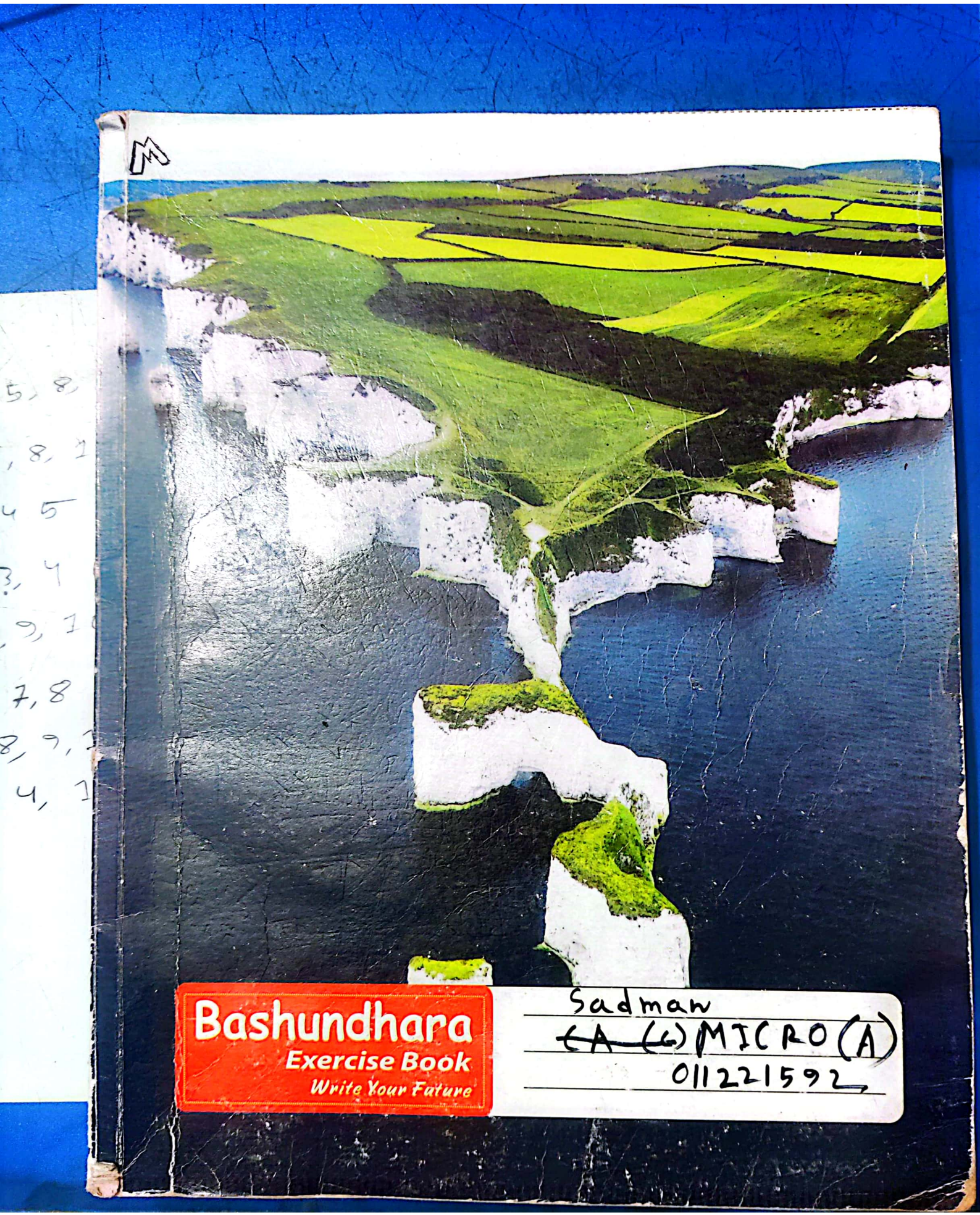


Sadman
~~(A)~~ (G) MICRO (A)
011221592

C.W
23.1.24

CA

Class → 1

Introduction

C.W
27.1.24

CA

* MIPS

→ sum = a + b [regular]

→ { mov AX, a
mov BX, b [Assembly language]
Add AX, BX
mov sum, AC

ROM → BIOS

→ SRAM, DRAM

→ Instruction set

CW
30.1.24

CA

* MIPS → 32 register ($0 — $31)
↓
Wikipbooks. org
(Number, name, comments)

* ($2 — $25.) → important for lectures.

*
Hello World → print ?

→
```
.data
mystr: .asciiz "Hello World\n"

.text
main:
    li $v0, 4 → value
    la $a0, mystr → address
    syscall → execute
```

li → load.
↓
system call

la → load address

li $V0, 10 → exit
syscall

CA

CW
3.2.24

Performance

opd code | operand

* Response time
* Throughput

→ Performance = $\frac{1}{\text{Execution time}}$

→ x is n time faster than J.

→ $\frac{\text{Execution time Y}}{\text{Execution time X}} = \frac{\text{Performance X}}{\text{performance Y}} = n$

* Fetch → Decryption → load → perform → save → output

CPU Time → Cpu clock cycles × clock cycle time

$$\text{CPU time} = \frac{\text{CPU clock cycles}}{\text{clock rate}}$$

$$\text{clock cycles} = \text{Instruction count} \times \underbrace{\text{cycles per instruction}}_{\hookrightarrow \text{CPI}}$$

$$\text{CPU time} = \frac{\text{Instruction Count} \times \text{CPI}}{\text{clock rate}} \times \text{clock cycle time}$$

$$= \frac{\text{Instruction count} \times \text{CPI}}{\text{clock rate}}$$

CW
6.2.24

CA

Performance depends on:

→ Algorithm

→ programming language

→ Compiler

→ Instruction set architecture

CW
10.2.23

* operation OP1, OP2, OP3

OP1 = OP2 operation OP3.

$2^9 = 512$

* h = 230;

A[10] = h + B[i]

li

~~addi~~, $s3, ~~#h+10~~, 230

~~lw, $t0, $s3, 4*i($s4)~~

~~add $t0, $t0, $s3~~

~~sw $t0, 40($s3)~~

h = $s2

A = $s0

B = $s1

i = $s3

```
mul i  $t1, $s3, 4
lw   $t0, $t1($s2)
add  $t0, $s2, $t0
sw   $t0, 40($s0)
```

11.2.24

online class - (1, 2, 3)

CA

C to Mips

operation name → destination variable → 2 source variable

* a = b + c + d

→ add a, b, c

add a, a, d

32 registers in mips:

$0, $1, ... $31

$s0, $s1 → save register

$t0, $t1 → temporary register.

$0 → zero register.

* f = (g + h) - (i + j)

f → $s0

g → $s1

h → $s2

i → $s3

j → $s4

add $t0, $s1, $s2

add, $t1, $s3, $s4

sub, $s0, $t0, $t1

P.T.O

* a = b + 2

addi $s0, $s1, 2

↳ add immediate

* a = b - 2

addi $s0, $s1, -2

* a = 7

addi $s0, $zero, 7

* a = 7 + 2

addi $s0, $zero, 7

addi $s0, $s0, 2

* a = b & c [b and c]

and $s0, $s1, $s2

P.T.O

* a = b + c [b or c]

or $s0, $s1, $s2

* a = b + 1

ori, $s0, $1, 1

* a = (b + c)' [nor]

nor $s0, $s1, $s2

* a = b >> 2 [shift right 2 times]

srl $s0, $s1, 2

performance

Relative performance

$$\frac{P_x}{P_y} = \frac{E_y}{E_x} \quad (\text{Response time / Execution time})$$

$$\frac{P_x}{P_y} = \frac{CR_x}{CR_y} \quad (\text{Throughput / clock Rate})$$

* CPU time = CPU clock cycles × Clock cycle time

* CPU time = $\frac{\text{CPU clock cycles}}{\text{Clock rate}}$

A: 2 GHz clock, 10s CPU time

Designing B

→ 6s cpu time

→ can do faster clock, but causes 1.2 × clock cycles A

How fast must computer B clock be?

→

cpu clock cycles → $10 \times (2 \times 10^9)$

2×10^{10}

new clock cycle → 2.4×10^{10} [1.2 × cc A]

so new rate,

clock rate = $\frac{2.4 \times 10^{10}}{6} = 4 \times 10^9$ Hz

~~* CPU time = CPU clock cycles ×~~

* Clock cycles = Instruction count × cycles per Instruction

* CPU time = Instruction Count × CPI × clock cycle time

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

A: Cycle time = 250 ps, CPI = 2.0

B: " = 500 ps, CPI = 1.2

Same IS A. * Which is faster?
* How much

A

CPU time

$= IC \times 2.0 \times 250 \text{ ps}$

$= IC \times 500$ ✱

↓ (faster)

B

CPU time

$= IC \times 1.2 \times 500 \text{ ps}$

$= IC \times 600$

$\frac{B}{A} = \frac{IC \times 600}{IC \times 500}$

$\Rightarrow B = 1.2 \times A$

#

Instruction	add	sub	beq
CPI for Instruction	1	2	3
IC, in sequence 1	2	1	2
IC " " 2	4	1	1

Avg CPI for seq 1 & 2

1:

Cycle: $(2\times1)+(1\times2)+(2\times3)$

$\rightarrow 10$

Avg CPI $= \frac{10}{2+1+2}$

$= \frac{10}{5}$

$= 2$

2:

Cycle:

$(4\times1)+(1\times2)+(1\times3)$

$\rightarrow 9$

Avg CPI $= \frac{9}{6}$

$= 1.5$

MIPS $= \frac{\text{Instruction count}}{\text{CPU time} \times 10^6}$

$= \frac{IC}{\frac{IC \times CPI}{\text{clock rate}} \times 10^6}$

$= \frac{\text{clock rate}}{CPI \times 10^6}$

W
13.2.24

Lecture - (4, 5), 6.

* CMOS

Power = Capacity load × Voltage^2 × Frequency

enhance power decrease the voltage & increase the frequency in such a way that the power will not change.

[Amdahl's law]

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

overall time minus affected time

if improvement factor negative, that will not possible.

But positive value improvement possible.

P.T.O

MIPS

* A[12] = h + A[8]

→ ~~lw $t0, 32($s3)~~
~~add $s2, $t0~~

lw $t0, 32($s3)
add $t0, $s2, $t0
sw $t0, 48($s3)

h = $s2
A = $s3

* h = 230;
→ A[10] = h + B[i];

addi, $t0, $s2, 230
muli, $t0, $s0, 4
lw, $t1, $t1($s4)
addi, $t0, $t0, $t1
sw, $t0, 40($s3)

h = $s2
A = $s3
B = $s4
i = $s0

or

sll $t1, $s0, 2

shift left one time → multiply by 2

two n → n n 2×2

n n → n^2

#

```
if (i == j) f = g + h
else f = g - h

beq $s3, $s4, IF
sub $s0, $s1, $s2
j EXIT
IF:
add $s0, $s1, $s2

EXIT:
```

f = $s0
g = $s1
h = $s2
i = $s3
j = $s4

P.T.O

#

if (i < j) f = g + h;

else f = g - h

f = s_0	
g = s_1	
h = s_2	
i = s_3	
j = s_4	#

```
slt  $t0, $s3, $s4
beq  $t0, $zero, ELSE
sub  $s0, $s1, $s2
J EXIT
IF: add $s0, $s1, $s2
EXIT:
```

If $t0 = 0
false
else
true

```
add  $s0, $s1, $s2
J EXIT
ELSE: sub $s0, $s1, $s2
EXIT:
```

$s4$ $s3$ $s0$ $s1$ $s2$

* if (i <= j) f = g + h

else f = g - h

remove the equal sign

converted code

if (i < j) f = g - h

else f = g + h

same

→

slt $t0, $s3, $s4

beq $t0, $zero, ELSE

~~beq $t0, $s1, $s2~~

~~EXIT:~~

~~sub $s0, $s1, $s2~~

sub $s0, $s1, $s2

J EXIT

ELSE: add $s0, $s1, $s2

EXIT:

if (i < j && i != k) f = g + h;
else f = g - h

$s3 $s4 $s5 $s0 $s1 2

→

slt $t0, $s3, $s4
bne $s3 $s5, JF
beq $t0, $zero, ELSE
ELSE: sub $s0, $1, $2
J EXIT.
IF: add $s0, $s1, $s2
EXIT:

→ Another solution:

slt $t0, $s3, $s4
bne $t0, $zero, L1
sub $s0, $s2, $s2
J EXIT.
L1: bne $s3, $s5, JF
IF: add $s0, $s1 $s2
EXIT:

loop

while (save[$i] == k) i++ ;

⇒

```
    lw  $t0, 20($S2)
LOOP: beq  $t0, $S1, INC
    J EXIT.
    INC: addi $S0, $S0, 1
    J LOOP
    EXIT:
```

for (i=1 ; i<w) i++

```
    addi  $S0, $zero, 1
LOOP: slt  $t0, $S0, $S1, INC
    J EXIT
    INC: addi $S0, $S0, 1
    J LOOP
    EXIT:
```

function

Argument reg → a_0 — a_3

Return Value reg → v_0, v_1

Return Address → $ra

stack pointer → $sp

Instructions:

jal func_address

jr $ra

function. address

jal (4000)

$ra = PC + 4 = 2008 + 4 = 2012

↓

[next line]

* int func (int g, h, i, j)

↓

[argument register]

$a0 → $a3

* int func (int g, h, i, j) {

int f, a, b = 20, c = 30

a = b + c

f = (g + h) - (i + j)

f = a + f

return f;

}

g h i j
a0, a1, a2, a3

f = $s0

a = $s1

b = $s2

c = $s3

for return f

add $v0, $s0, $zero

```
int main(){
int a=20, b=5, c=2, d=3;
func(a,b,c,d);
a = a+3;
d = b+c+a;
```

before function call [pushing]

```
addi $sp, $sp, -16      (empty place)
sw   $s0, 12($sp)
sw   $s1, 8($sp)
sw   $s2, 4($sp)
sw   $s3, 0($sp)
```

P.T.O

after function call [randint]

```
lw , $s3, 0($sp)
lw, $s2, 4($sp)
lw, $s1, 8($sp)
lw, $s0, 12($sp)
addi $sp, $sp, 16.
jr $ra.
```

We will push the amount of variables declare in the function

```
int func() {
    int a,b,c,d
}
```

Machine code

R-Type (add, sub, and, or, nor, sll, srl, slt, jr)

I-Type (lw, sw, andi, ori, addi, beq, bne)

J-Type (J, Jal)

R-Type

op	rs	rt	rd	shamt	func
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

op = opcode

rs = 1st source register

rt = 2nd " "

rd = destination register

shamt = shift amount

func = function

rd rs rt

* add $\$s_0$, $\$s_1$, $\$t_2$

op	rs	rt	rd	shamt	func
0	17	10	16	X	32
000000	10001	01010	10000	00000	100000

rd rs shamt

* sll $\$t_0$, $\$s_3$, 20

op	rs	rt	rd	shamt	func
0	19	X	8	20	0
000600	10011	00000	01000	10100	000000

* jr $ra → ($rs$)

op	rs	rt	rd	shamt	func
0	31	X	X	X	8
000600	11111	00000	00000	00000	001000

I-Type

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits.

* sw, $s0, 20($s1)

$s1 → rs, $s0 → rt

op	rs	rt	C/A
43	17	16	200

* andi $s0, $s1, 223

$s1 → rs, $s0 → rt

op	rs	rt	C/A
12	17	16	223
001100	10001	10000	0000000011011111

Example

bne $s1, $s2, 100

(for bne & beq $\frac{N}{4}$)

op	rs	rt	C/A
5	18	17	$\frac{100}{4} = 25$
000101	10010	10001	0000000000011001

J type

* J 2000

(for J type J, Jal = $\frac{N}{4}$)

op	address
2	$\frac{2000}{4} = 500$
6 bits	26 bits

* Jal 10,000

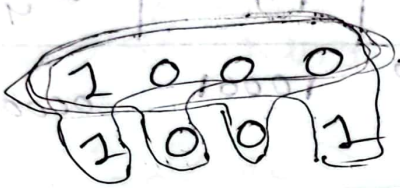
op	add
3	$\frac{10000}{4} = 2500$
000011	00000000000010011100010

Lecture-3

arithmetic

Multiplication

Multiplication by addition




```
  1 0 0 0
  0 0 0 0 X
0 0 0 0 0 X X
1 0 0 0 X X X
(+)
1 0 0 1 0 0 0
```

MR এর কোন বিট ধরে ধরে প্রত্যেকটি 0 & 1 এর সাথে MN এর প্রত্যেকটি 0 & 1 গুন করতে হবে, প্রথমে একবার গুন হয়ে গেলে পরের বার ~~একবার~~ দিয়ে এর ঘর করে বামে move করতে হবে.

replace all the cross with 0.

Shift left MN 1 time

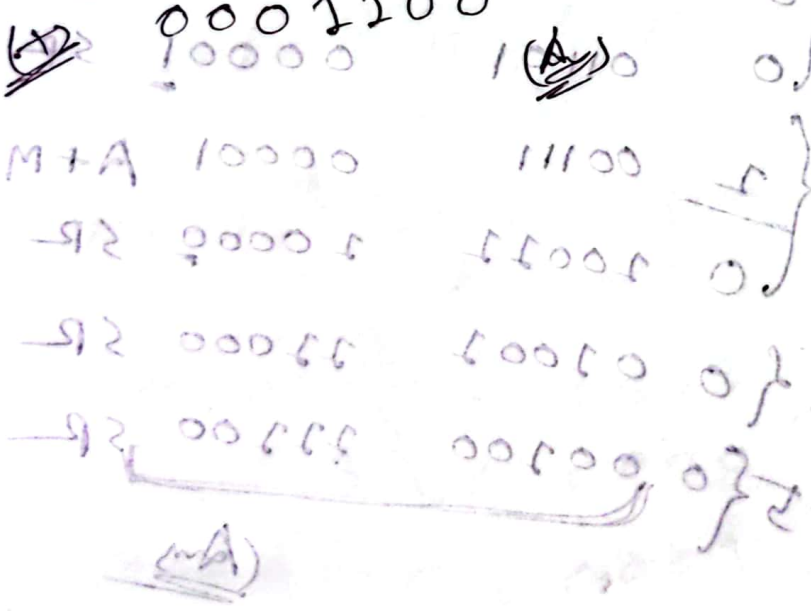
" Right MR 1 "

(Multiplication)

0100 (MN)

0011 (MR)

01100	step-1
01100X	2
00000XX	3
00000XXX	4
00001100	



Optimised multiplication

5 bit architecture

M	Q
26	6
11010	00110

steps

if ($Q_0 = 1$)
A + M, SR
$Q_0 = 0$, SR

C	A	Q	
	0	00000	00110 start
2	0	00000	00011 SR
2	0	11010	00011 A + M
	0	01101	00001 SR
3	1	00111	00001 A + M
	0	10011	10000 SR
4	0	01001	11000 SR
5	0	00100	11100 SR

(Ans

Division

Steps

SL (C, A, Q)

A = A − M

C = 1, Q_0 = 0, A = A + M

C = 0, Q_0 = 1

Q M

15 ÷ 7

1111 0111

		C	A	Q	
		0	0000	1111	
1	M+A	0	0001	111□	SL
		1	1010	1110	Q_0 = 0
		0	0001	1110	A = A+M
2		0	0011	110●□	SL
		1	1100	1100	Q_0 = 0
		0	0011	1100	A = A+M
3		0	0111	100●□	SL
		0	0000	1001	Q_0 = 1
4		0	0001	001●□	SL
		1	1010	001■	0, Q_0 = 0
		0	0001	0010	

Reminder Ans

Multiplication

13 x 3

M Q

1101 0011

$Q_0 = 1$

A+M, SR

$Q_0 = 0$, SR

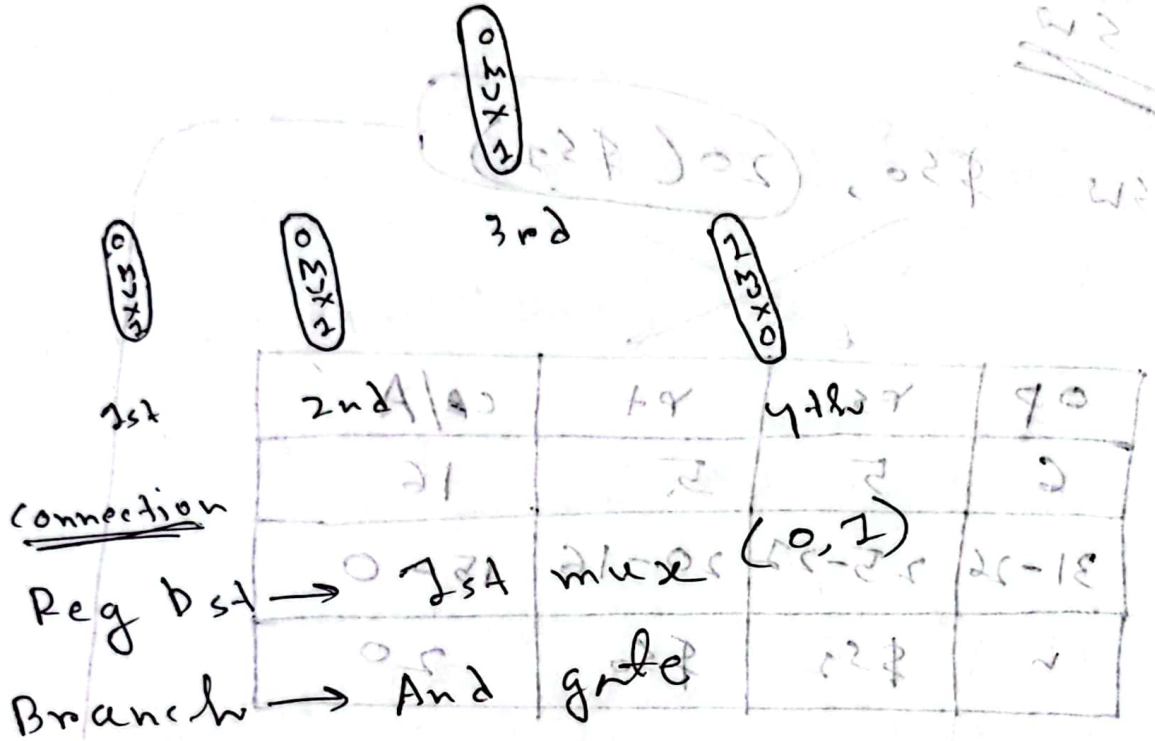
C	A	Q		
0	0000	0011		
0	1101	0011	→ A+M	1
0	0110	1001	→ SR	
1	0011	1001	→ A+M	2
0	1001	1200	→ SR	
0	0100	1120	→ SR	3
0	0010	0112	→ SR	4

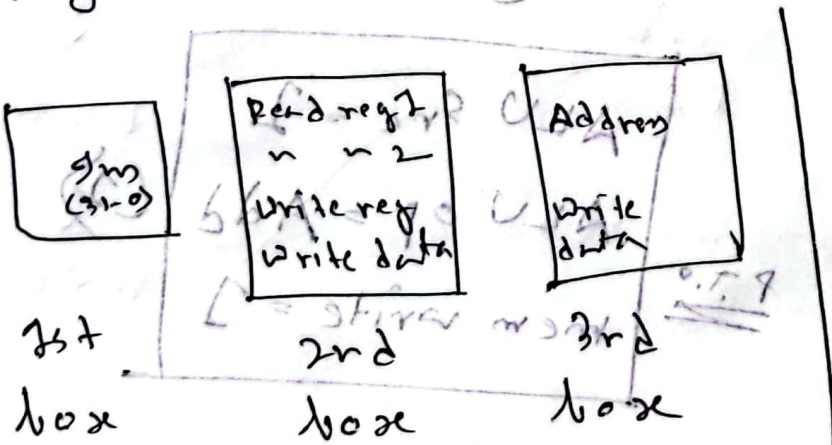
Single Cycle

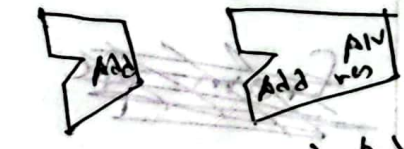
(X / 0) same
No wire will connect

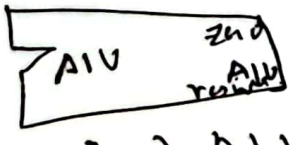
	Reg Dst	ALU SRC	Mem to reg	Reg write	Mem Read	Mem Write	Branch	ALU OP1	ALU OP0
Jmp									
R-Type	1	X	X	1	X	X	X	1 (name)	0 (name)
lw	X	1	1	1	1	X	X	0	0
sw	X	1	X	X	X	1	X	0	0
beq	X	X	X	X	X	X	1	0	1

ALUOp


2XC30
3rd
2XC30
1XC30
0XC32
1st
2nd
4th
connection
Reg Dst → 1st mux (0,1)
Branch → And gate
Mem Read → 3rd box
Mem to Reg → 4th mux (1,0)
AIU OP → AIU control → 3rd Adder
Mem Write → 3rd box
AIU src → 2nd mux (0,1)
Reg Write → 2nd box


Read reg 1
Write reg
Write data
Address
Write data
1st box
2nd box
3rd box


Add
Add
AIU res
1st Adder
2nd Adder


AIU
2nd AIU result

3rd Adder.

SW

SW $S0, 20($S1)

op	rs	rt	ca/A
6	5	5	16
31-26	25-21	20-16	15-0
w	$S1	$S0	20

1. Read (20-16) [Register]
2. Execute addition operation (25-21) && (25-0) [ALU]
3. store value of [20-16] in executed address [Data memory]

~~from PC~~

P.T.O

ALU Src = 1
ALU op = Add
mem write = 1
(1)

from pic

Ins (31-26) → control

Ins (25-21) → Read reg 1 → Read data 1

Ins (20-16) → Read reg 2

Ins (15-0) → mux (1) → ALU

ALU result → Address

read data 2 → Write data

PC → Add → mux (0)

4 →

PC

Same for lw / Add / beq

* LW

lw $s0, 20($s1)

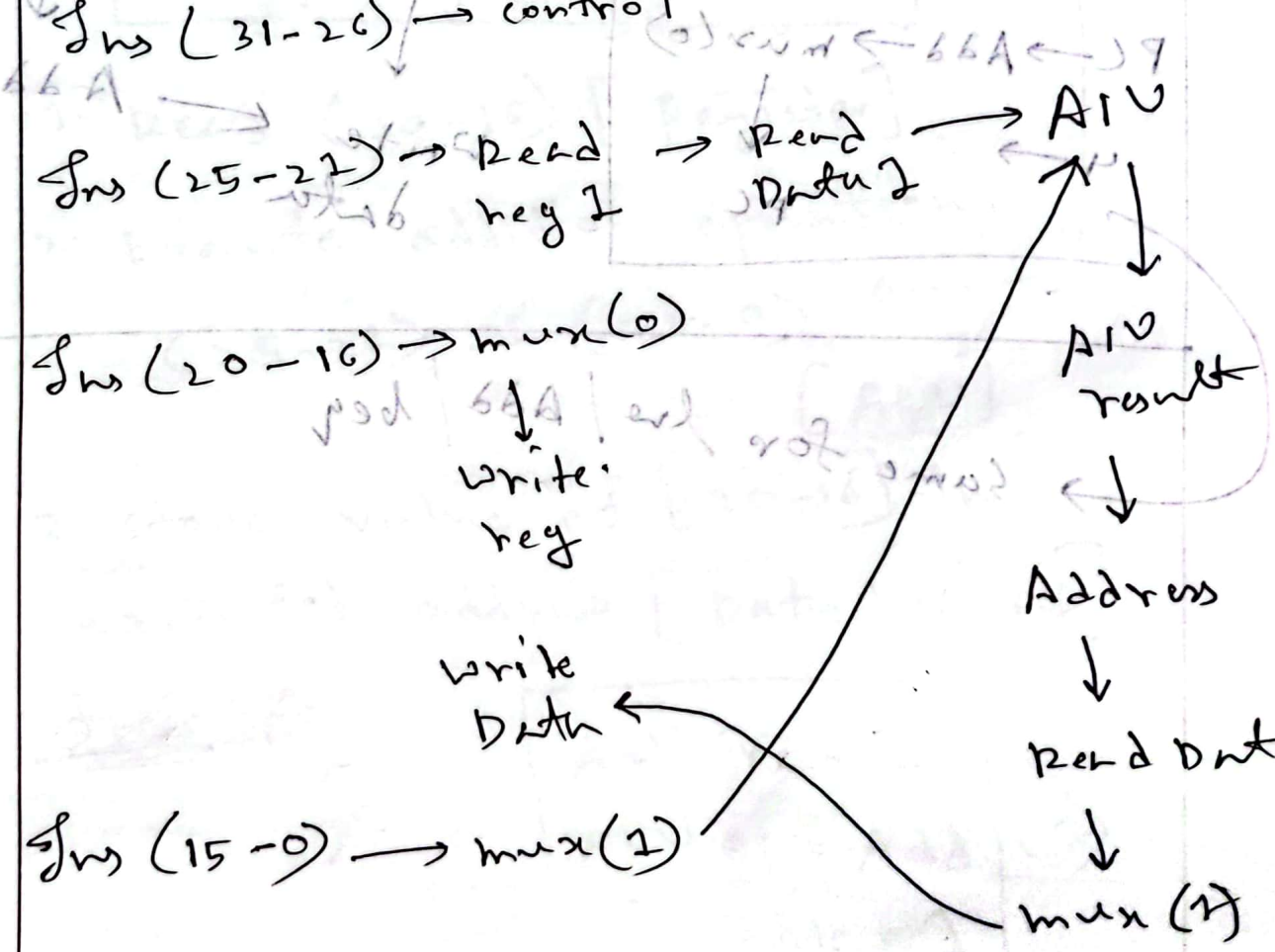
① Registers

② ALU

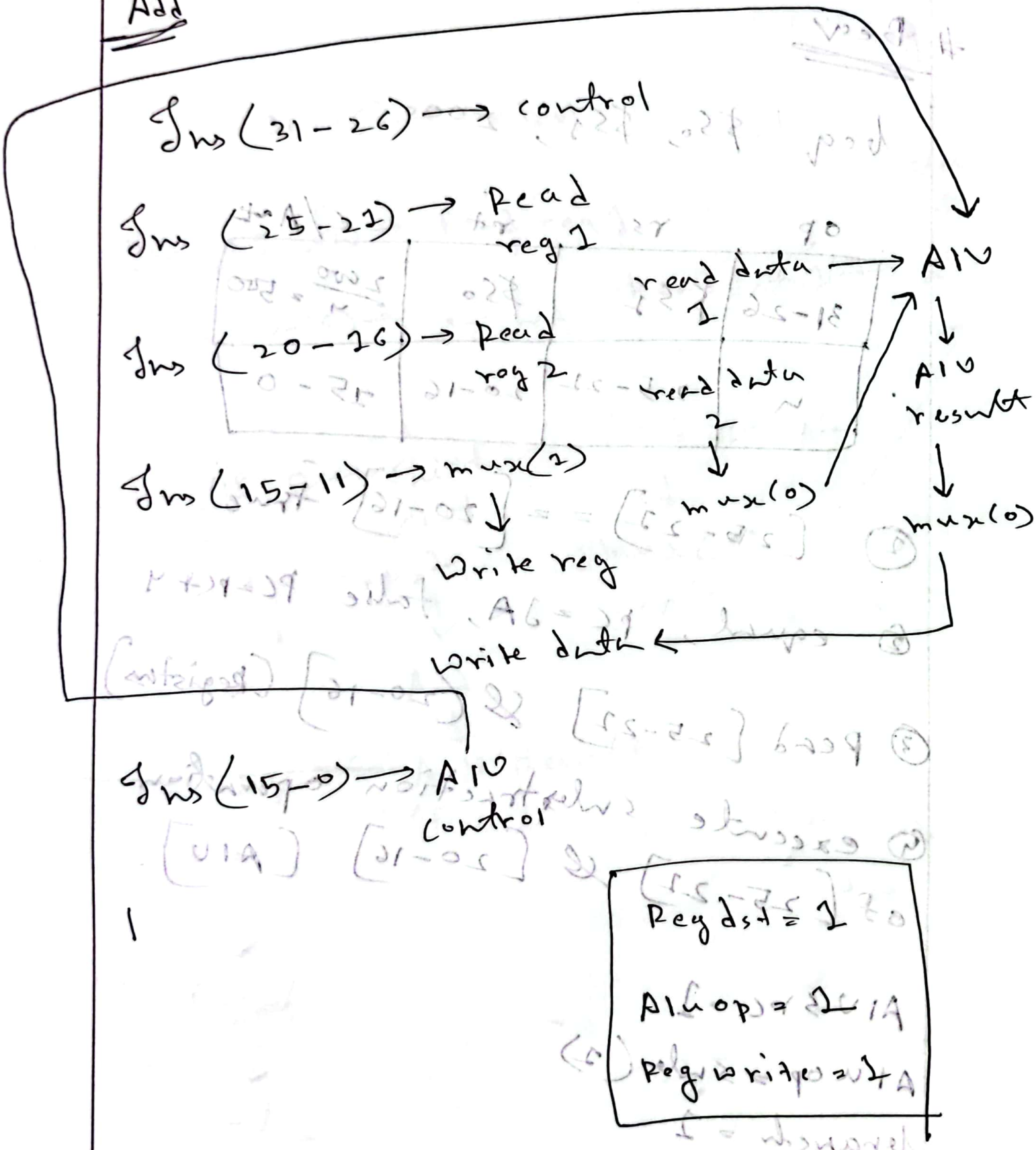
③ Data memory & Registers

ALU Src = 1
mem Read = 1
mem to reg = 1
reg write = 1

Pic



Add


Ins (31-26) → control
Ins (25-21) → Read reg. 1
Ins (20-16) → Read reg 2
Ins (15-11) → mux(1)
Write reg
Write data
Ins (15-0) → ALU control
read data 1
read data 2
ALU
ALU result
mux(0)
mux(0)
RegDst = 1
ALUop = 2'b10
RegWrite = 1'b1

1

Beq

beq $S0, $S1, 2000

op	rs	rt	IC/Adr
31-26	$S1	$S0	$\frac{2000}{4} = 500$
4	25-21	20-16	15-0

① [25-21] == [20-16] true

② equal, → PC = JA, false PC = PC + 4

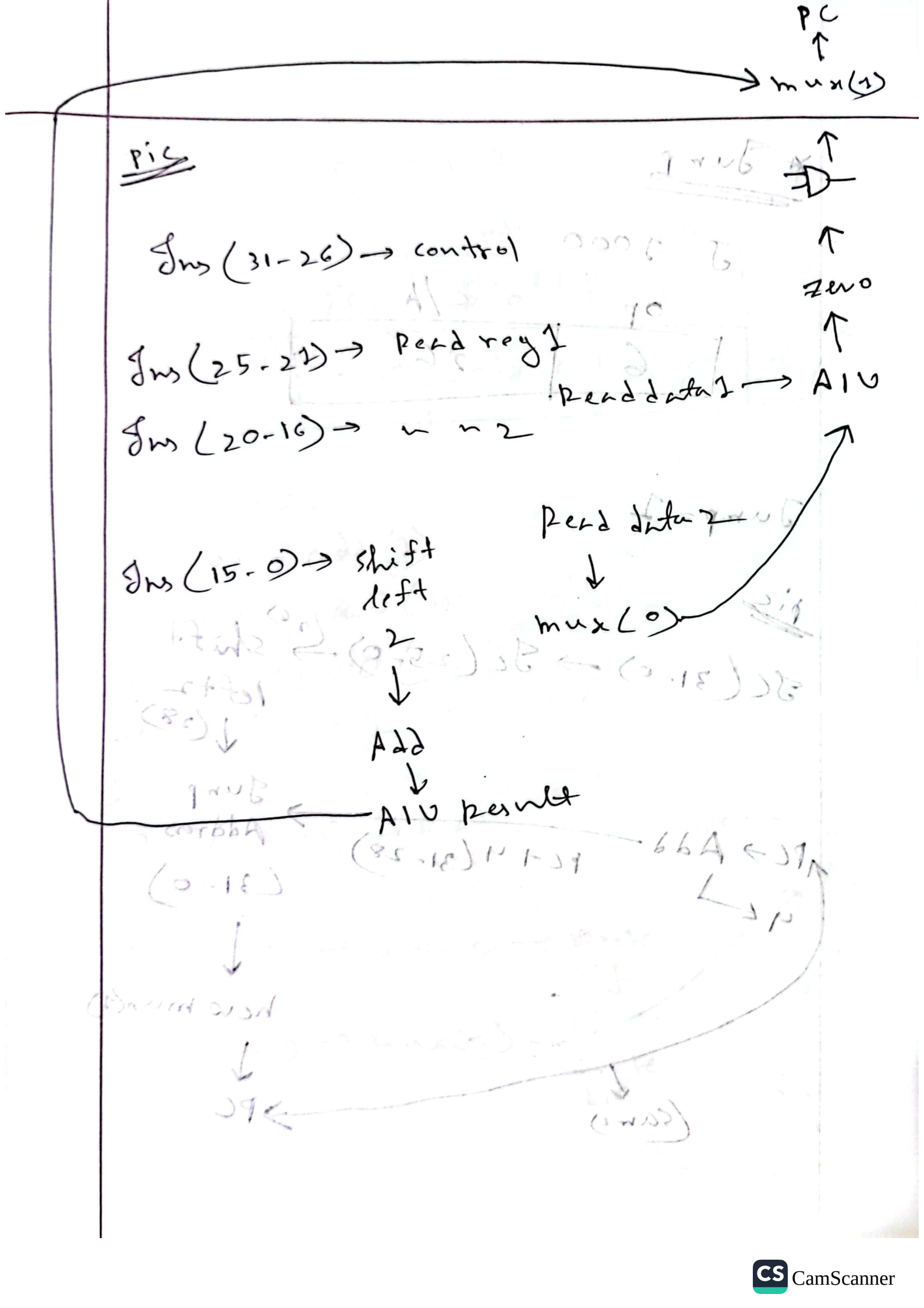
③ Read [25-21] & [20-16] (Registers)

④ execute subtraction operation of [25-21] & [20-16] (ALU)

ALUSrc = 1/0

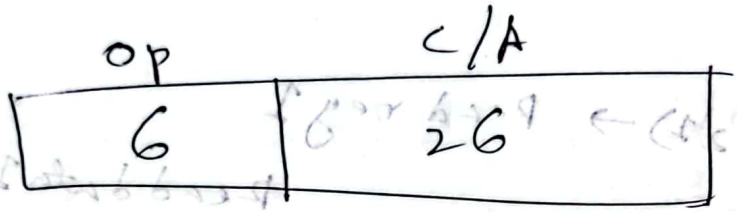
ALUop = sub (2)

branch = 1


PC
mux(1)
Pic
Ins(31-26) → control
Ins(25-21) → Read reg1
Ins(20-16) → " " 2
Ins(15-0) → shift left 2
Add
ALU Result
Zero
Read data1 → ALU
Read data2
mux(0)

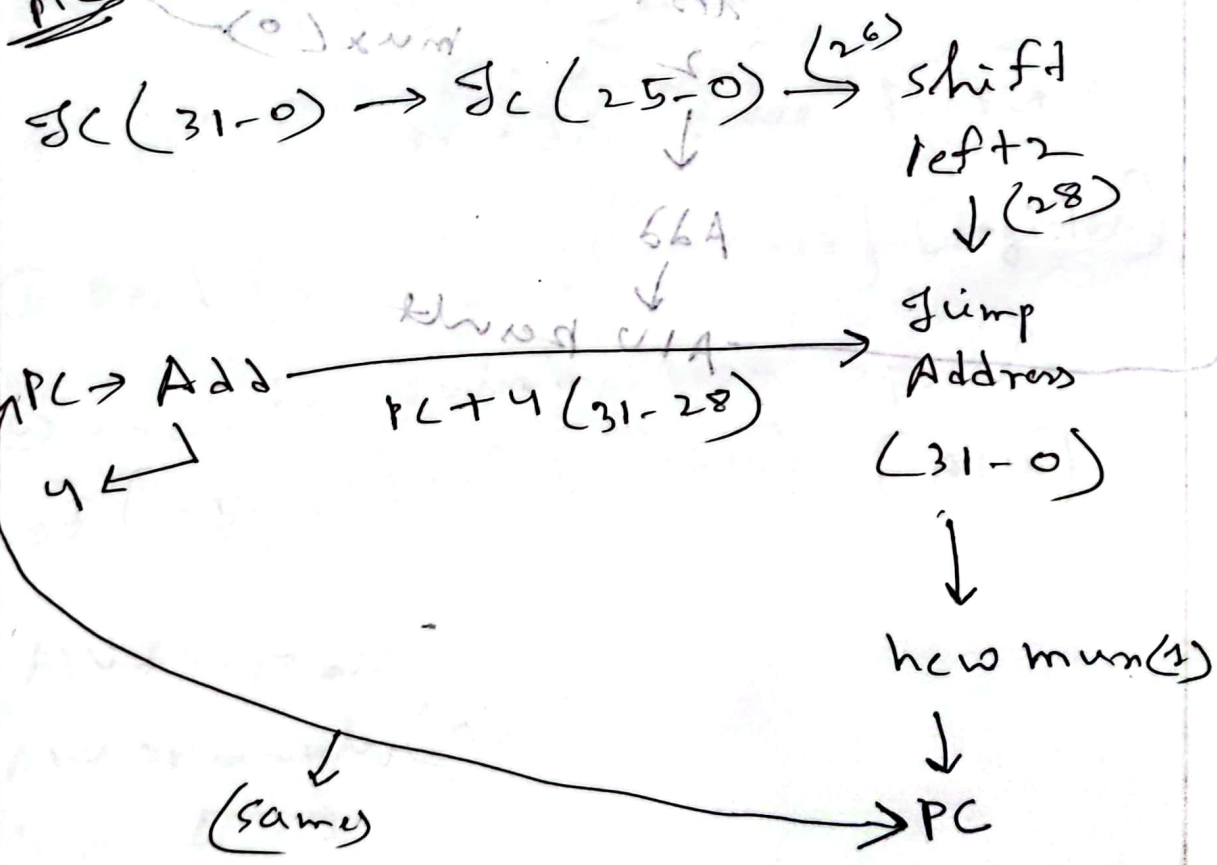
* Jump

J 1000



Jump = 1

pic



addi $s0 $s1, 1

→ dst

op	rs	rt	c/A
31-26	$s1	$s0	1
h	25-21	20-16	15-0

ALU src = 1

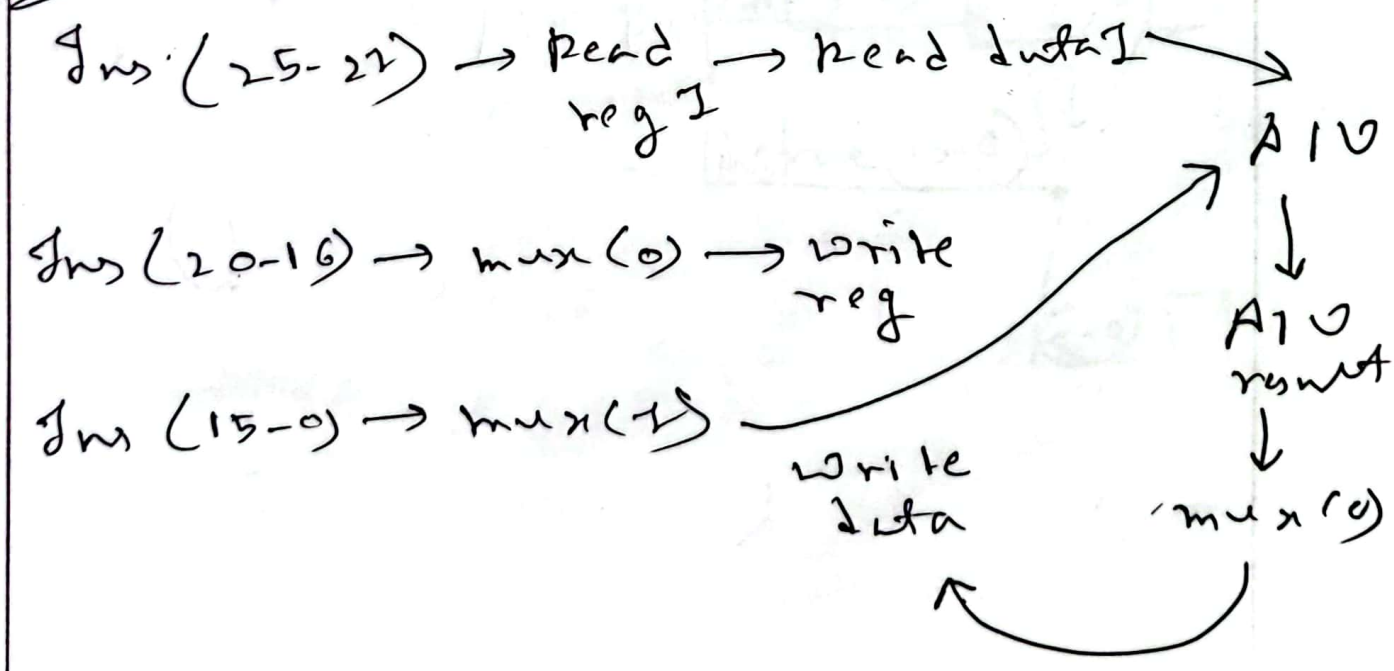
ALU opp = add (2)

Reg dst = 0 (mux result)

mem to reg = 0 (n)

reg write = 2

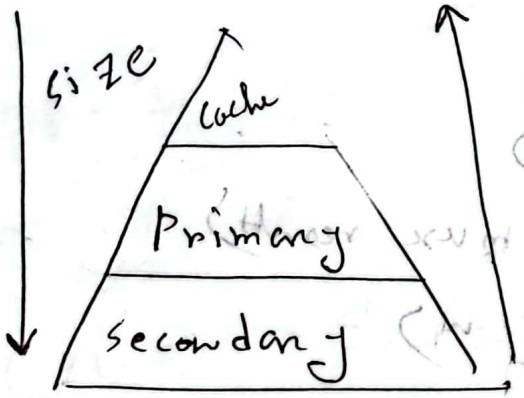
pic

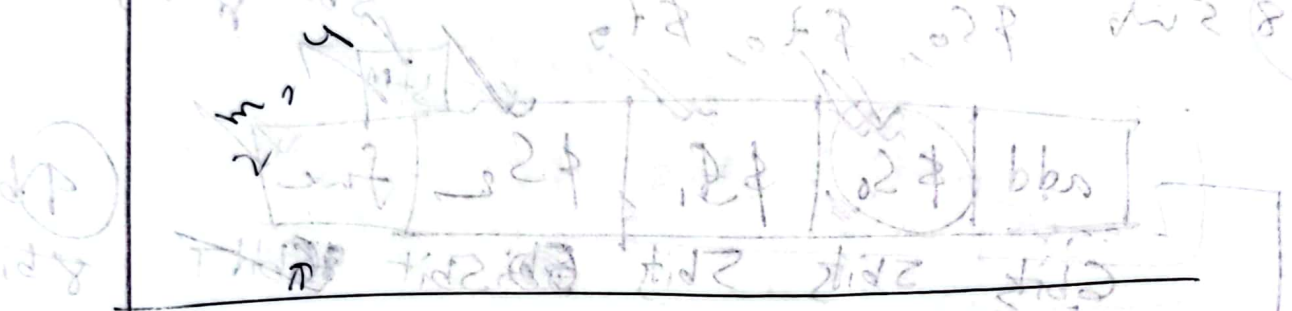


Primary → Ram, Rom

Secondary → HDD, SSD

Tertiary → CD, DVD, Pendrive





GW
4.5.24

(a) Per block

= 4 words = 16 bytes

$$= \frac{512 \times 1024}{16} \rightarrow = 32768 \text{ blocks}$$

$= 2^{15} \Rightarrow w = 15$

(b)

$2^w = 32768$

$\Rightarrow w = \log_2(32768)$

$\Rightarrow w =$

(4)

2^m words = 4

$\Rightarrow m = \log_2(4)$

4.5.24

CA

Cache

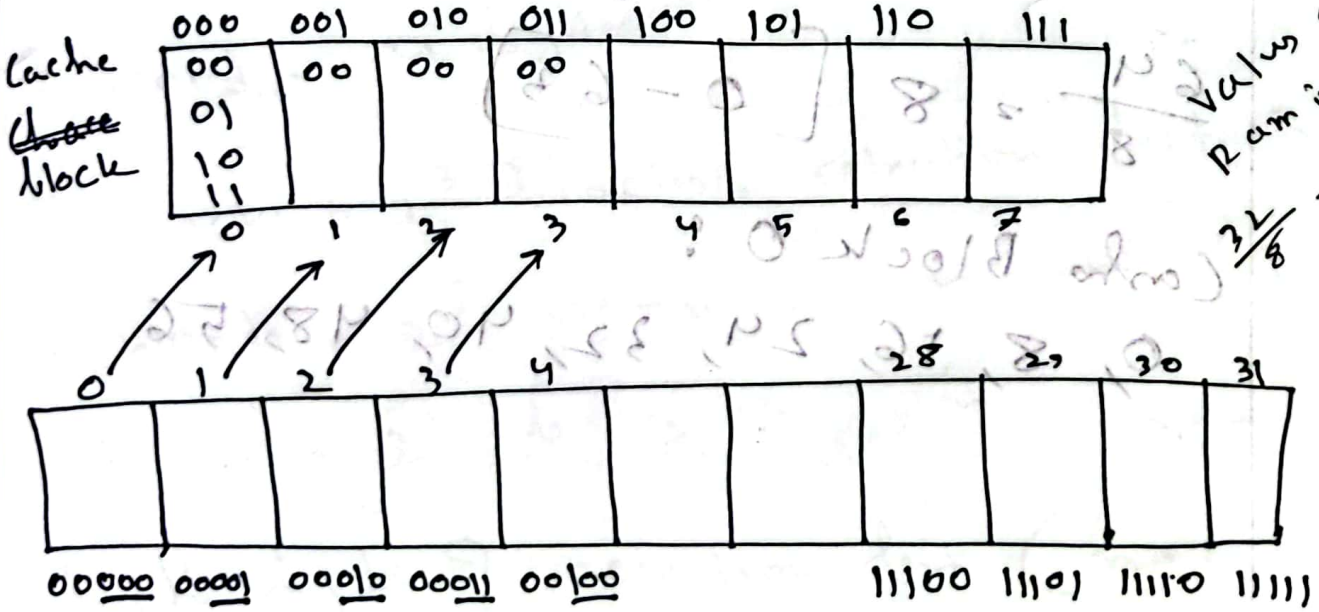
* upper level means cache memory
* lower " " Ram.

tag bits | index

00000 (0-7)
01000 (8-15)
10000 (16-25)
11000 (26-31)

Cache 8 block

Ram 32 block



Cache Block 1: 1, 9, 17, 25

Cache Block 2: 2, 10, 18, 26

(always plus 8)

[5 bits of last 3 bits match with the block number]

A)

* Cache block → 8

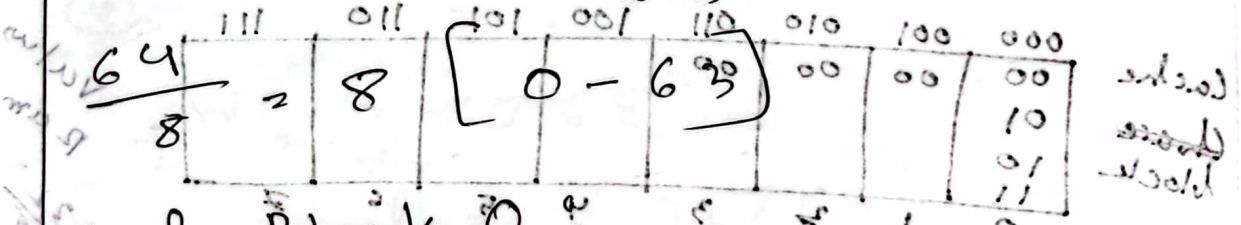
Main block → 64

$[2^6 = 64]$

(0-7) 00000
(8-15) 00010
(16-23) 00001
(24-31) 00011

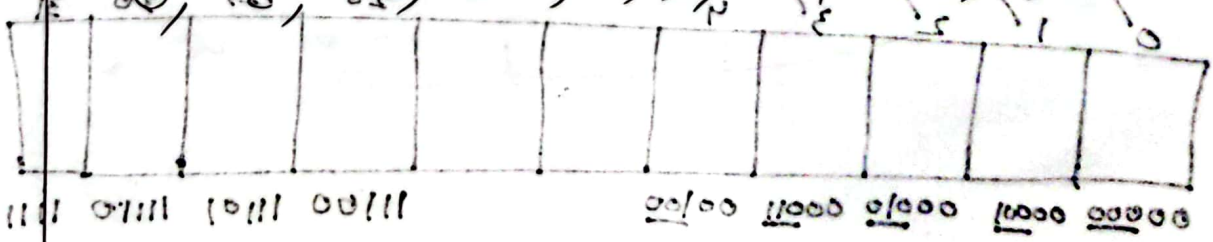
000 000
tag bits | index

$\frac{64}{8} = 8$ [0 - 63]



Cache Block 0:

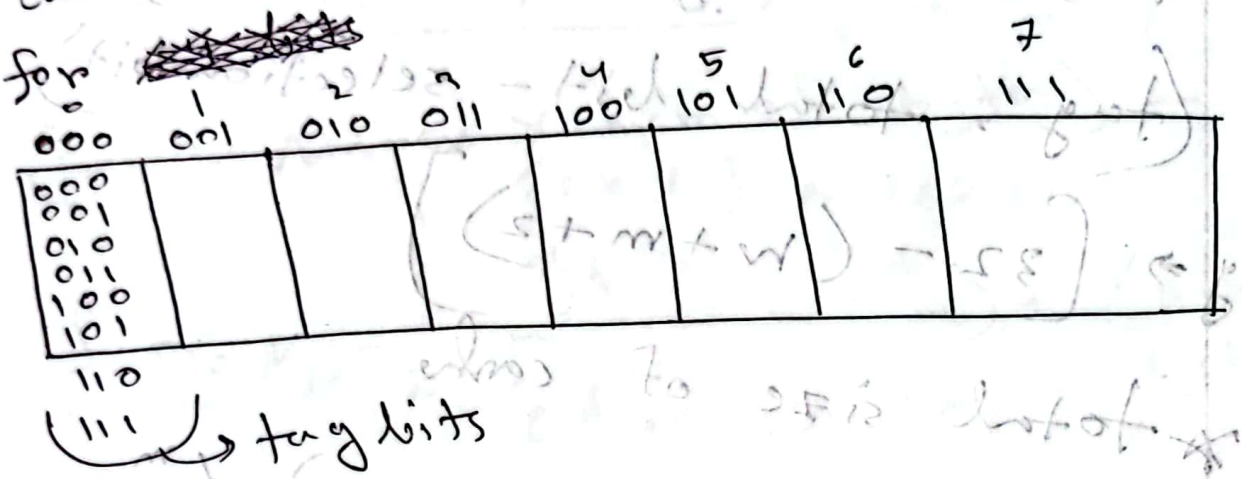
0, 8, 16, 24, 32, 40, 48, 56



Ram blocks → word address

Cache Blocks

for



Cache Blocks 4:

4, 12, 20, 28, 36, 44, 52, 60

From slide.

(1 byte = 8 bits)

cache size = 2^n block

block size = 2^{m+2} words

1 word = 4 bytes

8 blocks = 2^3 blocks = selection bit 3

2 words = 2^1 words = " " 1

4 bytes = 2^2 bytes = " " 2

which byte ? selection bit $(n+m+2)$

Index = selection bit

* 32 bit byte addressable,

(tag = total bit − selection bit)

tag ⇒ [32 − (n + m + 2)]

* total size of cache

⇒ $2^n \times [1 + \{32 - (n+m+2)\} + (2^m \times 32)]$

- 2^n ↓ number of blocks
- 1 ↓ valid bit
- $\{32 - (n+m+2)\}$ ↓ tag
- $(2^m \times 32)$ ↓ number of words ↓ total size in bits

* slide (29), 30

Selection bit = index + offset

(n) (m+2)

1 word = 4 bytes

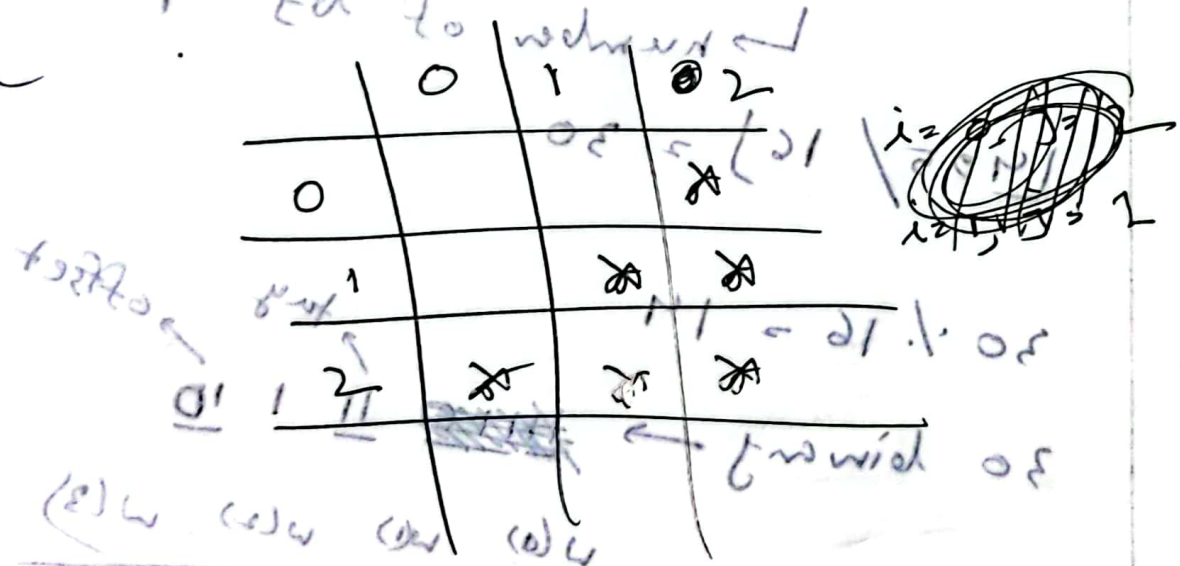
Block copied from:

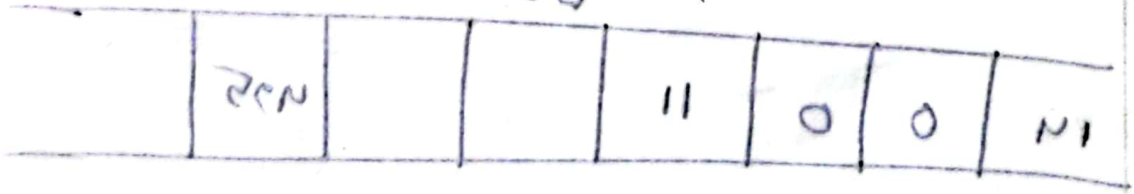
(Block Address × num of bytes) —

{(Block ~ ~ ~ ~ ~) + (num of bytes - 1)}

Block Address = address / bytes

Block number = Block address / total blocks





16 bytes / block

$16 \times 16 + (16 \times 2) + (16 \times 2 \text{ bits})$

bytes bits

$\Rightarrow 256 + 4 \text{ bytes} + 4 \text{ bytes}$

$= 264 \text{ bytes}.$

$\left\lfloor \frac{1024 \text{ bytes}}{16} \right\rfloor = 64$

↳ number of bytes per block

$\lfloor 495 / 16 \rfloor = 30$

$30 \% 16 = 14$

30 binary → ~~XXXX~~ 11 1 10

tag offset

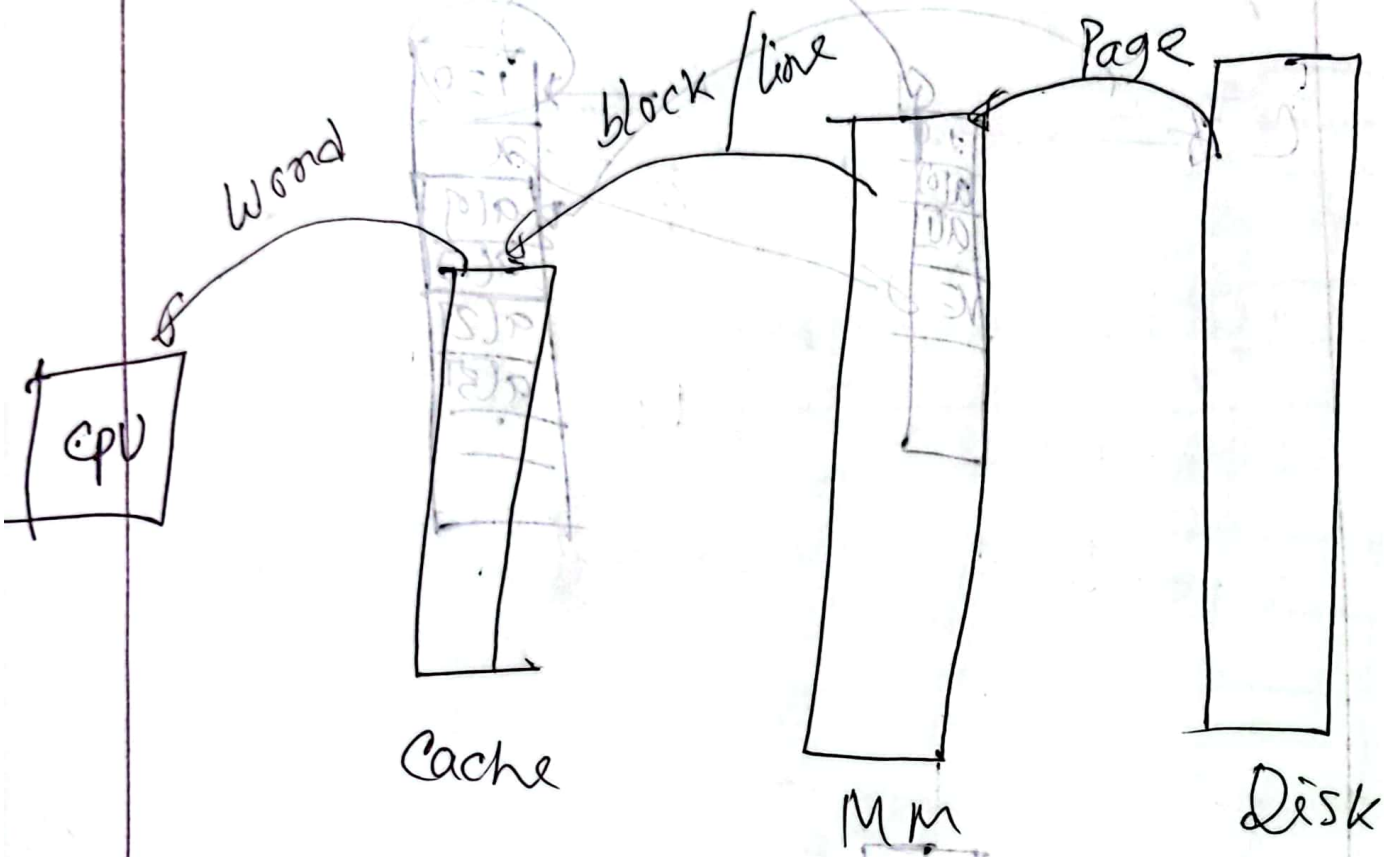
	w(0)	w(1)	w(2)	w(3)
14	0	0	11	495

8 word per block

56
64

4 word per block

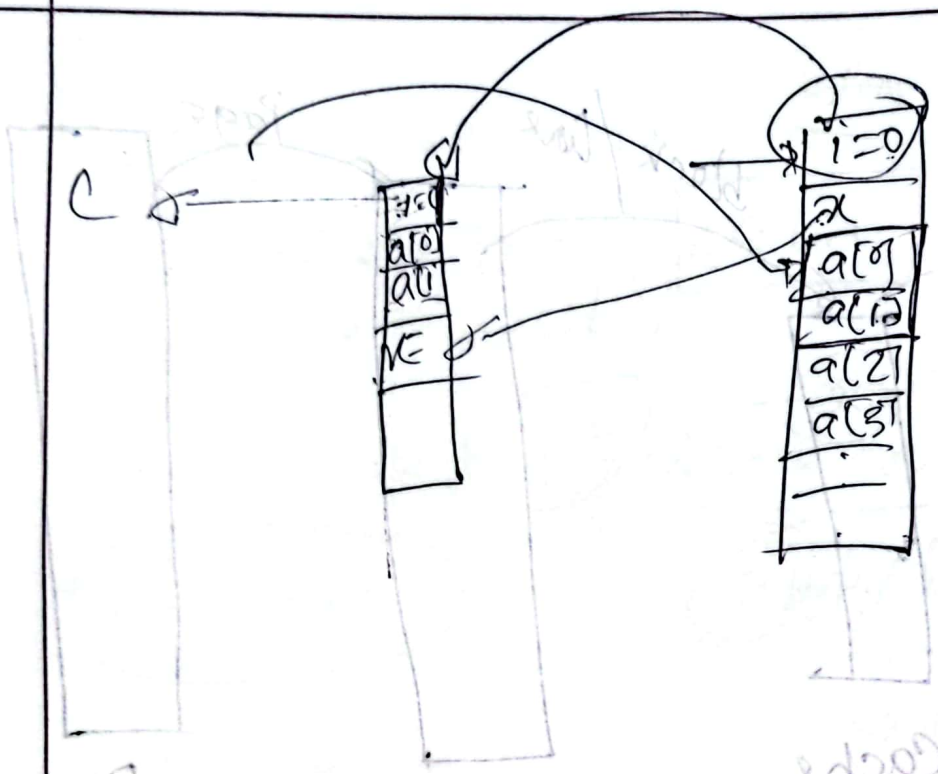
48
64

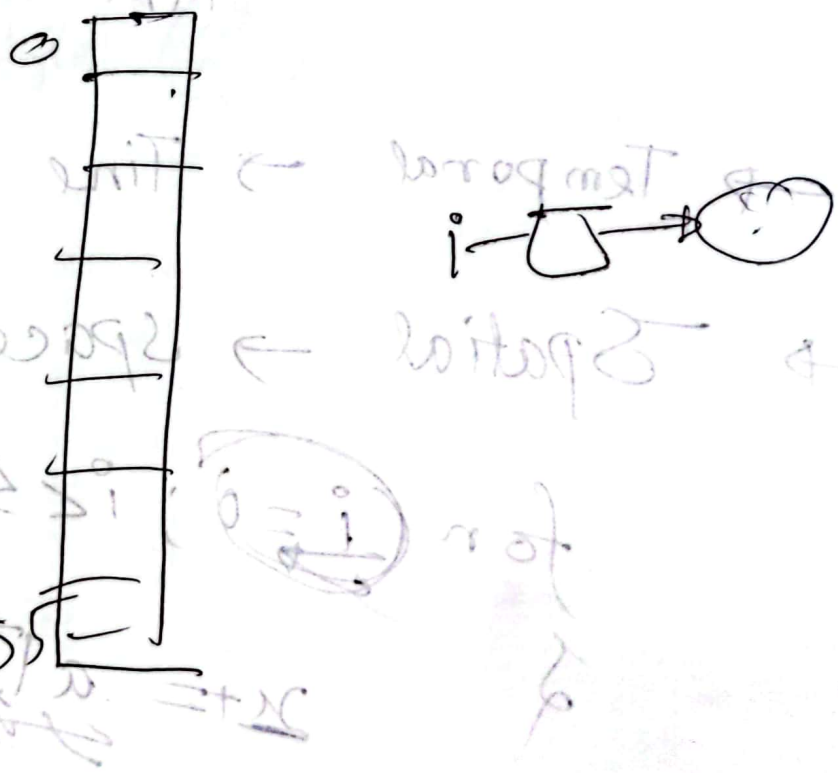


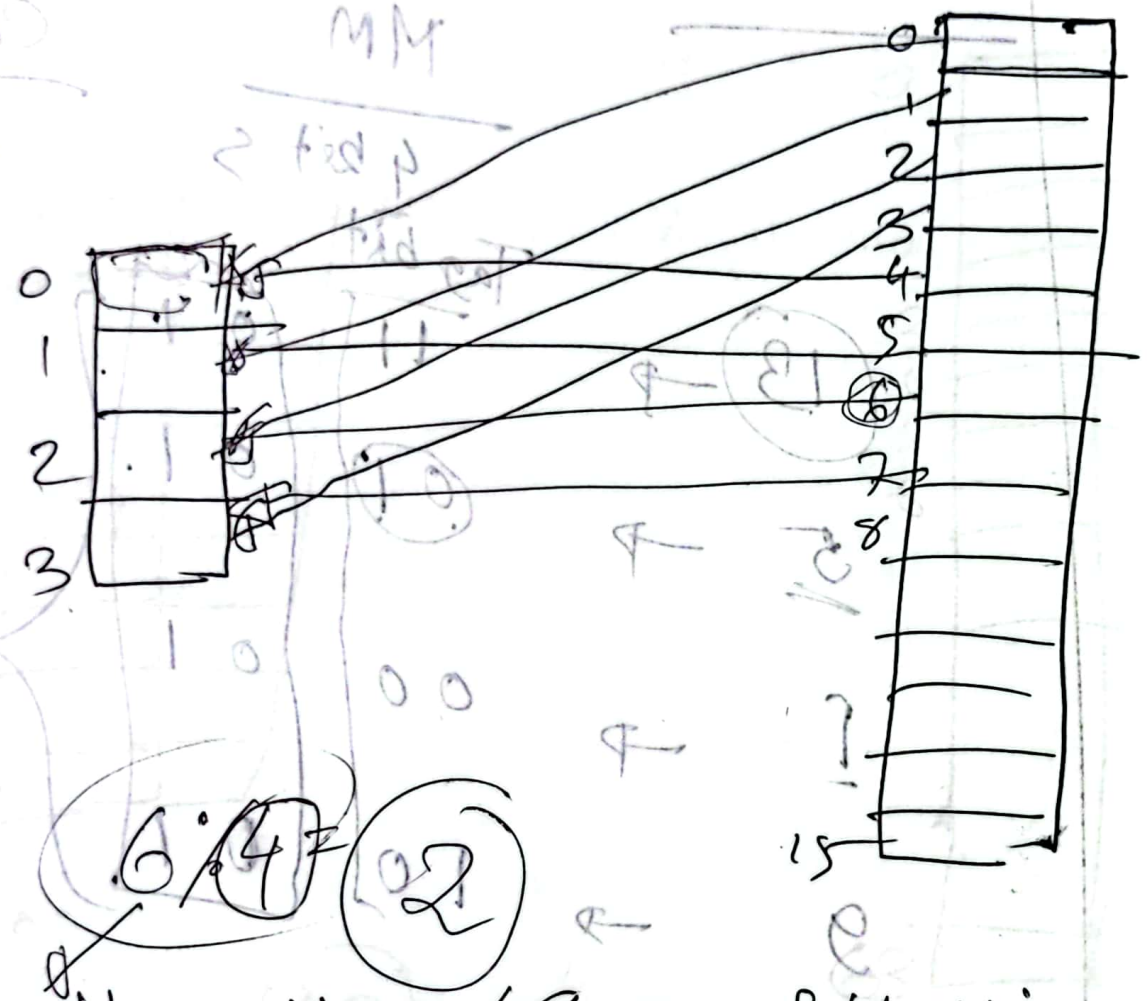
→ Temporal → Time

→ Spatial → Space

```
for (i=0; i<5; i++)
{
    x += a[i];
}
```


C
i=0


255



① block address ② No. of block in Cache

Cache

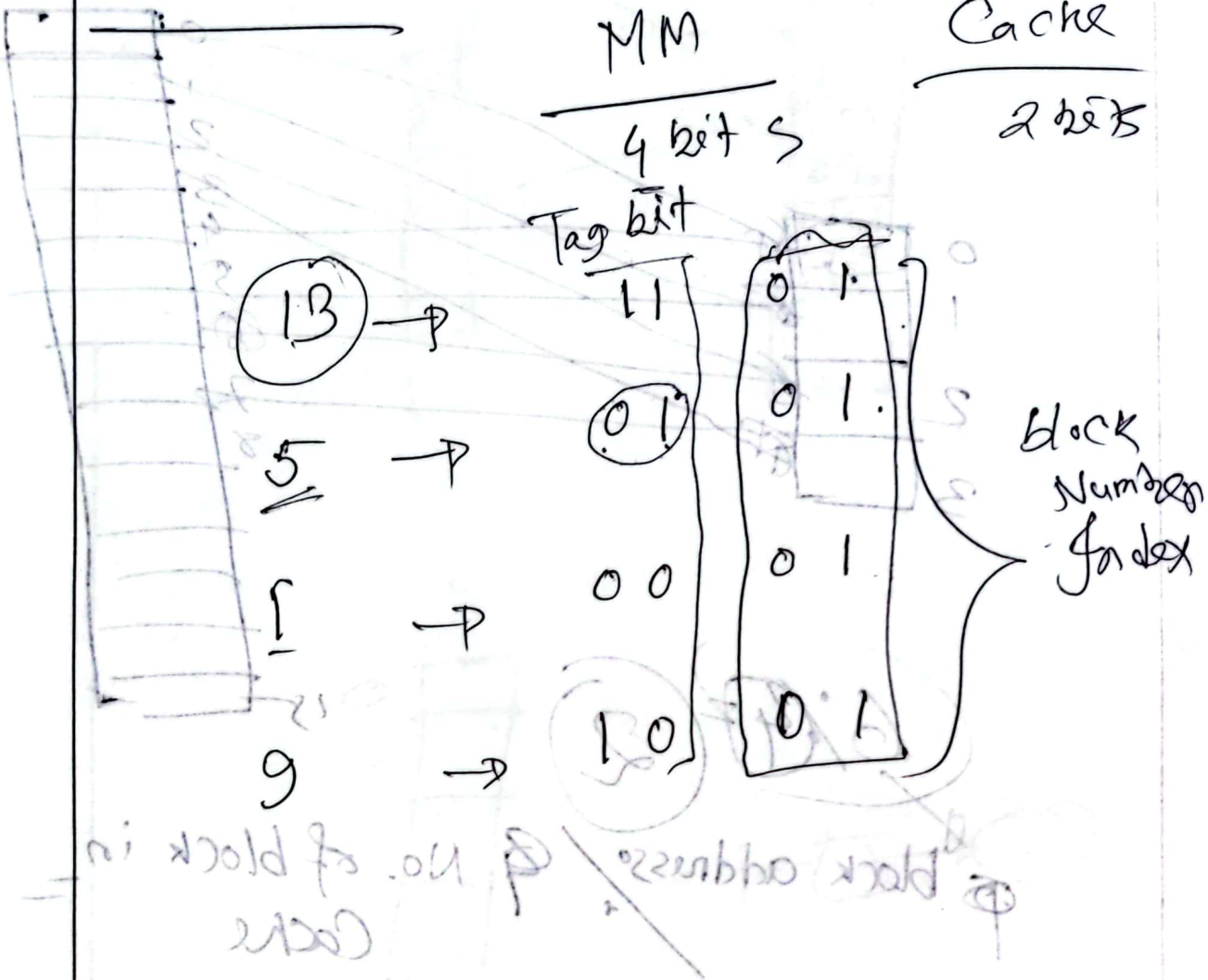
0: 0, 4, 8, 12

1: 1, 5, 9, 13

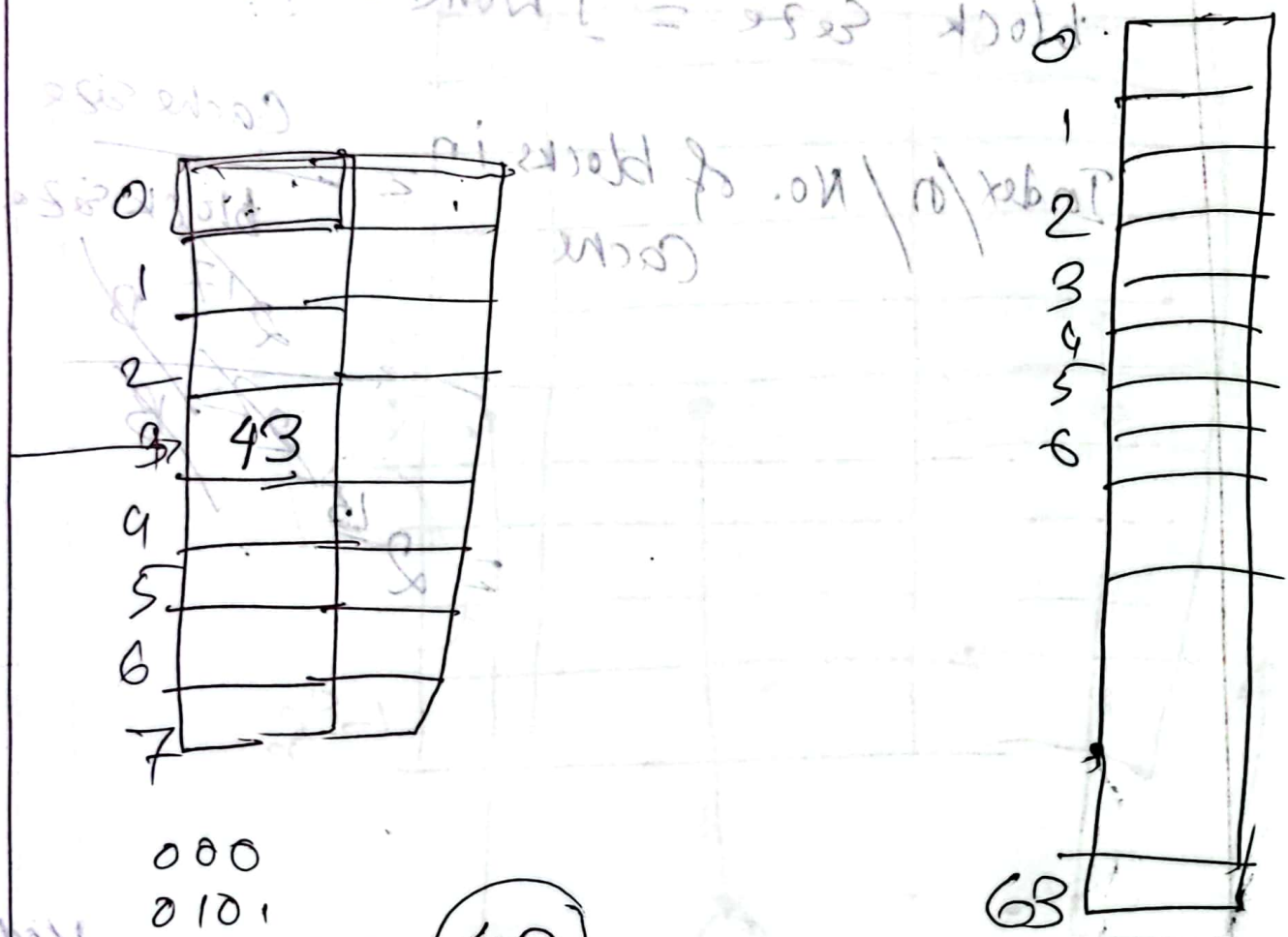
2: 2, 6, 10, 14

3: 3, 7, 11, 15

Tag bit:


MM
4 bits
Cache
2 bits
Tag bit
13
5
1
9
11
01
00
10
0 1
0 1
0 1
0 1
Block Number / Index

Ex: MIPS → (32 bit) → 4B → 1 Word



000
010
011 ; 011 (43)
100 ; 011
101 ; 011
110 ; 011
111 ; 011

3 : {

3, 11, 19, 27, 35, 43, 51, 59

Q. CPU = ~~32 bit~~

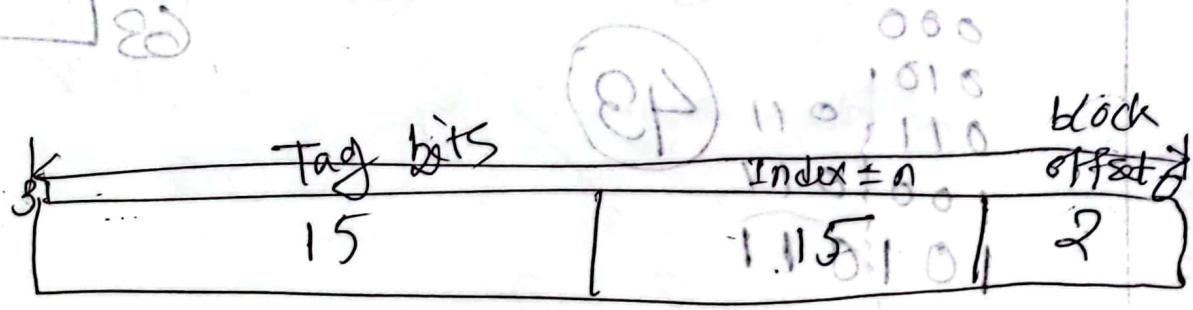
Cache size = 128 kB = $2^7 \times 2^{10} = 2^{17}$ B (2)

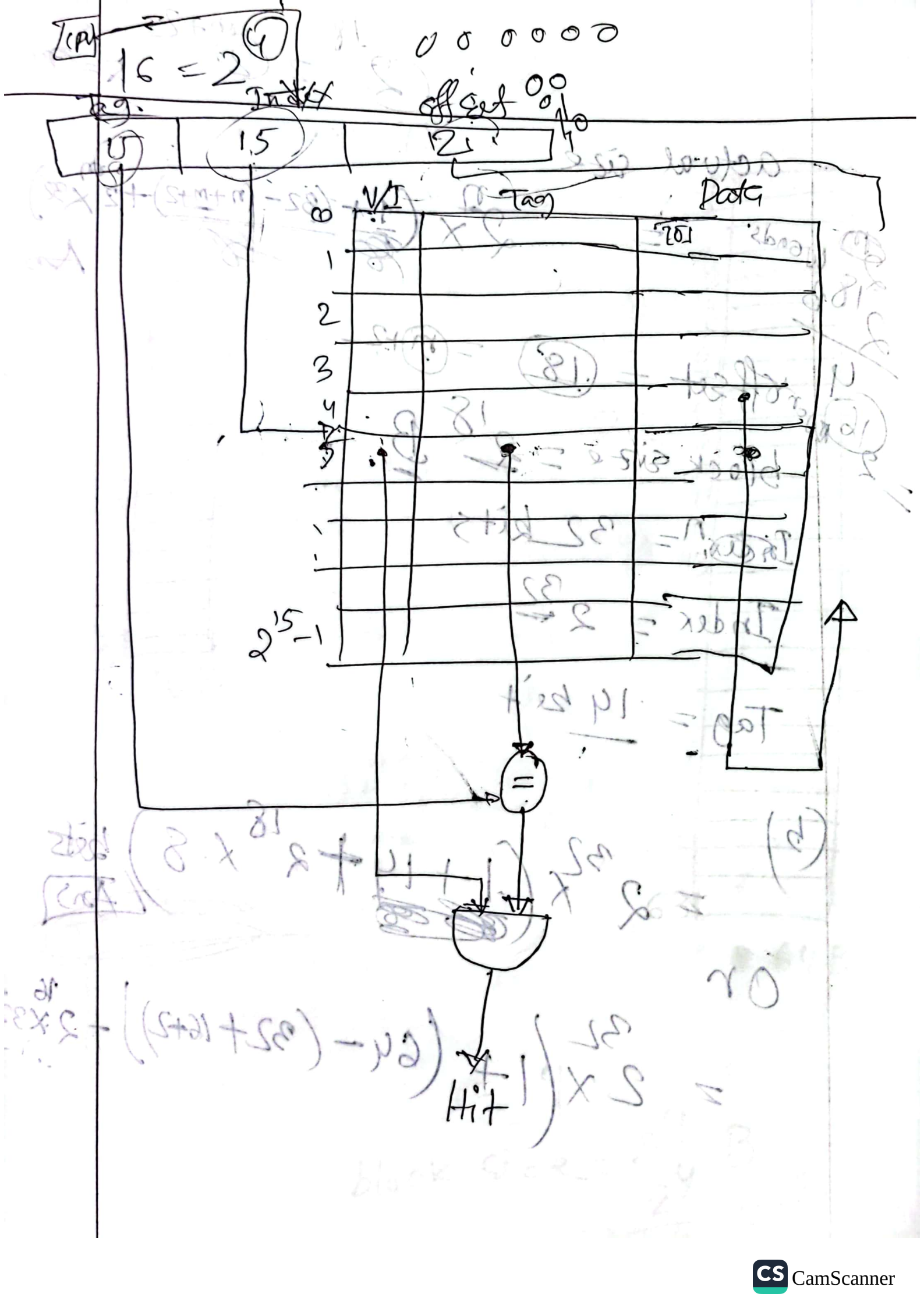
block size = 1 word = 4 B = 2^2 B

Index/n / No. of blocks in Cache = $\frac{\text{Cache size}}{\text{block size}}$

$= \frac{2^{17}\ B}{2^2\ B}$

$= 2^{15}$




16 = 2
Index
offset
15
12
V.I
Tag
Data
0
1
2
3
4
5
$2^{15}-1$
=
Hit

$2^{18} = 2^{m+2}$ $k = m$

actual size

$= 2^n \times \left(1 + (32 - (n+m+2)) + 2^m \times 32\right)$ Ans

2^{18} B

$\frac{2^{18}}{4}$ words

2^{16} words

offset = 18 = m+2

block size = 2^{18} B.

n = 32 bits

Index = 2^{32}

Tag = 14 bit

(b)

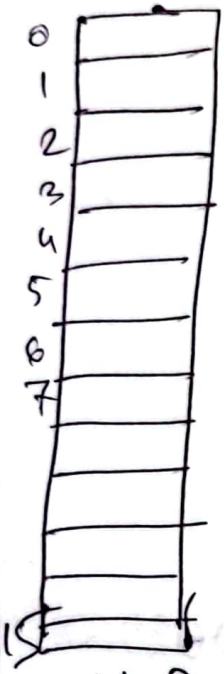
$= 2^{32} \times (1 + 14 + 2^{18} \times 8)$ bits (Ans)

Or

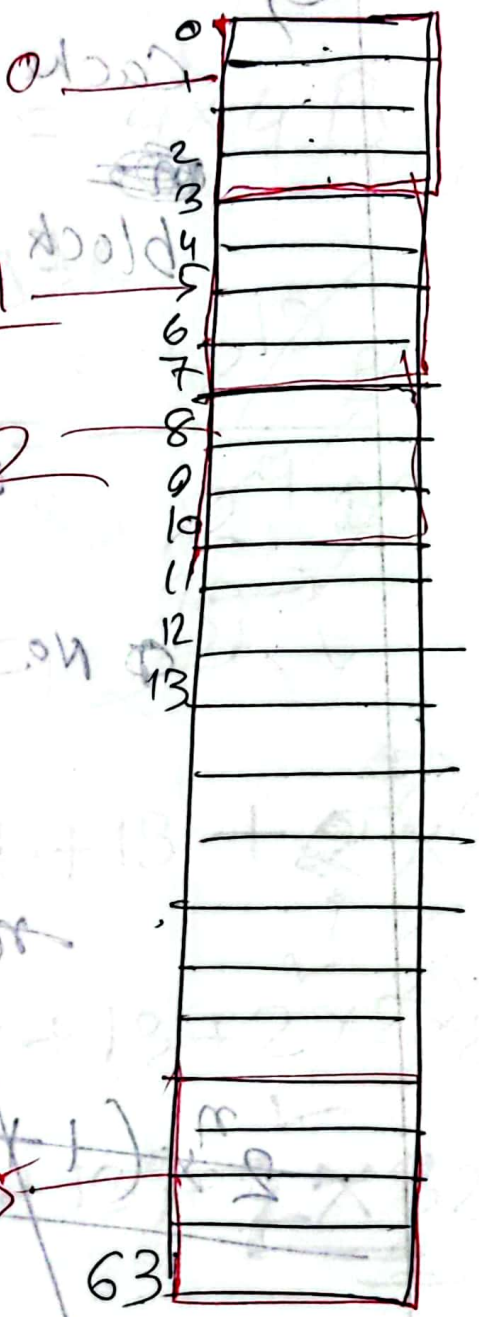
$= 2^{32} \times \left(1 + (64 - (32+16+2)) + 2^{16} \times \right.$

2

block size = 4B



16 B



64B

= 4

block size = 2^4 B

$= \frac{2^4}{4} = 2^2 = 4$ wo

(91)

Byte → bits (4)

Cache size = 2^{19} B

block size = 4 W = 4 × 4 B

= 2^4 B (4)

∴ No. of blocks = $\frac{2^{19}\ B}{2^4\ B}$

= 2^{15} (15)

cache actual size = $2^{15}(1 + 13 + 2^4 \times 8)$

= $2^{15}(1 + 13 + 2 \times 2^{4n} \times 8)$

= $2^{15}(1 + 13 + 2 \times 32)$

*

$$\text{block address} = \left\lfloor \frac{\text{given address}}{\text{block size}} \right\rfloor$$

$$= \left\lfloor 1200/16 \right\rfloor = 75$$

$$\text{block number} = 75 \% 64$$

$$= 11$$

4. Cache size = 4kB = 2^{12} B

block size = 4 word = 4 × 4 B = 2^4 B

No. of blocks = $\frac{2^{12}}{2^4} = 2^8$

1: → block address = $\lfloor 1/2^4 \rfloor = 0$

block number = $0 \% 2^8 = 0$. Miss

[0 – 15]

2: → block address = $\lfloor 2/2^4 \rfloor = 0$

block number = $0 \% 2^8 = 0$. Hit

10: → ~~Hit block address~~

4097: → block address = $\lfloor 4097/2^4 \rfloor = 256$

block number = $256 \% 2^8 = 0$ Hit

[4096 – 4111]

4098: → Hit

15: → Miss

(a)

Cache size = $2^8 \times 2^{10} = 2^{18}$

block size → words = $2^3 = 2 \times 4 = 2^3$

No of blocks = $\frac{2^{18}}{2^3} = 2^{15} = 32768$

12

block address = $\lfloor 42 / 2^3 \rfloor = 1$

block number = $1 \% 32768 = 1$

$2 \times 2^3 - (2^3 + 2^8)$

$\lfloor \frac{42}{8} \rfloor \% 2^{15}$ (miss)

42 → miss

block address = $\lfloor \frac{42}{2^3} \rfloor = 5$

block number = $5 \% 32768 = 5$

5×2^3

$\lfloor \frac{47}{8} \rfloor$

7 → miss

block add $= \lfloor 7 \div 2^3 \rfloor = 0$

block num $= 0 \% 32768 = 0$

15 → Hit

40 → Hit

39 → miss

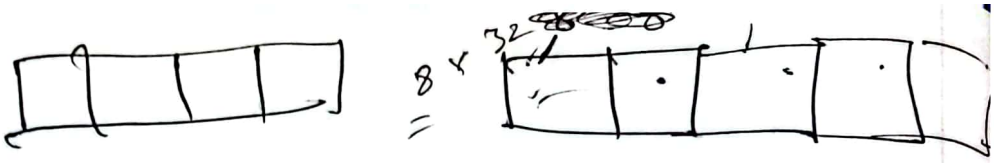
probability $= \frac{4}{6} = \frac{2}{3}$ [miss ratio]

20

cache size $= 2^8 \times 2^{10} = 2^{18}$

block size $= 2^2 \times 2^2 = 2^4 = 16$ → Element

No of blocks $= \frac{2^{18}}{2^4} = 2^{14} = 16384$



12 (miss)

block add = $\lfloor (12/2^4) \rfloor = 0$

block num = 0 % 16384 = 0

(0 থেকে 15) → (0 থেকে $2^4 - 1$ পর্যন্ত)

42 miss

block add = $\lfloor (42/2^4) \rfloor = 2$

block num = 2 % 16382 = 2

(offset) $2 \times 2^4 - \{2 \times 2^4 + (2^4 - 1)\}$

(32 - 47)

7 → Hit, 15 → Hit, 40 → Hit, 39 → Hit

miss rate = $\frac{2}{6}$

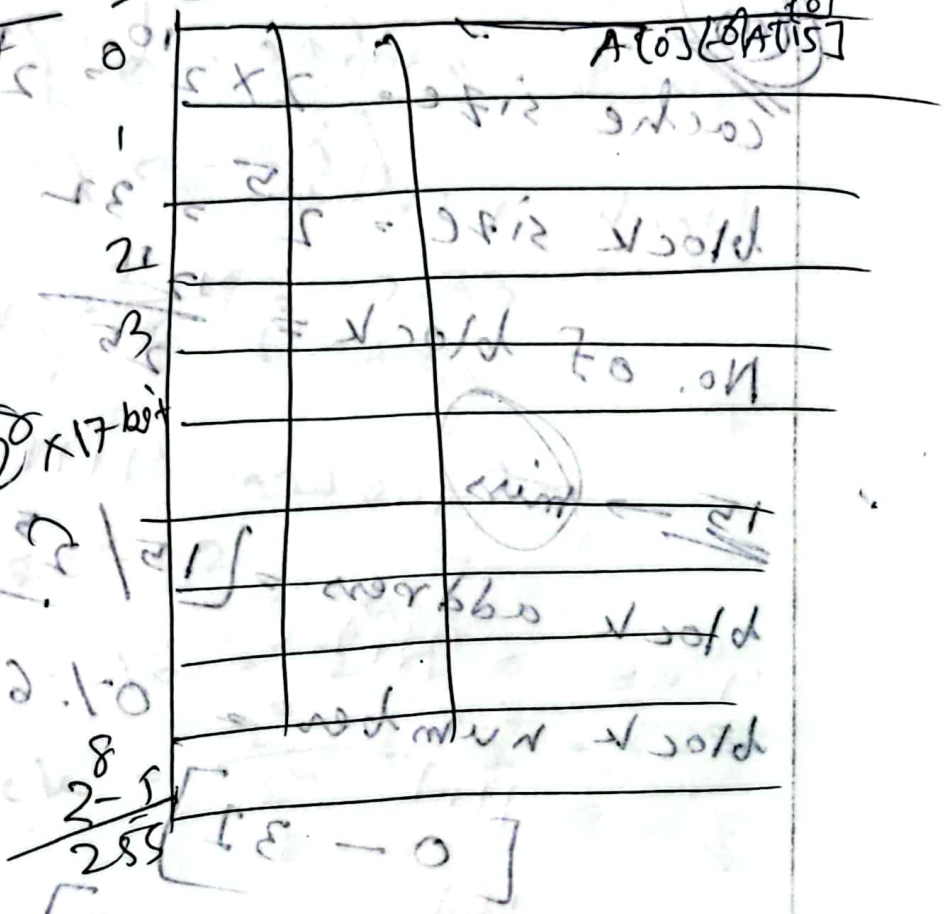
Cache size = 2^{15} B = 32 KB

block size = 128 B = 2^7 B

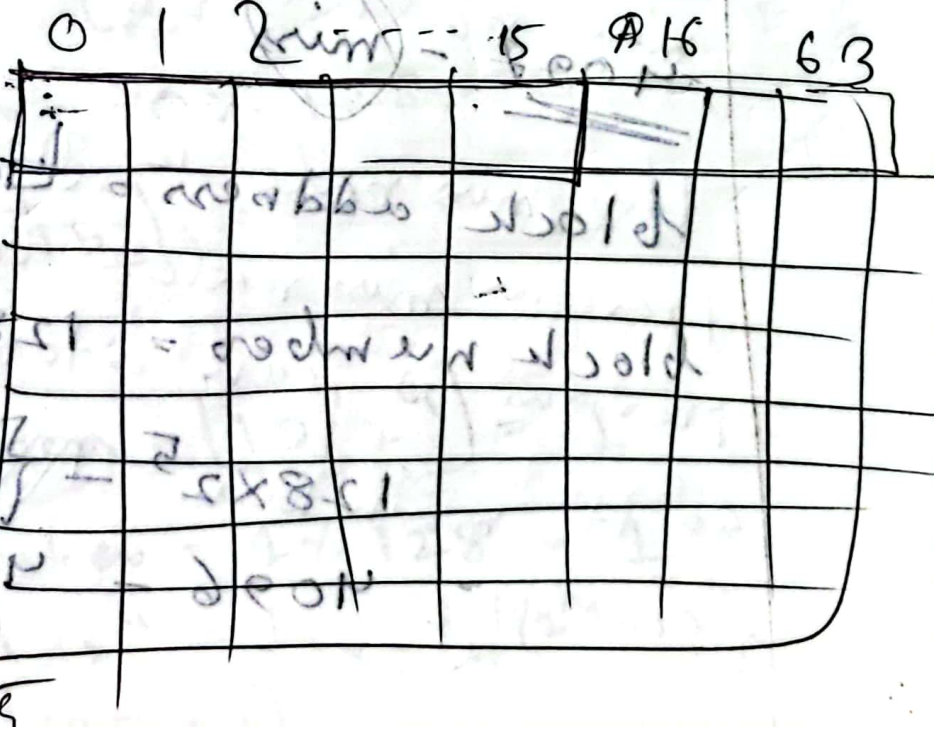
No. of block = $\frac{2^{15}}{2^7} = 2^8$

Tag = 17 bits

Tag directory = $2^8 \times 17$ bit



$\frac{128 \text{ bytes}}{8 \text{ bytes}} = 16$



No. of Miss in each row = 16

Total Miss = 4 × 64

Miss ratio = $\frac{4 \times 64}{64 \times 64} = \frac{1}{63}$

= 0.0625

Spring 22

3

(a)

cache size = $2 \times 2^{10} = 2^{11}$

block size = $2^5 = 32$

No. of block = $\frac{2^{11}}{2^5} = 2^6 = 64$

15 → miss

block address = $\lfloor 15 / 2^5 \rfloor = 0$

block number = $0 \% 64 = 0$

[0 - 31]

19 - hit [0 - 31]

4097 - miss

block address = $\lfloor 4097 / 2^5 \rfloor = 128$

block number = $128 \% 64 = 0$

$128 \times 2^5 - \{128 \times 2^5 + (2^5 - 1)\}$

$= 4096 - 4127$

4098 → hit [4096 or 4+2]

7 → hit [0 - 31]

30 → hit [0 - 31]

miss rate = $\frac{2}{6} = 0.33$

(b)

chache size = 2

block size = 16 = 2^4 H

No of block = $\frac{2^{11}}{2^4} = 2^7 = 128$ H

15 → miss

block address = $\lfloor 15/2^4 \rfloor = 0$

block number = 0 % 128 = 0

[0 → 15]

12 → miss

block address = $\lfloor 12/2^4 \rfloor = 0$

block number = 0 % 128 = 0

$1 \times 2^4 - \{1 \times 2^4 + (2^4 - 1)\}$

[16 - 31]

4097 → miss

block address = $\lfloor 4097 / 2^4 \rfloor = 256$

block number = $256 \times 128 = 0$

$256 \times 2^4 \rightarrow \{256 \times 2^4 + (2^4 - 1)\}$

[4096 – 4111]

4098 → Hit

7 → Hit

30 → Hit

miss rate = $\lfloor \frac{3}{6} \rfloor = 0.5$

this miss rate is greater than the previous miss rate. because in previous it can store more bytes. so it maintain spatial locality.

(4)

Tag size = 19 bits

Index = 7 bits

offset = 6 bits

Number of block = 2^7 blocks

bytes per block = 2^6 bytes per block

tag	index	offset
22	6	4

number of block = $2^6 = 64$

bytes per block = $2^4 = 16$

4	4	4	4

Spring-23

4.

(a)

cache size = $4 \times 2^{10} = 2^{12}$

block size = $4 \times 4 = 16 = 2^4$

No of block = $\frac{2^{12}}{2^4} = 2^8 = 256$

2 → miss

block add = $\lfloor 2/4 \rfloor = 0$

block number = $0 \% 256 = 0$

[0-15]

2 → hit [0-15]

10 → hit [0-15]

4097 → miss

block add = $\lfloor 4097/16 \rfloor \% 256$

block number = $(256 \% 256)$

$= 0$

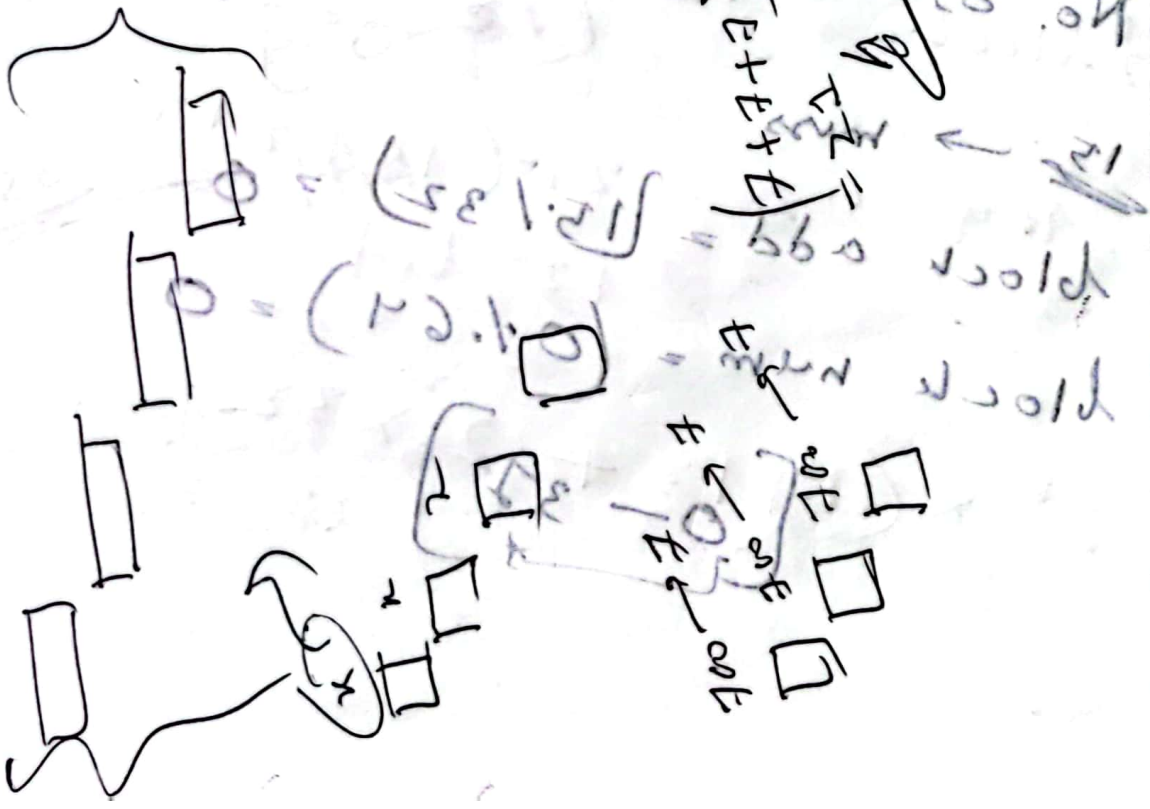
$256 \times 2^4 + \{256 \times 2^4 + (2^4 - 1)\}$

$= (4096 - 4113)$

4098 → hit [4096 – 4113]

15 → hit [0 – 15]

miss rate =



Cache size = 2 KB
= 2048 byte

~~block size = (32 × 8)~~
~~= 256 bits~~

~~No. of block = 8~~

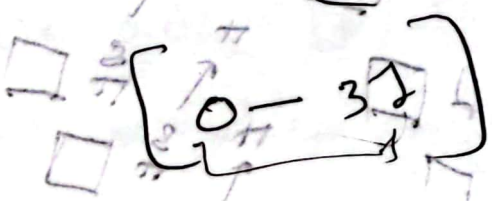
block size = (4 × 8)
= 32 bytes

No. of block = 64

15 → miss

block add = ⌊15/32⌋ = 0

block num = (0 % 64) = 0



0.00124

10 → Hit

4097 → miss

block add = (4097 / 32)

= 128

block num = (128 % 64)

= 0

[4097 — 4127]

4098 → Hit

71 → miss

[0-31]

30 → Hit

7.

Ans

cache size = 524288 bytes

block size = 16 x 4 = 16

no of block = 32768

(ii)

block size = 2^4 = m = 2

2

block size = 2^m

No of block = 2^n [2^{15}] n = 15

∴ tag size = 32 - (15 + 2 + 2)

= 13 bit

$2^n \times (1 + 13 + 2^2 \times 32$