Comp Architecture :-

Comp Arch is Comp + it's organisation

Comp has 5 basic components:

1) Processor —— control unit + Data path

Datapath – data travelling from one hardware component to another.

3 types of data path : control bus, data and address bus.

whether data should go from memory to R or vice versa whether read/write operation should be performed is done by control bus.

2) Memory

3) ~~I/O devices~~ Input

4) Output

18101089 computer: combination of hardware and software

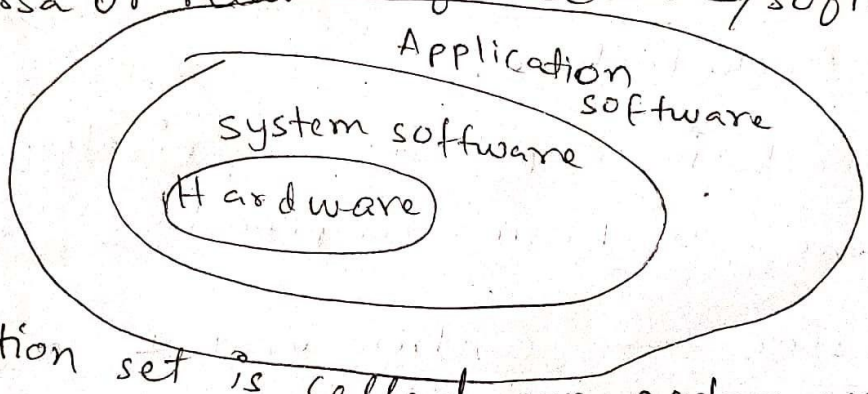
Computer Architecture: computer + it's instruc set organisation

(slide) Computer Architecture

→ Layer of Computer or relationship among software and hardware and instruction set.

Above green : software related
Below " : hardware. "

How we can go from low level to high and vice versa or relation of hardware/software with instruction set is called computer arch.



Compiler is system software

SAP :-

memory lower level, instruction store
higher " , data store.

You must know memory organisation
As a CSE engineer, you have to consider cost + needs. (you need to have hardware knowledge).

→ In MIDS arch, it is assembly lang.

a[i], b[i] → array // content should be moved to register.

c → variable stored in register

↳ 14 register

Arithmetic operations must involve register not memory in MIPS arch.

Read → load word (lw) } In MIPS
write → store word (sw) }

We need to know instruction format to convert high to low level.

Why?

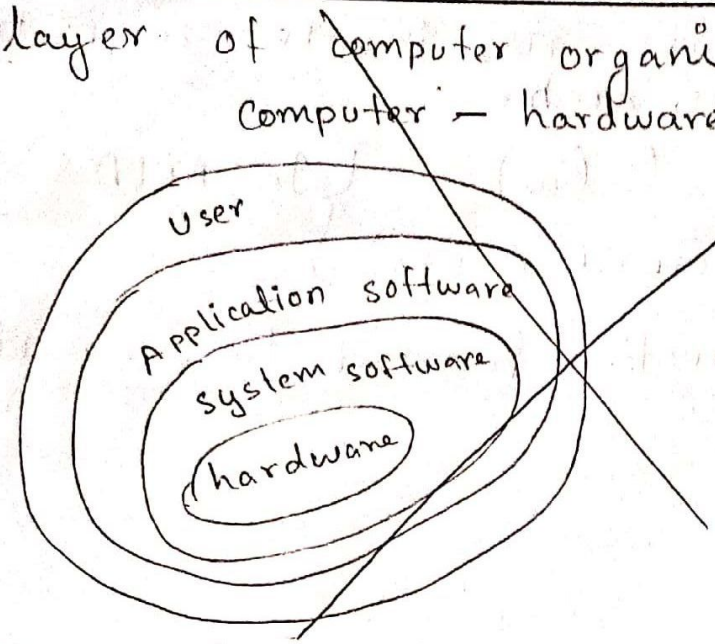
We need to know about generations, OS, application, arch changes for technological changes.
Depending on performance and technology, new generations of comp is developed.

Why?
1st parameter → Performance
How to measure performance?
i) Time Comparison.
Performance (slide)
Ques 1:
Depending on the work and hardware requirements, diff processor have diff performance, your personal requirements matter.

Ques 2:
OS does resource allocation, (new machine)
For new student, new pc should be added,
for faster performance, upgrade processor.

layer of computer organisation

Computer — hardware software



System software (OS)

Application software (MS-WORD)

Role of performance :-

parameters should be kept in mind.

Response time = CPU time + execution time

Throughput = Total work load

In a program, how many instructions are there, and how much time is required to execute all instructions.

Relative performance — comparison with other systems.

CPI — Clock per Instruction,

CPU clocking, instruction count - how many ins. in a program.

→ more passenger + more speed = better performance.

(slide) Comp Performance :-

* RT → Time it takes to execute ~~individual~~ program ins written by user.

→ how long you have to wait for database query.

18101089

CPU Time - OS allocation + Execution Time

Database Query - Collecting data from resource + ... time taken to run the data together is called response time.

Throughput - n ins execution response time is total workload + time

* throughput (Ans) new processor
* Response time (Ans) new machine

→ user CPU time concern more as when we're working for an application in an IDE or text editor and we compile and then we see output.

$P_x > P_y$ ∴ The lesser the execution time, the better performance.

$$\frac{1}{E_x} > \frac{1}{E_y}$$

$E_x < E_y$, So X is better.

Q. Afia = 10 sec Tahmid = 15 sec performance better by how much?

ET of A = 10 sec Execution Time

" " T = 15 sec

P of A = $\frac{1}{10}$

P of T = $\frac{1}{15}$

$\frac{P_A}{P_T} = n$ $P_A > P_T$

$n = 1.5$

18101089

After 1st page :

→ Every instruction has opcode for identification from instruction register.
→ This identification is same for all instructions.
→ If in a processor, face cycle requires 2 cycles, all instruction will require 2 cycles.
→ Many processor accesses memory to perform operations and many processor uses register. We use register to perform operations.
→ We do have to memory read/write. to read from memory and move to register, to write from register to memory, that's why memory reference → M+R. → requires. So the cycle will be diff here.
→ Register requires less time. So register refins is faster as you don't need to access memory or find out the memory address.
→ when programming, if you work with registers it'll be faster.
→ ase cycle and Execution Cycle, when you'll have to implement in hardware, you have to keep in mind small things.
→ Every block should be designed by you. Whether your processor is taking single or multi cycle for instruction execution. There is also pipelining, which is also parallel processing. And single multi cycle is sequential which means at a time the processor will execute multi instruction

18101089

(not multi cycle).

Cycle - ~~single cycle~~ Instruction will execute in single cycle.

" " " " multi cycle.

~~It~~ Pipelining - At a time, multiple instruction will be executed.

Identification - whether the searched data is in cache memory or main memory (RAM)

Placement - if in cache, then can be accessed from cache. If not, if in main, then where will you place? What will be the strategy?

Replacement - If cache is full, which data should be replaced by new data? There are algo for this.

If in cache, it'll be farter. How you will identify hit rate? What are the parameters?

mid - 20% - 10% - viva - CO - wise

→ when ques asked, must switch on camera.

18101089

Chapter - 1

1) CPU Performance and it's factors
2) Instruction performance and it's factors

Clock cycles:

Last lecture we have calculated performance based on execution time.

→ In a program, how many instructions are present, (not in seconds), but in cycle time.
→ How much time is required to run a program, is CPU execution time.

→ ① $\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$

CPU execution time = No. of cycles × cycle time / clock

→ cycle time = time between the ticks.
→ clock cycle time is inverse of clock rate.

↳ 1st performance equation.

If instead clock cycle time, clock rate is given. Inverse to find clock cycle time.

Perform inc, if clock cycle dec and clock rate inc perform inc

time / no. of cycles

→ No. of cycles differs depending on the type of instruction.

18101089

→ Mul is one kind of addition.
→ Diff hardware design of mul and addition.

CPU
Execution time for A = 10 sec clock rate = 400 MHz
" " " for B = 6 sec

No. of cycles for A = $\frac{10}{400\text{MHz}}$ 10×400 MHz
= 4000 M cycles

CPU time B = $\frac{1.2 \times \text{clock cycles of A}}{\text{clock rate B}}$

clock rate B = $\frac{1.2 \times 4000\text{ M}}{6}$

= 800 MHz

clock rate inc, performance inc. clock cycles dec, performance inc.
Viva: What clock rate should we tell the designer to target?
To run the same program in 6 sec, B must have twice the clock rate of A.

Instruction Performance:

CPU clock cycles: instruction for a program x avg. clock cycles per instruction (CPI)

(MIPS) - millions of ins per sec
cycles per sec - clock rate
ins per sec - ins rate

Execution time = Ins x avg CPI x cycle time

clock cycle replaced..

ISA - Ins set arch.

A c.c.t = 10 ns CPI = 2.0

E.T (A) = I x 2 x 10 ns = 20 x 10^{-9} s I

E.T (B) = I x 1.2 x 20 ns = 24 ns I

Machine B faster by 24 ns - 20 ns = 4 ns.

Cpu C.C = Ins x Avg CPI

C.C (A) = I x 2.0

C.C (B) = I x 1.2

cpu time = C.C x cycle time

time (A) = I x 2 x 10 ns = 20 Ins

time (B) = I x 1.2 x 20 ns = 24 Ins

$\frac{P_A}{P_B} = \frac{E_B}{E_A}$ A is faster

$= \frac{I \times 1.2 \times 20 ns}{I \times 2 \times 10 ns} = 1.2$

A is 1.2 times faster than B.

18101089

cpu performance equation ⇒ cpu time = Ins count × CPI × c.c.t

(or)

CPU time = $\frac{\text{Ins count} \times \text{CPI}}{\text{clock rate}}$, as the clock rate is inverse of c.c.t

CPI Example 11 :.

same machine, two code sequences

3 types of ins : A, B, C

A = 1 cycle

B = 2 "

C = 3 "

Instruction :

A = 2

B = 1

C = 2

1st code

A = 4

B = 1

C = 1

2nd code

1st code :

~~CPU time~~ =

5 =. 5 × CPI CPI = 1

time = 5 × 1 ×

18101089

(C.S.1) $CPU\ C.C = \sum_{i=1}^{n} (CPI \times C_i)$

$= (1 \times 2) + (2 \times 1) + (3 \times 2)$

$= 10$ cycles

(C.S.2)

CPU C.C $= (4 \times 1) + (1 \times 2) + (1 \times 3)$

$= 9$ cycles (faster)

$CPI_{(C.S.1)} = \frac{10}{5}$

$= 2.0$

CPI (" " .2) $= \frac{9}{6} = 1.5$

$\therefore$ C.P.I less, performance will be faster.

MIPS Example H.W

18101089

Chapter - 3 Instructions : Language of the Machine

MIDS is a processor, micr comp without interloop pipining states
MIPS - million ins per sec.
MIPS arithmetic ins uses exactly 3 variables.
that's why it is said to be restrictive.

3 operands : des, 1st source operand, 2nd " ".
register or operand. or register

→ There must be 3 operands.

If, A = x+y-z // assigned to register.

$A = x + y - z$; there are 32 registers.
S_0 S_1 S_2 S_3

s - means same register.

→ Add $to, $S1, $S2 ; to - temporary register

→ Sub $S0, $to, $S3 ;

⇒ MIPS arithmetic provides restriction 3 operands must be present, for simplication of hardware design.

4 design principle:

(1st) simplicity favours regularity.

→ every arithmetic ins format is same.

(2nd) Operands must be in registers (0-31) in 5 bits.

from variables, high level to low level has to be converted. Data length, memory length, register length must be represented in bits.

→ smaller is faster. If you can perform all operations and work within 32 bits, it be faster.

→ The more hardware, the more clock cycle time.

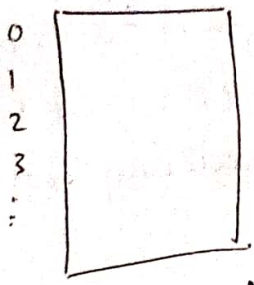
→ So, smaller is faster.

i = 0 to 50 A[i]

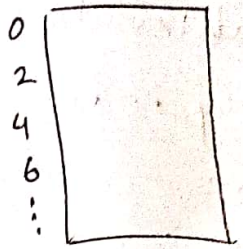
register splits to memory if not enough registers.

Memory Organisation:
(byte addressing) – 8 bit length for every address

8 bits



16 bits



Array A[i]
base register=A

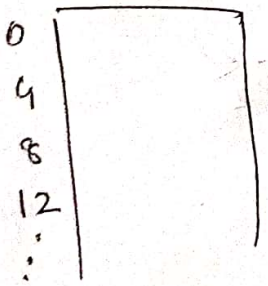
base and offset = physical.
base + offset = "
segment + offset. = "

MIPS, word = 4 bytes or 32 bits

Load / Store Instructions:
→ both are memory refrence instruction
load → memory to register (read)
store → R to m (write)

A[8] – memory's 8 no slot content

For MIPS



load → des, source
store → source, des

$s3 – A[ ] base register
32 – offset (8 x 4)
MIPS data 4 byte length is 32 bit which is 4 byte in byte addressing.
offset = index x 4 for MIPS (byte address)

A[12] = x + A[3] - y;

S_0 ← S_1 ← S_0 → S_3

1st step: variables to be assigned to registers.

MIPS Assembly code: [25 - 20 marks in mid] formatting wrong - 0 marks

1st instructions:

```
lw $to, 12($S0);
Add $to, $S1, $to;
Sub $to, $to, $S3;
Sw $to, 48($S0);
```

→ Variables gets directly assigned to registers after conipilation.

→ memory reference instructions should be read/write in register (Array).

memory read → load, memory write → store

A[2] = A[12] + B[3] + C[2];

S_0 ← S_0 ← S_1 ← S_2

```
lw $to, 48($S0);
lw $t1, 12($S1);
lw $t2, 8($S2);
Add $to, $to, $t1;
Add $to, $to, $t2;
sw $to, 8($S0);
```

A[i] = x - y - A[i+1] where i = 100.

S_0 S_1 S_2 S_0

```
lw $to, 404($S0);
Sub $to, $S2, $to;
Sub $to, $S1, $to;
sw $to, 400($S0);
```

A[100] = x - y - A[101];

CSE 317 High Level lang → Assembly language → Machine language → 0, 1

Machine language :-

There are 3 class of instruction format:

1) R-type (Register reference instruction)
2) I-type (Memory " ")
3) J-type (Jump instruction, jumps regular instruction).

1) R-type instruction format : (Arithmetic Instructions)

ALU has 2 parts → AU (Arithmetic Unit)
→ LU (Logical Unit)

→ AU, LU has different functions, ~~where is the~~ to which function it's going, execution is taking place for which type of instruction or which type of operation

For Arith Ins, the ins format is 32 bit long.

op	rs	rt	rd	shift	function code

6 fields total

① Operation code (length = 6 bit)

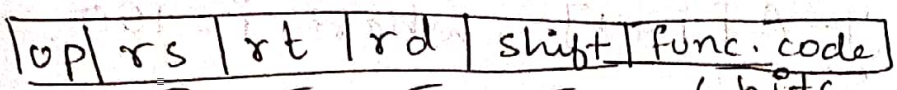
MIPS = 32 registers

Add $to, $S1, $S2;

- $S1 → source 1 (rs)
- $to → destination (rd)
- $S2 → source 2 (rt)
- Add → function defined

Data length = 32 bit.

Every register is 5 bits.



total = 32 bits

① Operation code ② 1st source operand or ~~source~~ register ③ 2nd source operand or register

④ Destination operand or register ⑤ shift amount

⑥ specific arithmetic function

Diff between func. code and op code?

→ By looking at op code, you can decide it is arithmetic operation or not.

→ If arithmetic, in arith which particular func that will be selected by func code.

→ In ALU, S_2 → Mode selection, AU or LU

→ Every ins 32 bits long.

A = B + C; write machine code for this ins

S_0 ← A, S_1 ← B, S_2 ← C

op code for arithmetic ins = 0

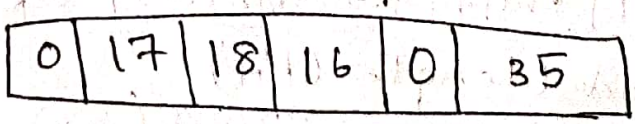
0 in 6 bits = 000000 // R-type op code

function code = 35 for add ins.

register no. { S_0 = 16, S_1 = 17, S_2 = 18 // No need to memorise opcode or register no. Technically assume reg no. (0 - 31) in 5 bits

Op code and func code in 6 bits.

In format, // assigning



Add $ S_0, $ S_1, $ S_2;

rd rs rt

In case of add sub, shift = 0.

① Assigning ② Represent in binary machine code.

op	rs	rt	rd	shift	func code
000000	10001	10010	10000	00000	100011
6	5	5	5	5	6

For sub, same reg can be used except ~~sub op code needed~~ f sub's func code needed / op code will be same.

Op code — R, I, J type func code - Add, sub, inc, dec for R-type.

→ helps to identify or convert low level to high level.

A = D + E + F;
↳ s_0 ↳ s_1 ↳ s_2 ↳ s_3

Add $to, $s1, $s2;
Add $s0, $to, $s3;

2 statement, 2 machine code.

1 full set question from this topic

2) I-type instruction.

lw $to, 32($s0);
↳ des. → cons → base

~~sw~~

sw $to, 12($s0);

high level → A[3] = B + A[8];

lw

6	5	5	16bits
op			offset

rs → base
rt → des for lw / source for sw
offset or cons.

base and source/des.

Eg:- lw = 42
sw = 45 } op code. to = 8
s0 = 16

i) | 42 | 16 | 8 | 32 |

= | 101010 | 10000 | 01000 | |

0000000000100000

ii) | 45 | 16 | 8 | 12 |

= | 10 | 11 | 01 | 10000 | 01000 | | → 00000000000001100

Example:-

So A[3] = X + B[100] − Y ; Convert this high level statement into MIPS Assembly machine code.

(S1) (S2) (S3)

~~lw $to,~~ $t_0 = 8, S_0 = 16, S_1 = 17, S_2 = 18, S_3 = 19, Add = 32, Sub = 35, lw = 42, sw = 45.

```
lw $to, 400($S2);
Add $to, $S1, $to;
Sub $to, $to, $S3;
sw $to, 12($S0);
```

MIPS Machine code:

| 42 | 18 | 8 | 400 | → binary

| 0 | 17 | 8 | 8 | 0 | 32 | → binary

| 0 | 8 | 19 | 8 | 0 | 35 | → binary

| 45 | 16 | 8 | 12 | → binary

Overview: shift left or right

i) sll, slr → R
ii) Addi, subi → add immediate → I
iii) beq, bne → branch equal, " not " → conditional Jump → I
iv) AND, OR, NOT... → logical operation → R
v) Jump → unconditional Jump → J

List of registers

J Exit;

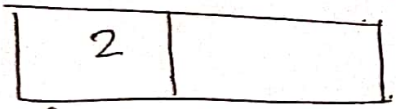
J Address → Label;

opcode

program counter → counts ins and indicates next ins.

Format of J-type:

OP Address: 2 is fixed for Jump



sll = 0 slr = 2

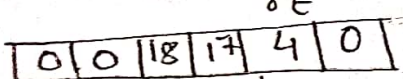
17

sll $S_1, $S_2, 4;

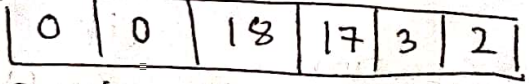
18

rd rt

$S_1 = S_2 << 4$



slr $S_1, $S_2, 3;



Book 2.4 Representing Instructions in the Computer.
→ Binary to Hex and Back.
→ OP code → 0 (R), 2 (J), Value (I)
Design principles explained in books.
Loop is unconditional Jump (J-type)
Pg:-78 MIPS machine language table.
Figure 2.18 MIPS register conventions.
Pg: 101 See before exam.
CT Syllabus: 1) Intro 2) Performance
3) Lang of comp → Instructions

R → slr → 2bits R → s ll → 3 bits

0	0	16	17	2	2

0	0	16	17	3	0

Ques type : Performance math, short ques, problem solving high level to low level statement, low level to high level conversion.

Definition : Comp Arch, Ins set, Comp layer, abstraction.

Answer to Question 1 :-

Comp A

clock cycle time = 250 ps CPI = 1.8

Comp B

clock cycle time = 500ps CPI = 1.2

For Comp A :

Cpu clock cycle = Instructions x Avg CPI

= I x 1.8

Cpu execution time = clk cycle x clk cycle time

= I x 1.8 x 250 ps

= I x 450 ps

For Comp B :

"

= "

" = I x 1.2

= I " x 1.8 x " 500ps = I x 900ps

$$\frac{P_A}{P_B} = \frac{E_B}{E_A} = \frac{I \times 900ps}{I \times 450ps}$$

$$= 2$$

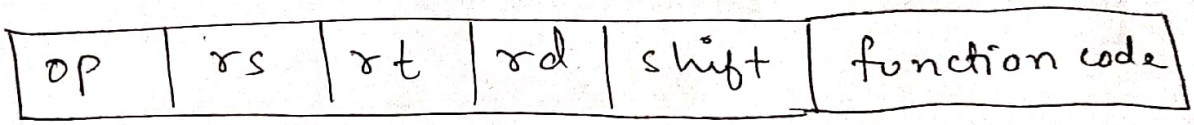
~~A is faster~~ A is 2 times faster than B.

Answer to Question 2:

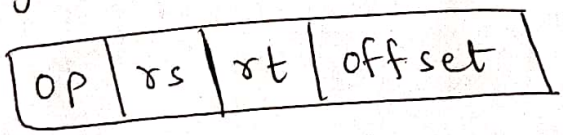
3 types of instructions:

1) R-type
2) I-type
3) J-type

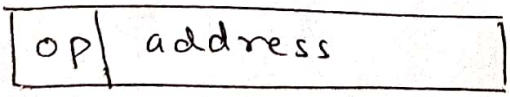
1) R-type



2) I-type



3) J-type



Answer to Question 3 :-

My ID = 18101089

$t_0 = 17$, $S_1 = 18$, $S_3 = 19$

$S_0 = 20$

$X = 89$

A[89] = 1 + A[32] - j;

S_0 S_1 S_0 S_3

① lw $t0, 128($S0);

② Add $t0, $S1, $t0;

③ Sub $t0, $t0, $S3;

④ sw $t0, 356($S0);

0x1713a0a03

① | 35 | 20 | 17 | 128 |

② | 0 | 18 | 17 | 17 | 0 | 32 |

③ | 0 | 17 | 19 | 17 | 0 | 34 |

④ | 43 | 20 | 17 | 356 |

(9)

43	18	17	356

=

101011	10010	10001	0000000101100100

Arithmetic for Computers :

Chapter 3

* watch lecture from beginning

ALU's base is Adder. Ins execution takes place with the help of adders.

-2 represent by doing 2's complement.

$$A - B = A + B' + 1;$$

(slides skipped)

→ MUX - input signals selection. (multiple input single output). ($2^n : 1$) If 2 unit input one 32 bit output.

→ MIPS data bus length - 32 bits.

→ HA can calculate single bit. When you work with more than one bit, you'll need FA(n bits)

→ In HA, Carry in doesn't work as input

16 no slide, not needed much for exam But when we study mul, all will be needed.

→ For mul in MIPS, there are 3 versions. Update for faster mul. 4th version for unsigned number also.

m bits x n bits = m+n bits product

Ver-2

M_n ← 00110 × 00100 → M_x

6×4 using 5 bits multiplier

	step	Mx	Mn	Product
0	1. Test $m_{x=0}$ 2. NOP 3. $\vec{P}$ 4. $\vec{M_x}$	00100 00010	00110	00000 00000 00000 00000
1	1. Test $m_x=0$ 2. NOP 3. $\vec{P}$ 4. $\vec{M_x}$	 00001		00000 00000
2	1. Test $m_x=1$ 2. $P_L = P_L + M_n$ 3. $\vec{P}$ 4. $\vec{M_x}$	 00000		00110 00000 00011 00000
3	1. Test $m_x=0$ 2. NOP 3. $\vec{P}$ 4. $\vec{M_x}$	 00000		00001 10000
4	1. Test $m_x=0$ 2. NOP 3. $\vec{P}$ 4. $\vec{M_x}$	 00000		00000 11000

Ver-1 of Mul:

M_x = 32 bits M_n and P = 2x 32 bits (64)

$M_x = M_0\ M_1 \ldots M_{31}$

1. LSB of $M_x = 0$, then step 2

a) " " " = 1, Add $P_{(R)} = M_n + P_{(R)}$, then step 2 ($M_{n(R)}$)

2. $M_n \rightarrow$
3. $M_x \rightarrow$

M_x = n bits Prod = M_n = 2 x n bits

eg:- Multiplier = 4bits (4 iterations)

$M_n \rightarrow 0101 \times 0011 = 15\ (1111)$ (M_x: 5 → 3)

iteration	steps	M_n (8)	M_x (4)	P (8 bits)	
0	initialisation	0000 0101	0011	0000	0000
1	1. $M_{x_0} = 1$ 1.a) $P = P + M_n$ 2. $M_n \rightarrow$ 3. $M_x \rightarrow$	0000 1010	0001	0000	0101
2	1. $M_{x_0} = 1$ 1.a) $P = P + M_n$ 2. $M_n \rightarrow$ 3. $M_x \rightarrow$	0001 0100	0000	0000	1111
3	1. $M_{x_0} = 0$ 1.a) NOP 2. $M_n \rightarrow$ 3. $M_x \rightarrow$	0010 1000	0000	0000	1111

4	1. $M_{x_0}=0$ 1.a) NOP 2. $\overleftarrow{M_n}$ 3. $\overrightarrow{M_x}$	0 1 0 1 0 0 0 0	0 0 0 0	0 0 0 0 1 1 1 1 Ans :- (15)

→ Product register one side always unused, so every bit generations requires power consumption. less bits used, faster operations performed.

→ It requires more Clk cycles.

Ver-2 LSB of Mx

1a $P_{(L)} = P(L) + Mn$; if 1

2 $\overrightarrow{P}$

3 $\overrightarrow{M_x}$

$M_n = M_x = 32$ bits

$P = 64$ bits

Mn and M_x = 'n' $P = 2 \times n$

$M_n = M_x = n$ bits

$P = 2 \times n$ bits

Ver-3

① Test lsb of P = step 2 if 0

② $\overrightarrow{P}$ $= P_{(L)} = P_{(L)} + Mn$ if 1.

iteration = Mx bits

$P(R) = Mx$

$Mn = n$

$P = 2 \times n$

No ③ step.

Motivating Booth's Algo : (for unsigned number mul)

→ Diff between arithmetic right shift and logical right shift.

→ arithmetic – msb will be copy of previous bit.

If $M_x = n$, iteration n;

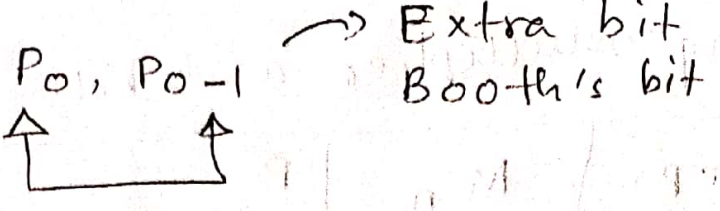
Till Mul for Mid.

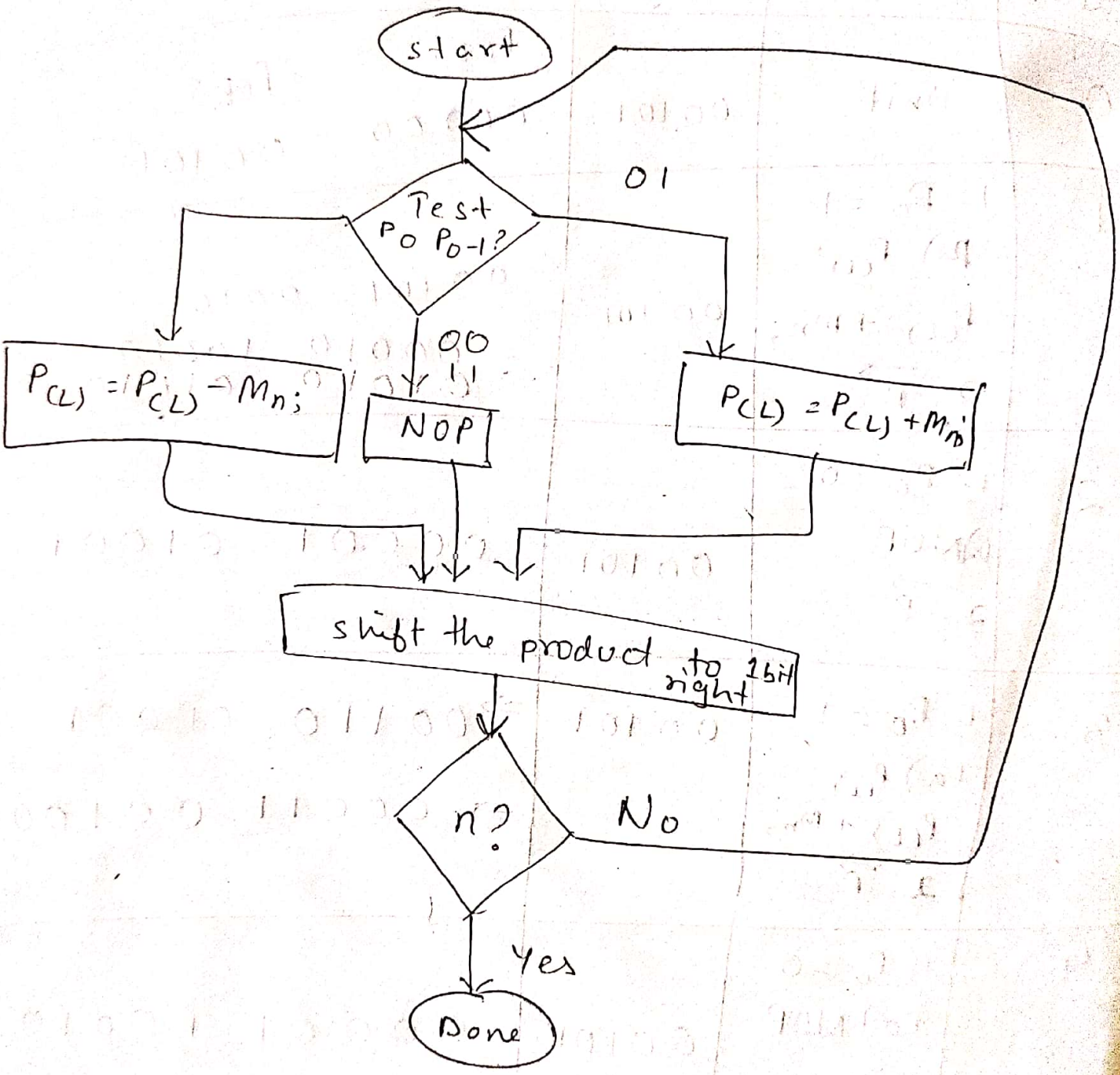
Ver 3

5 x 5 using multiplier as 5 bits

iteration	steps	Mn	P
			P(L) P(R)
0	init	00101	00000 . 00101
1	1- $P_0 = 1$ 1a) $P_{(L)} = P_{(L)} + Mn$; 2. $\overrightarrow{P}$	00101	00101 00101 → 00010 10010
2	1. $P_0 = 0$ 1a) NOP 2. $\overrightarrow{P}$	00101	00001 01001
3	1. $P_0 = 1$ 1a) $P_{(L)} = P_{(L)} + Mn$; 2. $\overrightarrow{P}$	00101	=> 00110 01001 → 00011 00100
4	1. $P_0 = 0$ 1a) NOP 2. $\overrightarrow{P}$	00101	00001 10010
5	1. $P_0 = 0$ 1a) NOP 2. $\overrightarrow{P}$	00101	00000 11001

Booth's multiplier (+/−)





$3 \times (-7)$ → 5 bits

Mx should be -ve

Mn ↵

$A + B' + 1$

$= A - B$

$Mn = 00011$

$Mn' = 11100$

$-Mn = \quad +1$

$11101 \rightarrow -Mn$

1. NOP
2. $P_{(L)} = P_{(L)} - Mn; 10$
3. $P_{(L)} = P_{(L)} + Mn; 01$

$-7 \rightarrow 00111$

11000

$\quad\quad\quad 1$

11001

$11001 = -7$

iteration	steps	Mn	P extra bit
0	initialisation	00011	00000 $\quad -7$ 110010
1	1. $P_0P_{0-1} = 10$ 1a) $P_{(L)} = P_{(L)}$ 2. Mn; $\vec{P}$	00011 00011	11101 $\quad$ 110010 11110 $\quad$ 111001
2	1. $P_0P_{0-1} = 01$ 1a) $P_{(L)} = P_L +$ Mn; 2. $\vec{P}$	00011	00001 $\quad$ 111001 00000 $\quad$ 111100
3	1. $P_0P_{0-1} = 00$ 1a) NOP 2. $\vec{P}$	00011	00000 $\quad$ 011110
4	1. $P_0P_{0-1} = 10$ 1a) $P_{(L)} =$ $P_{(L)} - Mn;$ 2. $\vec{P}$	00011	11101 $\quad$ 011110 → 11110 $\quad$ 101111

5	1. $P_0 P_{0-1} = 11$ 1a) NOP 2. $\overrightarrow{P}$	00011	11111 01011 ✗ Result

21 → 000010101

11111 01010

1

11111 01011

4 Ques 1 will (OR) Q4 → −21

1, 2 mandatory