CSE340

Moinul Islam
CSE Dept
Faculty:DJU
Semester:Summer'19

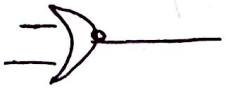
CSE-340

17/05/19

Lecture-1 & 2 || Combinational Logic-1

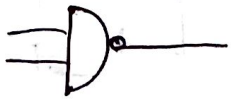
OR

NOR

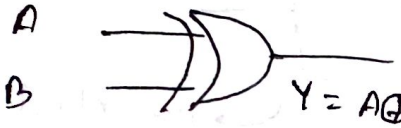


AND:

NAND



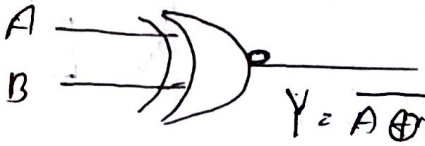
XOR:



$Y = A \oplus B$

$= A\bar{B} + \bar{A}B$

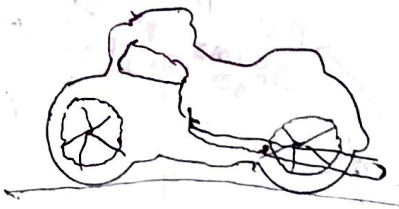
X-NOR:



$Y = \overline{A \oplus B}$

$= \bar{A}\bar{B} + AB$

ROM: (Read Only Memory)



input (Mem Add)		Output (Contents of Mem)		
A_1	A_0	Y_2	Y_1	Y_0
0	0	0	1	1
0	1	0	0	1
1	0	0	0	0
1	1	1	0	0

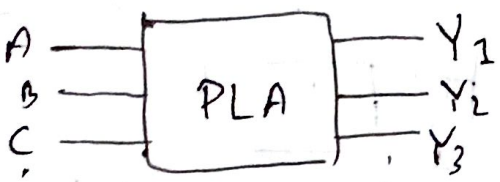
input → address of mem location

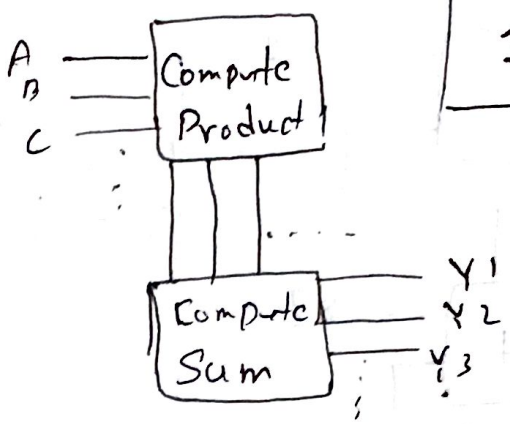
output → contact of that mem. ~~location~~

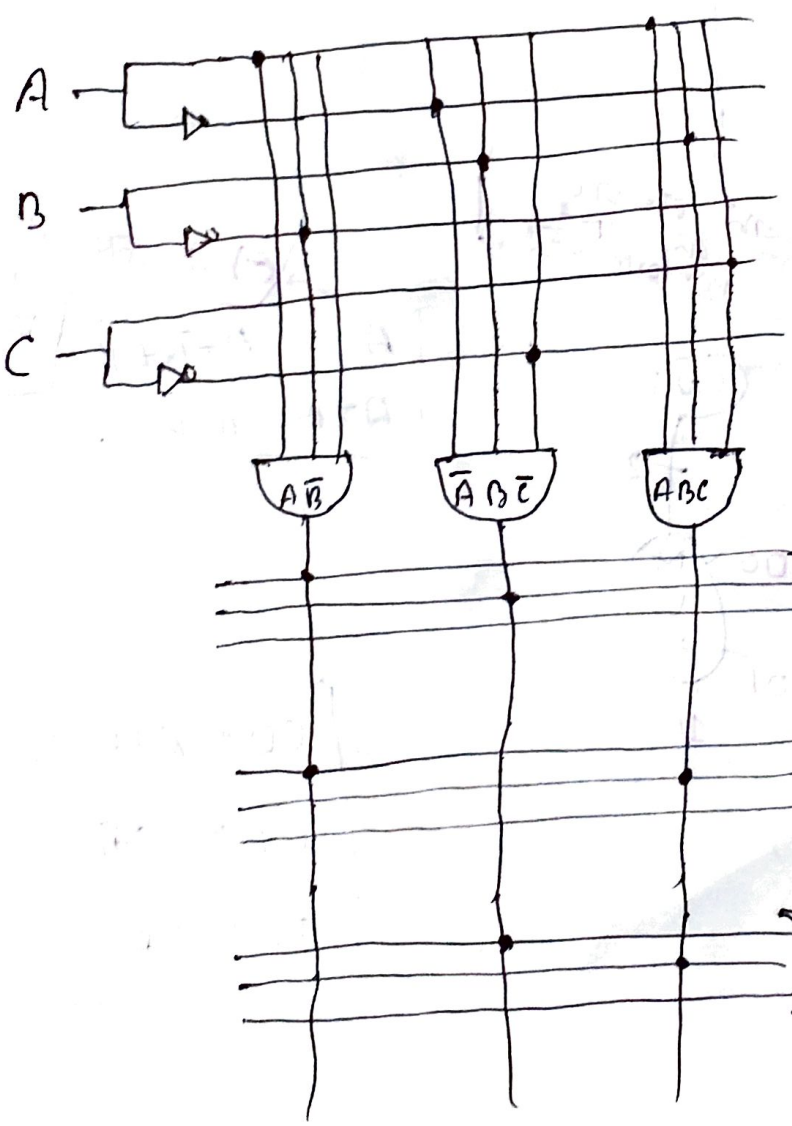
Programmable Logic Array (PLA)

- A circuit.
- SOP
- Hardware

1 byte = 8 bit.
1 KB = 2^{10} bytes.
1 MB = 2^{20} bytes
1 GB = 2^{30} bytes
1 TB = 2^{40} bytes







$Y_1 = A\bar{B} + \bar{A}B\bar{C}$

$Y_2 = A\bar{B} + ABC$

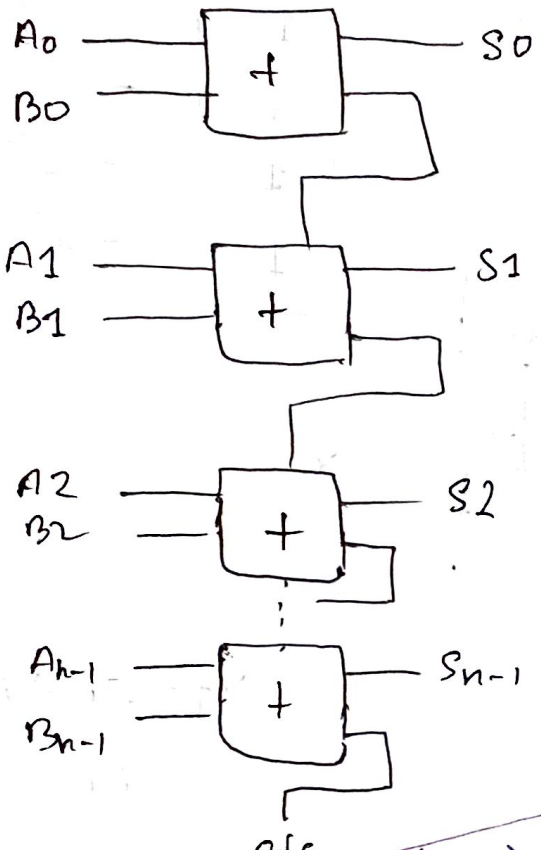
$Y_3 = \bar{A}B\bar{C} + ABC$

$Y1 = A\bar{B} + \bar{A}B\bar{C}$

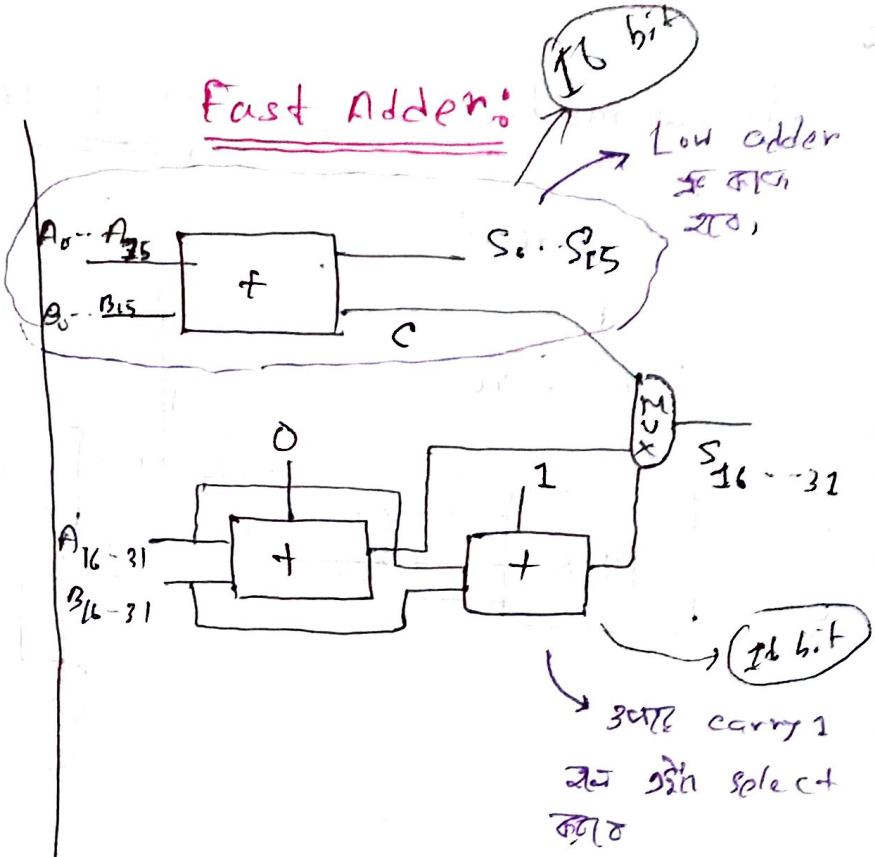
$Y2 = A\bar{B} + ABC$

$Y3 = \bar{A}B\bar{C} + ABC$

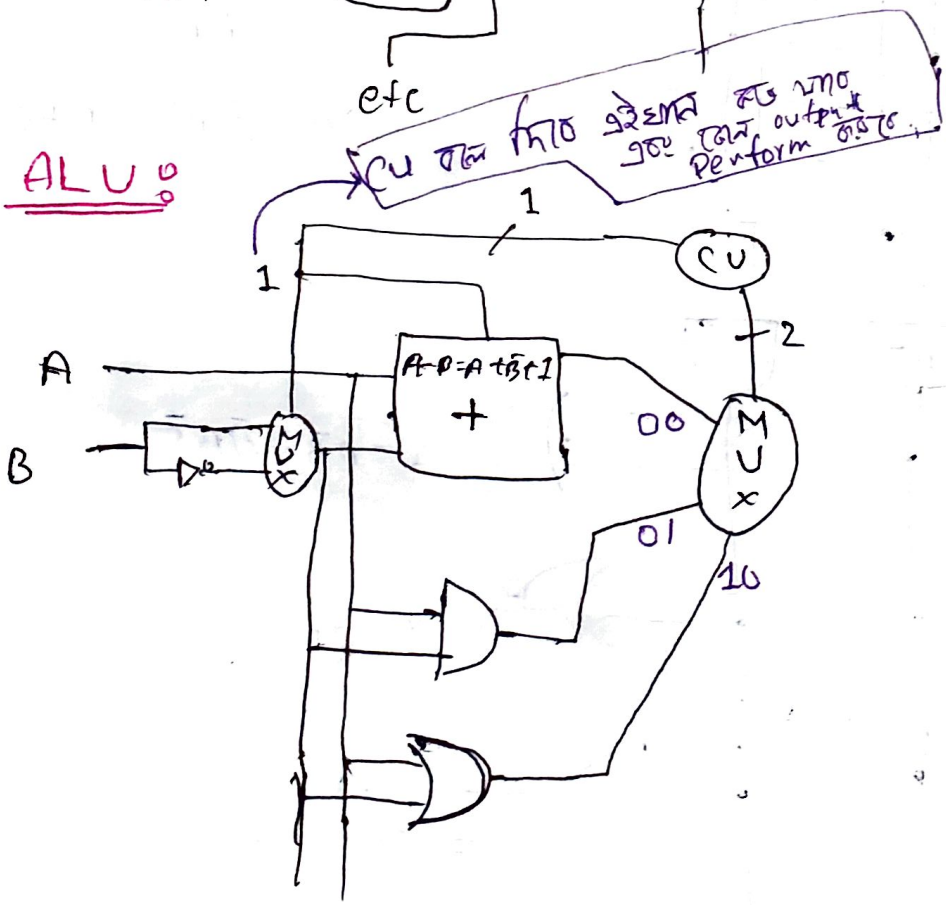
Ripple Carry Adder:



Fast Adder:



ALU:



(-) এর জন্য Adder কিভাবে কাজ করবে

$A - B = A + \bar{B} + 1$

$A + B = A + B$

00 : Add/Sub

01 : AND

10 : OR

Encoder:

- A combinational logical circuit
- Output specify which set of inputs has occured.

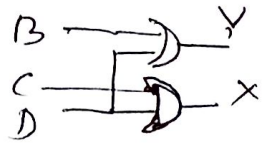
input $= 2^n$
output $= n$

2^2 ④
②

A	B	C	D	X	Y
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

কোন input দিলে কোন output পাব।

$Y = B + D$
$X = C + D$



Decoder:

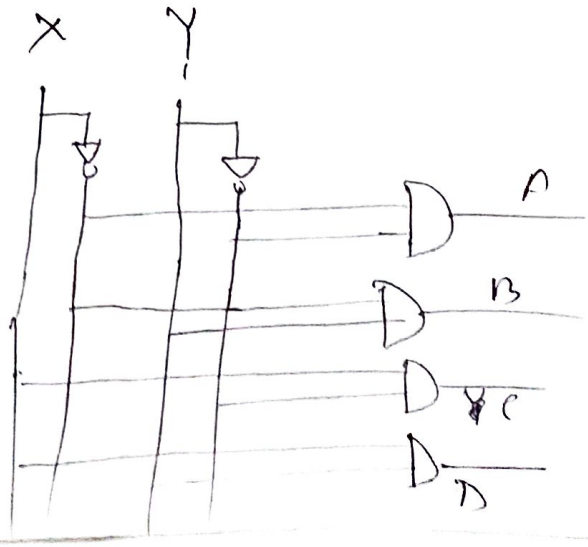
- takes a code
- turns on one of several output wires.

input $= n$
output $= 2^n$

X	Y	A	B	C	D
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

কোন input দিলে কোন output পাব।

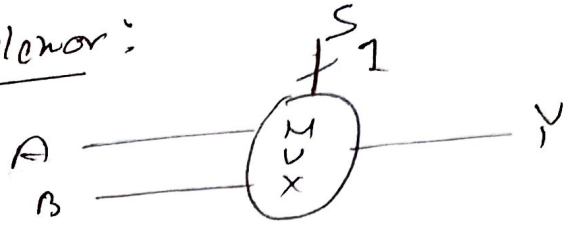
$A = \bar{X}\bar{Y}$
$B = \bar{X}Y$
$C = X\bar{Y}$
$D = XY$



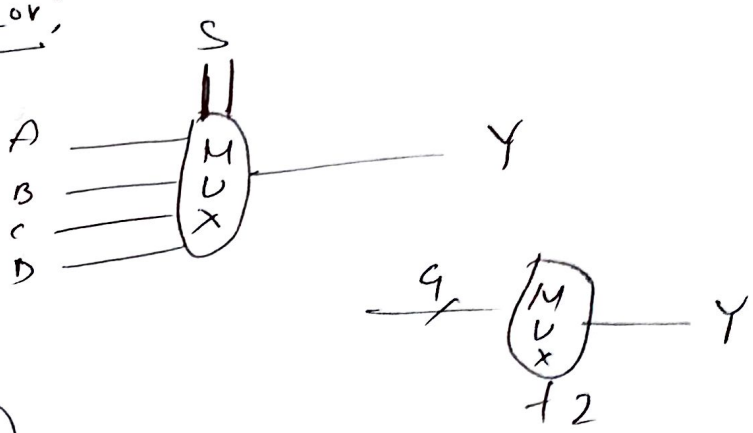
Multiplexer

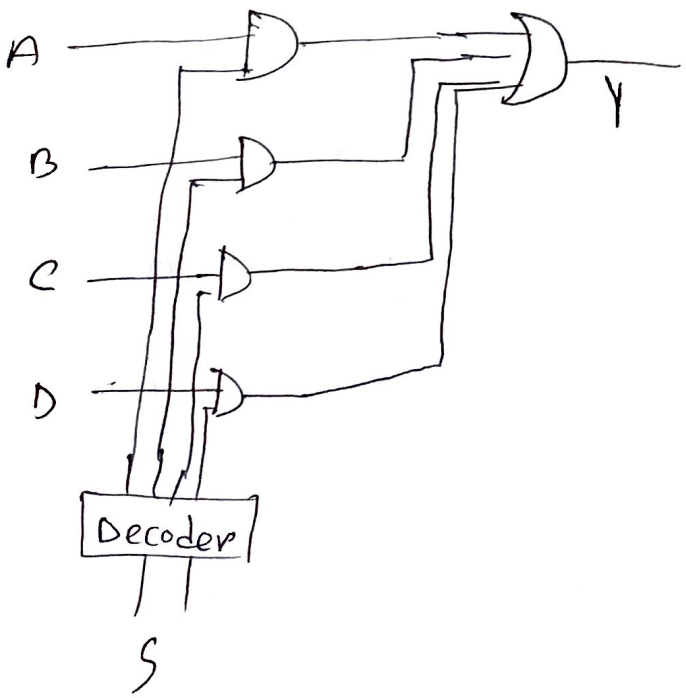
- used to select [illegible] of possible input var.
- take n input & choose among of them.

One bit multiplexer:



two bit multiplexer:



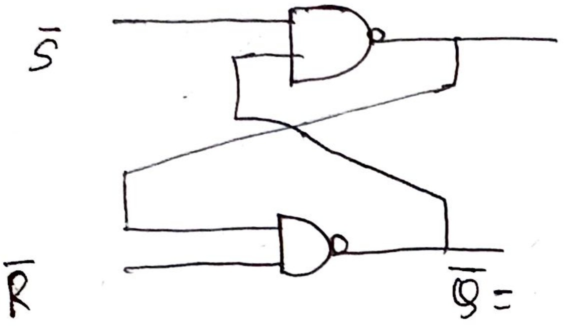


Sequential 1

20/05/19

RS flip-flop :

Nand Latch :



$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$
0	0	1	1 (forbidden)
0	1	1	0 (set)
1	0	0	1 (reset)
1	1	0	1 (Unchanged)

কোনো সময়ে '0' input এ আসতে দিনা

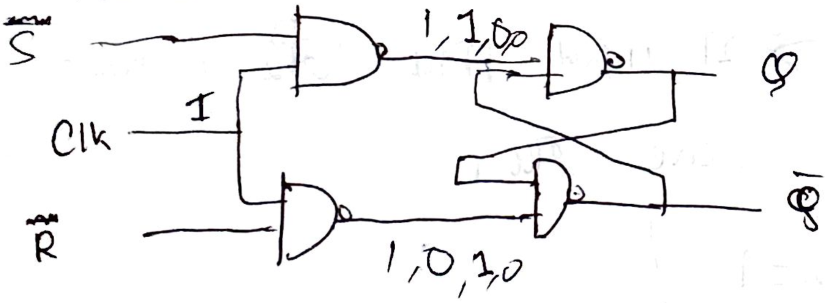
পূর্ববর্তী output, বর্তমানের input হবে এবং ~~প্রভাব~~ প্রভাব বিস্তার করবে।

NAND Gate :

I_1	I_2	Output
0	0	1
0	1	1
1	0	1
1	1	0

S R
~~0 0~~ 0 0
1, 1

SR flip flop :



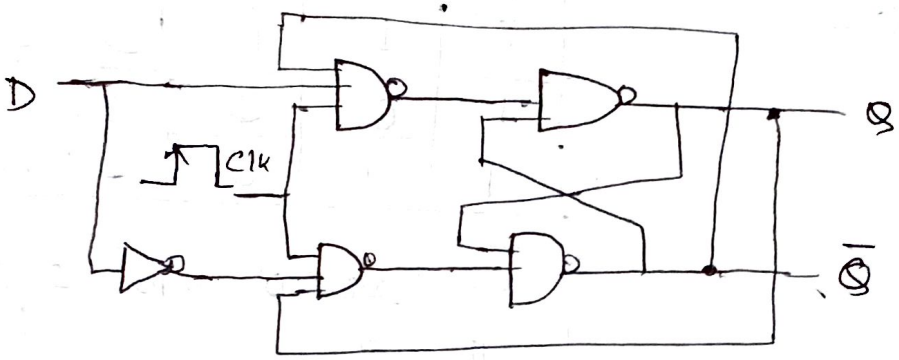
S R (0, 0)

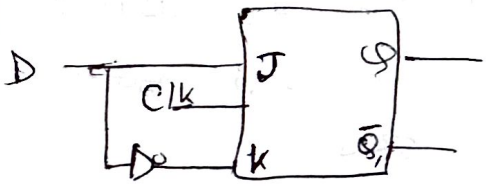
S	R	Clk	Q
0	0	↑	unchanged
0	1	↑	0 (reset)
1	0	↑	1 (set)
1	1	↑	forbidden.

↑ টেম্পো ওপরের দিকে

D-flip flop:

Data flip flop, যা ডাটা ঠিক store করে রাখে,





J	K	clk	Q
0	0	↑	আগেরটাই
0	1	↑	0
1	0	↑	1
1	1	↑	Toggle

D	clk	Q
0	↑	0
1	↑	1

→ যা input দিবে, তাই output এ save হবে।

D=0 মানে J=0, K=1

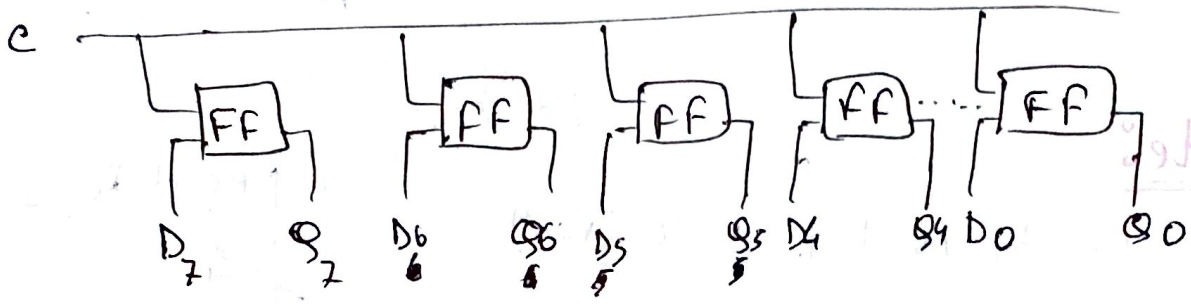
D=1 মানে, J=1, K=0

T	clk	Q
0	↑	আগেরটা
1	↑	toggle

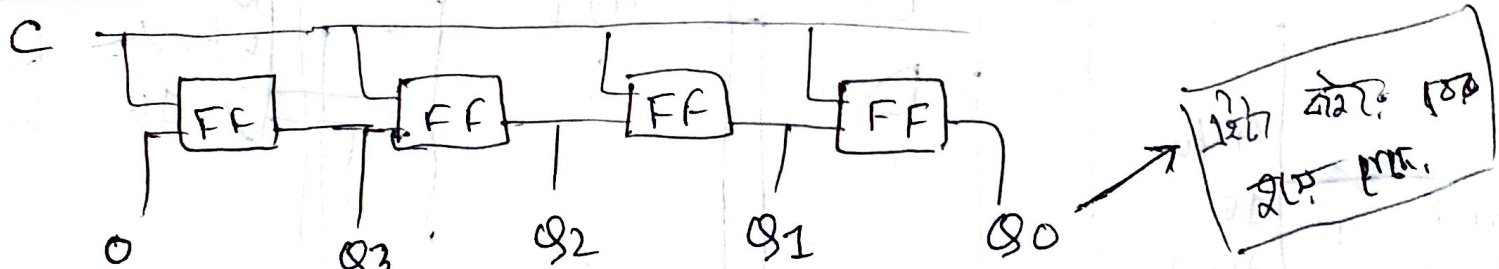
Register:

- A set of flip flop
- Holds the values of some var

8 bit Reg:

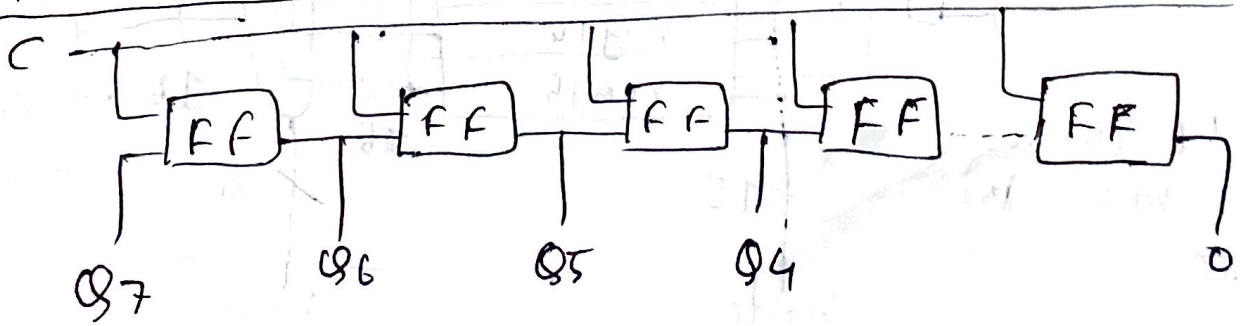


Shift Reg: 4 bit Right shift:

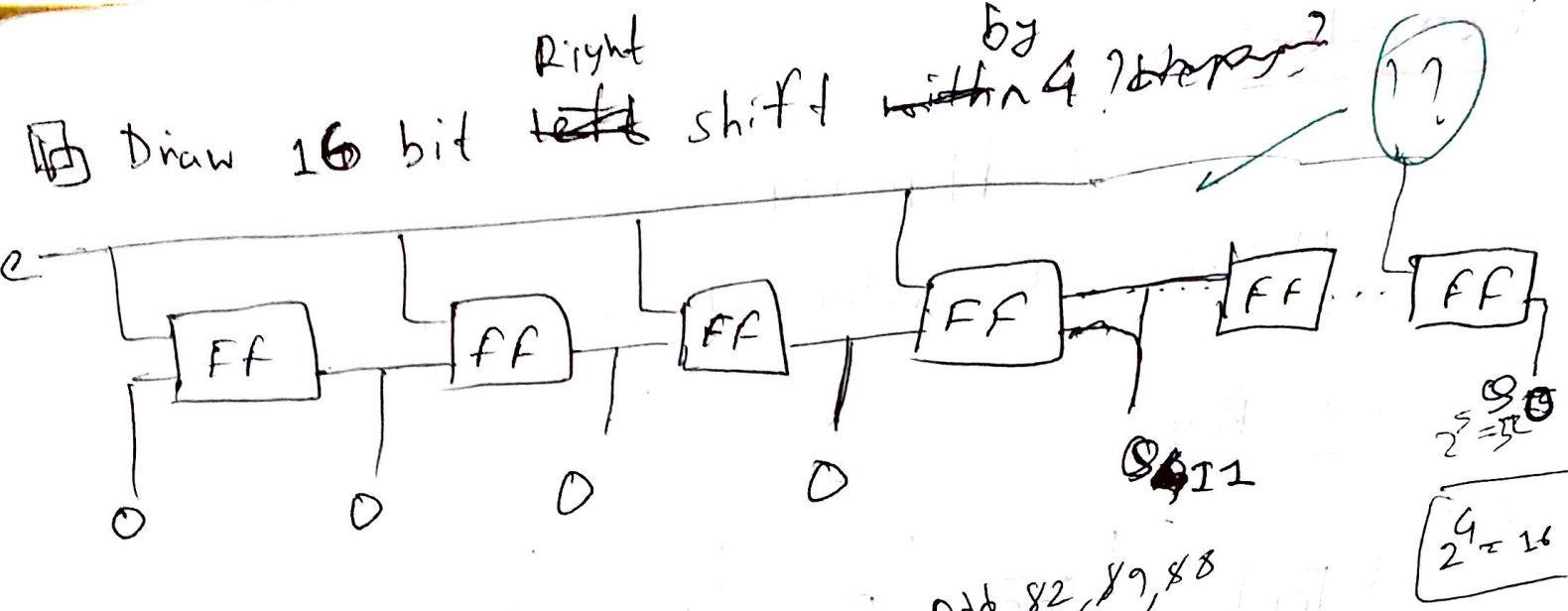


~~4 bit right shift~~

DRAW 8 bit left shift ??



Draw 16 bit ~~left~~ Right shift by ~~within~~ 4 ?



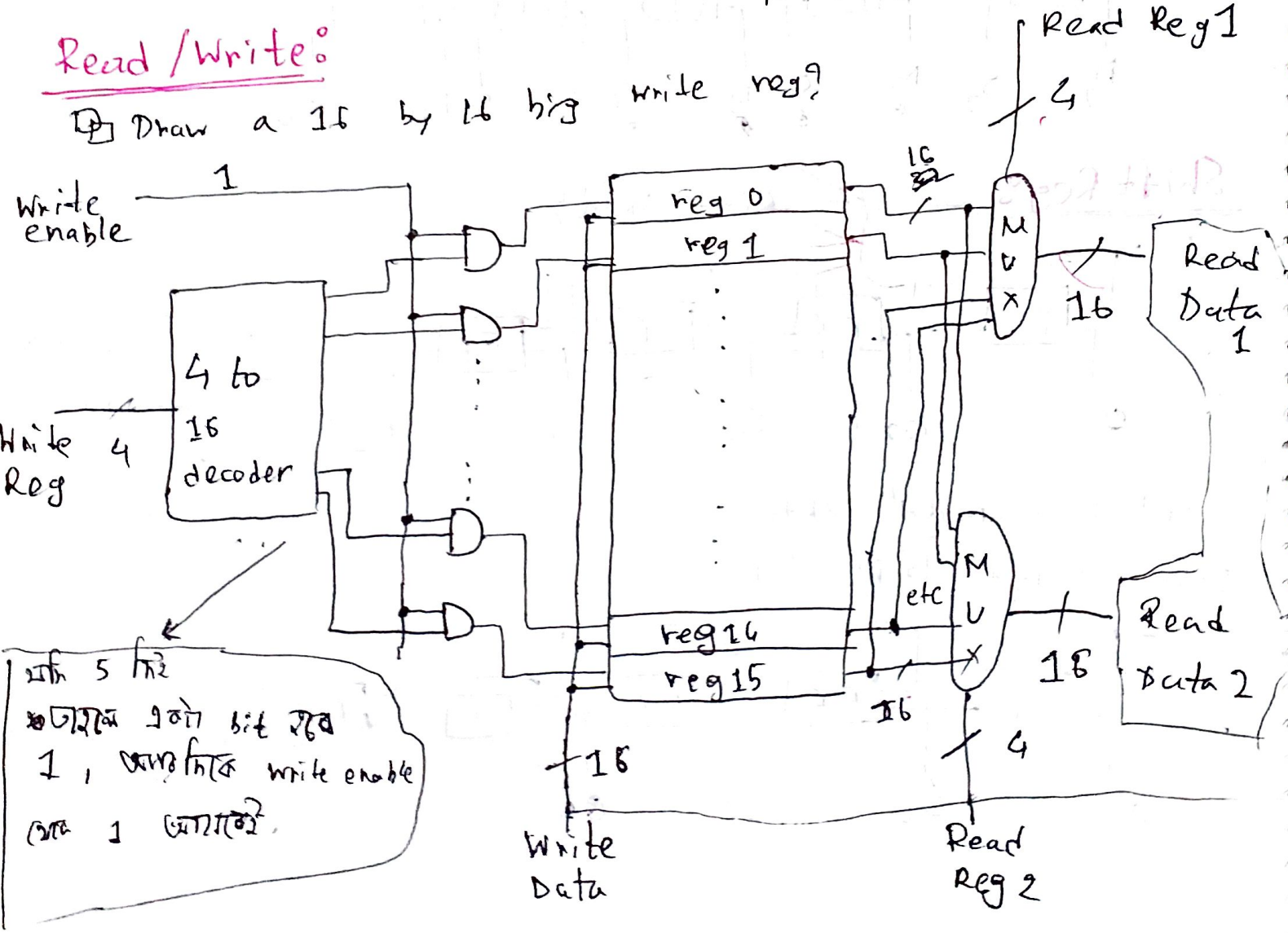
$2^5 = 32$

$2^4 = 16$

Add $2, $9, $8

Read / Write:

Draw a 16 by 16 bit write reg?



CSE-340

Chap 1 (ACH)

23/05/19

Laptop & Desktop:

- one machine
- one user
- at a time.

Server:

- one machine
- multiple user
- at a time.

Embeded Computer:

- Build for specific reason
- particular কোন কাজের জন্য use করা হয়।

Multi Core:

- Many CPU
- ~~One Core~~ single chip

Single Core:

- One CPU
- One ~~Core~~ Chip

Many Core:

- Many CPU
- Many physical ~~Core~~ chip

Example: Cloud server.

Embeded Core:

- Design a up to do a particular work.

Cloud:

- server বিভিন্ন জায়গায়
- use মাধ্যমে করতে পারবে,
- need internet.

Computers are pervasive:

- এখন com carry করা easy,
- land phone যেতে use হয় না,

Understanding Performance:

Algorithm.

- Determine num of operations executed.

Programming lan, compiler,

Architecture:

- Determine num of machine instructions executed per operation

Processor & mem system:

- Determine how fast instructions are executed

I/O system:

- Determine how fast I/O operations are execut

acronym

Below your Program:

Application Software:

- Written in high level lan.

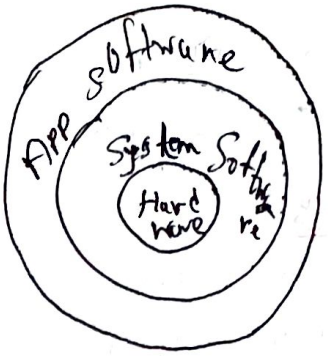
System Software:

Compiler: translate HLL code to machine code.

OS: managing mem & storage scheduling tasks.

Hardware:

Processor, mem, I/O controllers



Levels of programming Code:

(i) High level lan

(ii) Assembly lan: close to machine lan.

(iii) Hardware representation:

- Binary digits.
- Encoded instructions & data.

PMD: Personal mob device: small wireless device to connect the internet

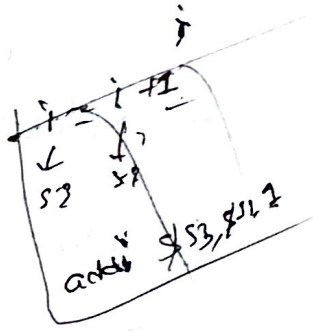
Cloud Computing: refers to large collection of servers that provide services over the internet

SaaS: Software as a service: delivers s/w & data as a service over the internet.

Web Search, Social Networking

Components of Computers:

- Same components for all kind of computer.
  - Desktop, server, embedded
- Input/output include
  - User Interface devices
    display, keyboard, mouse
  - Storage device.
    Hard Disk, CD/DVD, flash
  - Network adapters
    for communication with other computers



Inside the Processor (CPU):

Datapath: performs operation on data.

Control: seq of datapath, memory.

Cache Mem: Small fast SRAM for immediate access to data.

Abstractions:

- Abstraction helps us deal with @ complexity.
- Instruction set architecture (ISA)

- Application binary interface

- Implementation.

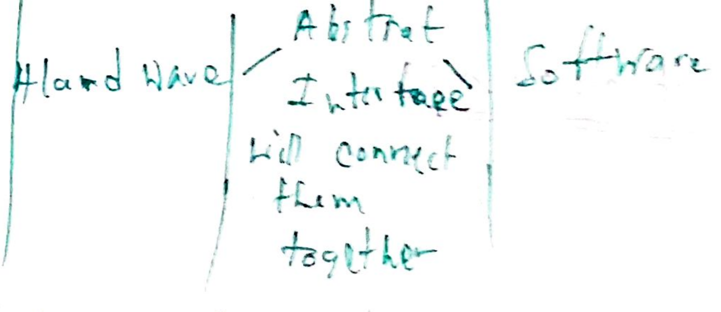
What is Computer Architecture?

- The science and art of
- designing: নতুন hardware কিভাবে তৈরি design করতে হবে।
- Selecting: Component select করতে হবে।
- Connect: যেগুলো আলাদা hardware কে যোগ.

- Designin the hardware / software interface to create a computing system
- that meets functional ~~ga~~ goals
- performance
- Cost

- Hardware programming Lan is very important.

Interface:



- Same problem ‸less competition time/ execution time যার কম স্টেপ সেটা compilers.
- ~~যার~~ - less instruction সেরা compiler.
- যদি C programming এ যে compiler আছে ~~এটা নিজেই~~ এটা ‸নিজেই hardware programming lan. So তার জন্য assembly compiler দিতে হয় না.

ISA:

- An **Abstract interface**
- between **hardware** & **software** (lowest level) of a machine
- that encompasses all the information necessary.
- to write a **machine language program**
- that will run correctly, including instructions reg, mem access, I/O, so on.

Data Path:

- A data path is a set of **functional units**
- that carry out **data processing.**
- Data paths along with Control unit, make up the CPU of a computer system.
- A larger datapath can only be ~~configured once~~. created by **joining** more then one together

Clock Cycle = Instruction Count × Cycle per Instruction

Instruction এর জন্য ~~Count~~ & এত সময় লাগে।

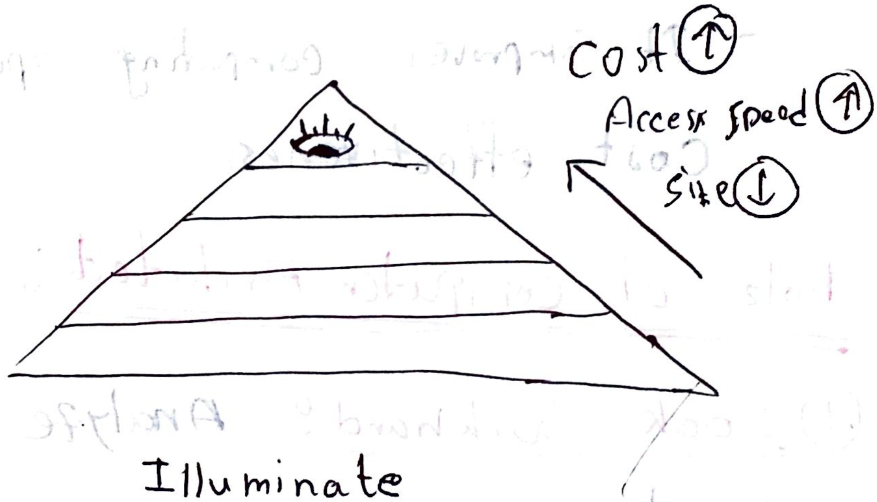
প্রতি instruction এর জন্য কতো cycle লাগবে

CSE-340 31/05/19

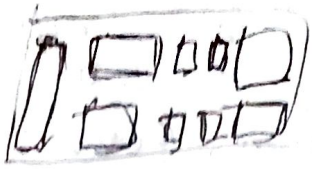
Lecture 1 (ACH)

Computer Mem Hierarchy:

- CPU
- reg
- ~~Cach~~
- ~~Chach~~
- cache Mem
- RAM
- Virtual Mem
- Hard Disk



Multi Processor:



- ~~it is a tightly coupled computer system.~~
- Multi Core
- A computer system  2 core
- two or more CPU
- each one shares the common main mem & I/O device
- It helps in simultaneous processing of programs.
- It improves computing speed, performance and cost effectiveness

Role of computer Architect:

(1) Look Backward: Analyze and evaluate the past.

(2) Look forward: Be a dreamer. Be creative.

(3) Look up: Understand important problems and their nature.

(4) Look down: Understand the capabilities of the underlying technology. যাতে design ~~redready~~ এমন না, technology support করে না.

$7 $5 $3

Response Time:

> ভাত রান্না করতে কেমন চাল নিয়ে আসিলে চুলে, চুলা রান্না করতে দেয়, শুরু হতে শেষ পর্যন্ত যে time লাগে তাই Response time

- Total time taken to complete a task/instruction
- start time to complete time.
- bed থেকে রান্না করা পর্যন্ত যে time.
- If you have 2 processor, one taken 5 s another one taken 3 s. Then 3 s is good one
- It is performance indicator.
- কম হলে ভালো।

Throughput:

- How many bits it can process in
- Per unit time. is throughput of a transmeter.
- How many bits it can recieve per unit time is the throughput of a reciever.
- Throughput can be the throughput of a channel/transmeter/reciever.
- Throughput of a processor: How many task/instruction you can process per unit of time.
- বেশি হলে ভালো,

① Replace a faster version?

– Response time কমে যাবে এবং throughput বেড়ে যাবে।

② Adding additional processor to a system?

– Throughput বাড়বে ~~এবং~~ কিন্তু response time same থাকবে। ~~টিকিট ম্যান একজন এখন টিকিট~~

টিকিট ম্যান আছে ২০ টা। ২টা মানুষ ~~ম্যান~~ থেকে যায় এটা। So ২ টা মানুষ যেন তাড়াতাড়ি শেষ হয়ে যায়, That means per unit of time এ বেশি কাজ হচ্ছে, আবার response time separately count করতে হয়, ২ জন মানুষের মাঝে ~~টিকেট~~ যেমন শেষ করতে same time লাগবে।

$$\text{performance} = \frac{1}{\text{Execution time}}$$

যত কম execution time তত বেশি performance

① X is faster then Y. Now calculate how much slow/fast are the execution time & Performance between each other?

- X is faster means,

$$\text{performance}_X > \text{performance}_Y$$

$$\Rightarrow \frac{1}{\text{execution time}_X} > \frac{1}{\text{execution time}_Y}$$

$$\Rightarrow \text{execution time}_Y > \text{execution time}_X$$

(2) X is n times faster then Y ?

- $\text{performance}_X = n \times \text{performance}_Y$.

$$\Rightarrow \frac{\text{performance}_X}{\text{performance}_Y} = n.$$

$$\Rightarrow \frac{\text{execution time}_Y}{\text{execution time}_X} = n.$$

(3) A take 10 sec & B take 15 sec how much faster A is then B?

- $\frac{\text{performance}_A}{\text{performace}_B}$ (faster) $= \frac{\text{execution time}_B}{\text{execution time}_A} = n$

$$\Rightarrow \frac{15}{10} = n$$

$$\Rightarrow n = 1.5$$

or,

$\text{performace}_A = \frac{1}{10}$

$\text{performace}_B = \frac{1}{15}$

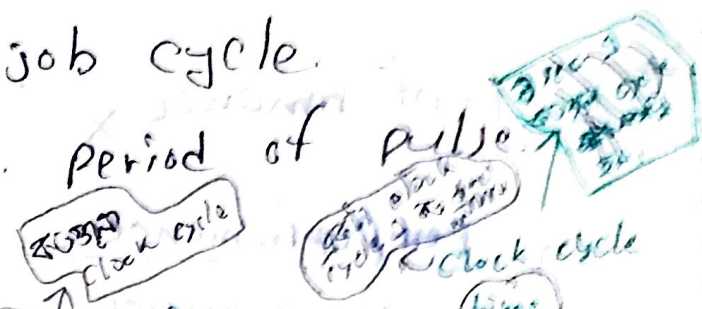
$$\text{ratio} = \frac{\frac{1}{10}}{\frac{1}{15}} = \frac{15}{10} \times \frac{15}{1} = 1.5.$$

Execution Time: - The time taken into the CPU. [CPU time]

- The time taken by ALU.
- ভাত রান্নার জন্য main time হচ্ছে, কতটা চাল সেদ্ধ হচ্ছে তা, চাল মাপা, ঢাকনা এইসব count এ ধরছি না।

Clock Cycle: - Unit of task

- Job is measured by job cycle.
- Time taken for one period of pulse.



CPU time / CPU Execution time for a program = Num of cycle for a program CPU takes × Per cycle time

Per cycle time → পর্যায়কাল,

$$\text{CPU Execution time for a program} = \frac{\text{CPU clock cycle for a program}}{\text{Clock rate}}$$

Clock rate → 1 sec এ কতগুলো clock cycle সেটার মান কত।

কম্পাঙ্ক → $T = \frac{1}{f}$

(1) Program runs 10sec on computer which has 2GHz clock. Program B run this program if in 6 sec B needs 1.2 time clock cycle then A. what is the clock rate of B?

$$\text{CPU time}_A = \frac{\text{num of clock cycle}}{\text{clock rate}}$$

$$\text{num of clock cycle} = \text{CPU time} \times \text{clock rate}$$

$$= 10 \times 2 \times 10^9$$

Again

$$\text{CPU time}_B = \frac{\text{num of clock cycle for a program}}{\text{clock rate}}$$

$$\text{Clock rate} = \frac{\text{num of clock cycle}}{\text{CPU time}_B}$$

$$= \frac{1.2 \times 10 \times 10^9}{6}$$

$$= 4 \text{ GHz}.$$

Ans

How many clock cycle
CPU takes to execute a program.

CPU num of clock Cycle = Instructions for a program × Avrg cycle per Instruction (CPI)

Performance equation

program → code → instruction

num of instruction taken by program
add
sub
sw

Avrg cycle per Instruction (CPI) → এক একটা instruction complete হতে এক এক time লাগে, So, Avrg নিব,

(1) Computer A has a clock cycle time of 250 ps & CPI of 2.0 for some program, Computer B has a clocy cycle time of 500 ps and a CPI of 1.2 for the same program. faster which one?

—

CPU clock Cycle time = num of Instruction × CPI

Let
num of Instruction for both = I.

∴ CPU clock ~~cly~~ cycle $time_A$ = I × ~~250~~ 2.0

∴ CPU clock cycle $time_B$ = I × ~~500~~ 1.2

CPU execution time$_A$ = num of clock cycle X per clock * cycle time.

$= I \times 2.0 \times 250$

$= 500$ Ips.

CPU execution time$_B$ = num of clock cycle x per clock cycle time

$= I \times 1.2 \times 500.$

$= 600 \times I$ ps.

$$\frac{\text{CPU Performance}_A}{\text{CPU Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A}$$

$$= \frac{600 \times I}{500 \times I}$$

$= 1.2$

CSE-340 14/06/19

Lecture-01 (ACH)

CPU time = CPU clock cycle for a program × clock cycle time

CPU clock cycle for a program = num of instruction for program × CPI

① Sequence 1 has (2+1+2) = 5 instruction count.

Sequence 2 has (4+1+1) = 6 instruction count.

CPU clock cycle of A = 2 × 1 = 2

" " " of B = 1 × 2 = 2

" " " " C = 2 × 3 = 6.

∴ CPU clock cycle of code sequence, 1 = 2+2+6 = 10

CPU clock cycle of code sequence, 2 = (4×1) + (1×2) + (1×3) = 9.

So, seq 2 is faster.

$$\text{CPI of seq. 1} = \frac{\text{CPU clock cycle for instruction}}{\text{Instruction count}}$$

$$= \frac{10}{5}$$

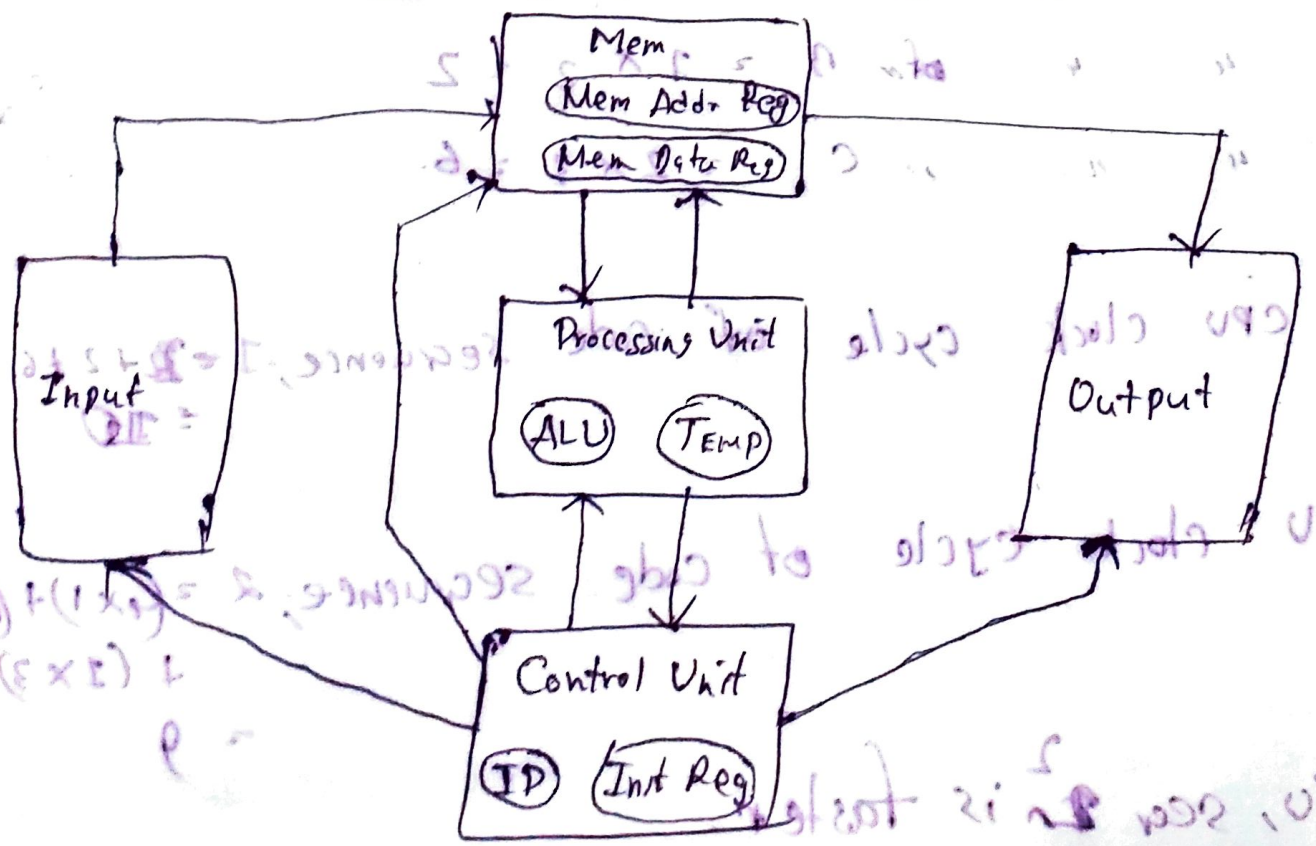
$$= 2$$

$$\text{CPI of seq. 2} = \frac{\text{CPU clock cycle for instruction}}{\text{Instruction count}}$$

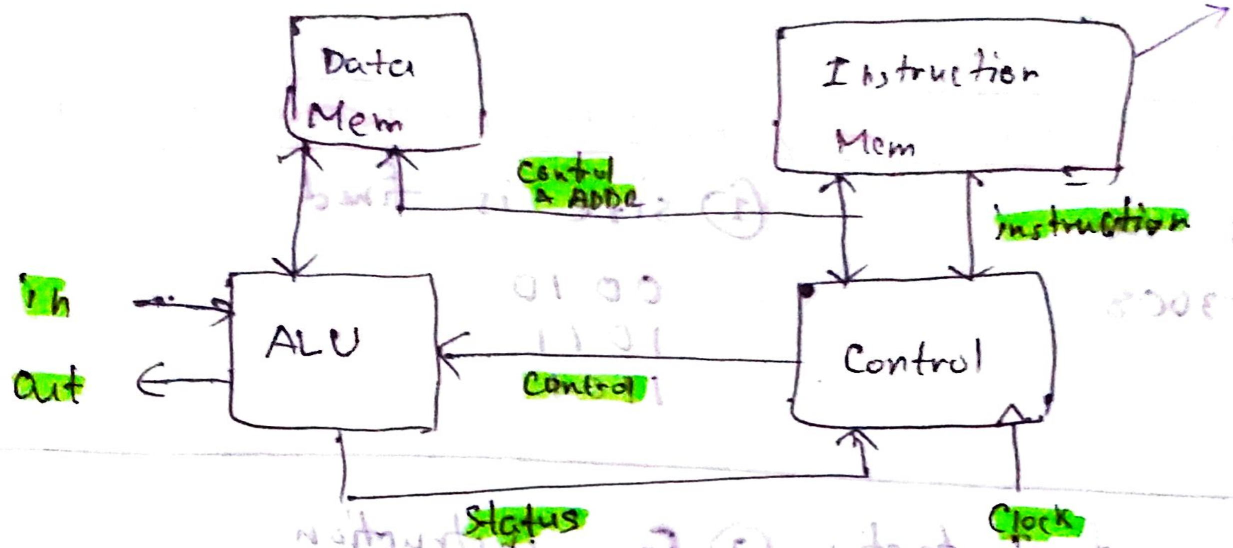
$$= \frac{9}{6}$$

$$= 1.5$$

The Von Neumann Model:



The Harvard Architecture:



Difference Between the Von Neumann & Harvard Model:

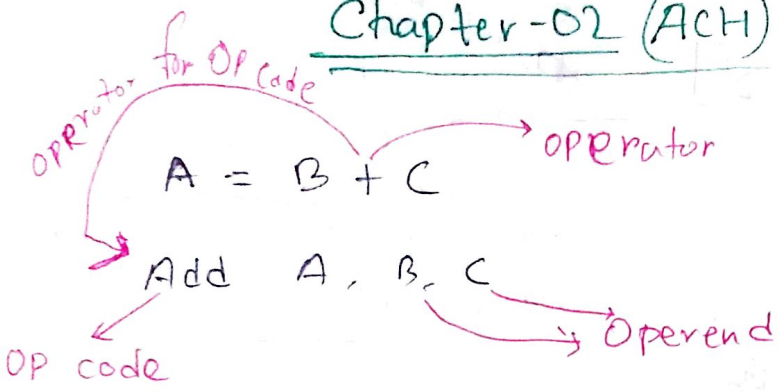
Von Neumann	Harvard Model
(i) Mem add is not seperated	(i) Mem has saperated.
(ii) ~~Complex~~ Slower. coz সবাই একটা bus use করে।	(ii) ~~Simple~~ faster. coz instruction fetch and data acess do not in same way.
(iii) একটা দিয়ে সবকিছু করা যায়।	(iii) কাজ চালায় জন্য একটা, মেমরি রাখার জন্য একটা. saperated
(iv) একটা mem নষ্ট হয়ে গেলে সব শেষ।	(iv) একটা mem নষ্ট হলে অন্যটা আছে।
(v) Either read করবে or write করবে, coz instructions and data use same bus system.	(v) Read & write একসাথে করতে পারবে।
(vi) Mem is same.	(vi) In some system instruction can be stored in read only mem while data mem requires read-write mem.

Difference Between RISK & CISC:

CISC (Complex Instruction Set Computing)	RISC (Reduced Instruction set computing)
① size is variable 00181630 08 10110 121314	① size is fixed 0010 1011 1100
② Large num of Instruction - Many addressing Mode	② few Instruction
③ Use more RAM and less Reg.	③ Use more reg and less RAM
④ Need better hardware & powerful processing	④ Less costly hardware would also work.
⑤ MOV AX, 5 MOV AX, BX, CX MUL CX	⑤ 16301180 → সবার id length সমান, ⑥ phone num → সবার same length.
⑤ Harvard Model	⑥ Von Neumann Model.

Chapter-02 (ACH)

14/06/16



4 byte

32 bit data is called = word

2^5 = 32 bit

1 byte = 8 bit

The MIPS Instruction Set:

32 bit MIPS instruction

→ all instruction length is 32 bit.

→ all reg also 32 bit.

① $f = (g + h) - (i + j)$

```
add  t0, g, h
add  t1, i, j
sub  f, t0, t1
```

→

```
add  $t0, $1, $2
add  $t1, $3, $4
sub  $0, $t0, $t1
```

lw R1, 45(R2)

R1 = mem(R2)

lw means:
load data to reg from mem

sw means:
store data to mem from reg

② $g = h + A[8]$

$1 ← g, $2 ← h, $s3 ← A

A → Data Mem. so Data mem থেকে reg তে আনতে,

offset / base add of A

lw $t0, 32($s3)

⇒ t0 = mem[s3 + 32]

base add, offset

add $1, $2, $t0

③ A[12] = h + A[8]

base reg $3 ; h → $2 ; A[8] → offset, base reg → $3

⇒ lw $t0, 32($3)

add $t0, $2, $t0

sw $t0, 48($3) → Mem[$3+48] = t0.

d (under $t0), s (under 48($3))

We need four byte for each word, so an offset of 10 words i.e. an offset of 40 bytes.

$zero reg means = 0
এর কোন value নাই,
0 কে কোন কিছুর সাথে
add করে check করে

add $t2, $s1, $zero

$s = saved reg.
$t = temp.
$gp = global pointer.
$fp = frame pointer
$sp = stack pointer
$a = Argument Reg
$v = Value Reg

```
for(int i=0; i<10; i++){        → $gp
    for(int j=0; j<i; j++)}     → $fp
        s.o.p("Depression");
    }
}
```

RAM

8 bit	A[0]
8 bit	A[0]
8 bit	A[0]
8 bit	A[0]
8 bit	A[1]
8 bit	A[1]
8 bit	A[1]
8 bit	A[1]

④ a = b + 5

addi $0, $1, 5

op	rs	rt	immediate
001000	00001	00000	0000000000000101

lw $7, 300($3)

op → lw, rt → $7, Const → 300, rs → $3

300 এর সাথে $3 এর [illegible] যোগ করে [illegible] তাকে immediate ও

beq $3, $2, level

op	rt	rd	Immediate

A[8] → offset, base

32($3) → base, offset } mem location add [illegible]. Form where you want to read and write

if (rs == rt)

→ level 1

না হলে level 2

⇒ beq rs, rt, level 1 (tan)

bne rs, rt, level 2

~~beq rs, rt, L1~~

Level 1:

Tan;

if(i==j)
a[i]=h+g+K[9]
else
a[i]=h+g-K[7]

4 bit sign

equivalent decimal num to binary

bits

1110

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

add $s1, $s2, $s3.

sub $s4

bne $s0, $s1, else

else: sub $s0, $s1

a=b+c-d

$t0

add $s0, $s1, $s3

Quiz-2

Wed

chapter-02

38

beq rs, rt, if

if: add $s0, $s1, $s2

sub $s0, $s1, $s2

Jump Exit.

mul $s0, $s1, 8

ix8 [$0]

while (save [i] == k) i+=1

$s6 $s7

i = i+1

4 = $(100)_2$

sll $t1, $s3, 2

২ ঘর shift করে মানে 4 দিয়ে গুণ করা

add $t1, $t1, $s6

lw $t0, 0($t1)

lw $t0, 0($t1)

offset পরিবর্তনশীল হতে পারে base এর মধ্যে.

lw $t0, $t1($s6)

beq $t0, $s5, NotExit

ix(4)10

ix(100)2

Not Exit: addi $s0, $s0, 1

J Exit.

CSE-340

Lecture-02

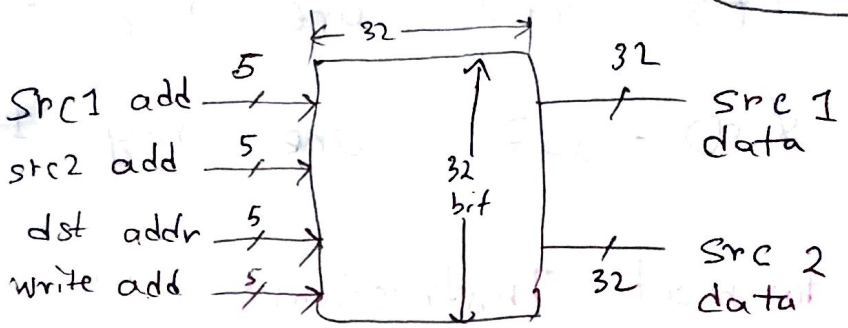
17/06/19

MIPS Register File:

- two read ports.
- one write port

সব 32 bit

Reg are faster then Mem. but reg files with more locations are slower.



Sign Extension:

ADDi $7, $3, 8 → constant যোগের 16 bit. But we need 32 bit.

- it is a hardware.
- if you put 16 bit then it will be 32 bit allways.
- sign num এর ক্ষেত্রে 2's complement হবে এবং সামনের সবগুলো 1 বসবে।
- unsign num এর ক্ষেত্রে সামনে 0 বসবে শুধু।

$(2)_{10} = (0000\ 0000\ 0000\ 0010)_2$

1's complement = 1111 1111 1111 1101

+1

1111 1111 1111 1110

$= (-2)_{10}$

Reg num:

\$t0 - \$t7 are reg. 8 - 15.

\$t8 - \$t9 are reg 24 - 25

\$s0 - \$s7 are reg 16 - 23.

sll \$t0, \$s0, 2

(save[i] = i)

R-format Instructions:

sll \$3, \$5, 7

op	rs	rt	rd	shamt	function
010010	00000	00101	00011	00111	100110

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

add \$t0, \$s1, \$s2

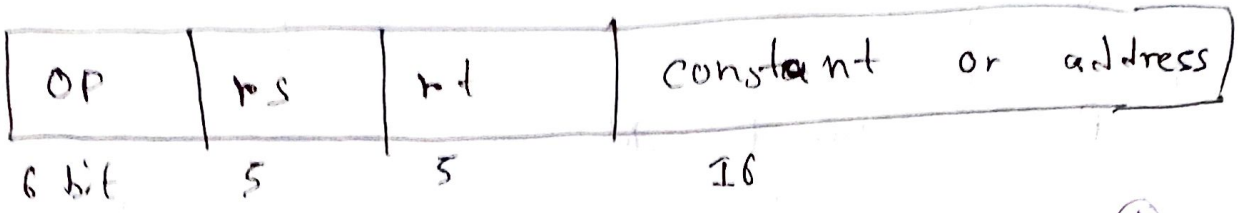
special	\$s1	\$s2	\$t0	0	add
0	17	18	8	0	32
000000	10001	10010	0100	00000	100000

$(00000010001100100100000000100000)_2 = (02324020)$

8   4   0

[illegible]

I - format Instruction:

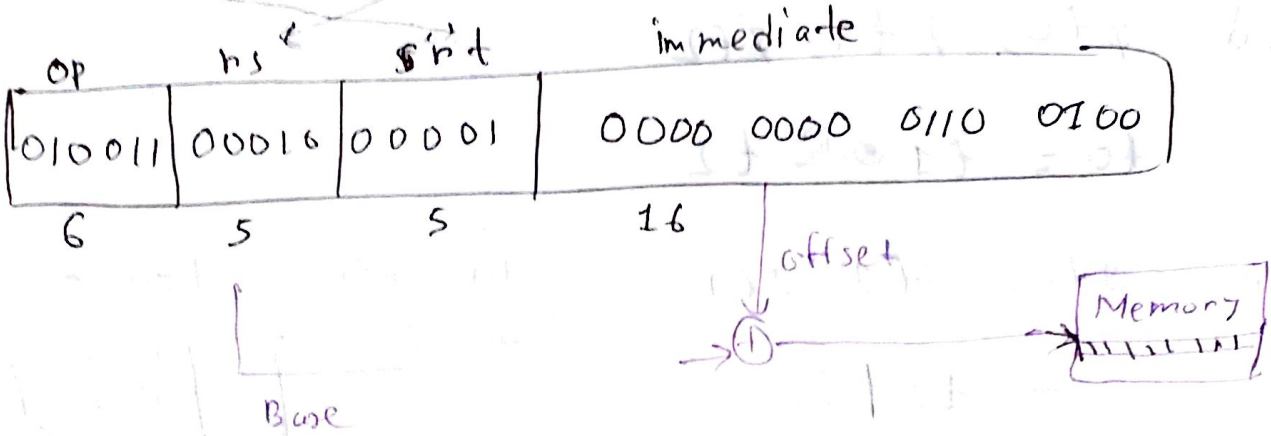


n(x)

lw $1, 100($2)

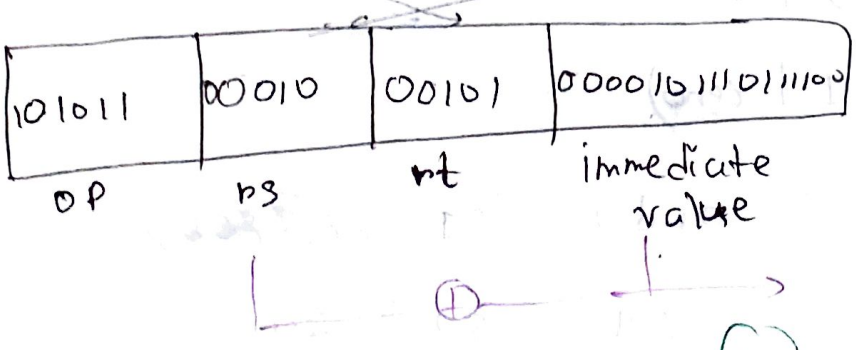
$1 = Mem[100 + $2]

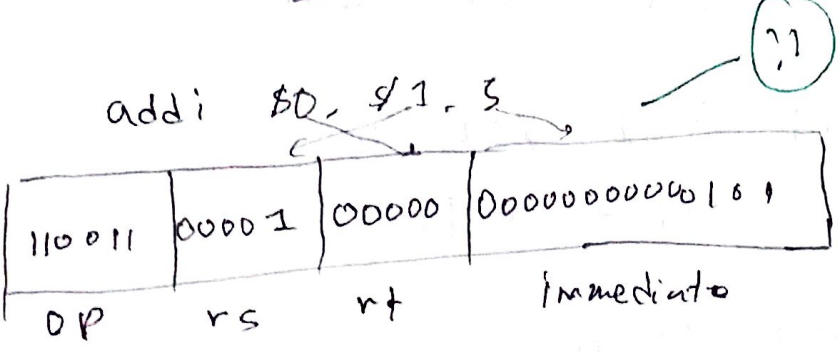
offset + base = Mem Location



sw $5, 3000($2)

Mem[$2 + 3000] = $5





lw $1, 100($2) $1 = Mem[100 + $2]
load data to reg from mem.
sw $55, 3000($2) Mem[3000 + $2] = $55
store data to mem from reg.

read

write

J-type:

6	26
op	target add

target add → jump করতে অনেক ক bit এর দরকার হয়।

> lui \$s1, 20
> \$s1 = 20 * 2^{16}

⑤ and \$t0, \$t1, \$t2

t0 = t1 & t2

⑥ or \$t0, \$t1, \$t2 → or gate এর logic table এর মত।

t0 = t1 | t2

1	0	1
1	1	1
0	0	0
0	1	1

⑦ Nor \$t0, \$t1, \$zero

t0 = ! (t1 | zero)

⑧ and immediate : andi \$s1, \$2, 20

s1 = s2 & 20

⑨ Or immediate : ori \$s1, \$s2, 20

s1 = s2 | 20.

⑩ ~~sst~~ sll \$s0, \$s1, ~~\$s2~~ 10

s0 = s1 << 10

(11) if (i == j) f = g+h
(i → s1, j → s2, f → s3, g → s4, h → s5)

else f = g−h.
(f → s3, g → s4, h → s5)

```
beq $s1, $s2, level1
  level1:
      add $s3, $s4, $s5.
    J Exit;
  sub $s3, $s4, $s5.
  Exit.
```

(12) while (save [i] == k) i += 1.
(save → $s1, i → $s2, k → $s3, i → $s4)

```
SLL $t0, $s2, 2
lw $t1, $t0 [$s1]
beq $t1, $s3, L1
L1: add $s4 = $s4 + 1
    addi $s4, $s4 + 1
Exit:
```

SLL → 2 ঘর left shift করলে 4 গুণ হয়ে যায়।

lw → $t1 = Mem[t0 + s1]

lw $t0, 48[$s1]
A [4]
16 [$s1]
Exit [$s1]

A [8]
32 [$s3]
$t0 [$s3]

CSE-340 19/06/19

Lecture-03

Overflow flag:

unsigh num: Carry and overflow same

```
  1 1 1 1
  1 1 1 1
  1 1 1 1
-----------
1 1 1 1 0
```

Carry

CF = 1
OF = 1

বলে দেয়া হয়েছে sign bit নিয়ে.

sign num: if $C_{in} \neq C_{out}$

$C_{in} = C_{out}$

```
  1 1 1 1
  1 1 1 1
-----------
1 1 1 1 0
```

Carry

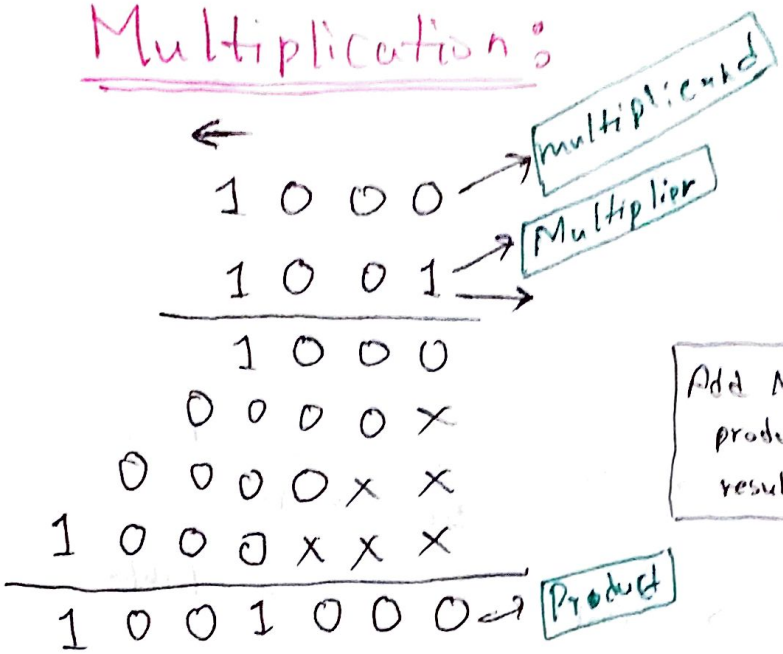
CF = 1
OF = 0

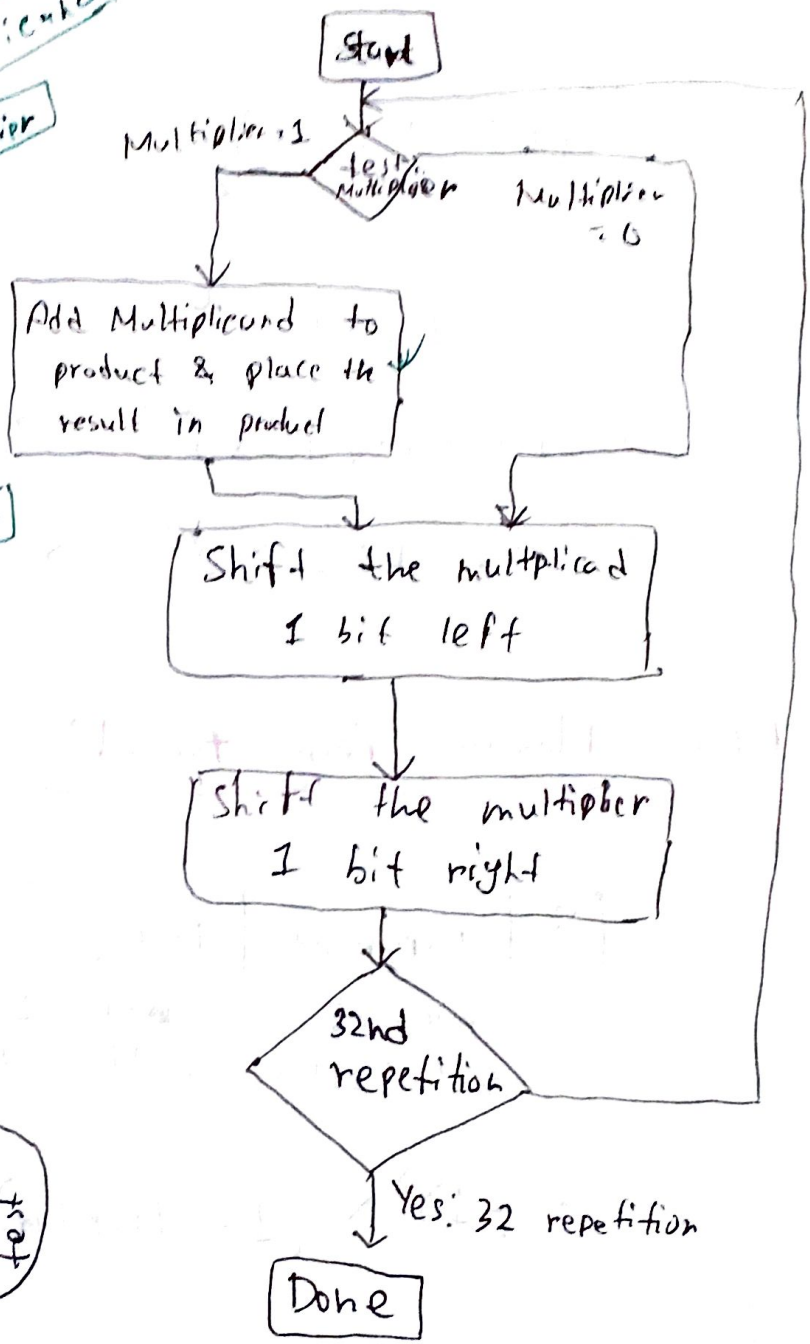
$C_{in} \neq C_{out}$

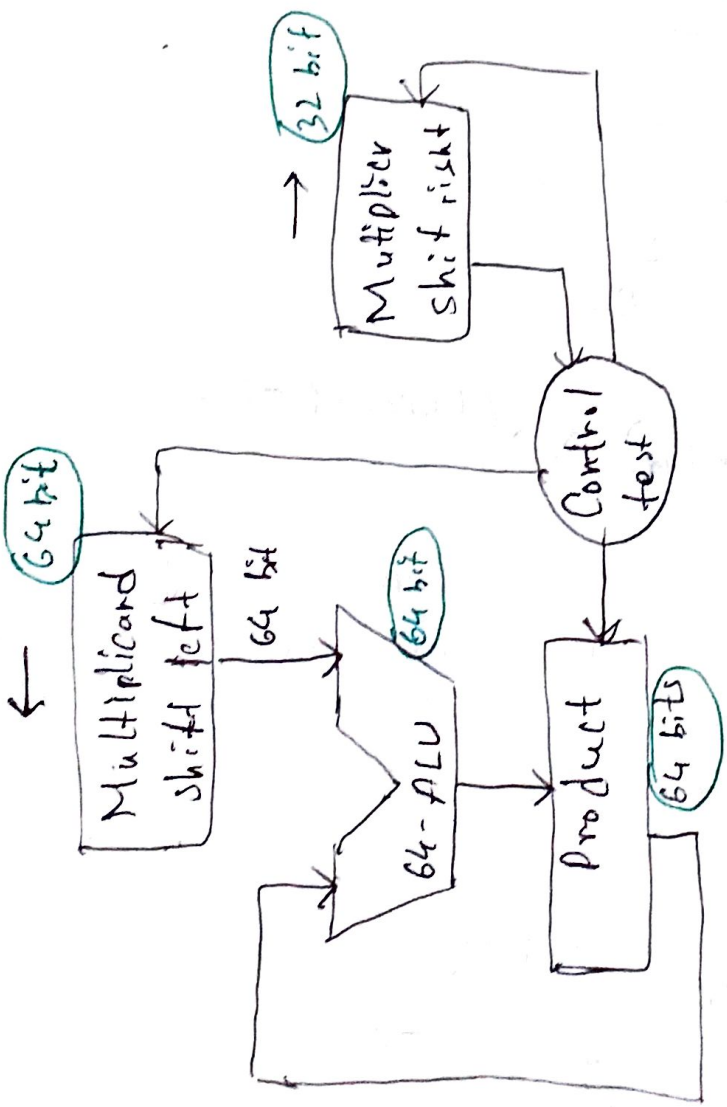
```
  0 1 1 1
  0 1 1 1
-----------
0 1 1 1 0
```

CF = 0
OF = 1

Multiplication:







Floating Point:

$$(11.1111)_2$$

$\Rightarrow 11.1111 \times 2^{0}$

$\Rightarrow 1111.11 \times 2^{-2}$ → সামনে বড় হয় so পিছনে ছোট করে দিই

$\Rightarrow 1.11111 \times 2^{+1}$ → সামনে ছোট হয়, so পিছনে বড় করে দিও

IEEE- floating Point format:

S	Exponent	fraction
	single: 8 bit	single: 23 bit
	double: 11 bit	double: 52 bit

এই format এ দিব →

$$x = (-1)^{S} \times (1 + \text{fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

① -0.75

যদি decimal এ zero থাকে ↓ ... zero then বাকিটা ...

$S = 1$ [(-1) যুক্ত থাকবে, and আমার (-0.75) so সামনে $(-)$ থাকায় 1, নাহলে 0 দিতাম, power এ (1) দিব]

$\text{fraction} = (0.75)_{10} = (0.11)_2$

$\Rightarrow 1.1 \times 2^{-1}$ → Exponent

[illegible] = 1000 → fraction

[illegible]

Exponent = -1 + Bias → single 의 경우 127, double 의 경우 1023

= -1 + 127

= 126

= 10111110$_2$

final format = | 1 | 01111110 | 10000000000000000000000 |

23 bit

② what num is represented by single precision float

1 10000001 00100000000000000000000

1.10×2^{-1}

0.11×2^{0}

S = 1

fraction = (010 0...00)$_2$

f/r = (10000001)$_2$ = 129

n = (-1)^1 × (1.01)$_2$ × 2$^{(129-127)}$

= (-1) × 1.25 × 2^2

= -5.0

0.01

= 0×2^{-1} + 1×2^{-2}

= 0.25

01

1.01×2^2

101.0×2^0

(101)

= 5

1 8 23

11 8 32

Lecture-02

29/06/19

Procedure call Instruction:

```
4:  addi R1, R0, 12
8:  jal  my_proc
12: add  R1, R2, R2

my_proc
80: addi R2, R1, 8
84: jr
```

Reg file

R0 : 0

R1 : 0 12 40

R2 : 0 20

R31 : 0 12

reg

address → (pc + 4)

8 + 4

= (12)

সিমুলেশন এইখানে 12 num add এ চলে যাবে

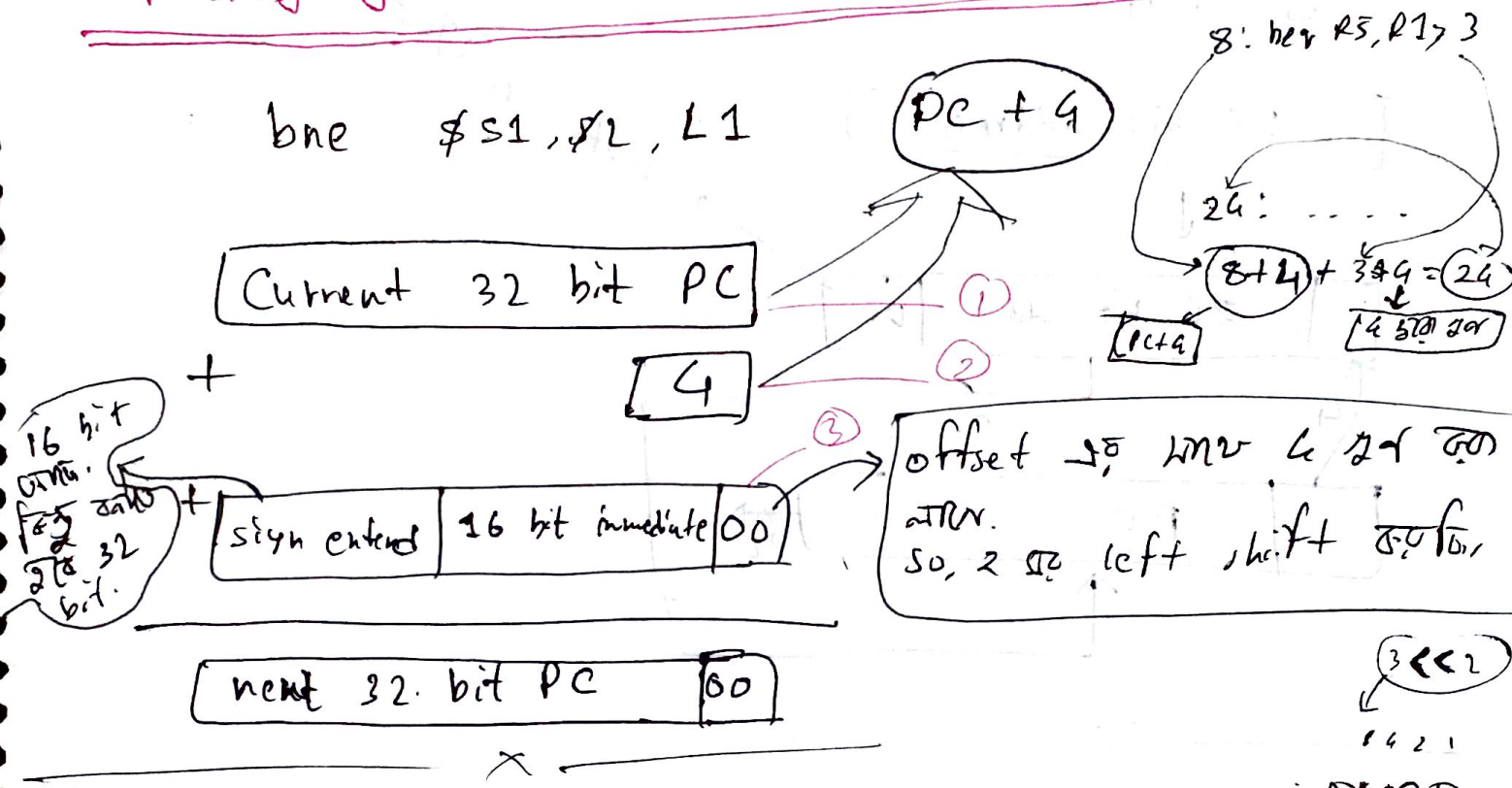
```
main(){
  :
jal my function(){
  :
}
my function(){
  :
jr $ra
}
```

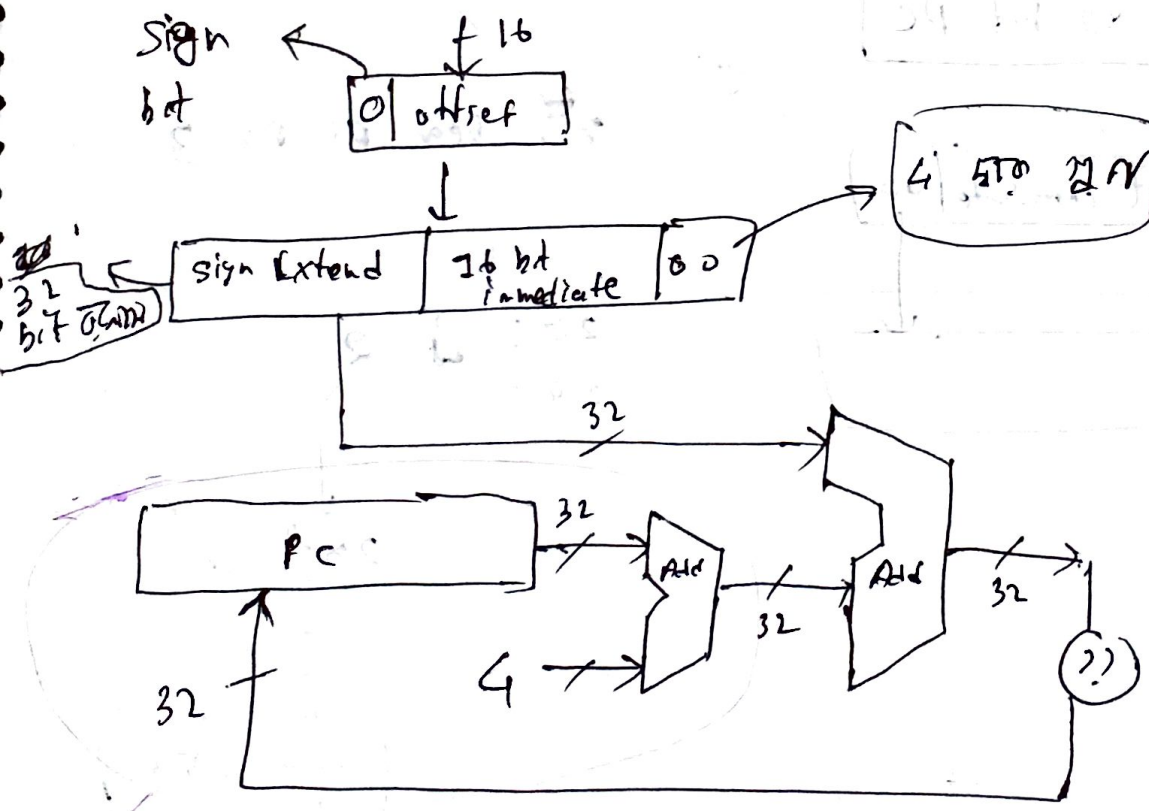
jal my function() → Jump এর সাথে সাথে. Store the return add. if the current add is PC. then it add (PC + 4) → it is the add where the value will return. (PC + 4) is a address & it will save in $ra reg

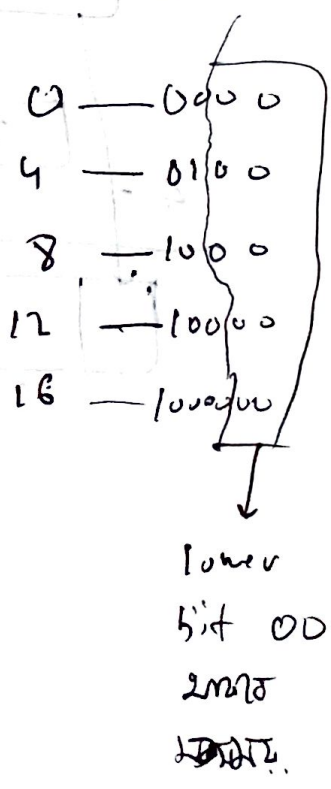
jr $ra → $ra তে যে address save করা আছে সেখানে যাবে এবং value এ ফেরত দেয়া.

Specifying Address Branch Destinations:

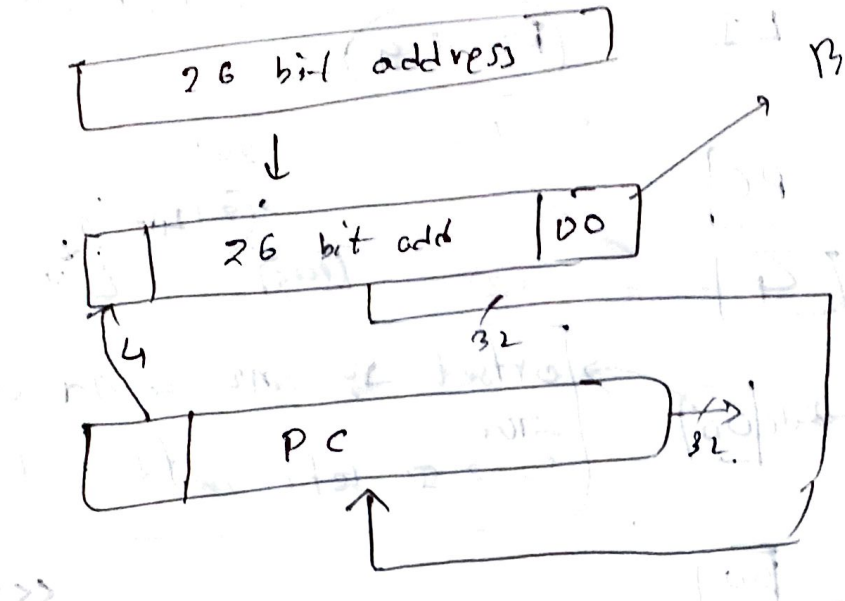
bne $s1, $2, L1

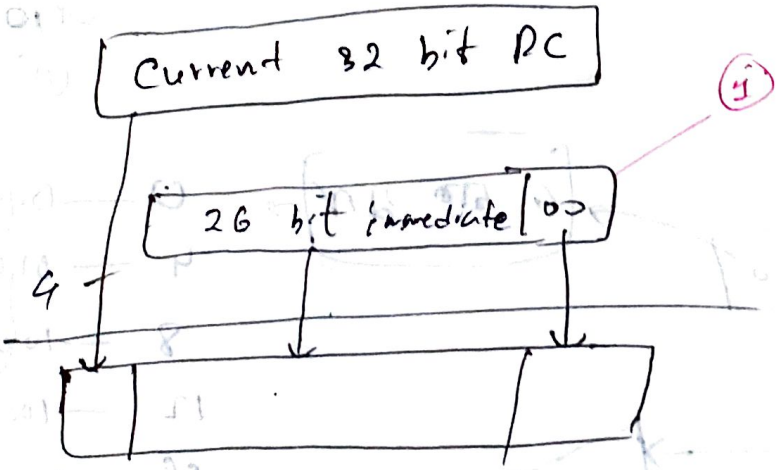






j label





8: bear R5_R1_3

20:

24:

$2 \times 2^2 = 8$

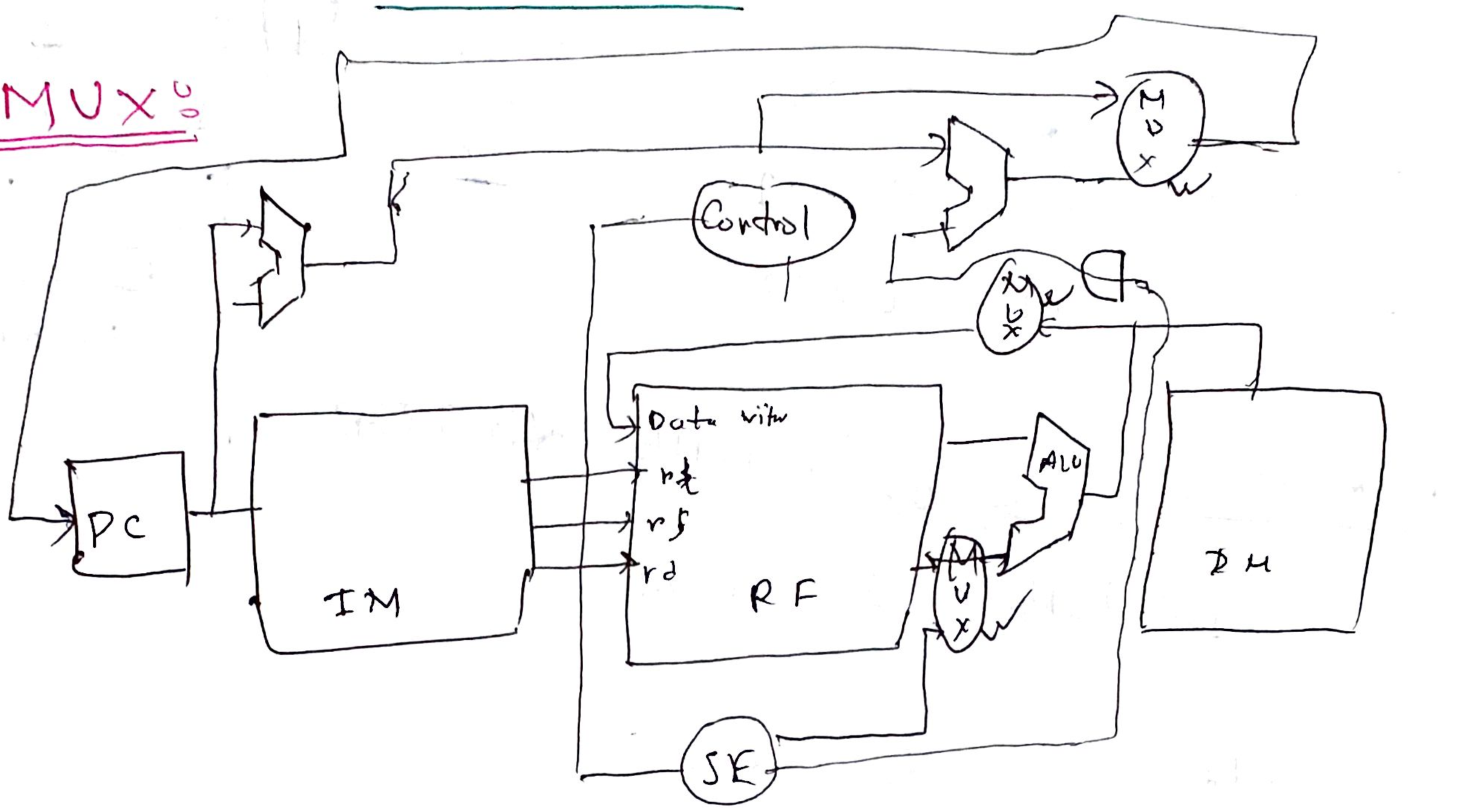
2 SH
left shift
mean 4 BTB
Dar

CSE-340

Lecture-04

19/07/19

MUX:



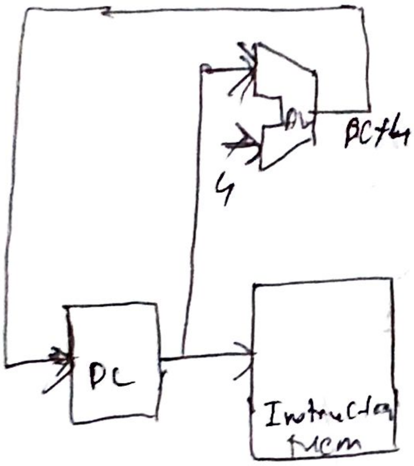
(i) PC+4 or PC+4+4 it is decided by that MUX. on the ~~base~~ beq.

(ii) whose output going to the reg. is it ~~Reg file~~ ALU result or Data Mem?

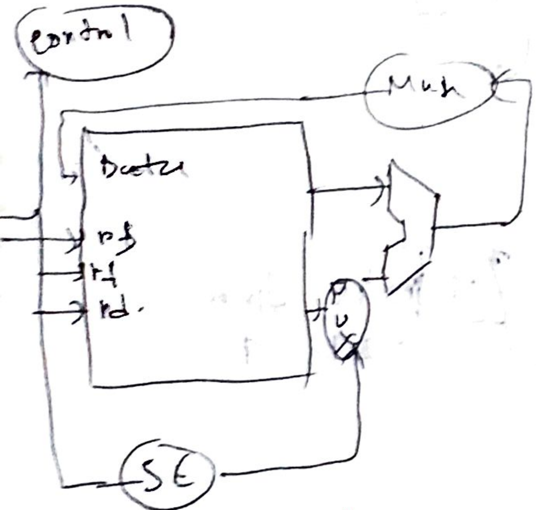
(iii) ~~ALU~~ ALU তে যাবে input কোনটা? Reg file এর 2nd ~~reg~~ result or S.E output?

Combinational Element: AND gate, ALU.

State Element: Reg file, Mem.



- The fetch is done here.
  1. PC holds the address of current instruction.
  2. Instruction Mem store the instruction & supply instruction given an add.
  3. ALU used for increment.



- Decode part done here
  1. in reg there can be written/read.

Control Signal:

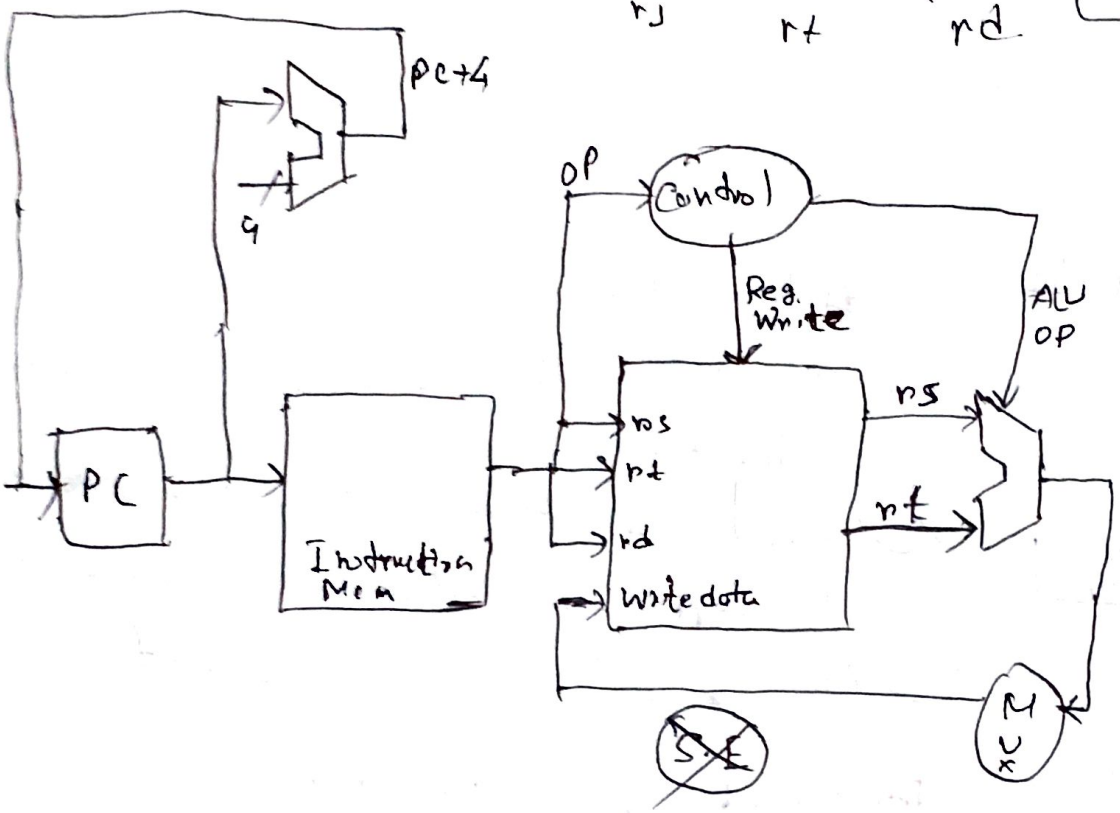
(i) Reg Dst	select rt as destination	select rd as destination
(ii) Reg Write		Reg file এর write data input select করে
(iii) ALU Src	reg file এর 2nd output selected	select output from S.E.
(iv) PC Src	select PC+4	select PC+4+ or Bew
(v) Mem Read		lw এর জন্য D.M কে read করে
(vi) Mem Write		sw এর জন্য D.M কে write করে,
(vii) MemTo Reg	Add এর জন্য ALU থেকে Reg এ	lw/sw এর জন্য D.M থেকে Reg ...

R-type:

Add $s10, $s12, $s13

rs rt rd

or, sub, and, sll, srl, -



I type:

~~lw $s2, $s3(500)~~

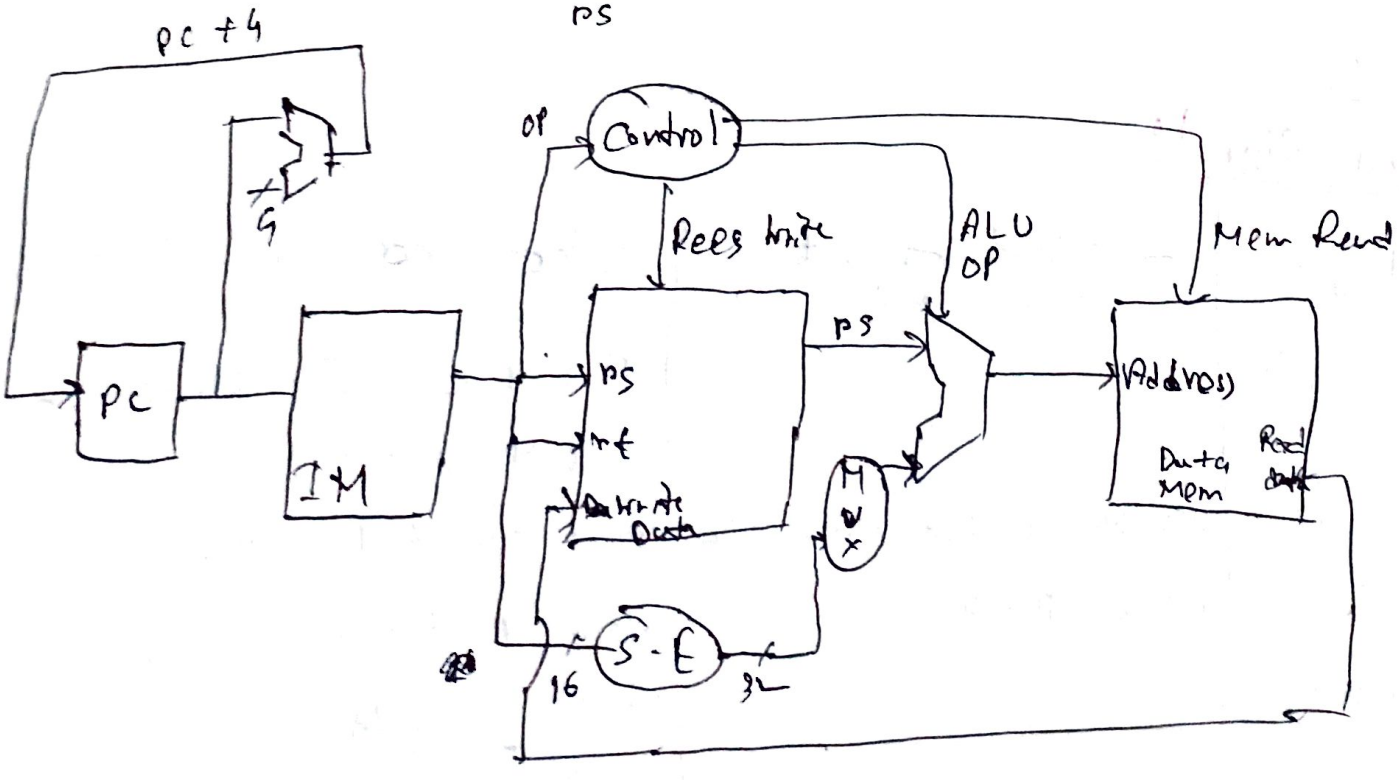
lw $s2, 500($s3)

op	rs	rt	Imm

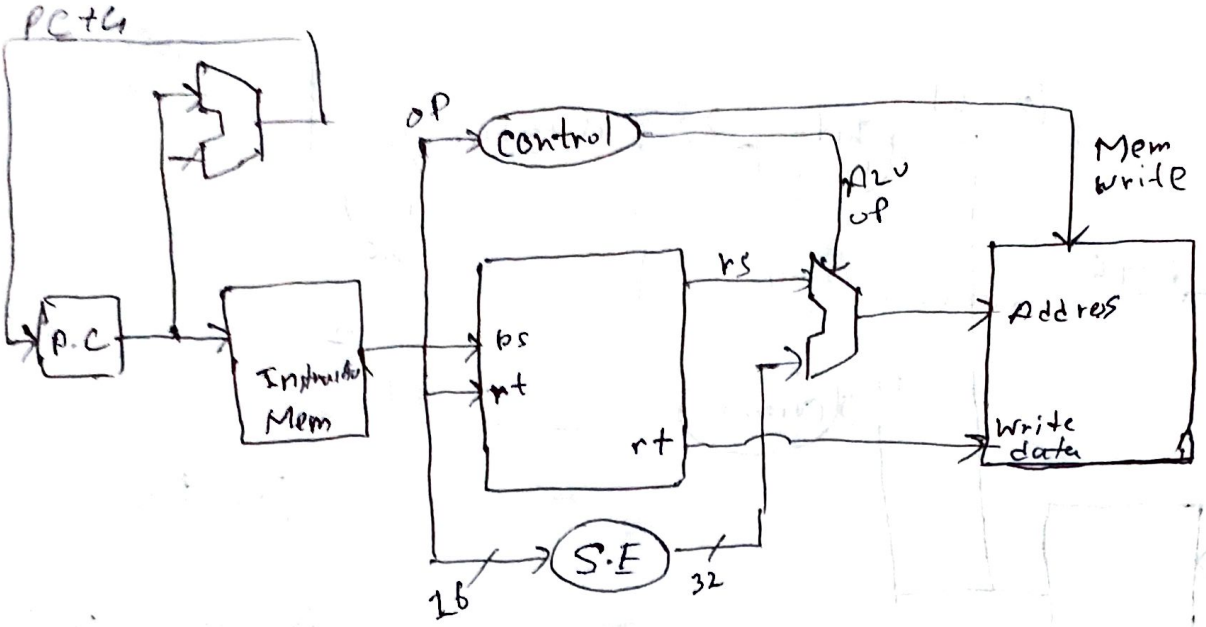
inm 31:26 25:21 20:16 15:0

rt

rs

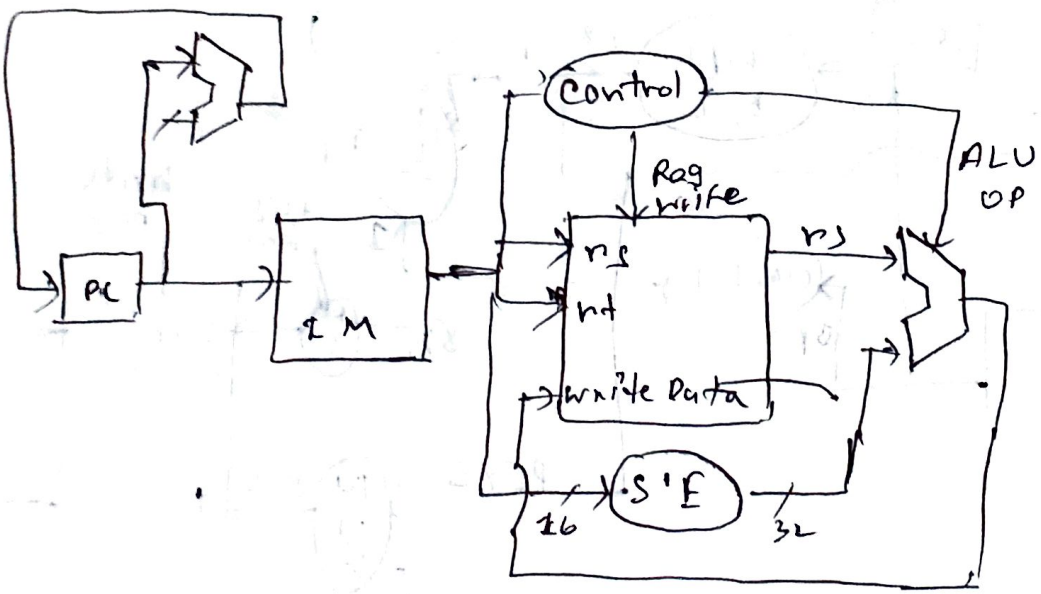


SW $S2, 500($3)

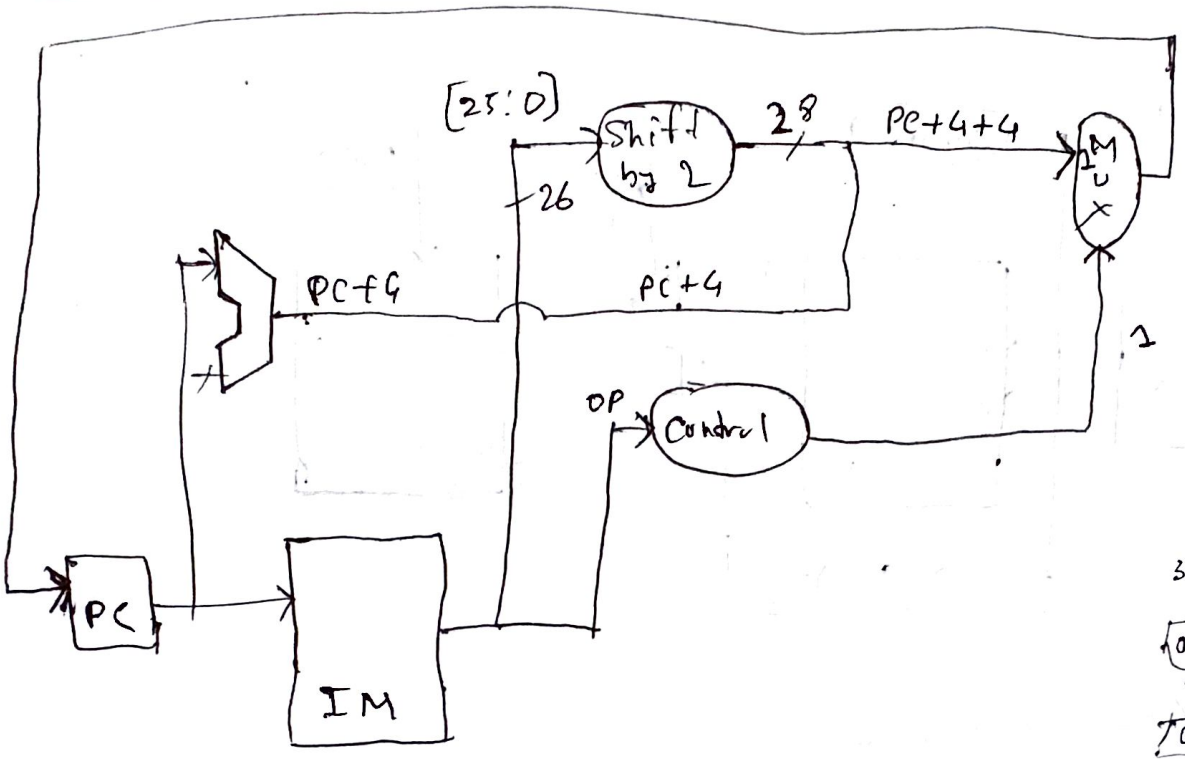


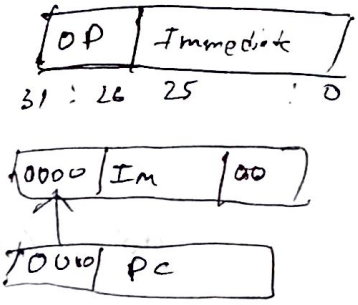
Addi $S2, $S1, 100

rs rt Imm

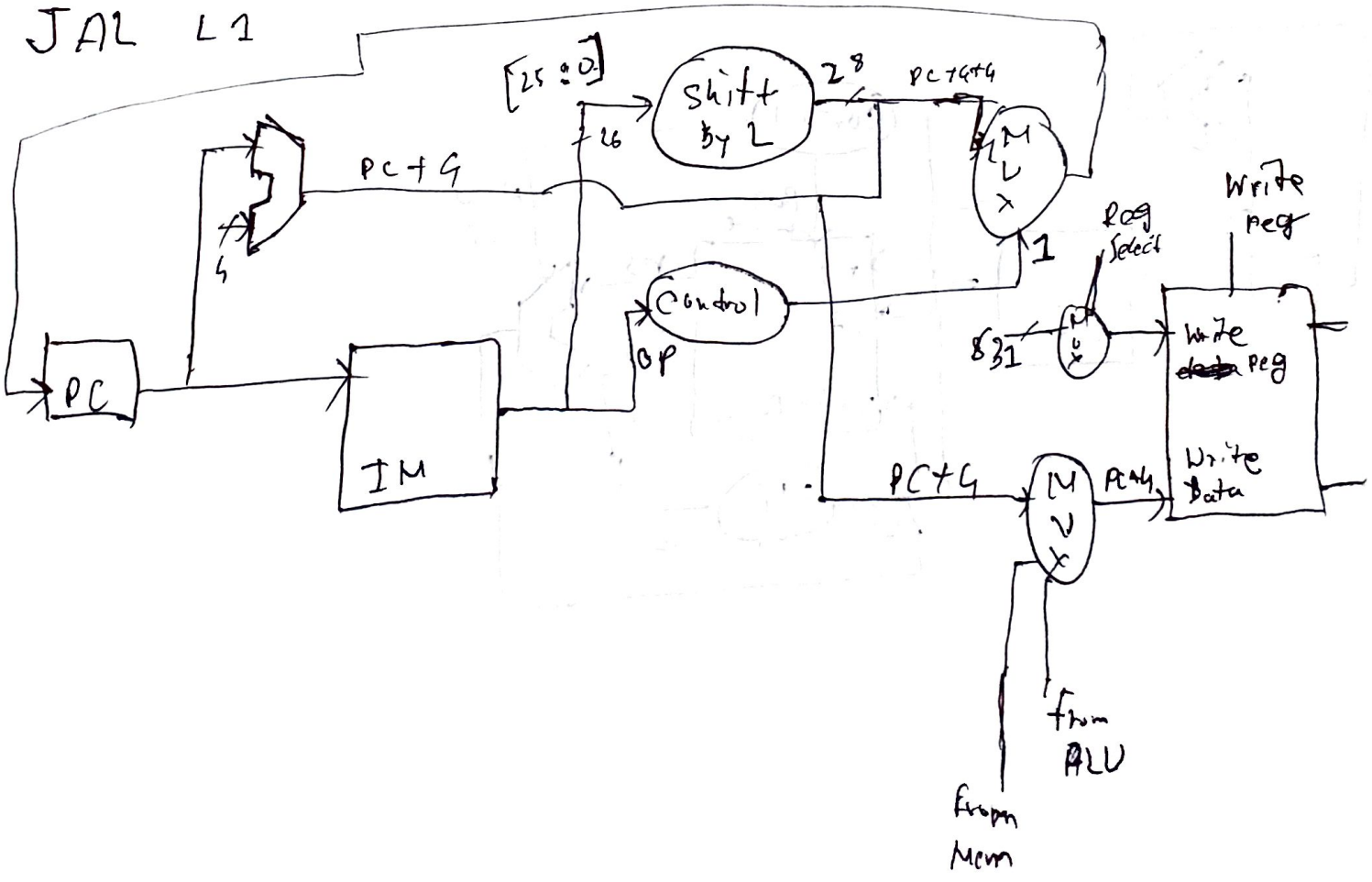


J-Type J 1030.

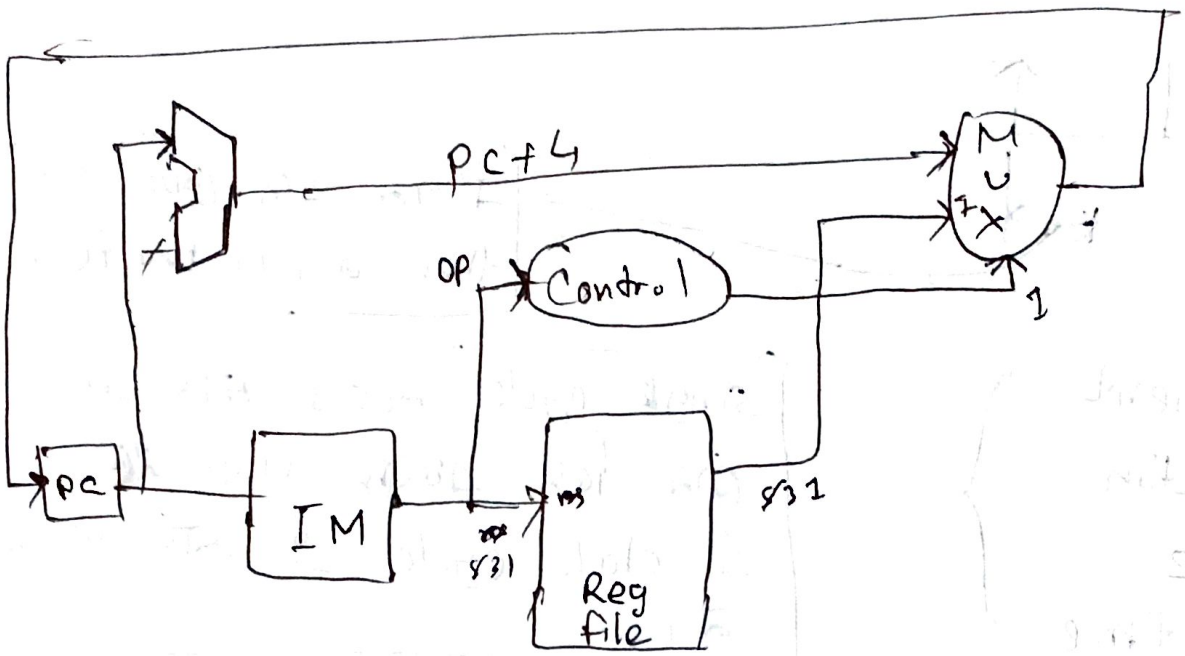




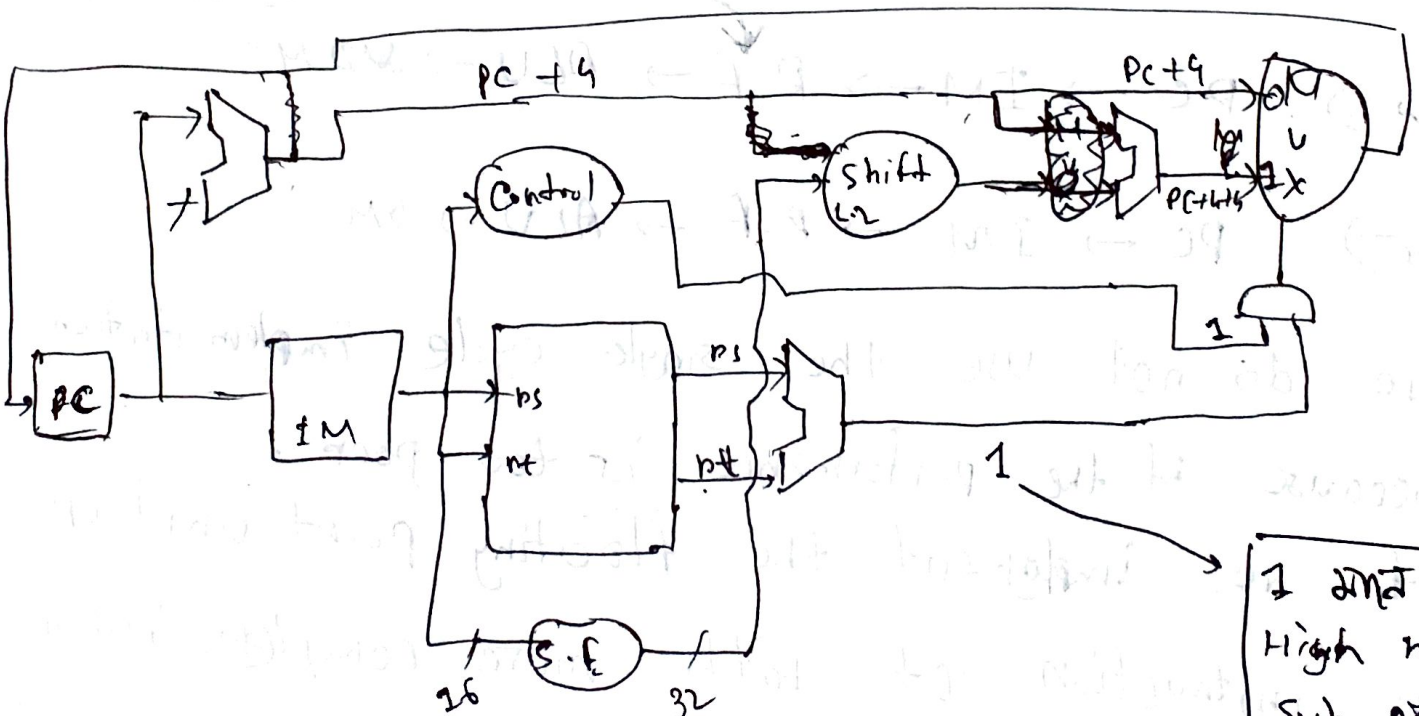
JAL L1



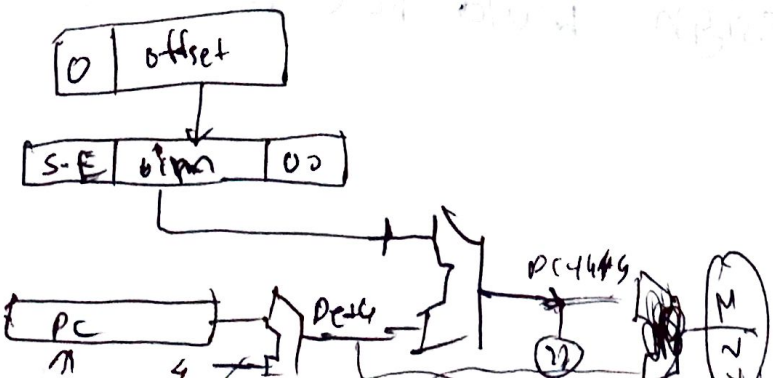
Jr $31



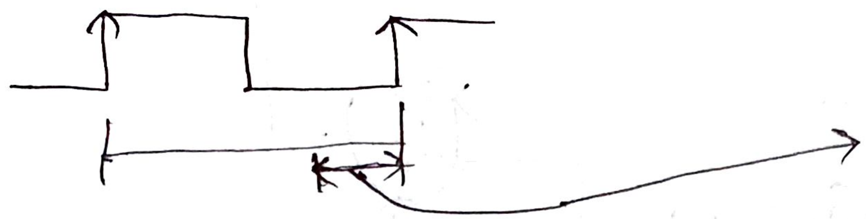
Beq $10, $12, Else



1 일때
High means
Sub 의 result
Zero,
Jump to Else



Performance Issue :



এইখানে যদি কোনো হয় then এইটা waste হবে

1 component
1 instruction
1 cycle
at a time

single cycle means ~~ALU কে~~ কোন জিনিস একবার use হলে ঐ clock cycle এ আর use হবে না।

lw সবচেয়ে time consuming.

lw ⇒ PC → IM → R.F → ALU → DM (write back → R.F)

sw ⇒ PC → IM → R.F → ALU → DM

We do not use the single cycle implementation because the performance is too poor. If we implement the floating point unit or an instruction set with more complex instruction, this single cycle design would not work at all.

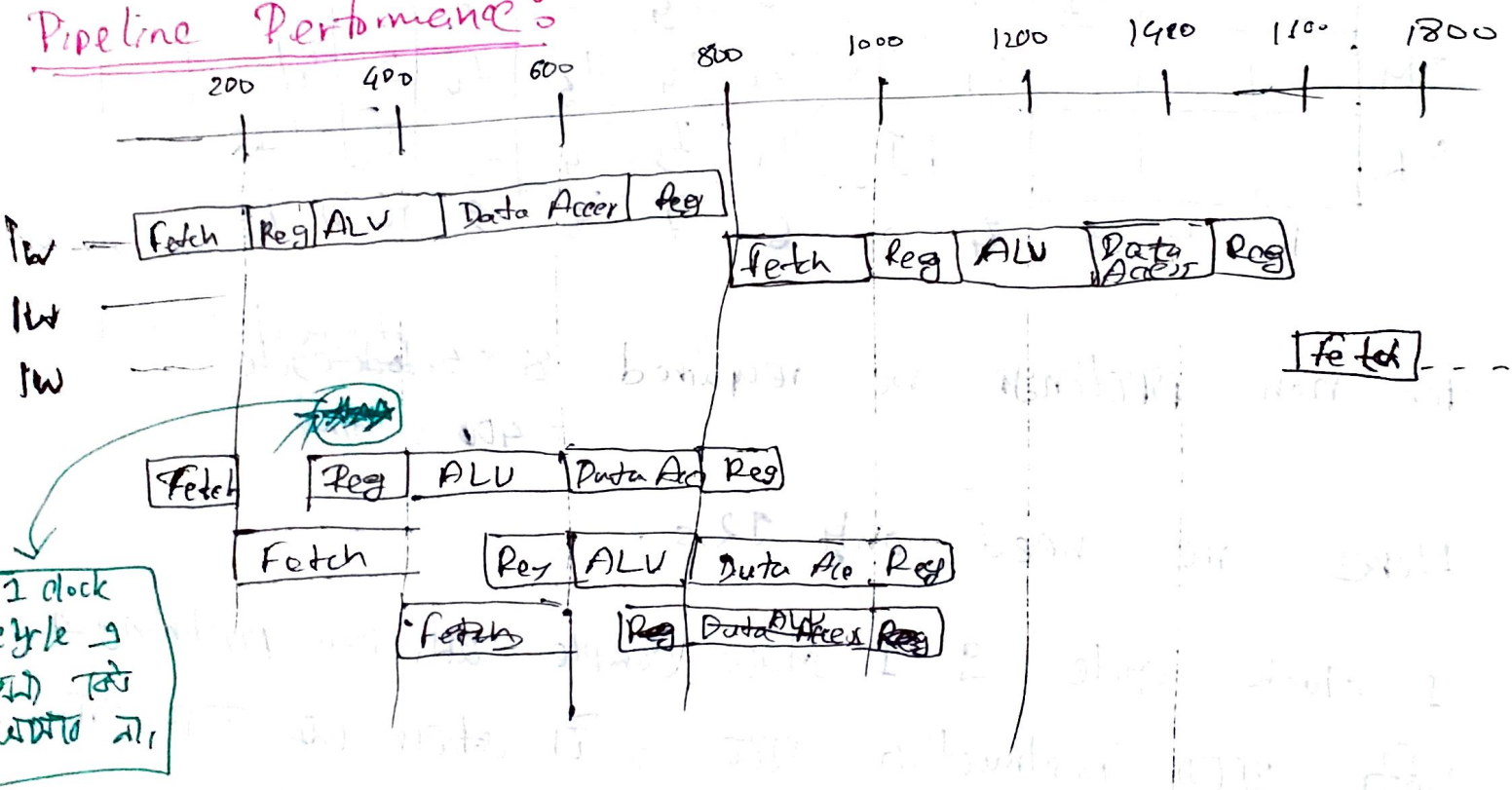
Pipeling:

- Pipeling is faster
- everything is working in parallel.

(F) (D) (E) (DM) (WD)

Five steps of MIPS instruction.

Pipeline Performance:



Pipeling Vs Non Pipeling:

8 instructions.

$k + (n-1)$

stages ↑	1	2	3	4	5	6	7	8	9	10	11	12
F	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8				
D		I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8			
E			I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8		
DM				I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	
WR					I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8

→ time

for non pipelings we recquired 8×5 ~~stage~~ clock cycle
$= 40$ s time.

Here we need only 12 s.

1 clock cycle এ 1 stage comple হয়। non pipling এ একটি পুরো instruction পুরো 5 টা stage শেষ করার পর অন্য instruction আসতে পারে। Pipeling এ একটা stage শেষ হওয়ার পর ঐ stage এ অন্য instruction এর কাজ শুরু হয়ে যায়।

$\therefore$ Speed up $= \dfrac{\text{Time of non pipeline}}{\text{time of pipeline}}$

$= \dfrac{40}{12} = 3.33$

Time between instruction $= \dfrac{\text{Time between non pipeline instruction}}{\text{num of pipeline}} = \dfrac{40}{5} = 8cc$

$\therefore$ Effeciency/utilization $= \dfrac{\text{we use how much}}{\text{total num of block}} = \dfrac{40}{60}$

Pipeline Hazard: unwanted event occur in pipeline.

① **Structural Hazard:**
- same clock cycle
- same resource (Reg)
- two or more instruction try to access

I1 : ADD R1, R2, R3 // $R_1 \leftarrow R_2 + R_3$

I2 : ADD R4, R5, R6 // $R_4 \leftarrow R_5 + R_6$

I3 : SUB R7, R1, R8 // $R_7 \leftarrow R_1 - R_8$

	1	2	3	4	5	6
F	I_1	I_2	I_3			
D		ADD R2, R3	ADD R5, R6	SUB R1, R8		
E			$R_2 + R_3$	$R_5 + R_6$	$R_1 - R_8$	
WB				$R_1 \leftarrow$ result	$R_4 \leftarrow R_5 + R_6$	$R_7 \leftarrow R_1 - R_8$

$R_7 \leftarrow R_1 - R_8$
$R_1 \leftarrow R_2 + R_3$
$R_7 \leftarrow R_1 - R_8$

	1	2	3	4	5	6
F	I_1	I_2	I_3			
D		SUB R1, R8	ADD R5, R6	ADD R2, R3		
E			$R_1 - R_8$	$R_5 + R_6$	$R_2 + R_3$	
WB				$R_7 \leftarrow$ result	$R_4 \leftarrow$ result	$R_1 \leftarrow R_2 + R_3$

code schedule possible

	1	2	3	4	5	6
F	I_1	I_2	I_3			
D		ADD R2, R3	ADD R5, R6	SUB R1, R8		
E			R2+R3	R5+R6	R1−R8	
WB				R1 ← result	R_4 ← result	R_7 ← result

→ Not the dependency coz, R1 এর value ready হয়ে গেছে so, এখন SUB করা যাবে. point is Reg এ এখন R1 এর কাজ হচ্ছে, একই D আর একই WB এর কাজ হচ্ছে, same CC তে. That is Structural Hazard

Solution:

- Delay with Bubble
- until I1 is totally finished.

I_4 sw \$5, \$500(\$10)

I_3 lw \$5, 500(\$80)

I_2 Add \$5, \$7, \$9

I_1 sub \$3, \$7, \$7.

	1	2	3	4	5	6	7	8
F	I_1	I_2	I_3	I_4	I_5	I_6		
D		Sub \$7, \$7	Add	lw	sw			
E			\$7−\$7	\$7+\$9	500(\$80)	500(\$10)		
DM				–	–	Addr	\$5 ←	
WB				\$←	\$5←		\$5←	X

Decode হয় Reg এ থেকেই, W.B হয় Reg এ.

Fetch হয় I-M যে যা আসে RAM এ, এবং Address save হয় DM যে যা থাকে RAM এ, same CC তে RAM use হচ্ছে দুটো instruction এর জন্য,

Data Hazard:

Dependency will be there

3 types of Data Hazard:

(i) Read after write Hazard (RAW)

(ii) Write after Read Hazard (WAR)

(iii) Write after write Hazard (WAW)

(I) Read after write Hazard:

I_1 ADD R_1, R_2, R_3 // $R_1 = R_2 + R_3$

I_2 SUB R_4, R_1, R_5 // $R_4 = R_1 - R_5$

dependency!

	1	2	3	4	5
F	I_1	I_2			
D		ADD R_2, R_3	SUB R_1, R_5		
E			$R_2 + R_3$	$R_1 - R_5$	
WB				$R_1 \leftarrow$	$R_4 \leftarrow$

R_1 ... value ... store ... num CC ... I_2 ... 3 num CC ... read ... delay, ... 5 CC ...

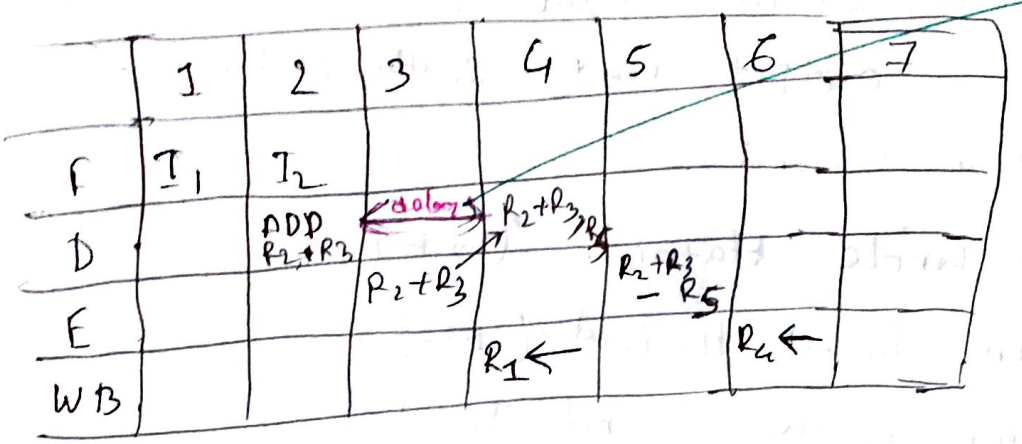
Solution:

(i) Delay the Pipe:

delay 2 CC

	1	2	3	4	5	6	7
F	I_1	I_2					
D		ADD R_2, R_3	delay		SUB R_1, R_5		
E			$R_2 + R_3$	$R_1 \leftarrow$		$R_1 - R_5$	$R_4 \leftarrow$
WB							

(ii) Operand forwarding Technique:



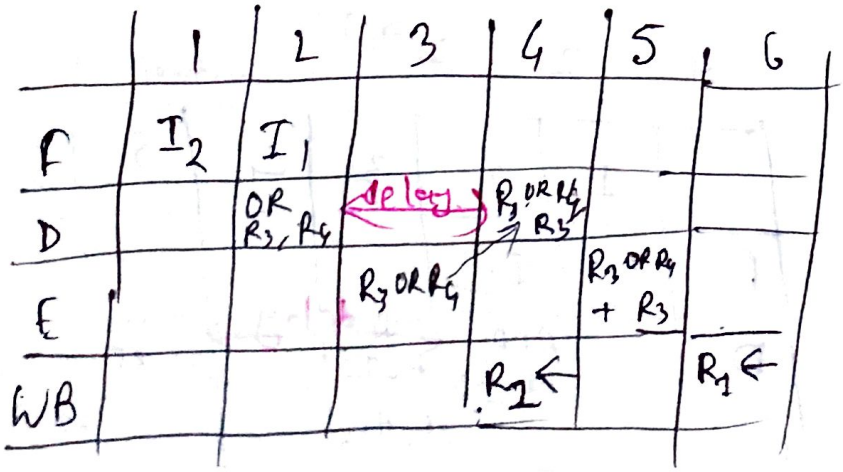
1 cc delay হবে, direct execution এর ক্ষেত্রে (R_2+R_3) কে দিয়ে দেয়া যাবে।

(2) Write after Read Hazard (RW Hazard / Anti-Dependency Hazard)

i : ADD R_1, R_2, R_3 ; // $R_1 \leftarrow R_2 + R_3$ → এখানে write হচ্ছে

j : OR R_2, R_3, R_4 ; // $R_2 \leftarrow R_3$ or R_4 → এখানে read হচ্ছে

Here ADD এর আগে OR execute হবে, কারণ এটি faster, তাই dependency চলে যাবে।



③ Write after write Hazard (WW Thread/output Dependency)

i $R_2 \leftarrow R_4 + R_7$

j $R_2 \leftarrow R_1 + R_3$

if as usaul i will happen first and j will be last. At the end ~~of the~~ we get updated value of R_2 from j. But if j happens first then there will be a problem. So, we set delay so that j will execute just after the i finish.

Code Scheduling:

```
   lw  $t1, 0($t0)
6  lw  $t2, 4($t0)
7
8  add $t3, $t1, $t2
9  sw  $t3, 12($t0)
10 lw  $t4, 8($t0)
11
12 add $t5, $t1, $t4
13 sw  $t5, 16($t0)
```

(??)

```
   lw  $t1, 0($t0)
6  lw  $t2, 4($t0)
7  lw  $t4, 8($t0)
8  add $t3, $t1, $t2
9  sw  $t3, 12($t0)
10 add $t5, $t1, $t4
11 sw  $t5, 16($t0).
```

(?)

independent গুলো সবার আগে রাখব,

মাঝের independent code গুলোই stall ফিল করে।

stall এ load এ 13টা cycle হবে।

for dependency:
1. stall এ 2টা stall
2. stall + forwarding এ 1টা stall

Control Hazard:

```
Add   $3, $2, $5
Sub   $6, $7, $8
beq   $5, $6, L1
MUL   $5, $7, $6     } × → flush
Div   $5, $9, $3     }
L1:
Add   $12, $10, $11
```

যখন program চলতে চলতে branch চলে আসে তখন jump করে new mem location এ চলে যায়, তত আগের যে instruction আছে, তারা removed করতে হয়, Thus the time lost as a result of this is Called Control Hazard. Branch হবে কিনা তা জানবো Ex stage এ, তখন তার instruction F, D, Ex এই 3 stage এ চলে যাবে।

S1
S2 - Beq → S6
S3
S4
S5
→ S6

	1	2	3	4	5	6	7	8
F	S1	S2	S3	S4	S6			
D		S1	beq S2, S3	S3		S6		
E			S1	S2 - S3 = 0			S6	
WB				S1				S6

~~too~~ losing 3 cc ~~then~~ Here.

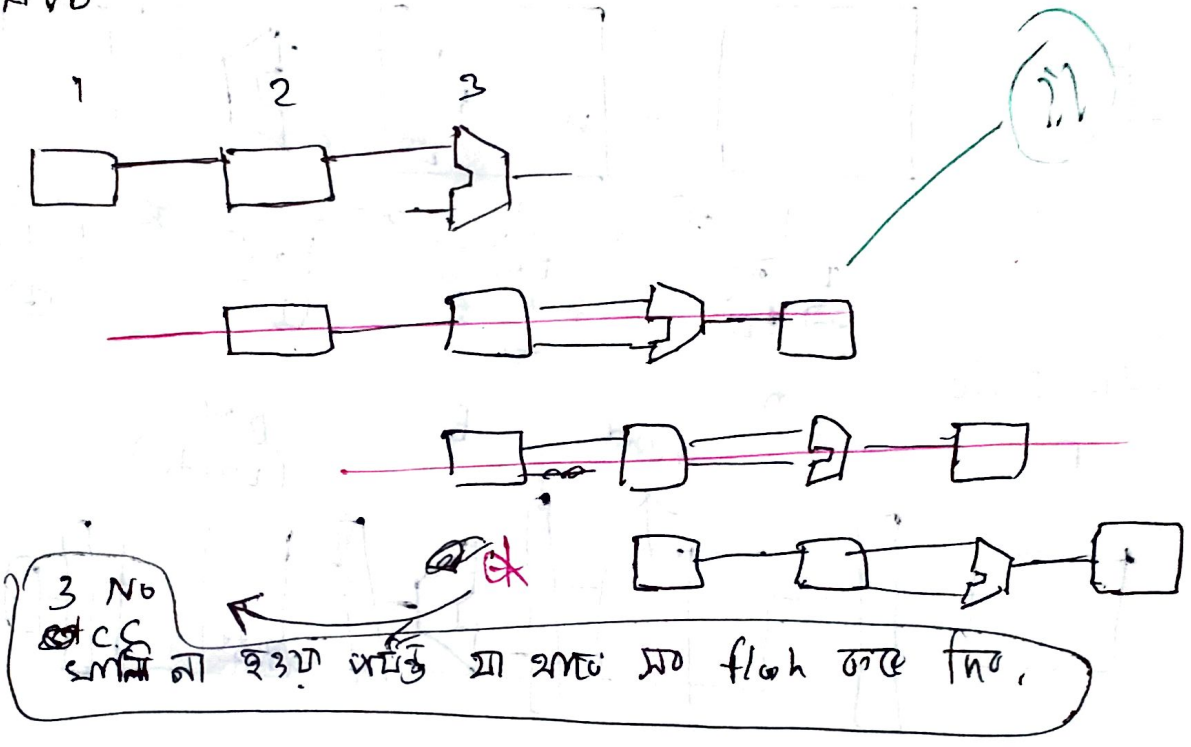
Solution: Flashing.
- control signal send signal (0 0 0)
- just set (0 0 0) keep all same.
- It will happen in 3 no cc of Execution
- & after those instruction of beq.

I_1 BEQ R1, R2, 50

I_2 AND R2, R3, R4

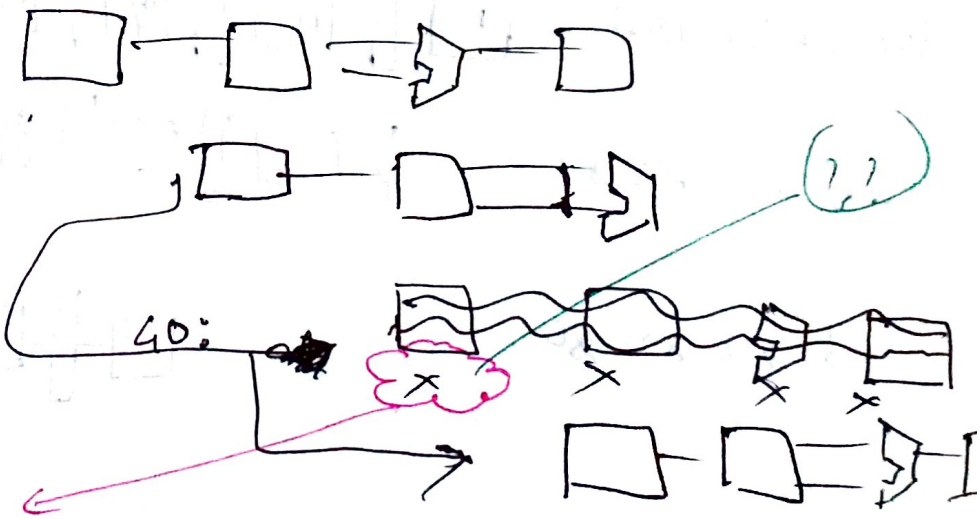
I_3 OR R1, R4, R5

I_4 ADD R2, R3, R4



(2) I Can do prediction.

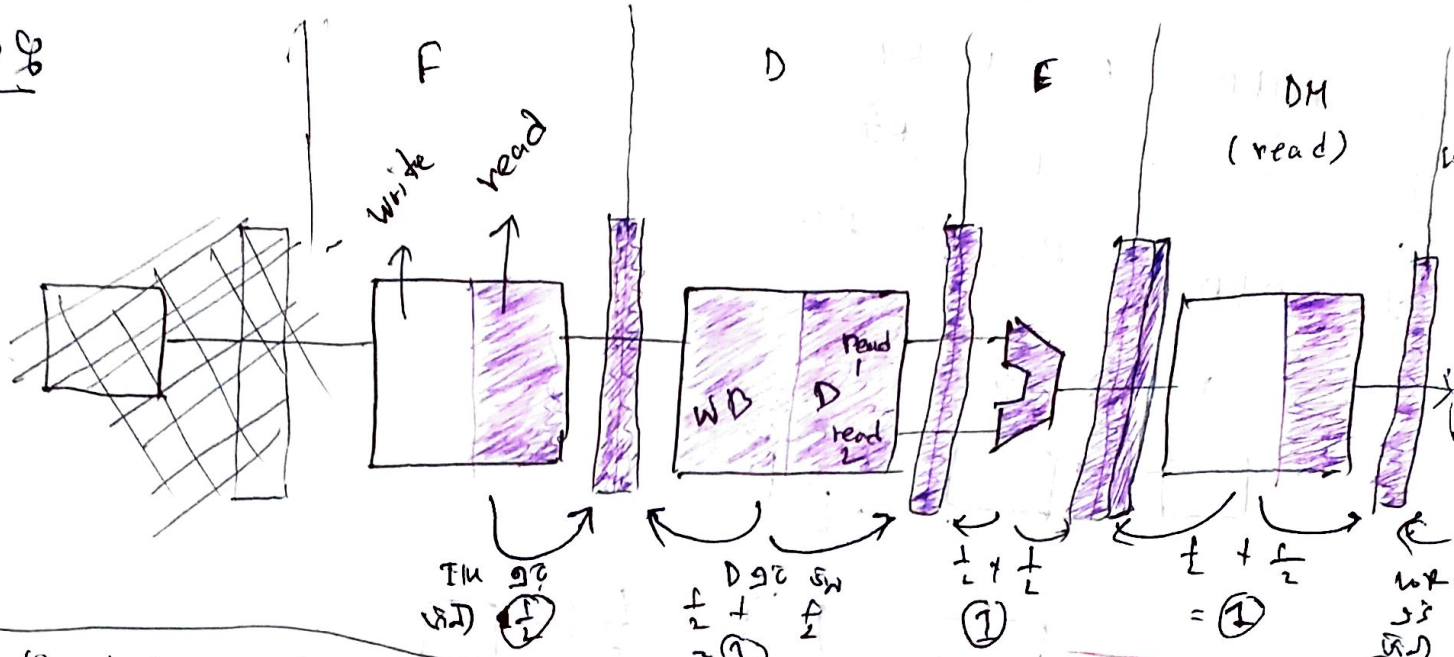
```
    add $4, $5, $6
    beq $1, $2, 40
    add $2, $3, $6
40: OR  $7, $8, $9
```



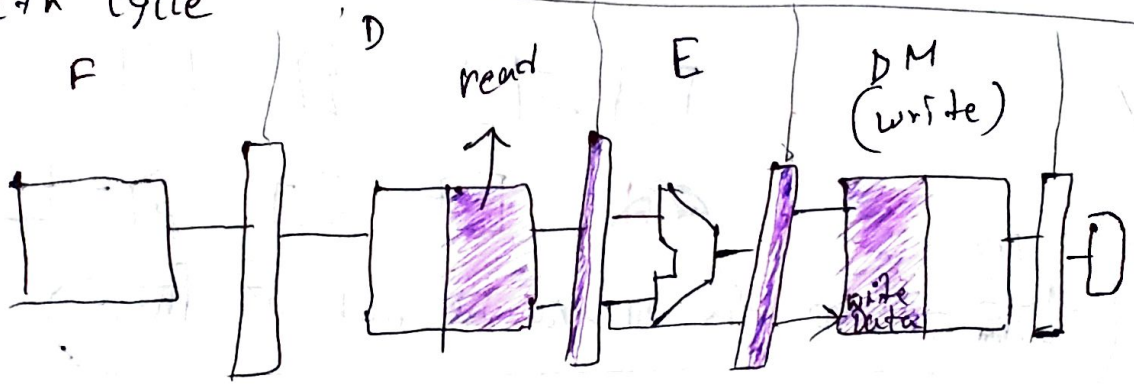
Add one bubble when you jump to the next add, ~~when~~ when jump is right after beq.

Pipeline Datapath:

lw:



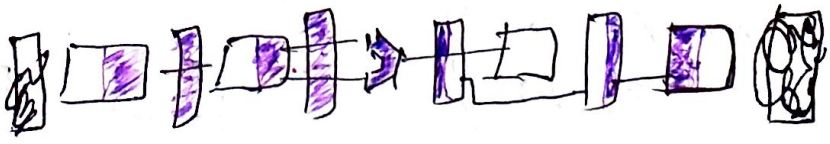
sw: 4th cycle



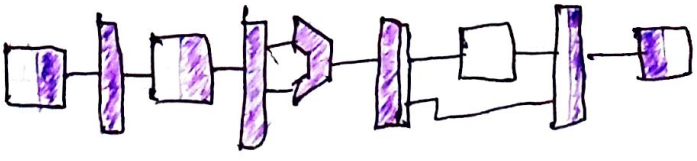
Multi Cycle Pipeline:

lw $10, 20($1)

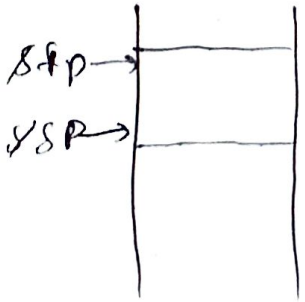
sub $11, $2, $3

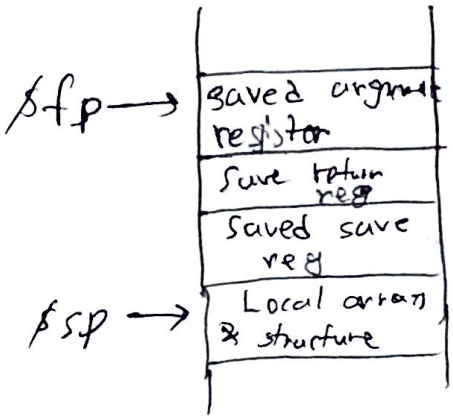


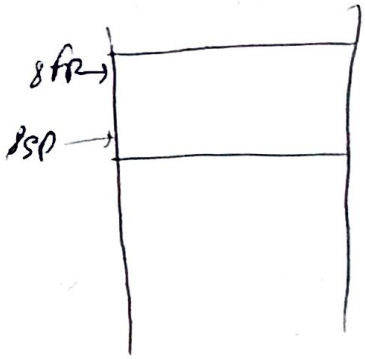
add $12, $3, $4



Stack Pointer & frame Pointer: sp points the top of the stack and the frame pointer points to the current active frame. Sometime you do not need frame pointer. You can use sp instead pointer ~~tot~~ instead. Then the fp becomes free and can be used in other works.







Stack pointer can be changed but frame pointer will not. This is like your mother (fp) and your girl friend (sp). During the push the sp is changed but fp holds the base address. So fp is used to access the local vars.

Block: - The minimum unit of information that can be either present or not present in a cache.

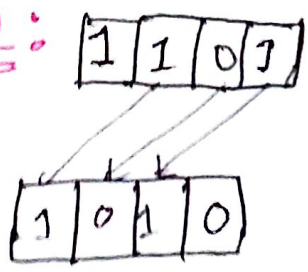
Hit Rate: The fraction of memory access found in a level of mem hierarchy.

Miss Rate: The fraction of mem access not found in a level of mem hierarchy.

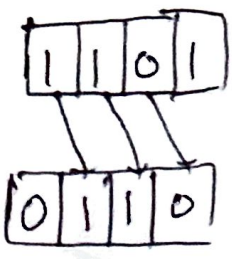
Hit Time: The time required - to access a level of mem hierarchy - the time needed to determine whether the access is a hit or miss.

Miss Penalty: - The time required to fetch a block - The time to access the block.

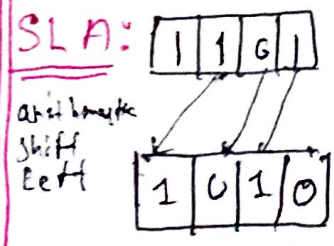
SHL:



SHR:

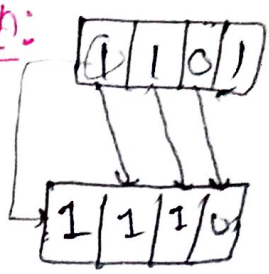


SLA:



normal left shift হবে,

SRA:



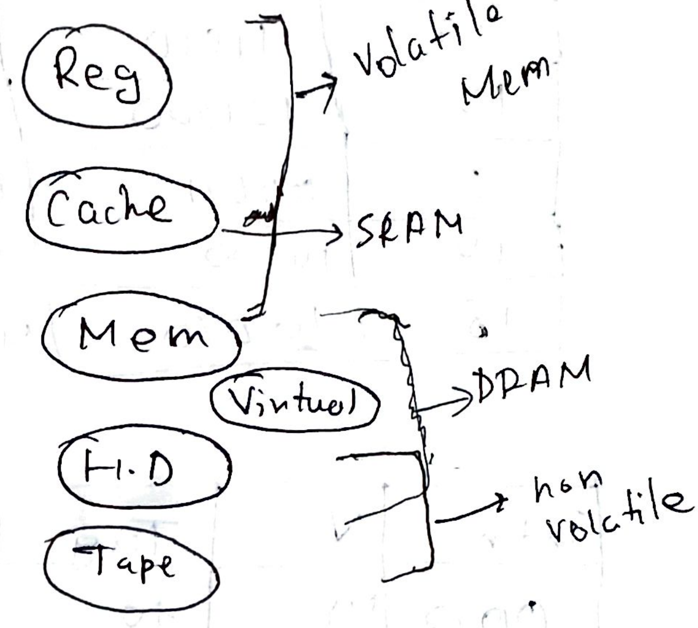
sign bit so কোনো change হবে না, direct copy হবে

CSE-340

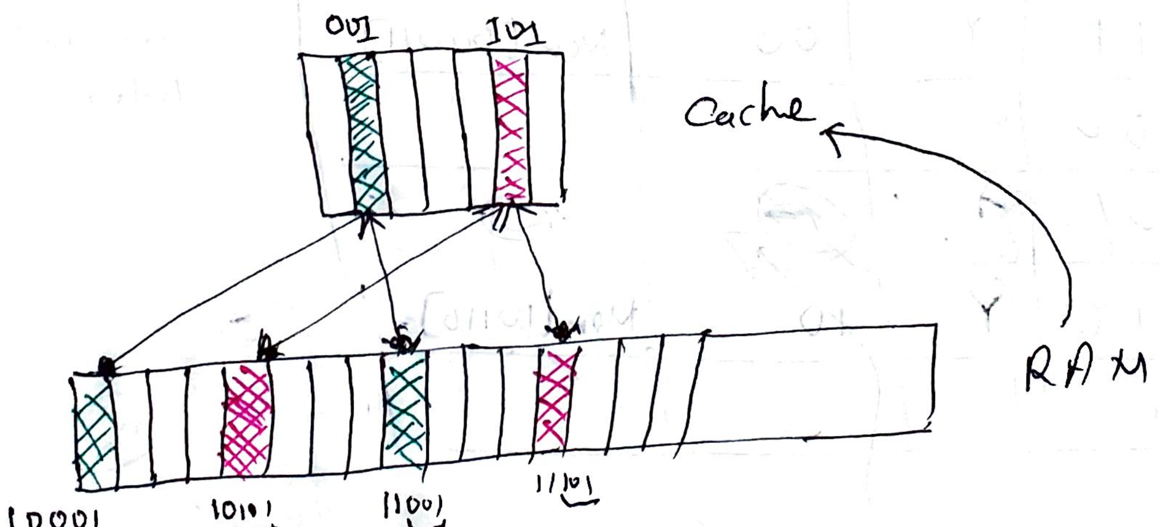
তারিখ 30/07/19

Chapter-05

Mem ~~Hei~~ Hierarchy:



How to do cache Mem Read and Cache Write?



Temporal ~~Data~~ locality: যে data ~~cache mem~~ এ বার বার use হবে তা cache mem এ রাখবো. (Cache)

Spatial locality: Where to put which instruction? We put outside of cache mem but not in H.D (RAM)

Read Advantage:

Word Add	Binary addr	Hit/miss	Cache
22	10110	Miss	110
26	11010	Miss	010
16	10000	Miss	000
3	00011	Miss	011
16	10000	Hit	000
18	10010	Miss	010

Can not go bcoz valid bit is 0.

So it is called Hit,

Now it will throw prev value to the dirty bit. then install the latest data.

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	~~N~~		
110	Y	10	Mem[10110]
111	N		

22 = 10110 → offset

26 = 011010 → index

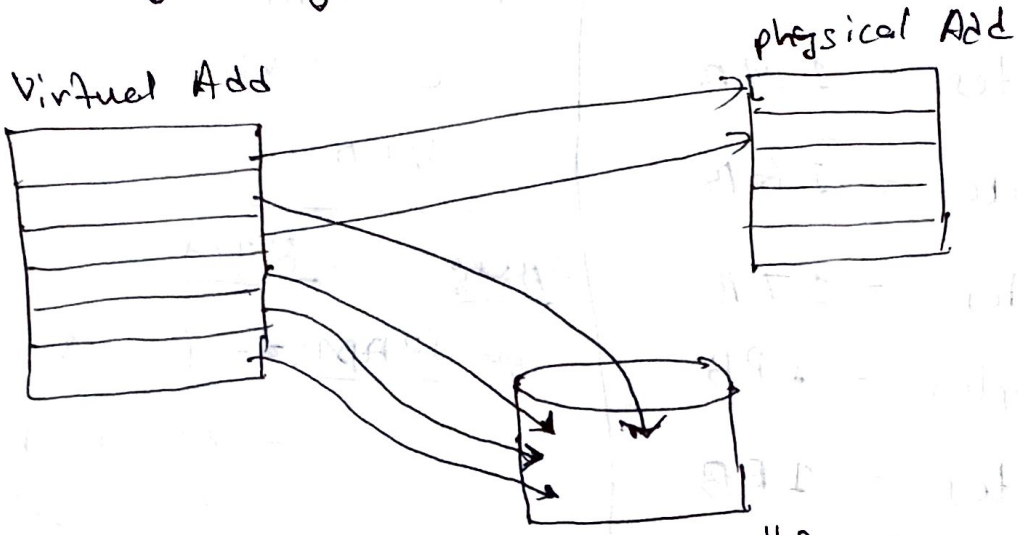
→ Tag.

$2^3 \times 8 \times 64$

2^6

Virtual Mem°

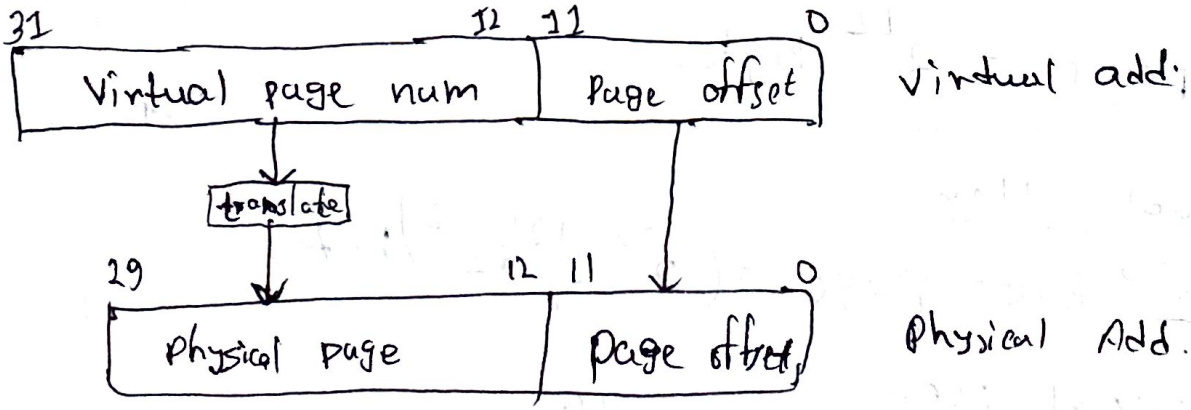
- Software control mem.
- physically lays on H-D.



Virtual mem can't access ~~the~~ the data.

It has logical ~~physical~~ add. which is converted to physical add.

physical mem: Cache, RAM, H-D.



2 bit is used for byte adjustment in physical Add.

Size of Mem°

8 bit = 1 byte

2^{10} bytes = 1 KB

2^{20} bytes = 1 MB

2^{30} bytes = 1 GB

2^{40} bytes = 1 TB

2^{50} bytes = 1 PB

2^{60} bytes = 1 EB

Volatile vs Non Volatile

RAM (SRAM, DRAM)
Cache
Reg

flash (SSD)
disk (HDD, CD, DVD)
ROM

---

1 page = 2^{12} bytes

Virtual Mem

2^{20} page = $(2^{20} \cdot 2^{12}) = 2^{32}$ bytes

physical Mem

2^{18} page = $(2^{18} \cdot 2^{12}) = 2^{30}$ bytes

2^{28} page = $(2^{28} \cdot 2^{12}) = 2^{40}$ bytes.

Floppy Disk.
CD, DVD.
HDD

RAM:

SRAM:- fast & Expensive
- Access in one clock cycle

DRAM:- slower but less expensive
- Access in 10 cc.

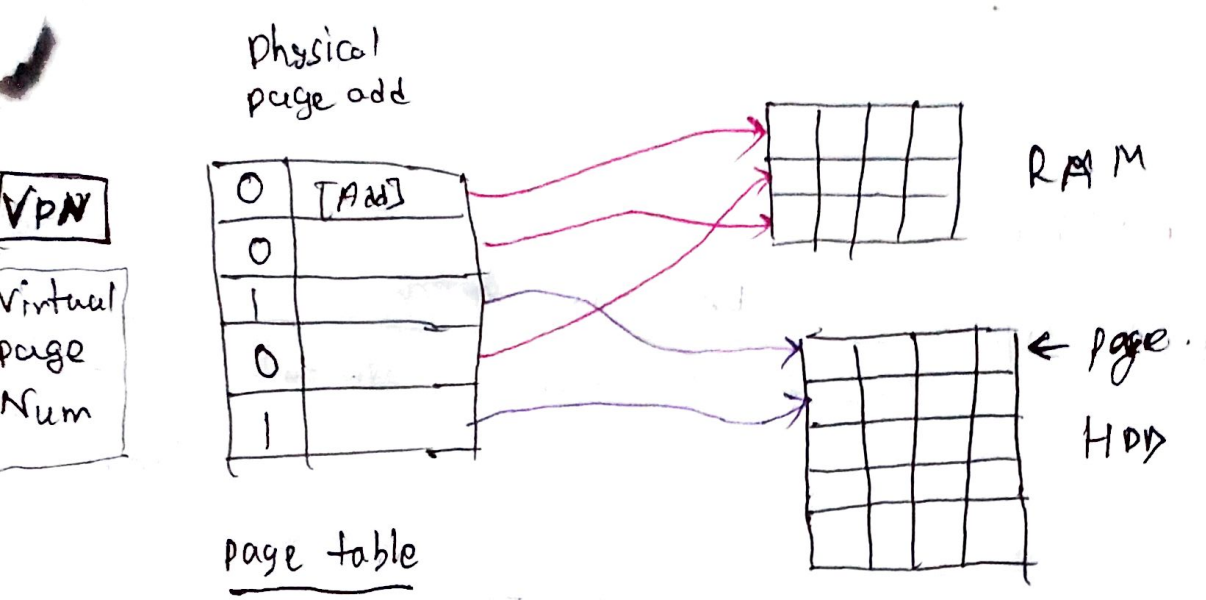
2^{32} byte = 1 page

2^{12} byte = 1 page

1 = $\frac{1}{2^{12}}$ page

2^{32} byte = $\frac{1}{2^{12}} \times 2^{32}$

= 2^{20} page



Page fault: When a program tries to access a add that is belongs to a page on HDD this page ~~is called~~ must be brought into main mem.

Bits in a cache:

1 word = 4 Byte

$$2^n \times \left[2^m \times 32 + (32 - n - m) + 1\right]$$

2^m × 32 : block size

$(32 - n - m)$: tag field

$+1$: field size.

(1,2)

① 16 KiB data & 4 word blocks, assuming 32 bit reg!!

$16\,KiB = 2^{10} \times 16$

$= 2^{10} \times 2^4$

- $4\ word = 2^m \Rightarrow 2^2 \therefore m = 2.$

$1024\ block = 2^{10} \Rightarrow n = 10$

$2^{10} \times \left[2^2 \times 32 + (32 - 10 - 2) + 1\right]$

Chapter - 6

CSE-340

10/08/19

Speed Up Challage:

$$\text{Speed up} = \frac{\text{En. time before}}{(\text{En. time before} - \text{En. time affected}) + \frac{\text{En time Affect}}{\text{Ammount of Improvement}}}$$

$$\text{Speed up} = \frac{1}{(1 - \text{fraction time affected}) + \frac{\text{fraction time affected}}{\text{Ammount of improvement}}}$$

(80)

$$\Rightarrow 70 = \frac{1}{\frac{80}{80}(1 - \text{fraction time affected}) + \frac{\text{fraction time affected}}{80}}$$

$$\Rightarrow 70 = \frac{80}{80 - 80\,\text{fraction} + \text{fraction}}$$

$$\Rightarrow 70 = \frac{80}{80 - 79\,\text{fraction}}$$

$$\Rightarrow 5600 - 5530\,\text{fraction} = 80$$

$$\Rightarrow -5530\,\text{fraction} = -5520$$

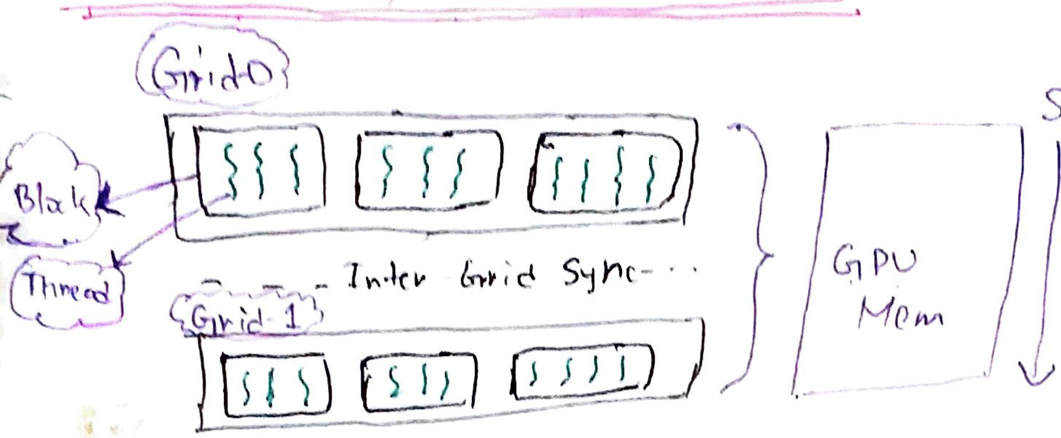
$$\Rightarrow \text{fraction time affected} = 0.999$$

$$(1 - 0.99) = 0.1\%$$

Ans: 0.1%.

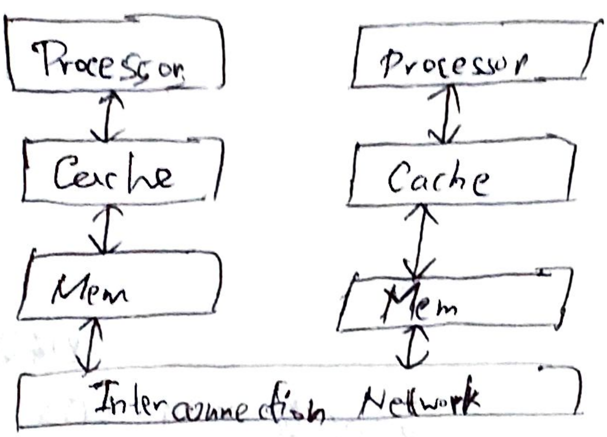
Streaming Multiprocessor Arc:

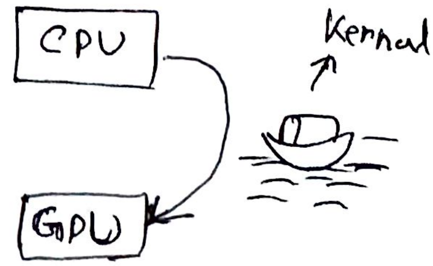
Software Arch/GPU Mem Structure:



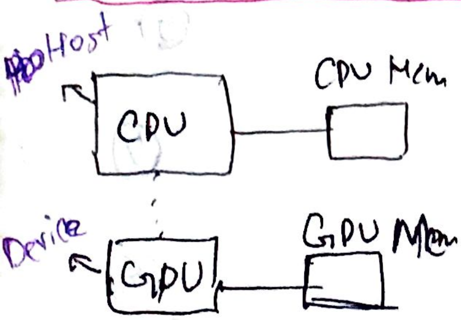
H/W	S/W
~~GPU Mem~~	GPU Mem
↓	↓
GPU	Grid
↓	↓
SM	Block
↓	↓
SP	Thread
↓	
DRAM	

Mess Passing:





A Basic CUDA Program Outline:



(i) Allocate the mem for array on host.

(ii) Allocate the mem for array ~~host~~ on device

(iii) CPU ~~Mem~~ থেকে CPU Mem এ array টা fill করায়.

(iv) CPU Mem থেকে তা পাঠায় দেয় GPU Mem এ.

(v) GPU Mem থেকে তা নিয়ে GPU এর কাজ সম্পন্ন হয়ে

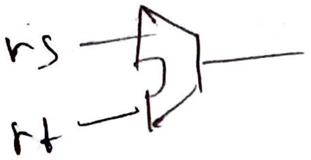
(vi) Then আবার GPU Mem এ ফেরত হয়ে সে তার পাঠিয়ে দে

(vii) CPU Mem এ CPU তে পাঠায় for correction.

(viii) Free the Host

(ix) Free the Mem

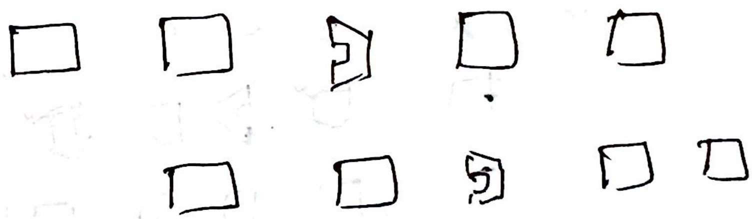
Add \$0, \$11, \$12



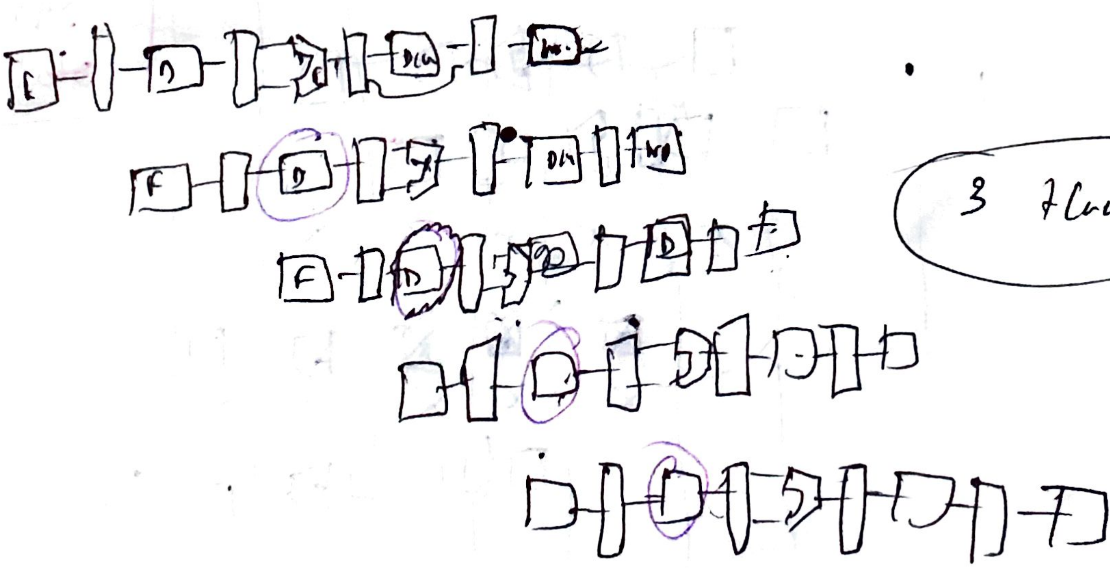
Double Data Hazards:

add \$1, \$1, \$2
add \$1, \$1, \$3
add \$1, \$1, \$4

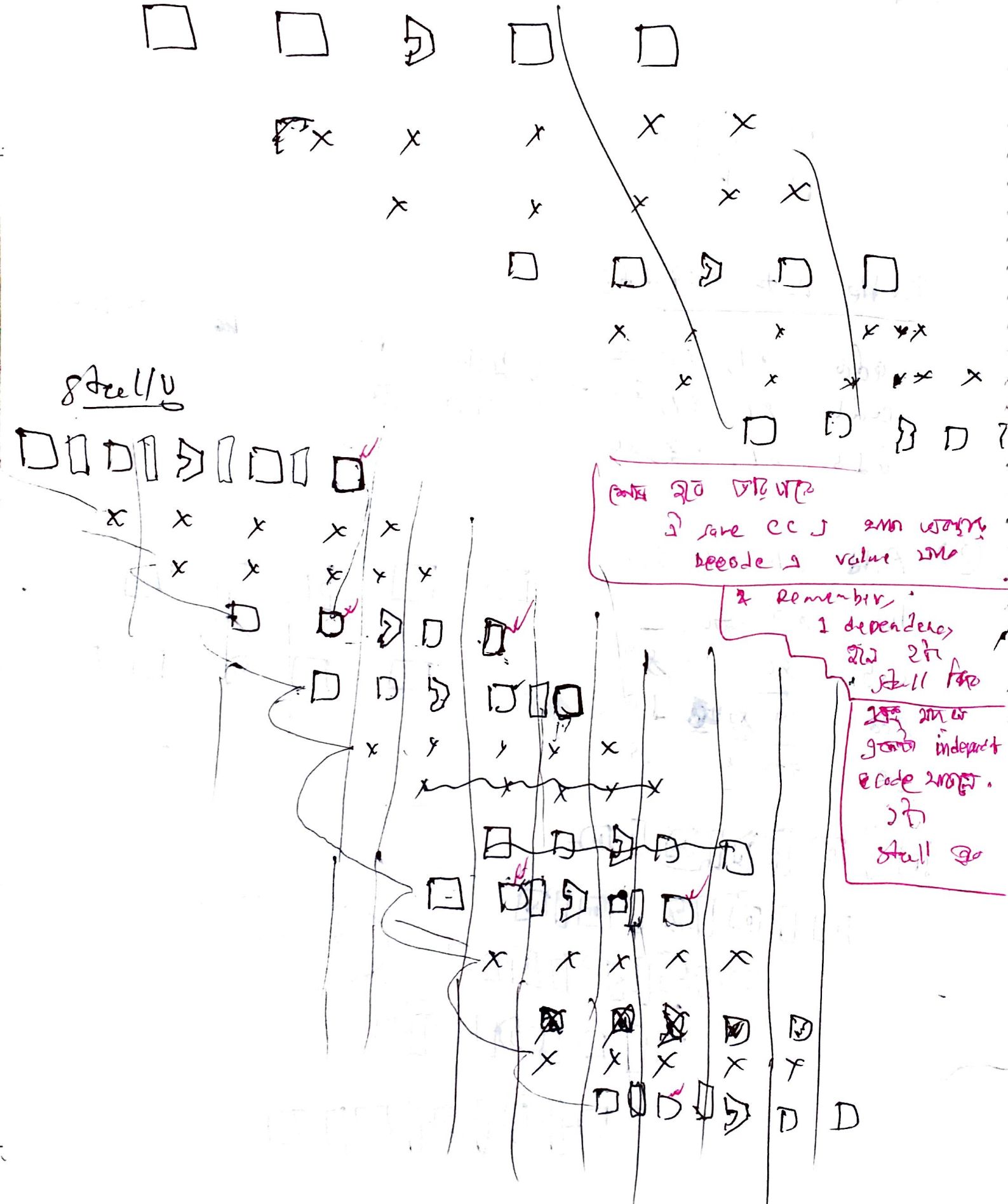
⇒ দুইটাতেই Ex থেকে ~~De~~ D তে forward করতে হবে.



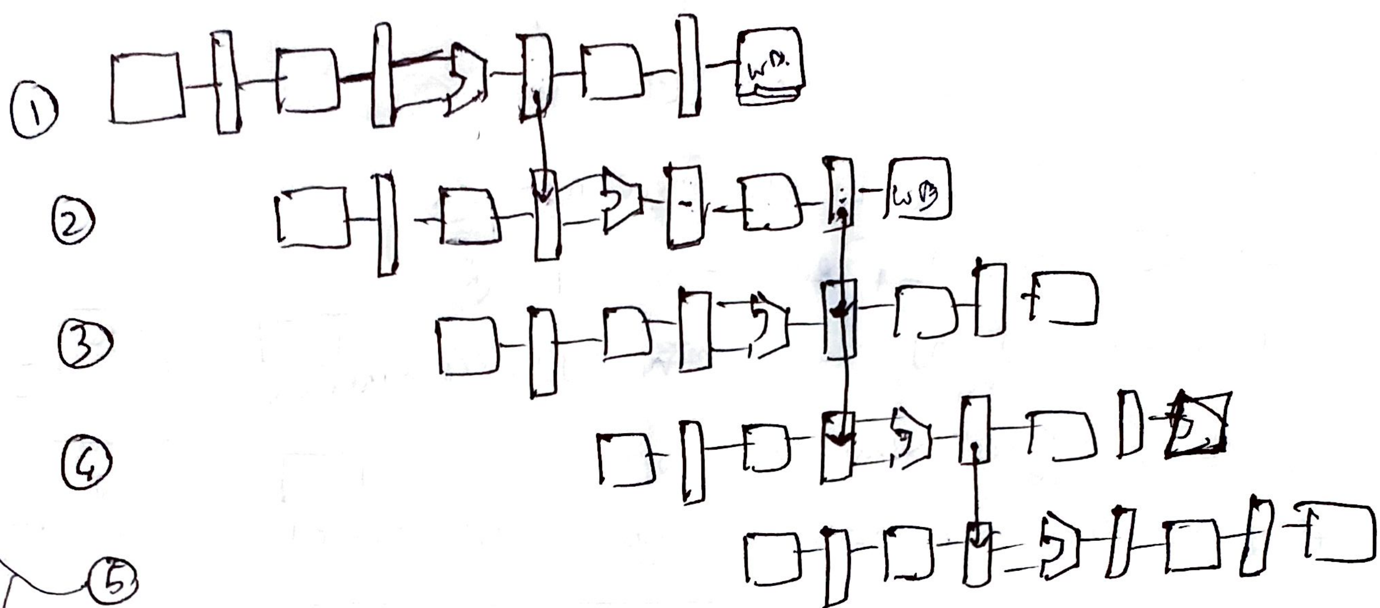
② Add \$10, \$11, \$12
lw \$7, 40(\$10)
sub \$6, \$8, \$9
Add \$9, \$7, 100
OR \$13, \$9, \$15



3 Forward


Stall

FWD & Stall:



যদি ~~reset~~ reschedule
এটি ওপর num এ same
থাকবে।