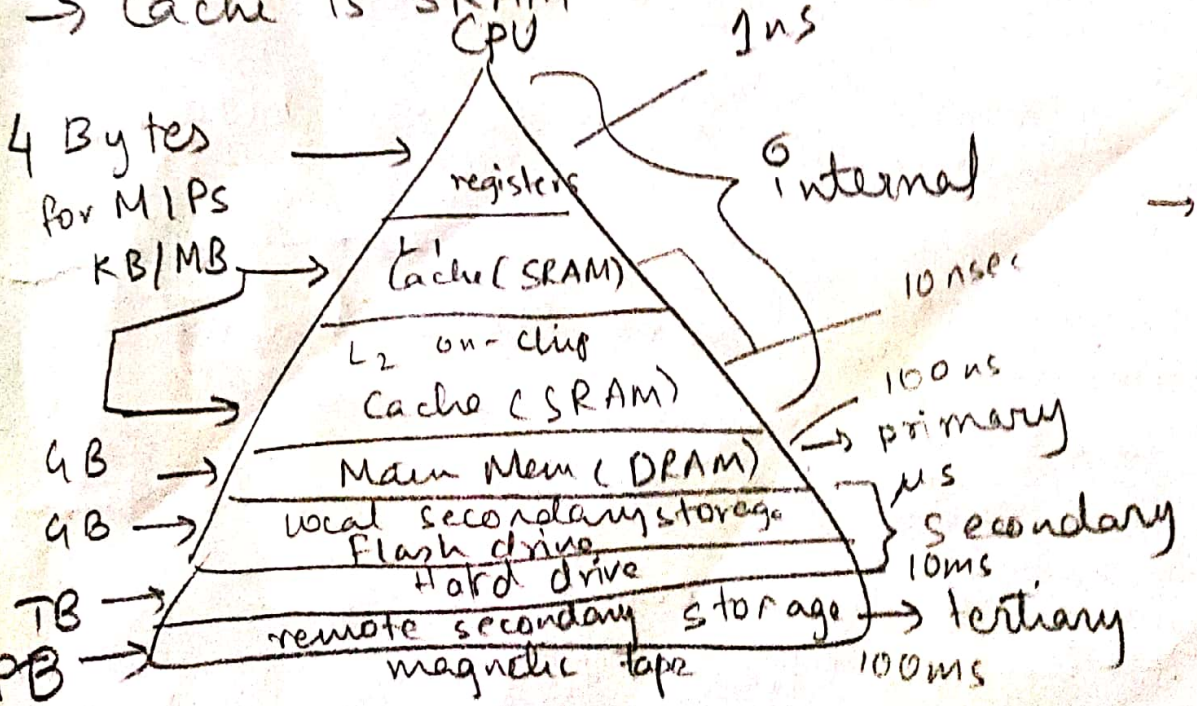
EEPROM → others by users, this electrically. EPROM - UV rays used for changing; EEPROM → electric field used for changing

Primary — related to processor

Memory hierarchy:

for making work carier, diff way storage levels maintained.
→ For time saving } why? Time cost related
→ Cost
→ Speed Time, speed related
→ Size

→ Cache is SRAM



→ further from CPU, mem size increases
→ bigger size, slower

Fig: Level of memory

→ Cache faster than main memory. Requests from CPU goes to memory levels, bigger storages will be slower due to access time, since bigger storages away from CPU more distance, so more access time. in memory, adjacent levels info are exchanged. (dis and time)

→ ~~cost~~ registers more costly as less quantity more required

→ Access time – memory size inc, access time inc.
→ Speed inc when size / capacity of memory dec.

↑ speed ↓ size ↓ Access time (AT)

↑ smaller, faster, expensive.

↓ Larger, slower, cheaper.

spatial → space, temporal → time

Lecture 15 (37:33)

array (data) – spacial som iteration → temporal.
(instruc) sequence – spatial cycle loop – temporal

→ mapping data transfer (42:50) between 2 adjacent levels.

→ If requested block in cache memory, then data hit, otherwise data miss. If not in cache memory, then bring from main memory.
hit time and miss time / penalty time.
Data from main mem copied to cache, then accessed by CPU delivered to CPU processor.
+ CPU request → ~~miss~~ hit time

The access time is hit time
From CPU accessed, then processor will check if the access time is hit time.
of that block
→ When cache full, replacement placement

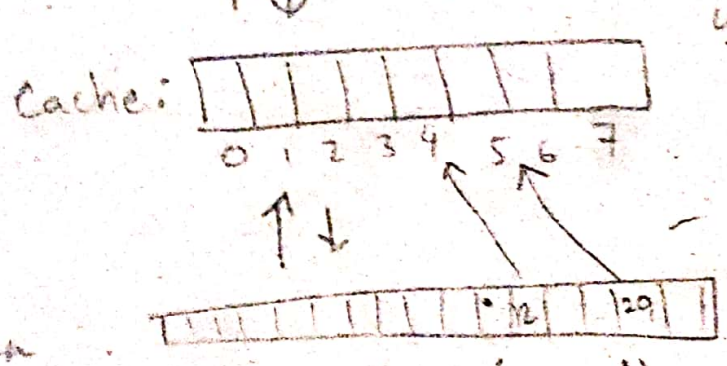
mapping function → ① Direct map
② Set associative
③ Fully associative

ECC memory — chipset that error correction, with RAM they does byte error correction.

Memory mapping functions:-

If cache has the data requested, the data is sent to memory (CPU). If not, goes to main memory.

1. Direct mapping → There are only one place (fixed).



Eqn:
requested address mod # of block in cache

$\frac{12}{8}$ (mod) ⇒ 4 (of cache)

$\frac{29}{8}$ ⇒ 5

memory blocks are given identifiers like token called tag.

3 requested addresses:

10 111 → index * both must match

11 001

10 110

---

2. Set associative → Block can be placed anywhere within the set.

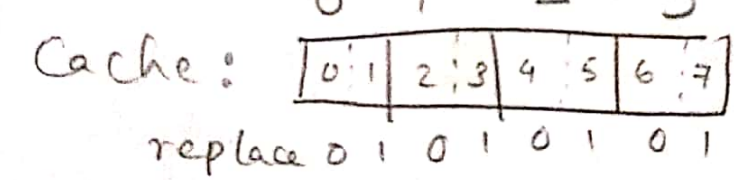
Here, set is 2^n (the power).

2-way set associative
4-way "
8-way "
n-way "

For 2-way,

# of block in cache ÷ frequency
How many sets (n-way)

i) ⇒ $\frac{8}{2}$ = 4 sets



ii) To which set?

(Requested Address) mod (# of set)

12 mod 4 = 0th set

→ Now, you can either place in 0/1 of 0th set

If 0 used, put in 1. If 1 used, put in 0.
If both has data, 2 algos:

* least recently used
* most recently used.

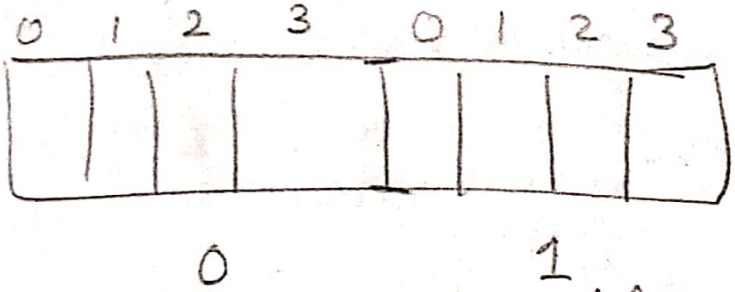
4 way:

i] $8 \div 4 = 2$ sets

ii] # of set = 2

$12 \div 2 = 0^{th}$ set

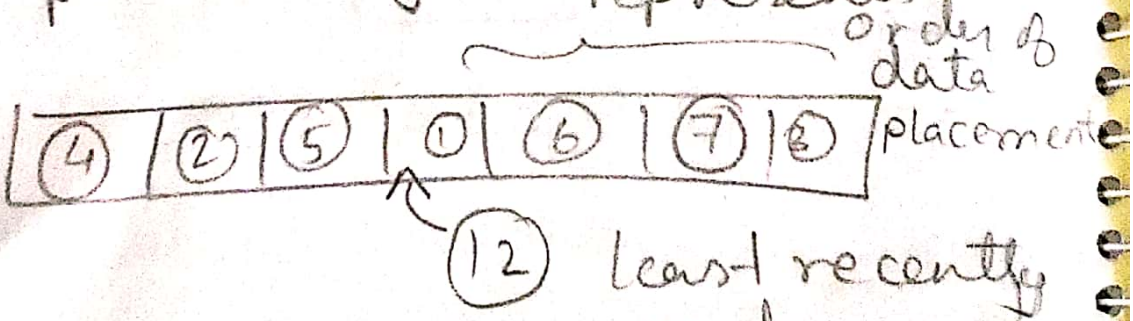
(mod)



If 27 is requested address,

$27 \bmod 2 = 1^{st}$ set

3) Fully associative → Block can be placed anywhere.



Example: Find the number of misses and hits for a cache with four 1 word blocks.

Given the following sequence of memory block access.

0, 8, 0, 0, 6, 8.

for each of the following cache configuration i) Direct mapped

ii) 2 way SA (LRU)

iii) Associative mapping

i) Direct mapping:

→ requested addresses: 0, 6, 8

→ 4 blocks of 1 word.

$0 \bmod 4 = 0$

$6 \bmod 4 = 2$

$8 \bmod 4 = 0$

Initially cache all slot = 0

requested address	Hit/ miss	Content of cache block after each reference			
		0	1	2	3
0	miss	~~Mem[0]~~			
8	miss	~~Mem[8]~~			
0	miss	Mem[0]			
0	hit	Mem[0]			
6	miss			Mem[6]	
8	miss	Mem[8]		Mem[6]	

misses = 5 # hit = 1

2) 2-way SA:

Sequence: 10, 8, 0, 6, 0, 8, 6, 12, 5)

Address mod No. of blocks

requested address	Hit/miss	content of cache after cad ref			
		Set 0		Set 1	
		0	1	0	1
0	miss	Mem[0]			
8	miss		Mem[8]		
0	hit	Mem[0]			
6	miss	LRU	~~Mem[6]~~		
0	hit	~~Mem[0]~~			
8	miss		Mem[8]		
6	miss	Mem[6]			
12	miss		mem [12]		
5	miss			Mem [5]	

We have in Cache 6, 12, 5.

miss = 7

hit = 2

3) Fully associative (0, 8, 0, 6, 0, 8, 6, 12, 5)

requested address	Hit/miss	Set 0		Set 1	
		0	1	0	1
0	miss	Mem[0]			
8	miss		Mem[8]		
0	hit	Mem[0]			
6	miss			Mem[6]	
0	hit	~~Mem[0]~~			
8	hit		Mem[8]		
6	hit			Mem[6]	
12	miss				Mem[12]
5	miss	Mem[5]			

5, 8, 6, 12 ✓

hit = 4

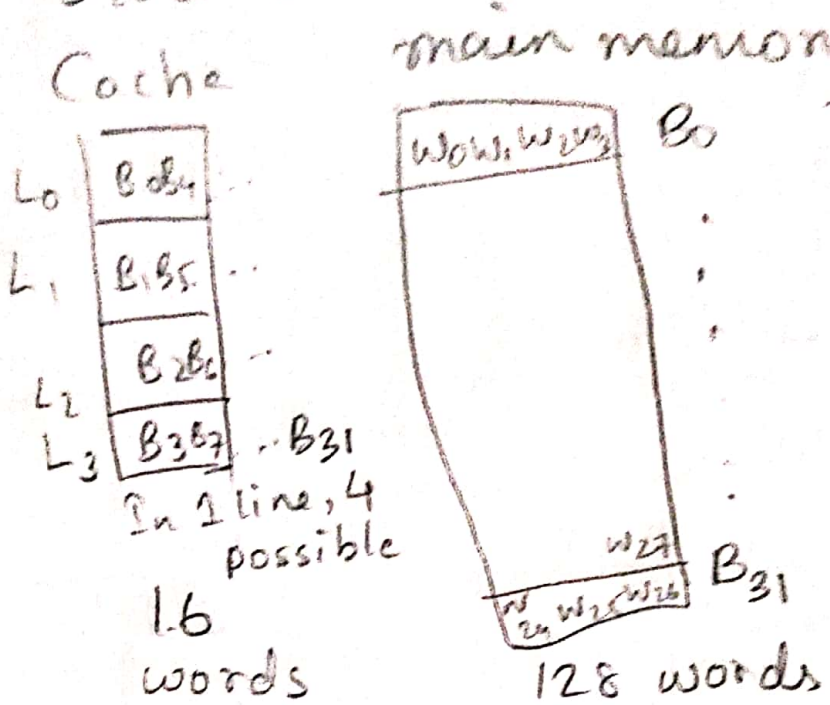
miss = 5

Memory Organisation:

Main memory = 128 words

Cache = 16 words

Block size = 4 words



$$\frac{\text{Cache size}}{\text{Block size}} = \text{Cache line}$$

$$\frac{16}{4} = 4$$

$$M.M \Rightarrow \frac{128}{4} = 32$$

K mod n → No. of lines in cache

↳ requested address

0 mod 4 = 0

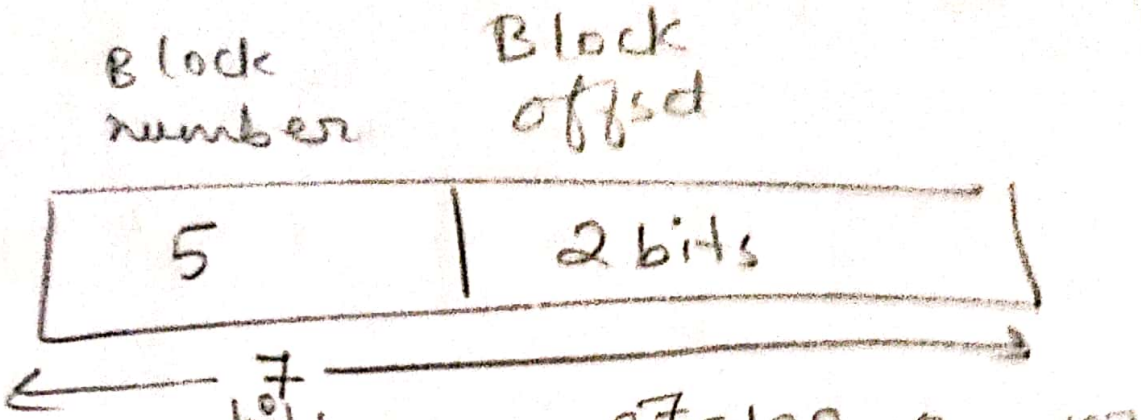
1 mod 4 = 1

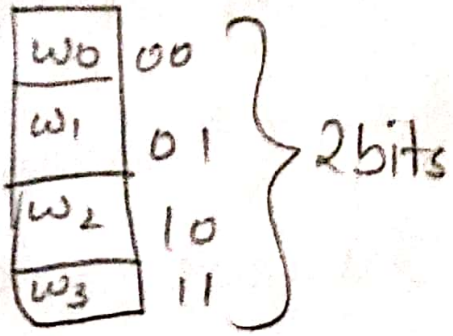
2 mod 4 = 2

3 mod 4 = 3

4 mod 4 = 0

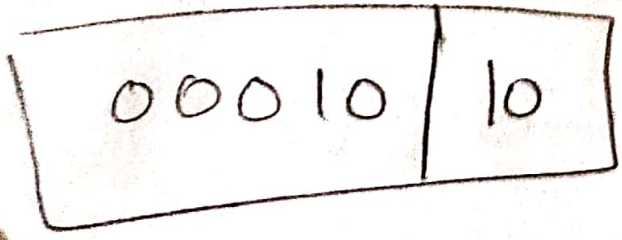
5 mod 4 = 1

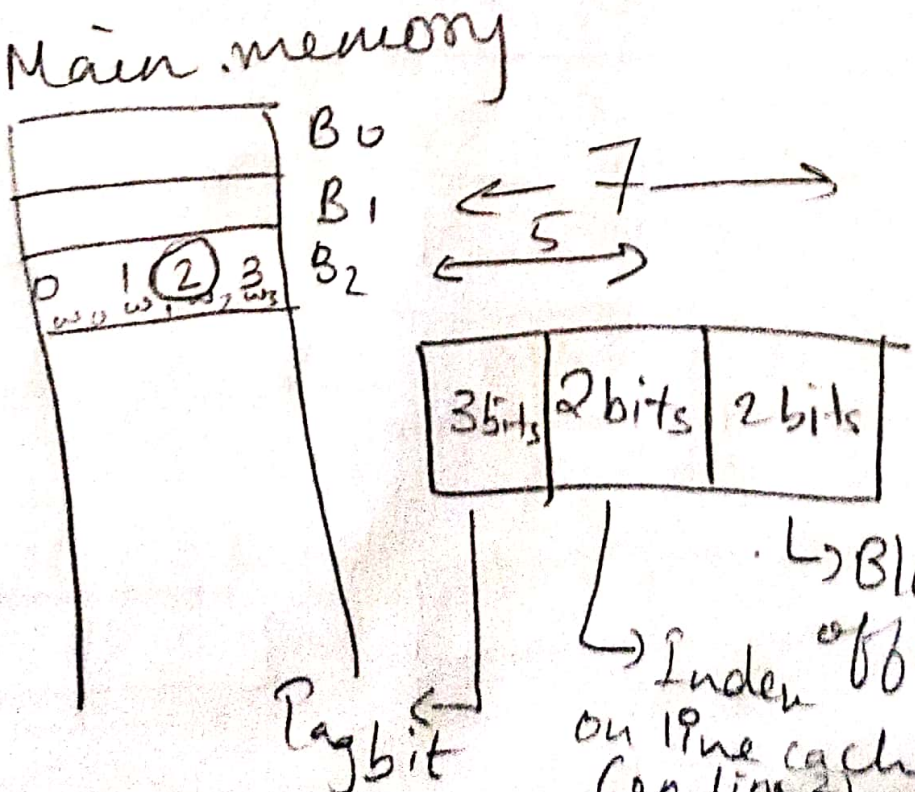




$2^7 = 128 = 0 \ldots 127$ for 128 words

→ Total physical address = 7 bits





When data travels, for matching tag bit is required.

001 01 00
001 01 01
001 01 10
001 01 11
} offset options

Tag, index, line no, block offset must match → data hit

physical address = 7 bits

$2^7 = 128$

4 lines → 2 bits (2^2)

8 lines → 3 bits (2^3)

Cache line = $\frac{32}{4} = 8$

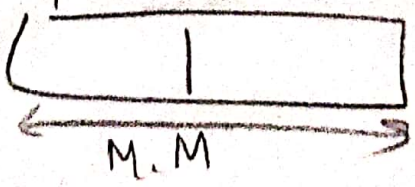
M.M = 64
C : S = 32
BS = 4

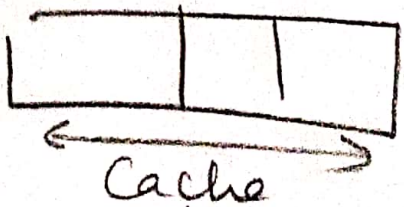
M.M lines = $\frac{64}{4} = 16$ blocks

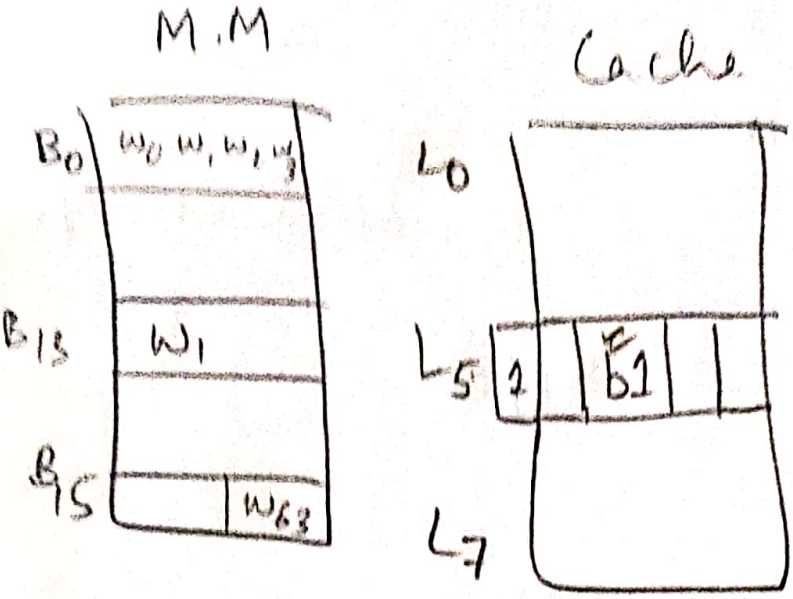
Physical address

$2^6 = 64$

total = 6 bits







~~4 × 4~~ = 16 blocks
$\frac{16}{4} = 4$

$2^6 = 64$ M.M

4 bits	2 bits

↓ blockno ↓ offset

Cache lines = 23
= 8

Cache
6 bits

≠ Direct mapping

1bit	3 bits	2 bits

↳ cache line or cache index

If given,

1 1 0 1	0 1

13 1
b.n b.0

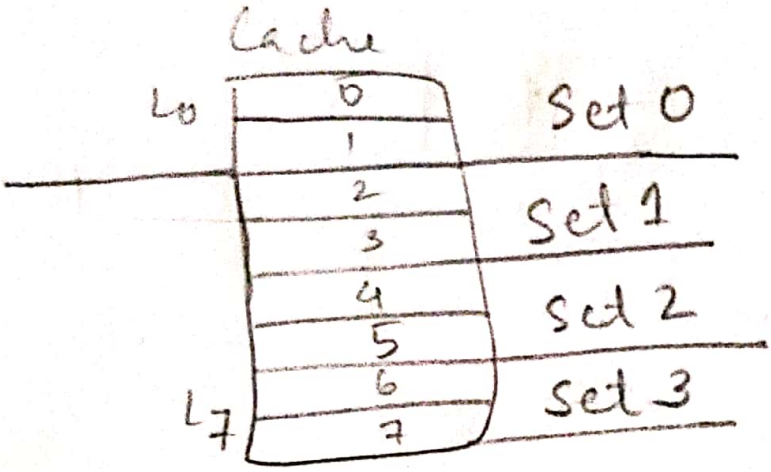
1	1 0 1	0 1

1
b.0

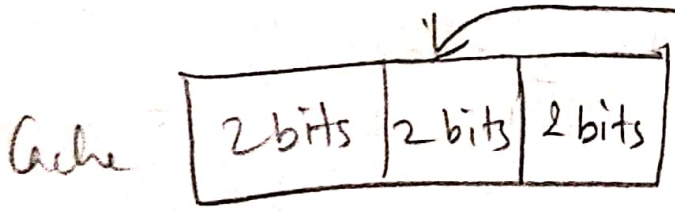
Set associative: 51:34

2-way SA

For last example



Cache lines / set = 8/2 = 4 sets
= 2^2 sets so,



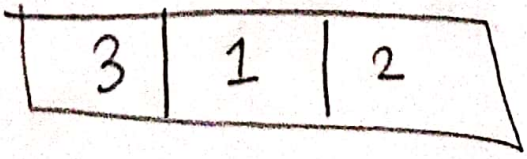
11 01 01

tag

no. of set (set 1) any from set 1

4 way

$= \frac{8}{4} = 2$ sets
$= 2^1$



Pipelining

stage / seg IF ID EX MEM WB

In pipline, all stage length are same.

So, you take min the stage that takes highest time for all the stages.

→ In Pip, for single ins it'll not give better performance.
The more the ins, Pip performance the better.

→ The lesser CPI, the better performance.

→ Max length is the min execution time.

IF - phare Cycle
ID - Decode Ins
Ex - Ins execution
Mem - Memory
WB - Write back

k (stag) n-ins

k + (n-1)

for MIPS, structural hazards of pip solved. IM & DM

⇒ Structural hazard: 2 ins → same hardware resource.

⇒ Control hazard: Before calculating result, jumps to another level, shows previous result (Conditional calcu problems). Level breaking

⇒ Data hazard: After decoding, registers identified where calcu being done. to being used for 2nd ins before updating for 1st (decode) Before WB, using that register for another ins. Wrong result

↳ Before calcu address, going to next ins.

For SAP 'St', For MIPs solved as Ins and data stored in diff memory.

For 'CH', stalling pipeline

IF - PC and IM part

ID - decoding, identification

Ex - Ins operation, which registers used, which type ins execution.

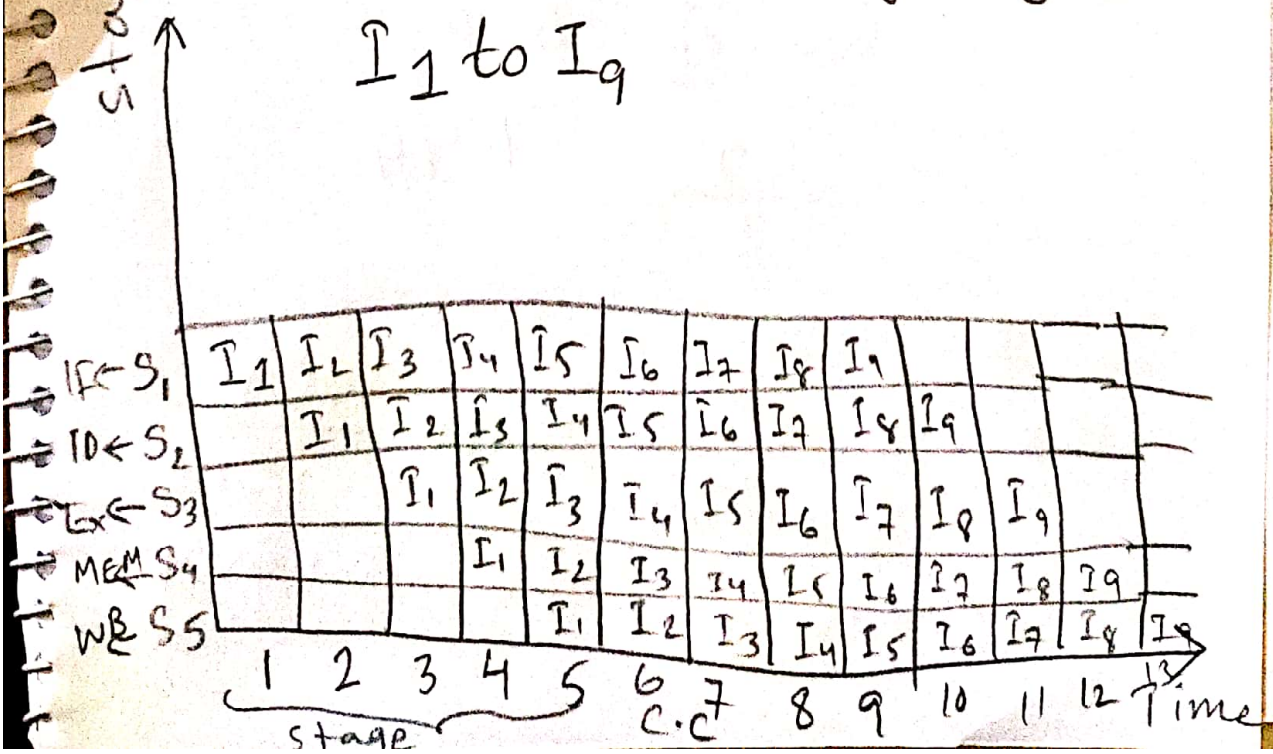
MEM - Memory read/write

WB - If not memory write, then register write.

---

9 Ins MIPs - 5 stage/segment

I_1 to I_9



n → Ins

Pip

1 × k + (n-1)

(1 c.c × 5) + (9-1)

5 + 8 = 13 c.c

NP

(k × n) = 5 × 9 = 45 c.c

$$\text{Speedup} = \frac{NP}{P} = \frac{45}{13} = 3.46$$

times faster will be P technique

$$CPI = \frac{\text{No. of C.C}}{\text{No. of ins}}$$ Pip

$$= \frac{13}{9} = 1.44$$

For I = 1000,

Pip 5 + (1000 - 1)

5 + 999

= 1004

$$CPI = \frac{1004}{1000}$$

= 1.004

Speed up $= \frac{5000}{1004} = 4.9$

The more instructions, the more the speedup.

Efficiency / Utilization

$$\frac{5 \times 9 \ (k \times n)}{13 \times 5 \ \text{Total block}}$$

$= 45 / 65$

$= 0.692$