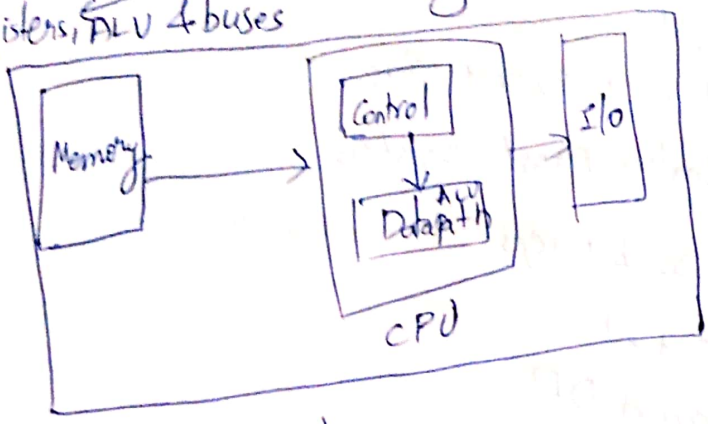
Von-Neuman model

stored program concept

The five classic components of computer system are :-

Control Unit, Datapath, Memory, I/O, CPU

Registers, ALU & buses



Architecture	Organization
Data structures Data types instruction set instruction format addressing modes	

MIPS → RISC ← { ARM, 8086
CISC

↓ Microprocessor without Interlocking Processing System

TB

Computer Organization and Design : The Hardware / Software Interface, Patterson & Hennessy

Lab

Verilog HDL : A Guide to Digital Design and Synthesis by S Palnitkar

MIPS Register Set

→ 32 bit Processor

→ 32 bit address/data

→ GPRs

- 32 regs
- R0 - R31
- Each reg is 32-bit wide

→ FPR (Floating point registers)

- 32 32-bit regs
- f0 - f31
- Paired DP

→ System Registers

- PC, hi & Lo.

GP Register File

32 registers (0 to 31)

Register names begin with a $

- $0, $1, $2, .. $31
- $a0 - $a3, $s0 - $s7

Not all registers for GPs

R0: reading return zero but cannot write

R31: used in function calls

Software convention for Registers

0	zero	constant 0
1	at	reserved for assembler
2	v0	expression evaluation &
3	v1	function results
4	a0	arguments
5	a1	
6	a2	
7	a3	
8	t0	temporary: caller saves
.	.	
.	.	
15	t7	
16	s0	callee saves (callee must save)
.	.	
23	s7	
24	t8	temporary (cont'd)
25	t9	
26	k0	reserved for OS kernel
27	k1	reserved for OS kernel
28	gp	Pointer to global area
29	sp	stack pointer
30	fp	frame pointer
31	ra	Return address (HW)

fixed instruction format (MIPS)

instruction format types < R-Type, I-type, J-type

Design 1: Simplicity favours regularity

Suppose we want to add 4 variables

① a ← b + c + d + e

```
add a, b, c
add a, a, d
add a, a, e
```

② d = a − e

```
sub d, a, e
```

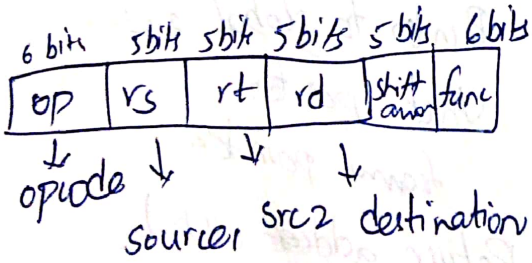
③ a = b + c

```
add a, b, c
```

④ f = (g + h) − (i + j)

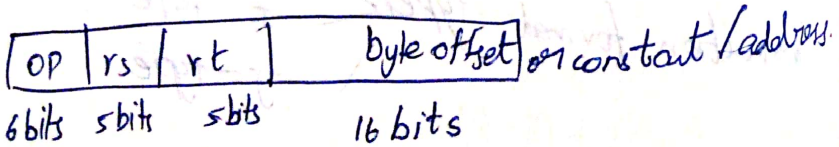
```
add  t0, g, h
add  t1, i, j
sub  f, t0, t1
```

R-type

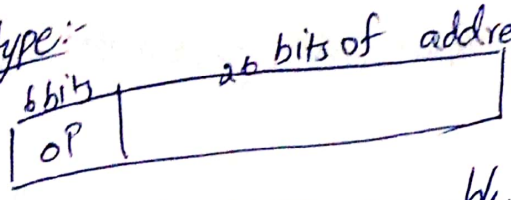


For all r-type opcode is 000000

I-type



J type:-



Translate the MIPS assembly instruction

(1) add $t0, $s1, $s2

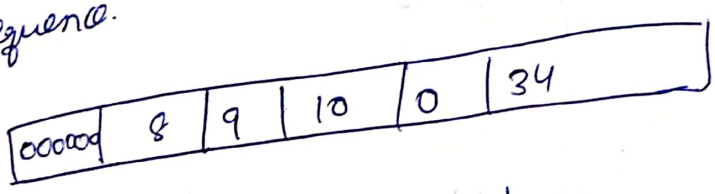
To binary instruction format

000000 10001 10010 01000 00000 100000

0 2 3 4 4 0 2 0

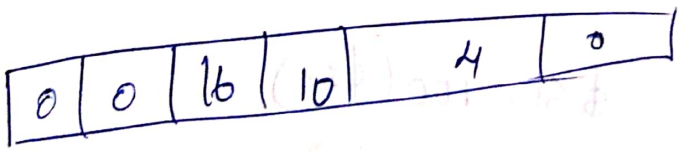
func → for sub is 34

What MIPS instruction is represented by the following bit sequence.



(2) sub $t2, $t0, $t2

(3) sll → shift left logical
(4) srl → shift right logical } rs will be zero as there are only two operands rt & rd.



(5) and — and $s1, $s2, $s3
(6) or — or $s1, $s2, $s3
(7) nor — nor $s1, $s2, $s3
(8) xor
(9) slt — slt $s1, $s2, $s3
{ comparison

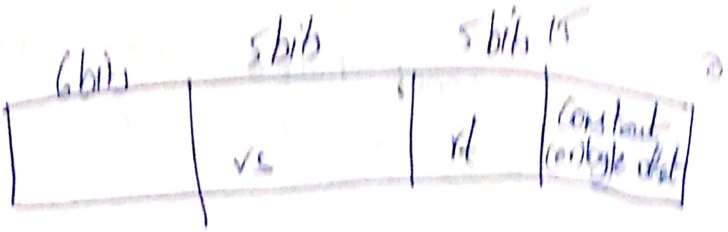
How will you the content from one register to another in mips

add $r1, $r2, $d0

or $r1, $r2, $d0

and $r1, $r2, $r2

I-Type



① addi $s1, $s2, #100

$s1 → rt, $s2 → source rs

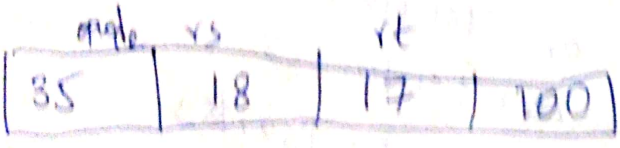
lw $s1, byteoffset($s2)

$s1 → rt, $s2 → rs

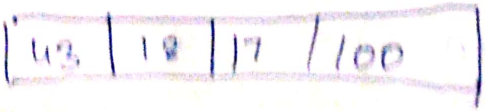
sw byteoffset($s2), $s1

byteoffset → rs, $s1 → rt

lw $s1, 100($s2)

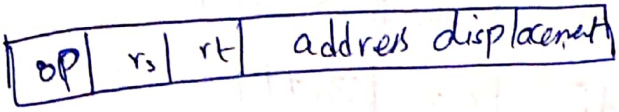


sw $s1, 100($s2)

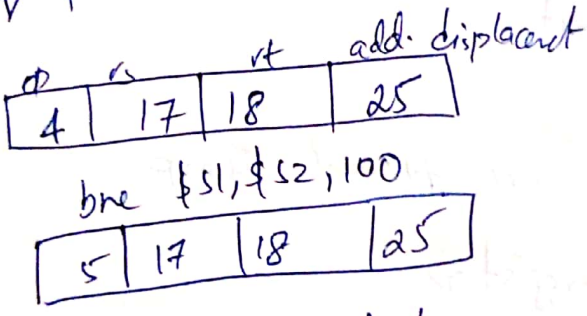


```
lw    $s3, 200($t1)
add   $s3, $s2, 300($t1)

add   $s4, $s2, $s3

sw    200($t1), $s4
```

```
beq $s1, $s2, L1
bne $s1, $s2, L2
```




```
beq $s1, $s2, 100
```



f, g, h, i, j & k are constats
so what is compiled MIPS

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

```
    beq  $s3, $s4, IF
    sub  $s0, $s1, $s2
    beq  $zero, $zero, END
IF:
    add  $s0, $s1, $s2
END:
```

Compile the following while using MIPS assembly.

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

Assume that i, k corresponding to $s3, $s5 and ba
of array save is in $s6.

```
loop: sll  $s1, $s3, #2
      add  $t1, $s1, $s6.
      lw   $s2, $s0($t1)

      bne  $s2, $s5, exit
      addi $s3, $s3, #1
      j loop
exit:
```

J → Jump unconditional Jump

Jr → Jump register

Jal → Jump and link

J imm # Jump absolute

opcode: 2

6 bit opcode, 26-bit address

new PC = (PC $0xf0000000) | immm<<2

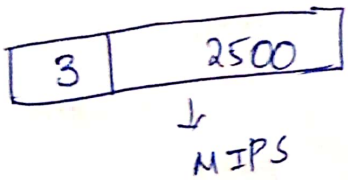
J 10000

2	2500

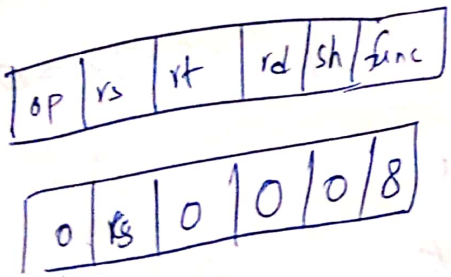
② Jal imm # Jump & Link

opcode = 3

r31 ← PC



③ Jr $rs



④ Jalr $s, $t # Jump reg and link

rd ← PC

PC ← rs

mult $s2, $s3 # Hi, Lo ← $s2 * $s3

multu $s2, $s3 # Hi, Low ← $s2 * $s3

mfhi $s1

mflo $s2

div $s2, $s3 — Lo ← $s2 / $s3; Hi ← $s2 % $s3 (Result rounded towards zero)

divu $s2, $s3

Compare & Branch Instructions :-

Compare to zero and branch :-

blez rs, offset # R[rs] <= 0 then b

bgtz rs, offset

bltz rs, offset

bgez rs, offset

bltzal rs, offset

bgezal rs, offset

how do you branch if r2 = 0

beq $r2, $zero, Offset

sub $t0, $r2, $r3

blt $t0, offset

lui - load upper immediate

↳ i-type instruction

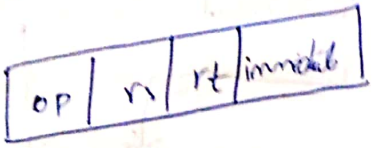
lui $t1, 255

What is the MIPS assembly code to load this 32-bits constat into reg $s0?

lui $s0, 61
addi $s0, 2304

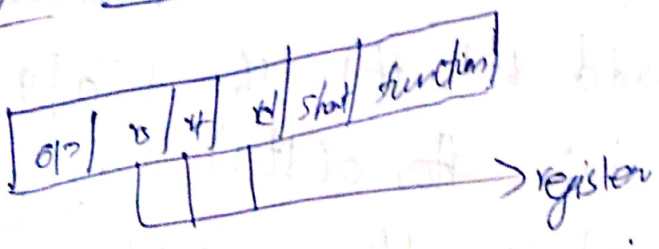
Addressing modes:-

1. Immediate addressing :- where the operand is constant within the instruction itself. The operand is 16 bit of itself



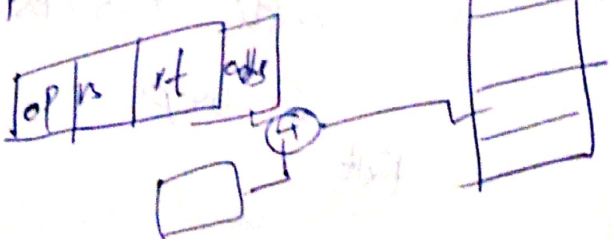
addi
ori
xori
andi

2. Register Addressing Mode :-



3. Base or displacement addressing mode

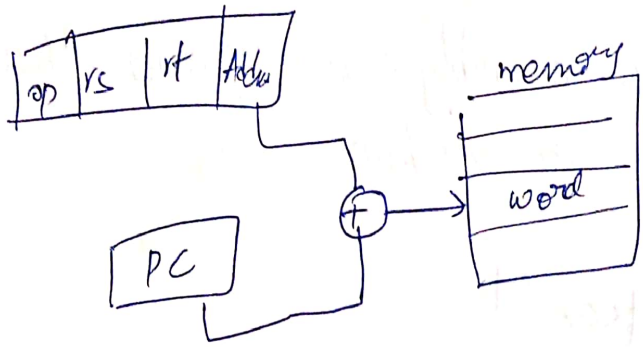
lw
sw



PC Relative addressing mode

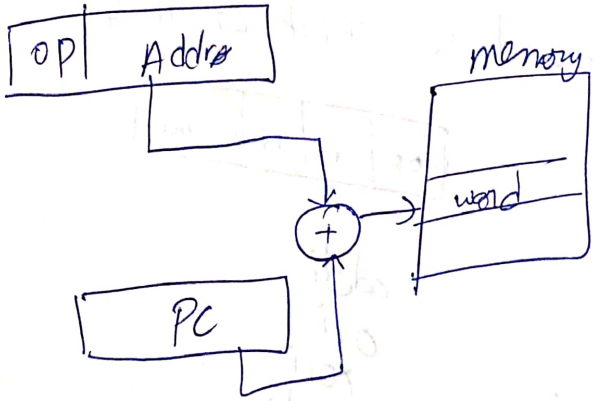
beq

bne

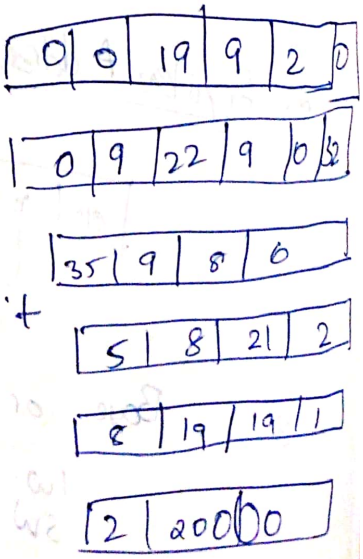


Pseudo direct addressing

where the jump address is 26 bit of the instruction which is left shifted by 2 times and then concatenated with 4 bits of PC.




```
loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    loop
Exit
```



Performance :-

Do the following changes to a CS increases throughput? (or) Decrease response time? (or) both?

Replacing processor in a computer with the faster version

Adding additional processors to a system that uses multiple processors for seperate tasks? For example: searching www

Response time or Execution time is the time between the start and completion of a task.

To maximize performance we want to minimize response time or execution time for same task

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

$$\text{Performance}_x > \text{Performance}_y$$
$$\text{Execution time}_x < \text{Execution}_y$$

If X is n times faster than y

If compute A runs a prog in 10s and computer B run is 15s runs. how much faster is A to B?

A is 1.5 times faster than B.

How do you calculate the CPU ... if only no. of clock cycles used for the Pr... is given?

n x clock cycle time

CPU execution time for a program = CPU clock cycles for a program * clock cycle

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

$$\text{clock rate} = \frac{1}{\text{clock cycle time}}$$

Our fax program runs in 10 s on machine A which has 2 GHz clock. We are trying to help a computer designer build a computer B which will run in 6 sec. The designer has determined a substained increase in clock rate is possible but this increase will effect the rest of CPU design causing computer B to require 1.2 clock cycle as A. What clock rate should we designer to target.

(4 GHz)

$$\frac{10}{6} = \frac{x}{2}$$

$$6x = 20$$

$$x = \frac{20}{6}$$

$$= 4$$

$$x = \frac{20}{6} \times 1.2 = 4 \text{ GHz}$$

Machine A has 250MHz frequency and machine B has 500 MHz frequency.

Comparing code segments :-

Suppose we have two implementations of the same ISA. Computer A has 250 ps. CPI of 2 for some program. Computer B has 500 ps and CPI of 1.2

$$\frac{A}{B} = \frac{250 \times 2}{500 \times 1.2} = \frac{1}{1.2}$$

B is faster than B by 1.2 times

CPI for each instruction class			
	A	B	C
CPI	1	2	3

Instruction counts for each instruction class

Code seq	A	B	C
1	2	1	2
2	4	1	1

A compiler designer is trying to design between two codes. The hardware designer has supplied following facts. For a particular high level language the compiler that require two code sequences

(a) which code sequence most instructions.
(b) which will be faster.
(c) what is CPI for each sequence

Time - seconds/program = $\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{clock cycles}}{\text{Instructions}} \times \frac{\text{seconds}}{\text{clock}}$

= no. of instructions × CPI × clock cycle

Components of performance :-

There are four basic concepts of performance

(i) CPU Execution time for a program (seconds)

(ii) Instruction count

(iii) clock cycles per instruction (CPI)

(iv) clock cycle time $\left(\frac{1}{\text{frequency}}\right)$

Instruction mix indicates how much percentage of instructions belong to each kind of classes.

It is a major of dynamic frequency of instructions across one or many programs.

An 800 MHz AMD processor, CPI = 1.2 for a MP3 compression. A 1 GHz pentium has CPI = 1.5. Both have identical ISA. which processor is better.

$$\frac{P_A}{P_B} = \frac{\frac{1.5}{10^9}}{\frac{1.2}{800 \times 10^6}}$$

$$= \frac{15}{10^9} \times \frac{800 \times 10^6}{12} = \frac{40}{40} = 1$$

10^{00}

What is clock cycle time for a machine with clock rate 200 MHz

$$\frac{1}{F} = \frac{1}{200 \times 10^6} = 5 \text{ ns}$$

1. Amdhal's Law:-

The performance enhancement of an improvement is limited by how much the improved feature is used.

For example:

$$\text{Execution time after improvement} = \frac{\text{Execution time effected by improvement}}{\text{Amount of improvement}} + \text{Execution time uneffected.}$$

Suppose you could speed up float point FP by 2 times but the program has 20% FP operations

$$E_{\text{after float}} = \frac{0.2}{2} + 0.8$$

$$= 0.9$$

Common case Fat → h/w
→ s/w

1. Amdhal's law
2. MIPS
3. MFLOPS
4. Benchmarks

Instruct types & CPI

What is a range CPI for following machine

OP	freq	cycles
ALU	50%	1
Load	20%	5
Store	10%	3
Branch	20%	2

$0.5 \times 1 + 0.2 \times 5 + 0.1 \times 3 + 0.2 \times 2$

$= 0.5 + 1 + 0.3 + 0.4$

$= 2.2$

MIPS - million instructions per second

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

$$= \frac{\text{Instruction count}}{10^6 \times \text{Instruction count} \times CPI \times \text{Clock time} \times 10^6}$$

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

MFLOPS → million floating point operations per second.

$$MFLOPS = \frac{\text{No of Fl. operations}}{\text{execution time} \times 10^6}$$

→ A benchmark is a standard of measurement or evaluation.

→ A computer benchmark typically a computer program that performs a strictly defined set of operations which is called a workload and

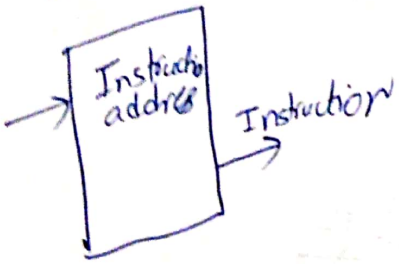
returns some form of result for example.
spec is a benchmark that is used
BIPS, BOPS, GIPS, GOPS

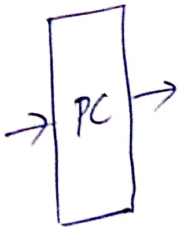
MIPS failure
↓
It doesn't take into account the instruction set
MIPS rate depends on the programs and its instruction mix
MIPS are not constant on a style machine
MIPS can vary inversely with performance.

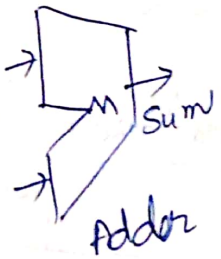
Instruction Execution

IF + ID + execute + memory ptr + write back
↓
FO

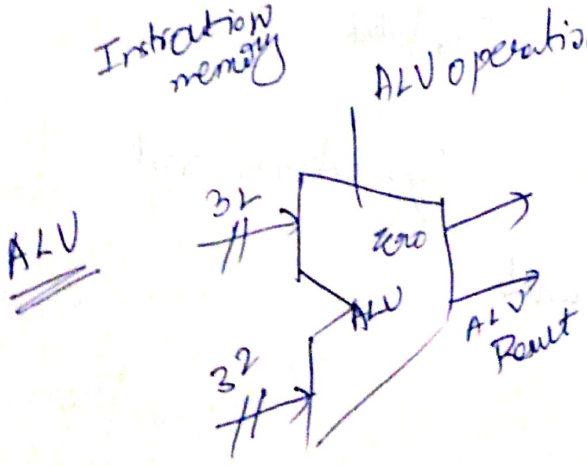
Elements of datapath:-



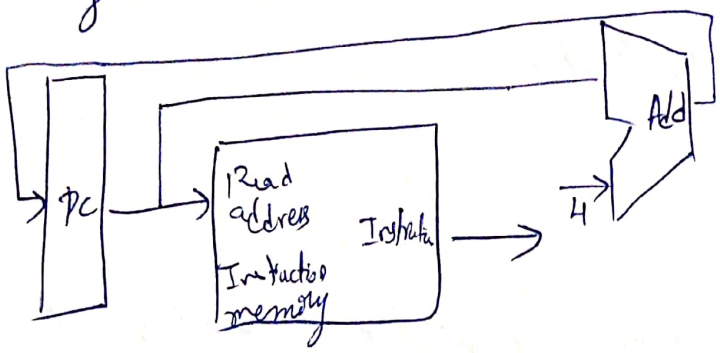




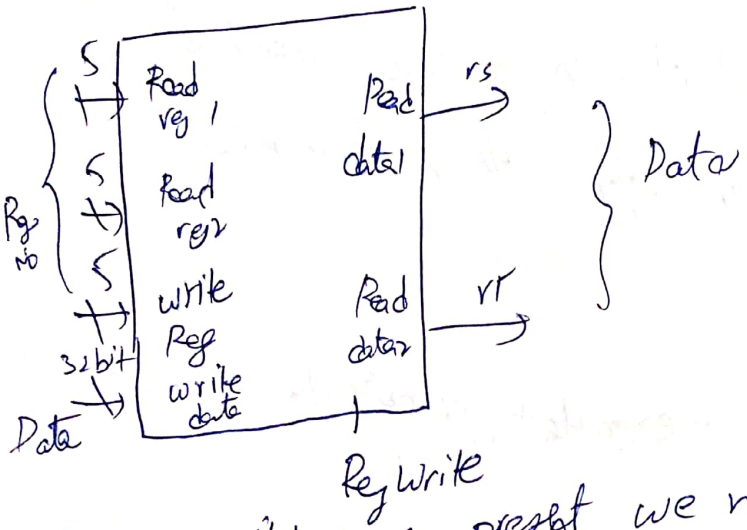
Instruction memory



Fetching and Incrementing the PC :-

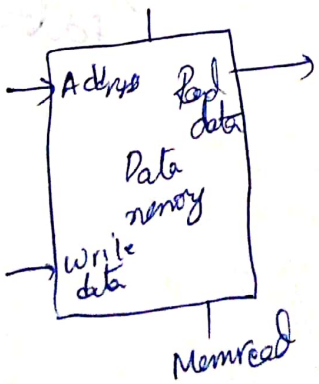


Registers

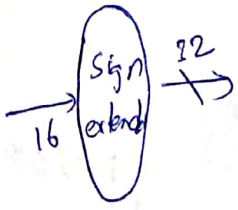


Whenever writeback is present we need to enable RegWrite.

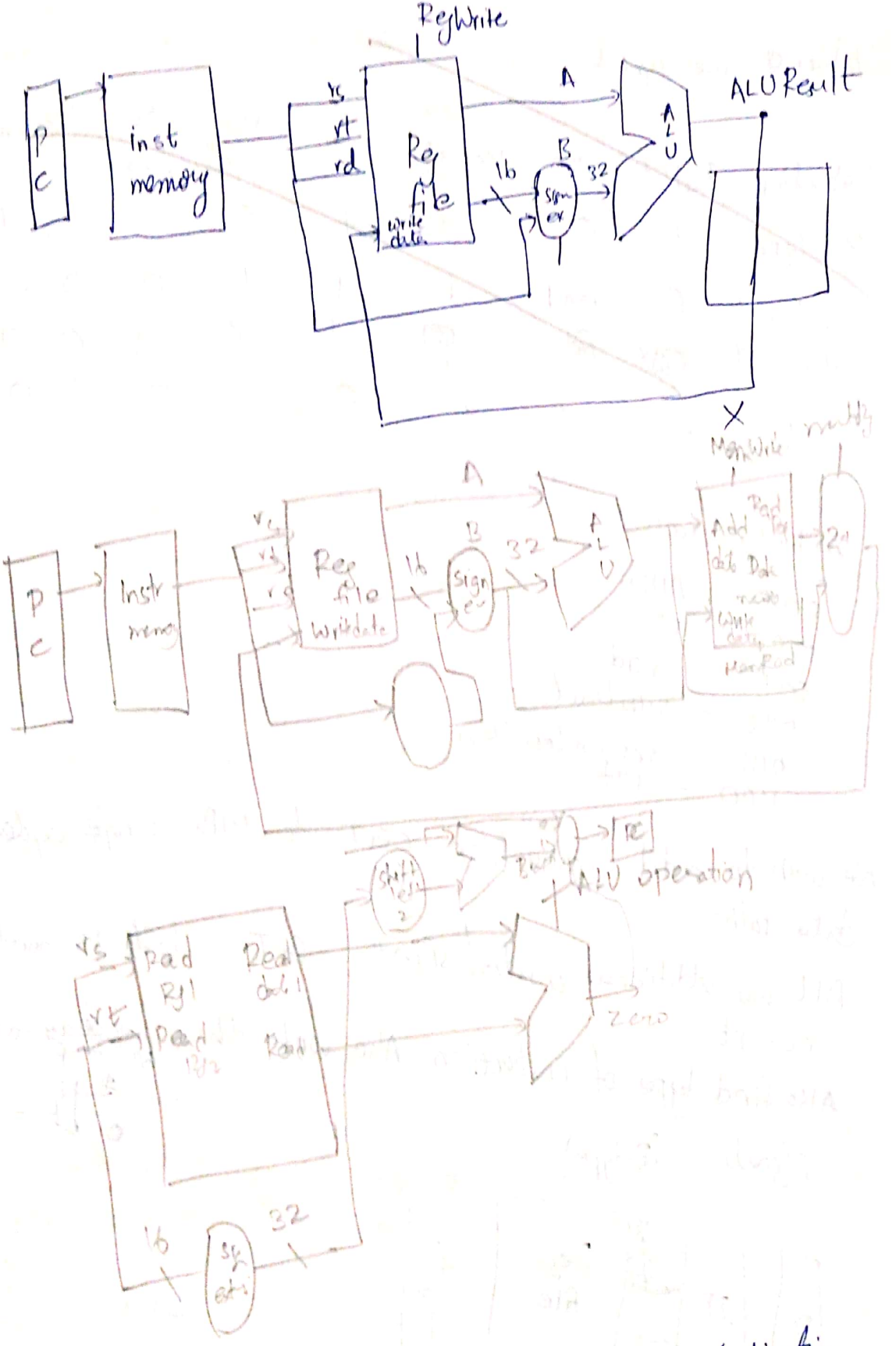
Data Memory Unit :-



Data memory unit



sign extension unit



RegDest → decides whether rt or rd is the destination.

Branch

MemRead

MemtoReg

ALUOp

MemWrite

ALUOp1	ALUOp2	
0	0	→ add
0	1	→ subtrac
1	0	→ R-Type

Values of various signals

Instruction	RegDst	ALUSrc	MemtoReg	Reg write	Mem Read	Mem write	Branch
R-Type	1	0	0	1	0	0	0
loadword	0	1	1	1	1	0	0
storeword	x	1	x	0	0	1	0
beq	x	0	x	0	0	0	1

4-bit ALU control

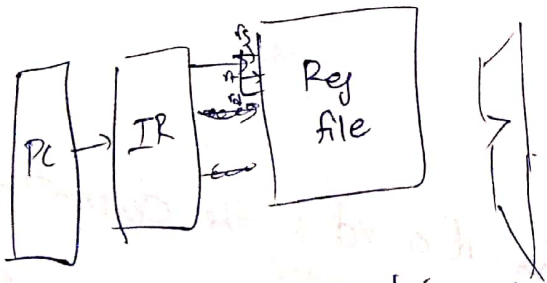
0000 - AND
0001 - OR
0010 - add
0110 - subtract
0111 - set on less than
1100 - NOR

We wish to add instruction CGT to MIPS single cycle data path.

ALU has additional control signal GT which is asserted

rs > rt

Also find type of instruction. Also write the necessary control signals. (R-type)



Find out the cycle time for single datapath design when memory units 200ps, ALU adder, 100ps reg file Assume other wires have no delays. ~~200+100+50~~ = 350 p.s.

	IF	ID	EX	MEM	WB	total
R-type	200	50	100	X	50	400

addi	200	50	100	×	50	400ps
lw	200	50	100	200	50	600ps
sw	200	50	100	200	×	550ps
B	200	50	100	×	×	350ps
J	200	50	100			200ps

clock cycle time = 600ps

Which of the following implementation will be faster & by how much.

① SCI

② An implementation where every instruction execution in one cycle using variable length clock

To compare the performance assume following instruction mix

- 25% load
- 10% store
- 45% ALU
- 15% branch
- 5% Jump

SCI takes 600ps — 600ps

2nd one takes 447.5ps — 447.5ps

$\frac{2nd\ Imp}{SCI} = \frac{600}{447.5} = 1.34$ times

	R-type	ori	lw	sw	beq	Jump
Rtype	1	0	0	×	×	×
ALUSRC	0	1	1	1	0	×
MemtoReg	0	0	1	×	×	×
RgWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Jump	0	0	0	0	0	1

Assume the following memory access is 2ns, register read is 1ns, ALU function takes 2ns

How much time it will take for the following instructions in single cycle data path.

Add/sub $2+1+2+1 = 6$ ns

ori $2+1+2+1 = 6$ ns

load $2+1+2+1+2 = 8$ ns

store $2+1+2+2 = 7$ ns

Branch $2+1+2 = 5$ ns

What will be the time for critical path?

8 ns

Arithmetic inst – 48%

load – 22%

store – 11%

Branch – 19%

If program has million instruction?

Execution time = ?

no.of instructions × CPI × clock cycle

$= 10^6 \times 8 \times 10^{-9}$

$= 8 \times 10^{-3}$ sec

$= 8$ ms

Disadvantages of SCI:-

(i) Long cycle time - All instructions take as much as time as slowest instruction

(ii) It only implements simple instructions. as it does not support complex floating point operations.

(iii) It does not have fancy addressing modes like 8086.

(iv) It requires extra hardware. → memory (data + instruction)
→ separate Adders

Is MIPS a perfect major of performance?

Consider the following measurements for a program

Measurement	Computer A	Computer B
IC	10 billion	8 billion
Clock rate	4 GHz	4 GHz
CPI	1.0	1.1

(a) which computer has higher MIPS rating

(b) which computer is faster.

$$MIPS = \frac{IC}{Execution \times 10^6}$$

$$= \frac{10 \times 10^9 \times f}{n \times CPI \times 10^6}$$

$$= \frac{4 \times 10^9}{10^6}$$

$$= 4000$$

$$\frac{4000}{1.1} = \frac{40000}{11} = 3636$$

$$\frac{10 \times 10^9 \times 1}{4 \times 10^9} = 2.5 \text{ s}$$

$$\frac{2}{\frac{8}{4} \times 10^9 \times 1.1} = 2.2 \text{ s}$$

Computer B is faster.

	A	B	C
CPI	1	2	3

	IC count in billion		
	A	B	C
Compiler 1	5	1	1
Compiler 2	10	1	1

4 GHz

10 billion

15 billion

(a) Which code sequence will execute fast acordig to MIPS

(b) Which is faster according to execution time

Compiler 1 → $\frac{5}{7} \times 1 + \frac{1}{7} \times 2 + \frac{1}{7} \times 3 = \frac{10}{7}$

Compiler 2 → $\frac{10}{12} \times 1 + \frac{1}{12} \times 2 + \frac{1}{12} \times 3 = \frac{15}{12}$

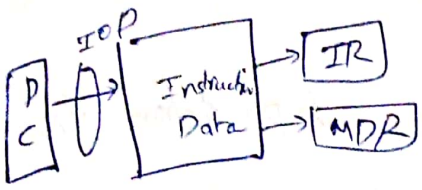
$\frac{4 \times 10^9}{10^6 \times \frac{10}{7}}$ = 2800

$\frac{4 \times 10^9}{\frac{15}{12} \times 10^6}$

(b) Compiler 1 → $\frac{10}{4} = 2.5$ s

Compiler → $\frac{15}{4} = 3.75$

www.spec.org



① Instruction Fetch

IR ← Mem[PC]
PC ← PC + 4
} clockcycle 1

② Instruction Decode & Register Fetch:-

A ← Reg[IR[25:21]]
B ← Reg[IR[20:16]]
ALUout ← PC + (signextend(IR[15:0]) << 2)

③ Execution, Memory address computation, or branch completion

if (A == B) PC ← ALUout

ALUout ← A op B

ALUout ← A + signextend(IR[15:0])

④ Memory access or R-type instruction completion step :-

Reg[IR[15:11]] ← ALUout

For sw → mem[ALUout] ← B

MDR ← Mem[ALUout]

(5) Reg[IR[20:16]] ← MDR

Fig 5.30 is a summary of the steps taken to executes only instruction class.

For Jump instruction in step 3

PC ← {P[31:28], IR[25:0], 00}

Suppose an instruction mix is 25% load, 10% stores, 11% branches, 2% jumps and 52% ALU. Calculate avg CPI. for a multicycle datapath.

$0.25 \times 5 + 0.1 \times 4 + 0.11 \times 3 + 0.02 \times 3 + 0.52$

$= 1.25 + 0.4 + 0.33 + 0.06 + 0.52 \times 4$

4.12

1, 1.25, 0.4, 0.33, 0.06, 2.04

Avg CPI for multicycle = 4.12

2.04 + 0.52 × 4 = 2.08

Suppose you have thousand instructions to be executed compared the performance of style cycles implementation vs multicycle vs pipeling.

Pipeling

Time space diagram

1 → IF ID IE MEM WB

2 → IF ID IE MEM WB

IF ID IE MEM

1. Throughput
2. Latency

For single cycle implementation clockcycle time is 8 ns.

$1000 \times 8 \times 10^{-9} = 8\mu s$

$= 8000\ ns$

$1000 \times 4.12 = 4120$ cycles

$= 4120 \times 2$

$= 8240\ ns$

$1000 \times 2 = 2000 + 8$

$= 2008\ ns$

Pipelin latency for n instructions

n → instruction in k stage pipeline and suppose we have t as cycle time. time taken

$= kt + (n-1)t$

$$\text{speedup} = \frac{\text{time taken by unpipelined}}{\text{time taken by pipelined}} = \frac{nkt}{kt + (n-1)t}$$

if $n \gg k$ $= k$

$n \ll k$ /

$n = k = \frac{k}{2}$

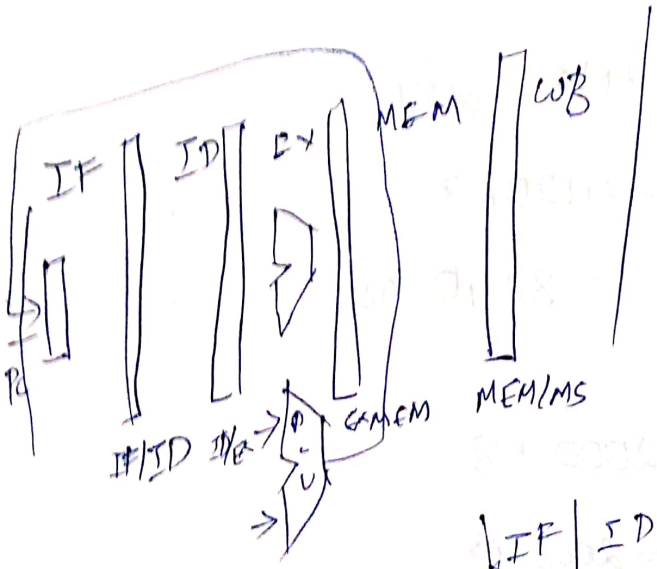
① Maximum speedup from pipelining

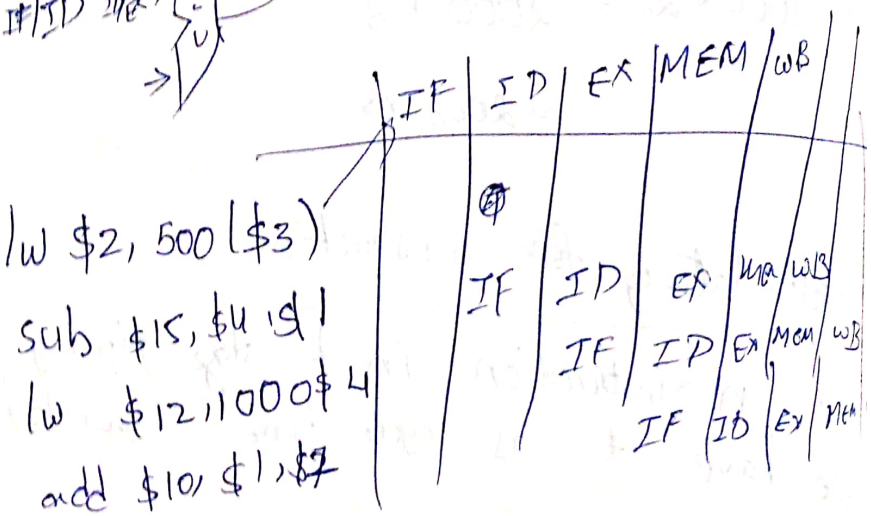
② Pipelining Advantages

③ Pipelining Datapath → Memory, Write Port } Structural Hazard

④ 3 Principles of Pipelining → Adder

⑤ What max is needed in datapath





Pipeline principles

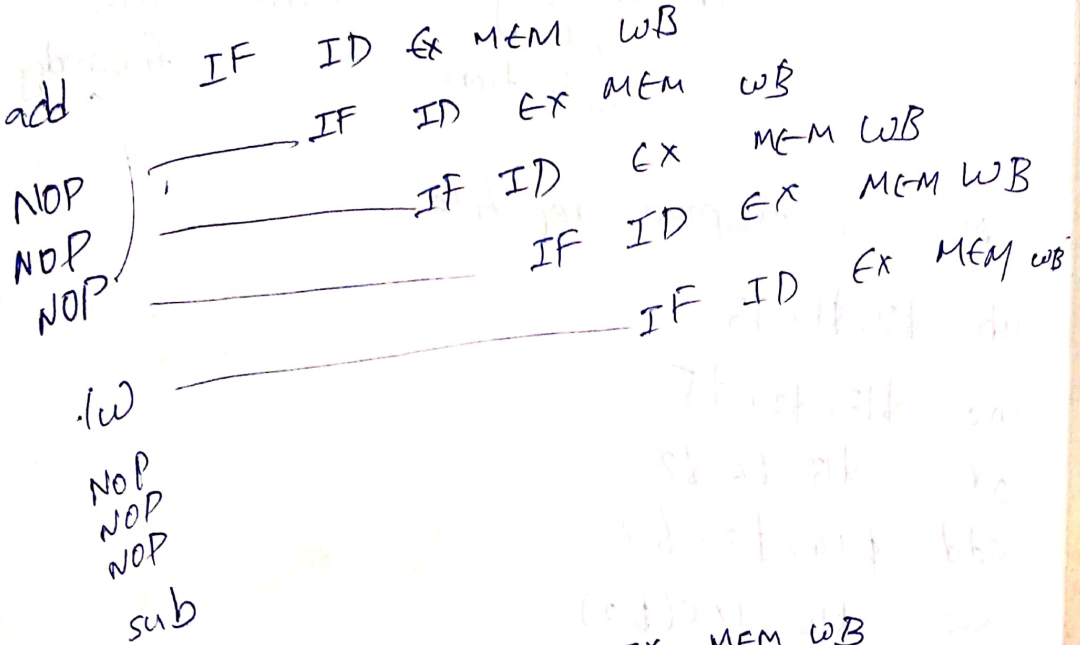
1. All instructions that share a pipeline must have the same stages in the same order.
2. All intermediate values must be latched each cycle.
3. There is no functional block reuse.

data dependency ⟶ data hazard

add $3, $10, $11 → IF ID EX MEM WB

lw $8, 50($s3) → IF ID EX MEM WB

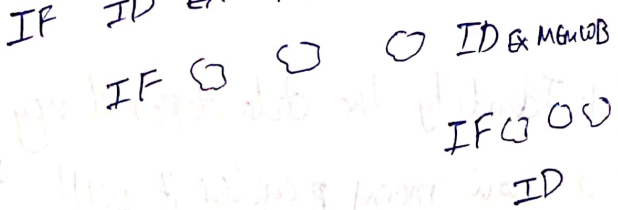
sub $11, $8, $7 → IF ID EX MEM WB



add $3, $10, $11

lw $8, 50($s3)

sub $11, $8, $7



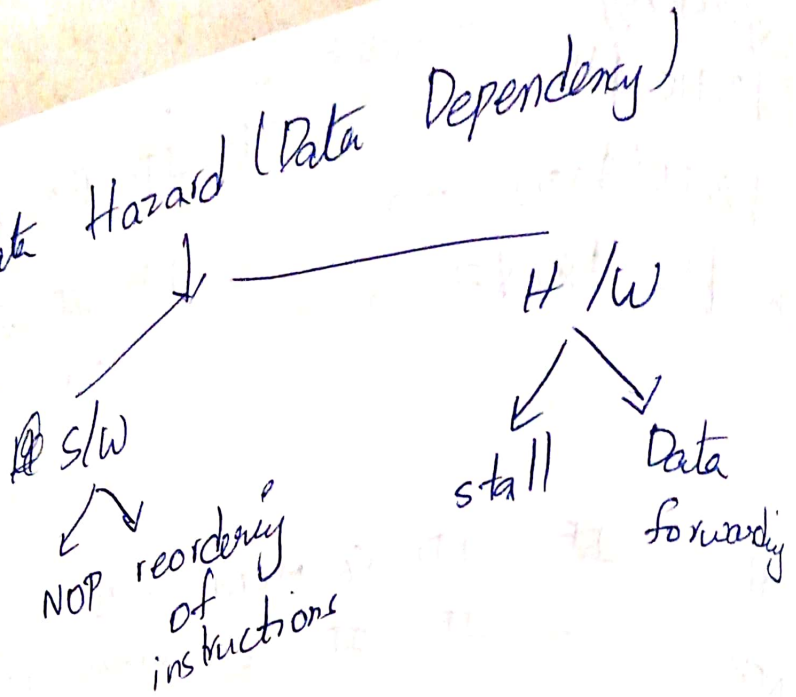
Transparent register file:-

In transparent register file in the first half of the clockcycle the register is written & in the next half register is read.

We can reduce one stall by using transparent register file.

To remove data dependency or data hazards we have two solutions one is software and other is hardware.

Data Hazard (Data Dependency)



Assume no transparent reg file

```
sub  $2, $1, $3
and  $12, $2, $5
or   $13, $6, $2
add  $14, $2, $2
sw   $15, 100($2)
```

1. Identify the data dependency of code
2. how many cycles it will take without considering transparent regfile
3. Where are no ops required

```
sub  $2, $1, $3
and  $4, $2, $5
or   $8, $2, $6
add  $9, $4, $2
slt  $1, $6, $7
```

sub IF ID EX MEM WB

and — IF ID EX MEM WB

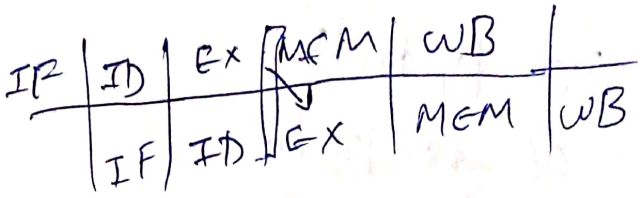
or — IF ID EX MEM WB

IF ID EX MEM WB

IF ID EX MEM WB

Forwarding (Register bypassing or short circuiting) :-

add $2,$3,$4
or $5,$3,$2

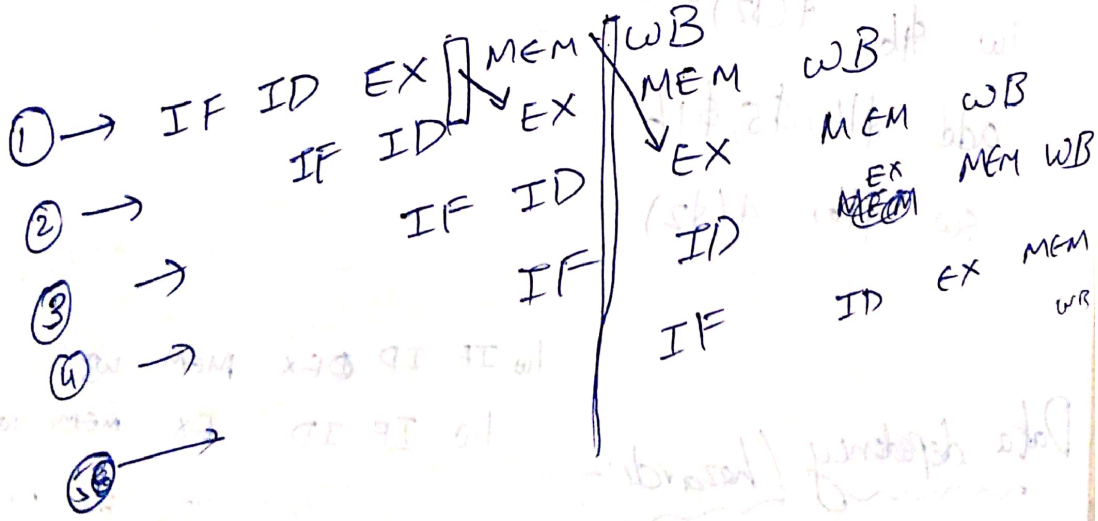


EX→EX → ALU output is → input in next

Forward A } H/w needs to be shayed.
Forward B }

(1) sub $2, $1, $3
(2) and $12, $2, $5
(3) or $13, $6, $2
(4) add $14, $2, $2
(5) sw $15, 100($2)

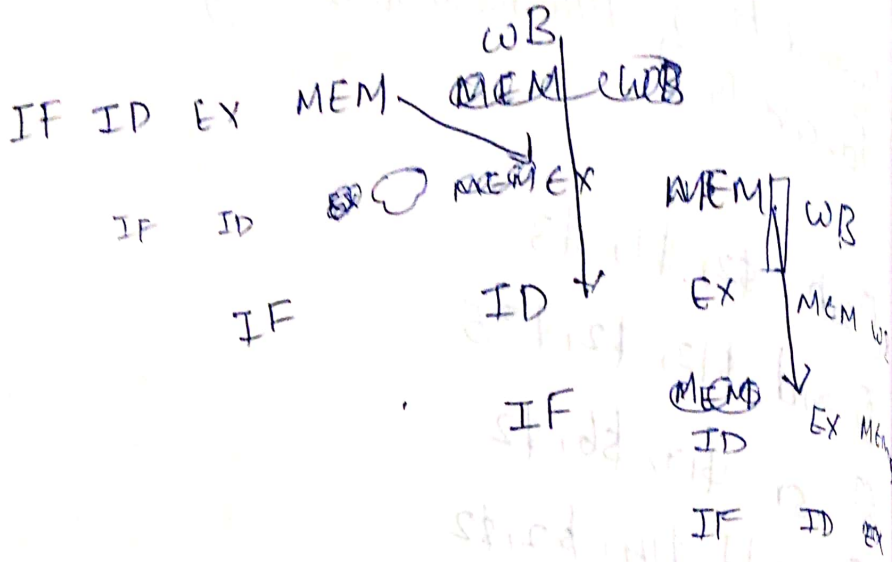
Forwarding is inplace as well as wayward by the



MEM→EX

1. lw $2, 20($1)
2. and $4, $2, $5
3. or $8, $2, $6
4. add $9, $4, $2
5. slt $1, $6, $7

EX → EX
MEM → MEM
WB → ID



After lw, we may have to put one stall.

lw $15, 0($2)
lw $16, 4($2)
add $14, $5, $16
sw $16, 4($2)

swap

lw IF ID EX MEM WB
lw IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB

Data dependency / hazards:-

Data dependency
- → RAW - Read After Write → True Data dependency
  - add $1, $2, $3
  - add $4, $1, $2
- → WAR → write after read (anti-dependency)
  - add $1, $2, $3
  - sub $2, $3, $4
- → WAW → write after write (output)

Branch Prediction

- Static Prediction
  - Forward → NOT taken
  - Backward - taken
- Dynamic
  - one bit
  - two bit

BHT - Branch History table

BTB = Branch Transition table.

```
for(i=0; i<10; i++)
{
   ==
   ==
}
```

	i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9	i=10
prediction	NT	T	T	T	T	T	T	T	T	T	T
Actual	T	T	T	T	T	T	+	T	T	T	NT

Prediction Accuracy = $\frac{\text{No. of times predicted value is right}}{\text{Total no. of times predicted}}$

$= \frac{9 \times 100}{11} = \frac{900}{11} = 81.81\ \%$

One bit branch predictor

Two bit branch predictor

Branch Buffers

Correlating & Hybrid Predictors:-

- Global Prediction bits
- Local Prediction bits
- Actual prediction
- Number of bits

Used in modern processors.

Arithmetic Operation

Perform multiplication when multiplicand is 1000 and multiplier is 1001

(1000) ← multiplicand
× 1001 → Multiplier

```
  1000
 0000x
0000xx
1000xxx
---------
1001000
---------
```

```
  0010
× 0011
------
  0010
 0010x
0000xx
0000xxx
---------
0001110
---------
```

2 × 3

```
  11
  10
× 11
----
  10
 10x
----
 110
----
```

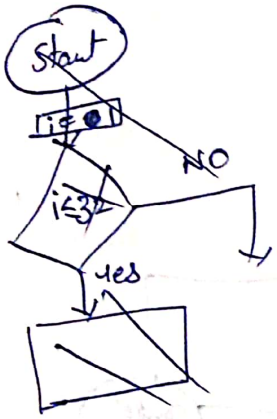
2 × 3

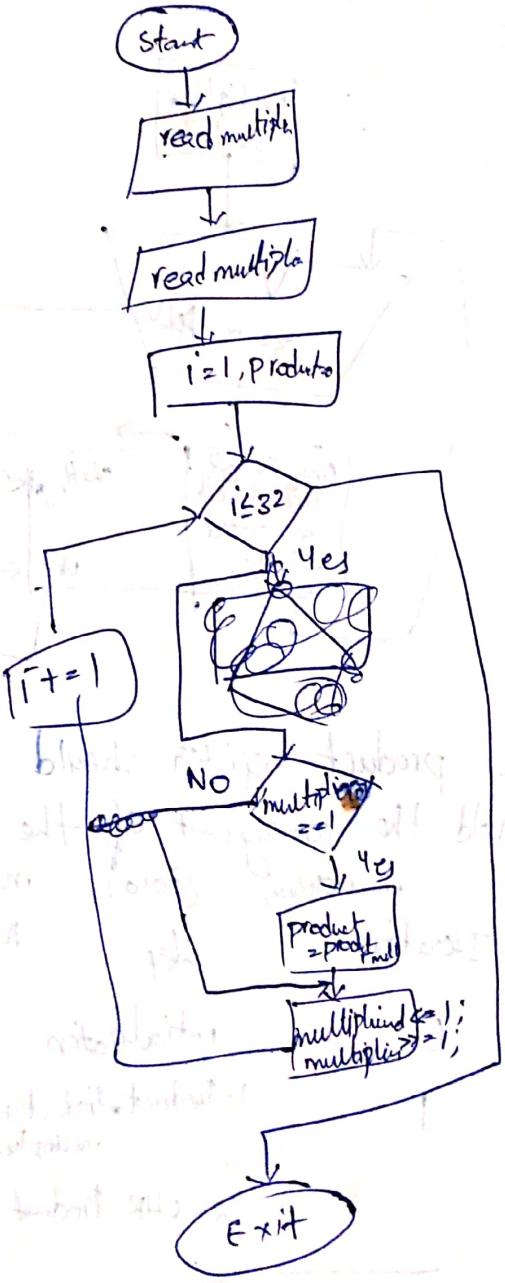
Multiplication algorithm sequential version :-

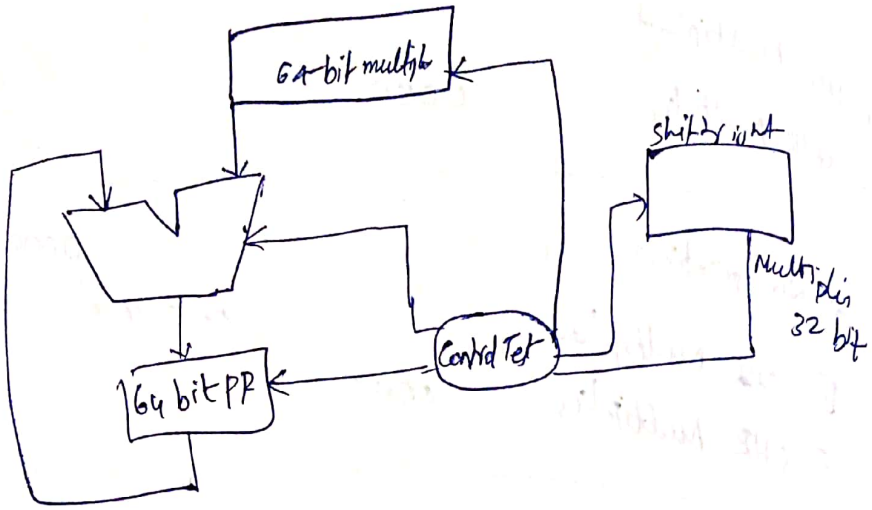
m+n bits are required to represent all possible products.

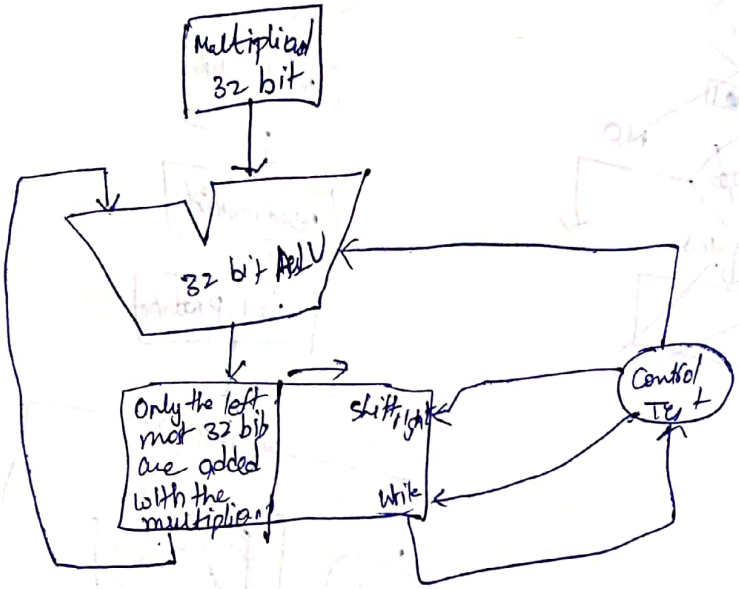
Iteration	Step	Multiplier xxxx	Multiplicand xxxx xxxx	Product xxxx xxxx
0	Intial value	0011	0000 0010	0000 0000
1	a. Product = Product + multiplicand b. SHL Multiplicand c. SHR Multiplier	0001	0000 0100	0000 0010
2.	a. Product = Prod b. SHL multiplicand c. SHR Multiplier	0000	0000 1000	0000 0110

3.	a. No operation b. SHL Multiplicand c. SHR Multiplier	0000	0001 0000	00000110
4.	(a) No operation (b) SHL Multiplicand (c) SHR Multiplier	0000	0010 0000	0000 0110









The product register should really be of 65 bit to hold the carryout of the ~~Reader~~ adder.

multiplicand (0010) multiplier (0011)

Iteration	Step	Multiplicand	Product
		xxxx	xxxx xxxx
0	initialization	0010	0000 0011
1	1. Product = Product + multiplicand	0010	0010 0011
	2. SHR Product		0001 0001
2	1. Product = Product + multiplicand		0011 0001
	2. SHR Product		0001 1000
3.	1. NOP		
	2. SHR Product		0000 1100

4. 1. NOP
2. SHR Product 0000 0110

Signed multiplication:-

multiplicand = +4 multiplier = −5

Iteration	step	Multiplicand	Product
		0×××	0××× 0×××
0	Initialization	0100	0000 0101
1	1. Product = Product + multiplicand		0100 0101
	2. SHR Product		0010 0010
2.	1. NOP		
	2. SHR Product		0001 0001
3.	1. Product += multiplicand	0100	0101 0001
	2. SHR Product		0010 1000

010100 → +20

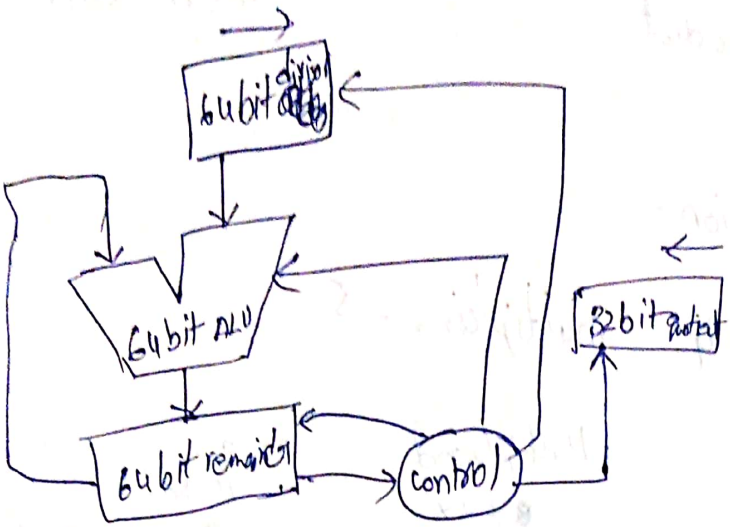
10101 1
+1
101100

Suppose the following is actual value of whether branch taken or not calculation branch prediction for one bit NT

T NT T T T T T NT T T
(NT) T NT T T T T T NT T

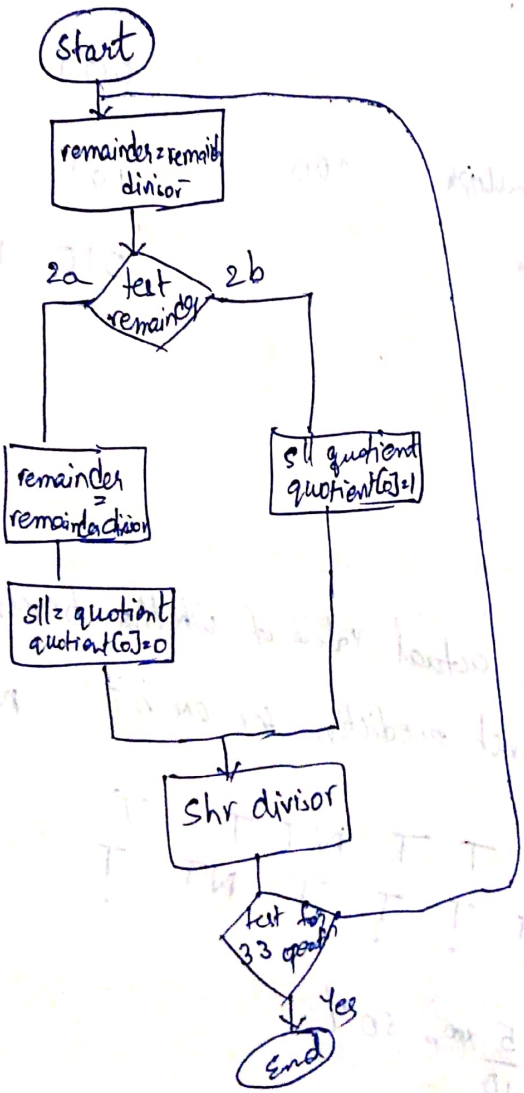
5/10 × 100 = 50%.

T NT T T T T T NT T T 60%.
NT NT T T T T T T 80%.



divisor / dividend / quotient

remainder

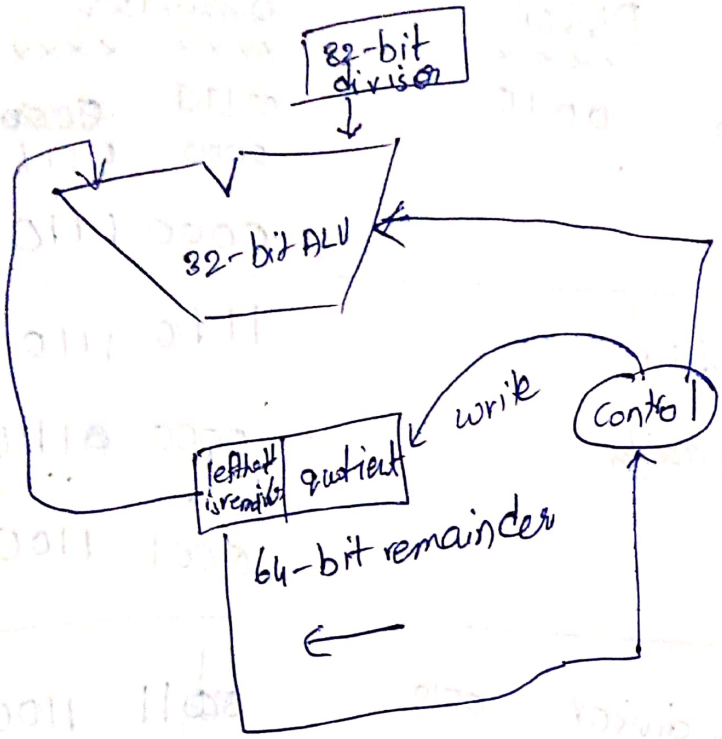


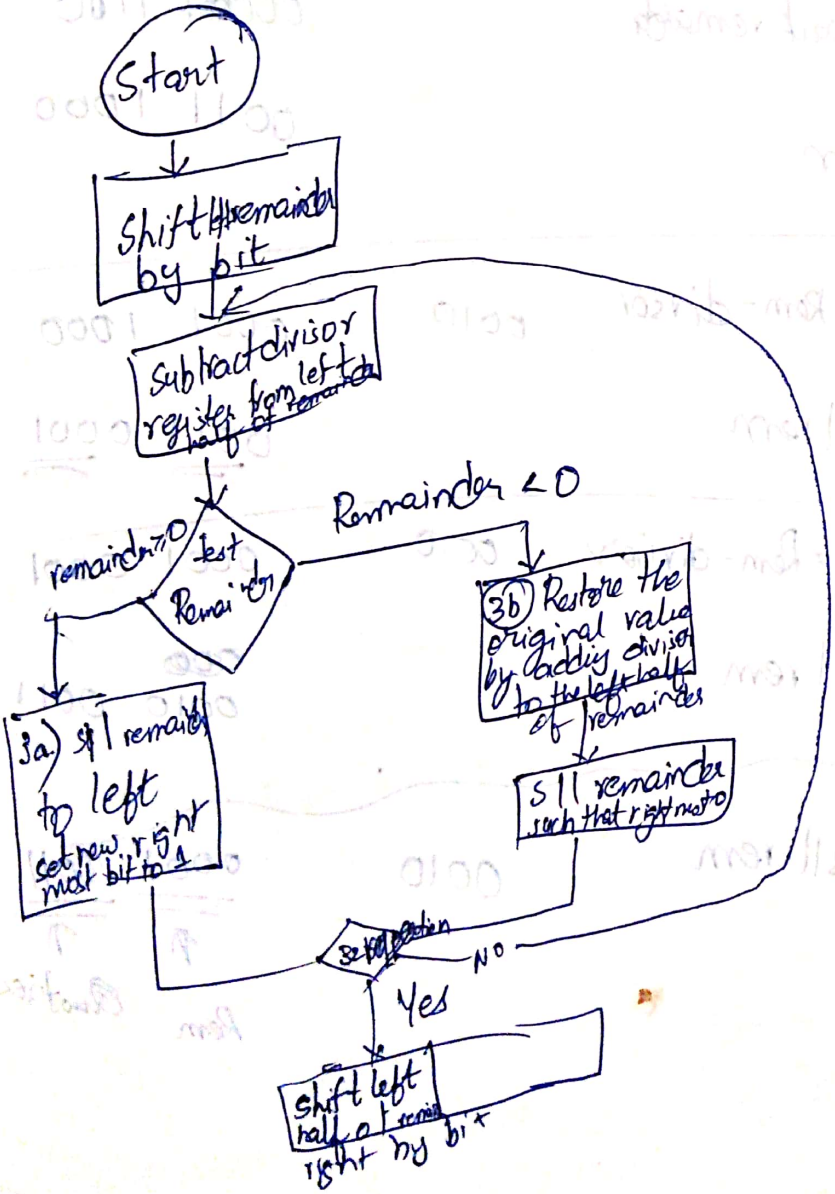
Iteration steps		Quotient	Divisor	Remainder
		xxxx	xxxx xxxx	xxxx xxxx
0	Initialization	0000	0011 0000	0000 1101
1	Remainder = Remainder - divisor		0011 0000	1101 1101
2a	Rem = Rem + divisor			0000 1101
	sll quotient	0000		
	3. shr divisor		0001 1000	
2.	Remainder = Remainder - divisor			1111 0101
2a.	Remainder = Remainder + divisor	0000		0000 1101
	shlf quotient			
	3. shr divisor		0000 1100	
3.	Remainder = Remainder - divisor			0000 0001
2b.	slf quotient	0001		
	3. shr divisor		0000 0110	
4.	Remainder = Remainder - divisor			1111 1011
2b.	remainder = remainder + divisor	0010		0000 0001
	sll quotient			
	3. shr divisor		0000 0011	
5.	Remainder = Remainder - divisor	0100		1111 1110
2b.	Remainder = Remainder + divisor			0000 0001
	3. shr divisor		0000 0001	

Divide 7/2 (Basic Division)

Iteration	steps	Quotient	Divisor	Remainder
		xxxx	xxxx xxxx	xxxx
0	Initialization	0000	0010 0000	0000
1.	1. Remainder = Remainder - divisor			1110
	2a. Remainder = Remainder + divisor	0000		0000
	sll quotient		0001 0000	
	3 shr divisor			
2.	1. Remainder = Remainder - divisor			1111
	2a. Remainder = Remainder + divisor	0000		0000
	sll quotient		0000 1000	
	3. shr divisor			
3.	1. Remainder = Remainder - divisor			1111
	2a. Remainder = Remainder + divisor	0000		0000
	sll quotient		0000 0100	
	3. shr divisor			
4.	1. Remainder = Remainder - divisor			0000
	2b. sll quotient	0001		
	3. shr divisor		0000 0010	
5.	1. Remainder = Remainder - divisor			0000
	2. sll quotient	0011		
	3. shr divisor		0000 0001	

Advanced Division Algorithm





7/2

Iteration	steps	Divisor	Remainder
		xxxx	xxxx xxxx
0	Initialization	0010	0000 0111
	sll rem		0000 1110
1.	Rem = Rem - divisor		1110 1110
	3b Get back remainder		0000 1110
	sll rem		0001 1100
2.	Rem = Rem - divisor	0010	0011 1100
	3b Get back remainder		0001 1100
	sll rem		0011 1000
3.	Rem = Rem - divisor	0010	0001 1000
	3a. sll rem		0011 0001
4.	Rem = Rem - divisor	0010	0001 0001
	sll rem		0010 0011
Final step	sll rem	0010	0001 0011

Rem Quotient

The advanced division algorithm also works for signed numbers where in while doing the computation, we ignore signs.

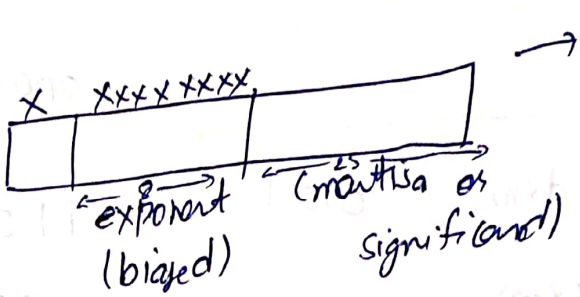
Perform 14/5 using advanced division algorithm

Iteration	Steps	Divisor	Remainder
		xxxx	xxxx xxxx
0	Initialization	0101	0000 1110
	sll rem		0001 1100
1.	Rem = Rem-divisor	0101	1100 1100
	get back rem		0001 1100
	sll rem		0011 1000
2.	Rem = Rem-divisor	0101	1110 1000
	get back rem		0011 1000
	sll rem		0111 0000
3	Rem = Rem-divisor	0101	0010 0000
	sll rem		0100 0001
4.	Rem = Rem-divisor	0101	1111 0001
	get back rem		0100 0001
	sll rem		1000 0010
Final step	sll rem	0101	0100 0010
			rem quotient

Floating point :-

IEEE-754 representation for floating points

1. Single precision floating point → 32
2. Double precision floating point → 64
3. extended double precision → 10 bytes
4. Quadruple Precision Floating → 16 bytes



→ bias 127

1101.001×2^3

↳ ① normalized no

1.101001×2^6

xxxx xxxx xxxx xxxx

0 1000 0101 1010 0100 0000 0000 000

⇒ 42D20000 → can be from 1 to 254 as 0, 255 is reserved.

Biased AE +127

= 127+6

= 133

AE range

-126 to 127

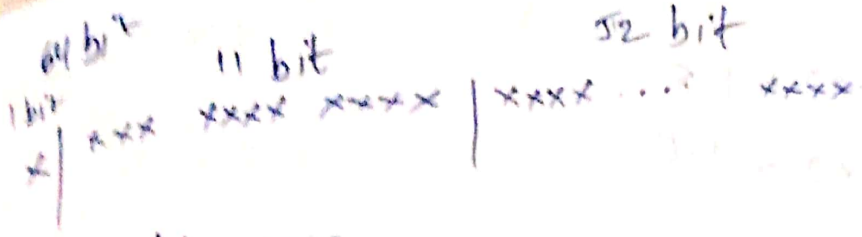
$2^{127-1} = 2^6$

16 | 133

8 - 5

1000 0101

10-A
11-B
C-12
D-13



bias = 1023

BF - AE + 1023

$(100.25)_{10}$

represent in single & double precision

2 | 100
2 | 50 - 0
2 | 25 - 0
2 | 12 - 1
2 | 6 - 0
2 | 3 - 0
1 - 1

1100100.01

1.1001000×2^6

1023

0.0000 0110

0.25 × 2 = 0.5

0.5 × 2 = 1.0

0 1000 0101 100 1000 0000 0000 0000 0000

42C80000

0 010 0000 0101 1001 0001

20591000 0000 0000

$(-0.75)_{10}$

0.75×2 = 1.5

0.5×2 = 1.0

0.11

1.1×10^{-1}

126

7 - 14

1 0111 1110 100 0000 0000 0000 01111110

0000 0000

BF400000

~~1022~~

~~1000~~

1 100 0000 1010 0000 0000 0000 0000

2

29

-1.01×10^{2}

$= -5$

① $0.11010001 \times 2^{10101}$

49D100000

② $-0.11010001 \times 2^{10101}$

(C9D10000)

③ $0.1101001 \times 2^{-10011}$

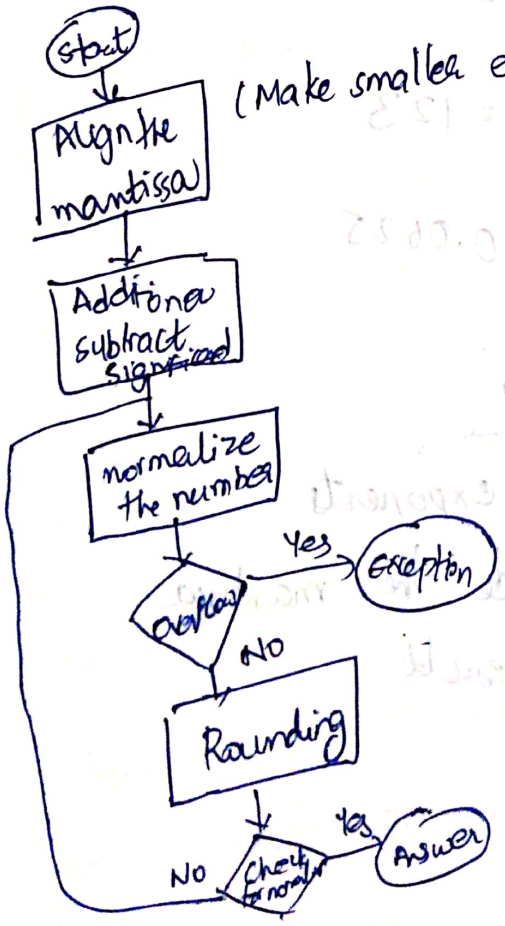
35D10000

$-0.11010001 \times 2^{-1001}$

BsD1 0000

Sigle precision		Double precision		Object Representation
Biased Exponent	Fraction	Biased Exponent	Fraction	
0	0	0	0	0 (zero)
0	Nonzero	0	nonzero	±denormalized number
1-254	Anythig	1-2046	Anythig	Floatig point
255	0	2047	0	±infinity
255	Non zero	2047	nonzero	Nan (Not a number)

Floating point addition :-



Perform the addition $(0.5)_{10}$ with $(-0.4375)_{10}$ using single precision

$0.5 = 0.1 = 1 \times 2^{-1}$

$-0.4375 = -(1.11 \times 2^{-2})$

$1.11 \times 2^{-2} = 0.111 \times 2^{-1}$

1.000×2^{-1}

0.111×2^{-1}

0.001×2^{-1}

1.0×2^{-4}

$-4 + 127 = 123$

$\frac{1}{16} = 0.0625$

Multiplication & division

(i) Add or subtract exponents

(ii) Multiply or divide the mantissa

(iv) Normalize the result

4×10^9

8×10^9

Rounding methods:-

There are four kind of roundings supported by IEEE-754 format:-

Add 3 bits at the end

g → guard bit
r → round bit
s → sticky bit (can be many) but it is 1 if any of them is 1.

(i) Round towards zero (also called truncation)

0.7783
↓
0.778 (3 places)
↓
0.77 (2 places)

1.1101
↓
1.11

1.001
↓
1.00

-1.1101
↓
-1.11

-1.001
↓
-1.00

If u have a number with g, r, s bits just leave them off

(ii) Round towards $+\infty$ (positive infinity):-

(i) It is applied to only positive numbers and no change if applied to negative numbers.

(ii) If any of g, r, s bit is 1 add 1 to LSB (obtained from exponent stage in addition/subtraction)

1. 1100 0000 0000 0000 0000 1000

(iii) Rounds towards $-\infty$ (negative infinity)

(i) Round down to infinity

(ii) If any of g, r, s bit is 1 add 1 to LSB

(iv) Rounds towards to nearest even :- (default)

	g	r	s	
(1)	0	x	x	→ does not matter r,s no change
(2)	1	1	x	→ Add 1 to LSB
(3)	1	0	1	→ Add 1 to LSB
(4)	1	0	0	→ Add 1 to LSB if LSB = 1

1 1100 0000 0000 0000 000 000 111

Add 1

1 1100 0000 0000 0000 000 000 010

No change

1 1100 0000 0000 0000 000 001 111

$[1.0 \times 2^{-1}] \times [-1.110 \times 2^{-2}]$

S	E	F
1	10000000	1100 0000 0000 0000 0000 1111
1	10000010	1110 0000 0000 0000 0000 1001

Perform addition and all four roundings

(1) round towards zero
C1940008

(2) rounds toward $+\infty$ (no change)

(3) round towards $-\infty$ (@C1940009)

(4) round to nearest even
C1940008

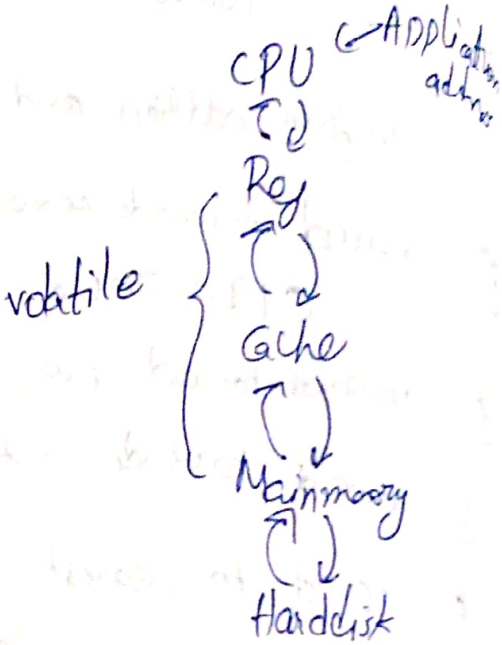
Memory

Memory Organization characteristics

- Location
- Capacity
- Units of transfer - Rate of transfer → property of communication
- Access method
- Performance
- Organisation

Access methods

Sequential
Random
Direct
Associative



Performance

~~Associative~~ Access time (t_a)
Memory cycle time (t_c)
cycle time = t_a + recovery
Transfer rate (b) $b = \frac{w}{t_c}$

Direct Mapping

(Blockaddress) % No. of blocks

set associative cache:-

no-of sets

set - at least two blocks

block - two way set associative mapping

4 blocks → direct
↓ → two way
fully associative

no-of misses

0, 8, 0, 6, 8

0000 1000 0000 0110 1000

Tag 1	data	Tag 2	data	Tag 3	data	Tag 4	data
0		8		6			

(3) misses
2 hit

	Tag 1	data	Tag 2	data
0	011		100	
1				

3 misses
replace for 6

110

direct map

0 → miss
8 - miss
0 → miss
6 → miss
8 → miss

5 misses

4C's model

Compulsory miss:- The misses that happen when the block is accessed for the first time.

Capacity miss:- The miss that happen when the capacity of cache is reached.

Suppose the cache has 1024 words and block of 1 word. How many bits are used to the cache. Suppose there are 2^{32} words in memory. How many bits are there for tag

No. of cache blocks = $\frac{1024}{1}$ = 1024

No. of bits for cache = 10

No. of bits for tag = 32 - 10 = 22

Block offset :- It indicates the no. of bits those are required to address each byte in a block.

cache bits = 10

tag bits = 30 - 10 = 20

(32-2)

2 for block offset

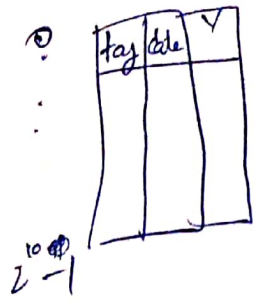
How may total bits are required for a direct mapped cache with 16kB of data and 4 word blocks. Assume (16×2^{10}) a 32 bit address.

how many bytes per block = 16 bytes

Size of cache = 16×2^{10}

No. of cache lines = 2^{10}

Tag bits = 32 − 10 − 4 = 18 bits.



Bits for one entry = 18 + 1 + 16 × 8

= 147 bits

total bits = $2^{10} \times 147$

= 18.4 kB

Performance

$T_{access} = t_{hit} + P_{miss} \times Penalty_{miss}$

Avg memory access = Hit time + Miss Rate × Miss penalty L1

Miss penalty L_1 = Hit time L_2 + Miss Rate L2 × Miss penalty L2

Suppose

L_1 hit time = 1 cycle

L_1 miss rate = 5%

L_2 hit time = 5 cycles

L_2 miss rate = 15%

main memory penalty = 100 cycles

penalty L_2 = 5 + 0.15 × 100

= 20 cycles

A

L_1 hit time = 1

L_1 miss = 5%

L_1 miss penalty = 100

Access = 1 + 0.05 × 100

= 6 cycles

pendrive

Ans. 1 + 0.05 × 20

= 1 + 1

= 2 cycles

4 C's model

1. Compulsory
2. Conflict
3. Capacity
4. Coherence