Chapter 3

Arithmetic for Computers

এই chapter টা Calculative হবে Calculation based
Overflow detect করার জন্য

Arithmetic for Computers.

⇒ Operations on ~~Computer~~ integers.

- Addition & Substraction
- Multiplication & Division.
- Dealing with Overflow

⇒ Floating Point Real numbers

- Representation and operations.

This chapter will deal with these things.

এই chapter এ আমরা signed number নিয়েই deal করবো

Addition & Substraction:

Integer Addition:

For this example we are considering 6 bit arch

7 + 6

```
 000 111 → 6 bit representation of +ve 7
 000 110 ⇒ 6 bit representation of +ve 6
────────
 001 101
```

MSB bit 0 হলে Positive

MSB bit 1 হলে negative

দুইটি number addition বা substract করার সময় wrong result আসলে overflow.

২টি positive number add করলে result positive আসার কথা, in some cases result negative আসতে পারে; কিন্তু তা তো হওয়ার কথা না। বিধায় এখানে Overflow হয়েছে।

Similarly, ২টি negative number add করলে result negative আসার কথা; but somehow result positive আসলে Overflow.

- Integer Addition
- Integer Substraction.

একটা n bit system নিয়ে কাজ করলে তার unsigned number এর range

$-2^{(n-1)}$ to $+2^{(n-1)}-1$

সাধারণভাবে আমরা কিছু arithmetic operation করলে যে result আশা করি তা না এসে ভুল result আসে, তাকে বলে overflow.

Integer Addition.

** ৩টা কারণে Overflow হতে পারে।

- If result out of range.
- Adding a positive number and a ~~negative~~ Positive number

  MSB bit

  ↳ Overflow if $\boxed{\text{result sign}}$ is 1

  ↳ একটা (+ve) এর সাথে একটা (+ve) যোগ করলে result (+) আশা করা তাই আসে result sign 0 আশা করা.

- Adding two negative numbers

  ↳ Overflow if result sign is 0.

  ↳ দুইটা (-ve) number যোগ করলে result (-ve) আশা করা; তাই আসে result sign 1 হওয়ার কথা.

যদি একটি positive number এর সাথে একটি negative number add করি তাহলে no overflow

প্রত্যেকটি calculation binary তে করছি, Signed binary number নিয়ে deal করছি.

Considering 4 bit architecture

7 + 6

$$\begin{array}{rcccc} & 0 & 1 & 1 & 1 \\ + & 0 & 1 & 1 & 0 \\ \hline 0 & \boxed{1} & 1 & 0 & 1 \end{array}$$

↳ এখানে MSB bit টা 1 → আসে 2টা positive number add করার পর neg number পাচ্ছি.
So, overflow.

→ Actually result টা 01101 আসছিলো কিন্তু,
4 bit arch এর কারণে 4 discard হয়
overflow হচ্ছে, so বাকি portion অনুযায়ী

কত bit architecture follow করছি সেটা উল্লেখ রাখতে হবে.

Examples:

↳ নিচের example গুলোর জন্য 5 bit architecture consider করছি

∴ range for 5 bit signed number:

$$-2^{(5-1)} \text{ to } 2^{(5-1)}-1 = -16 \text{ to } 15.$$

$11 + (-1) = 10$

→ 10 is inside the range.

$$\begin{array}{rl} 01011 & \longrightarrow 11 \\ +\ 11111 & \longrightarrow -1 \\ \hline \boxed{1}\ \boxed{0}1010 & \end{array}$$

↳ Updated MSB as 5 bit arch

~~Result 6 bit but system~~
~~5 bit so first এর 1 কাটা~~
~~দিতে, so বাকি 5 bit অনুযায়ী~~
~~consider করবো,~~

Steps:-

- আগে range বের করে নিতে হবে
- Then calculation করতে হবে।
- Check করতে হবে result out of range কিনা। যদি range এ থাকে then check করবে result sign টা। তারপর দিয়ে করে judgement হবে either it's overflow or not.

MSB bit বা result sign 0 so positive আসার কথা positive ই আসছে so no overflow।

Binary to Decimal Conversion জানতে হবে,

(P-7-0)

▨ Example 2 [Adding two positive numbers]

$11 + 15 = 26$

⇒ Initial decision : Over flow ; as 26 is out of range.

$$\begin{array}{rl} & 01011 \rightarrow 11 \\ + & 01111 \rightarrow 15 \\ \hline & \textcircled{1}1010 \end{array}$$

↳ MSB টা Result sign

$\textcircled{-}\{(1\times2^4) + 1\times2^3 + (1\times2^1)\}$

$= -26$

'-' আসছে bcuz signed number

এখানে আমরা ২টা positive number add করতেছি so result ও positive আসার কথা ; but calculation করার পর দেখা গেল Result sign is 1 ; which indicates it's an negative number.

∴ So overflow confirmed.

▨ Example 3 [Adding two negative number]

$-11 + (-15) = -26$ (Overflow)

$$\begin{array}{r} 10101 \\ 10001 \\ \hline \textcircled{1}00110 \end{array}$$

এই extra 1 টা discard হবে যেহেতু an 5 bit arch, বাকি portion নিয়ে check করব,

result আসার কথা negative । But result sign ~~[illegible]~~ is 0. which indicates positive number.

∴ Overflow

Integer Substraction:

- 2 টা negative number এর মধ্যে substract করলে বা 2 টা positive number এর মধ্যে substract করলে no overflow

- Integer Substraction এ 3 ভাবে overflow হতে পারে।

  ⇒ Out of range.

  ⇒ Substracting a positive value from a negative value

  - If result sign is 0 then Overflow.

  ⇒ Substracting a negative value from a positive value.

  - If the result sign is 1 then overflow.

- 8 − (−4) = 12 আসার কথা.
- −8 − (+4) = −12 আসার কথা but আসছে positive so Overflow.

Examples:

▨ 3 - (-15) = 18 [Substracting neg number from a pos number]

⇒ 18 is out of range; so overflow.

$$\begin{tikzcd} 3 - (-15) \arrow[d, bend right] \\ 3 + 15 \end{tikzcd}$$

$$\begin{array}{r} 00011 \rightarrow 3 \\ +\ 01111 \rightarrow 15 \\ \hline 10010 \end{array}$$

Result was supposed to be positive; but our result sign is 1; which indicates negative number.

So, overflow.

▨ -8 - (+10) [Substracting a positive number from a negative number]

= -18

⇒ -18 out of range so overflow.

$$\begin{tikzcd} -8 - (+10) \arrow[d, bend right] \\ -8 - 10 \arrow[d, bend right] \\ -8 + (-10) \end{tikzcd}$$

$$\begin{array}{r} 11000 \\ +\ 10110 \\ \hline \cancel{1}\,\textcircled{0}\,1110 \end{array}$$

Result sign.

As, result sign is 0; which was supposed to be 1 in this case. So, overflow.

Dealing With Overflow

C language Overflow handle করতে পারে।
↳ এটার জন্য তারা MIPs এ "addu, addui, subu" instruction use করে,

অন্যান্য language এ Overflow deal করার, তবে exception raise করাতে হয়,
↳ As তারা add, addi, sub instruction use করে।

▨ So, how to deal or Overcome Overflow?

⇒ Follow the 4 steps stated below:

(i) If the current instruction have overflow then PC holds the address of that instruction.

(ii) Then PC stores the address of that instruction in the Exception Program Counter (EPC) register.

(iii) Then the PC jumps to the function which handles overflow. This function contains the set of instructions to overcome overflow.

(iv) Lastly, PC retrives EPC address using mfc0 (move from coprocessor reg) instruction and resume instruction

" যেই instruction এ overflow হচ্ছে PC সেই instruction এর address hold করবে. Then EPC register এ গিয়ে PC সেই instruction এর address store করবে. Then সেই function overflow handle করতে সাহায্য করে direct ঐ function এ jump করবে। Then ঐ function এ কাজ শেষ হলে mfc0 instruction use করে EPC থেকে সেই instruction এর overflow ঘটেছিলো তার address retrive করে PC তে দিবে. Then বাকি কাজ normally resume হতে থাকবে."

EPC তে ঐসব instruction এর address জমা থাকে যেসব instruction এ overflow occur হয়েছিলো.

Coprocessor থেকে কিছু নেওয়ার জন্য mfc0 use হয়

Multiplication:

2 টা approach.

- Long Multiplication Approach
- Fast Multiplication Approach

Long Multiplication Approach:

- Length of product is the sum of operand lengths
- 32 bit Arch এর জন্য,
  - Multiplicand and Result এর জন্য 64 bit
  - Multiplier 32 bit এর জন্য
  - so As, 64 bit একসাথে রাখা যায় না register এ, so 2 টা register লাগবে, একটা register এ MSB রাখবে; আরেকটায় LSB রাখবে.

Multiplicand এর size হবে arch এর size এর double

Example: [Considering 4 bit arch.]

$$
\begin{array}{r}
1000 \\
\times\ 1001 \\
\hline
1000 \\
0000 \\
0000 \\
1000 \\
\hline
1001000
\end{array}
$$

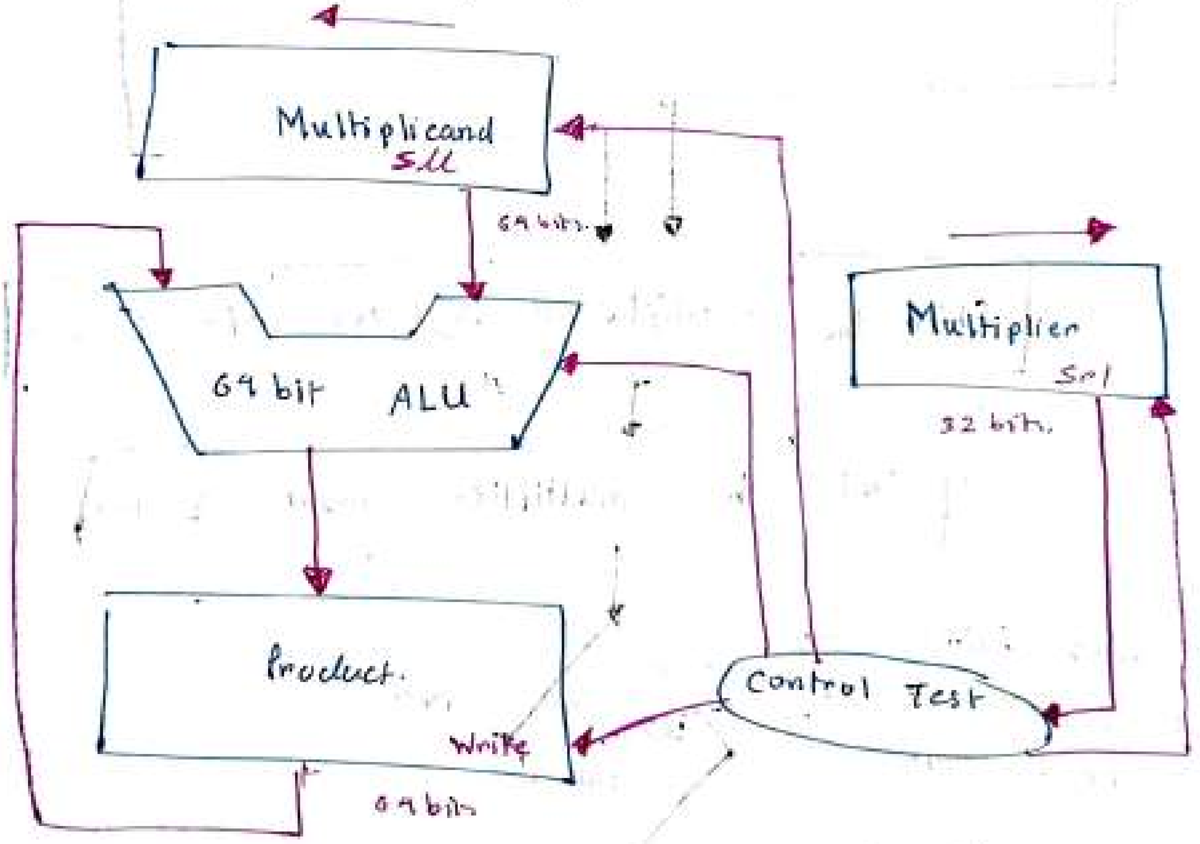
1000 → Multiplicand

1001 → Multiplier

1001000 → Product

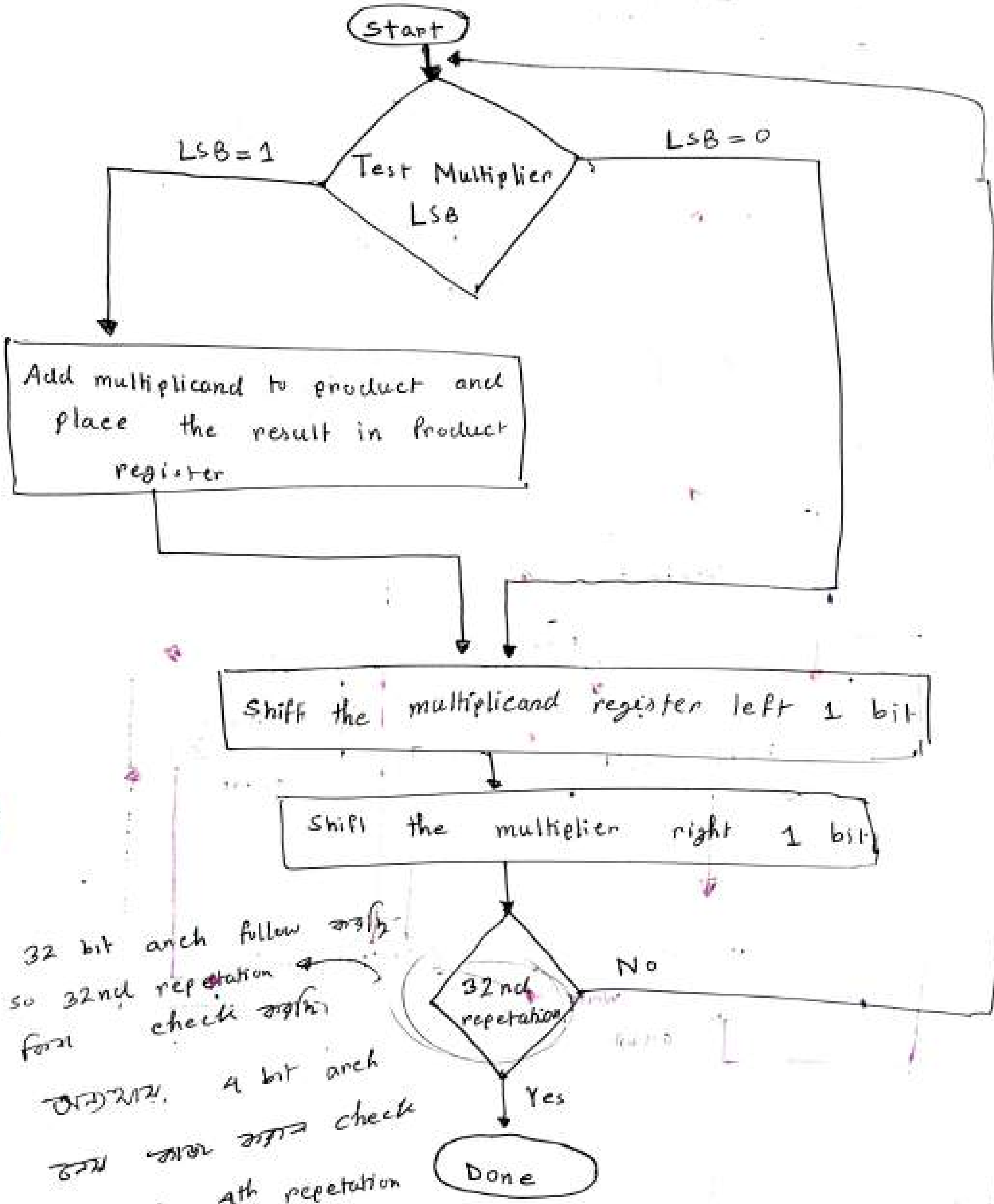
এখানে product এর জন্য যে register আছে তা maximum 8 bit hold করতে পারবে, এখানে product এর size 7 bit so, তা এখানে রাখা যাবে

Control Test @ ALU কে Control Signal পাঠায় Operation করার জন্য,



Product initially 0 থাকে,

Number of iteration = number of bit in multiplier = যত bit এর arch. follow করছি তত bit.


Start
Test Multiplier LSB
LSB = 1
LSB = 0
Add multiplicand to product and place the result in Product register
Shift the multiplicand register left 1 bit
Shift the multiplier right 1 bit
32nd repetation
No
Yes
Done

32 bit arch follow করছি so 32nd repetation দিয়া check করছি
অন্যথায়, 4 bit arch হলে আমরা check করতাম 4th repetation দিয়া।

Example: [4 bit arch.]

▨ 8 × 9

⇒ Multiplicand = 8 = $\underbrace{0000\ 1000}$

↳ an 4 bit arch. follow করছি। so, multiplicand length is the double of arch. bit.

Multiplier = 9 = 1001.

Iteration	Multiplicand	Multiplier	Product
	0000 1000	1001	00000 000
1	0000 1000	1001	0000 1000
	0001 0000	1001	0000 1000
	0001 0000	0100	0000 1000
2	0001 0000	0100	0000 1000
	0010 0000	0010	0000 1000
3	0100 0000	0010	0000 1000
	0100 0000	0001	0000 1000
4	0100 0000	0001	0100 1000
	1000 0000	0001	0100 1000
	1000 0000	0000	0100 1000

∵ $(0100\ 1000)_2 = (72)_{10} = (8) * 9$

এখন যদি 8 * (-9) = -72 করতে দেখাতে বললো

Then 8 এবং 9 এর positive value নিয়ে কাজ করবো। Last এ যে final result আসবে তার 2's complement করে দিব। Simple.

Optimized Multiplier

⇒ Perform steps in parallel: add / shift.

⇒ One cycle per partial-product addition

- That's ok if frequency of multiplication is low.

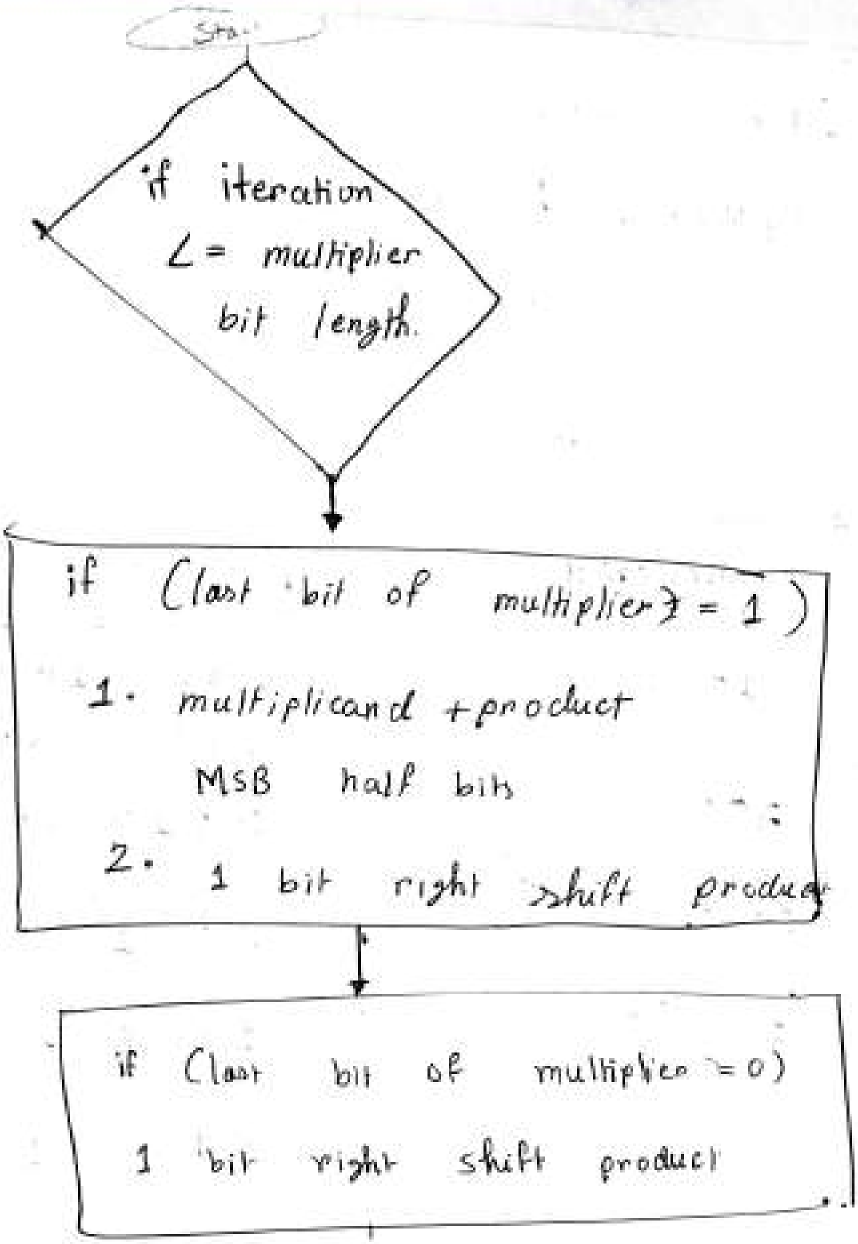
⇒ এখানে, multiplicand ও multiplier 32 bit just product 64 bit.

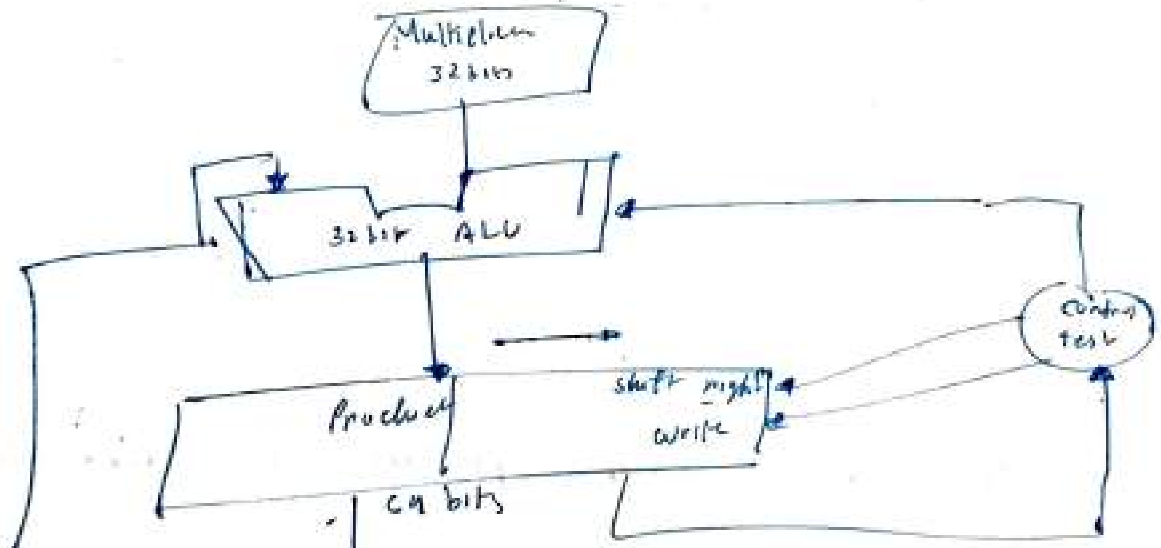
⇒ Product এর 64 bit দুইভাগে দিবো.

multiplic
32 bit
32 b

- LSB 32 is used for multiplier
- MSB 32 bit এ ~~[illegible]~~ Multiplicand এর সাথে ALU এর কাজ হয় কাজ হয় তার জন্য

Flowchart:


Sta
if iteration <= multiplier bit length.
if (last bit of multiplier) = 1)
1. multiplicand + product MSB half bits
2. 1 bit right shift product
if (last bit of multiplier = 0)
1 bit right shift product


Multiplicand 32 bits
32 bit ALU
Product
64 bits
shift right
write
Control test

▨ Example: [4 bit arch]

▢ Perform multiplication between 8 and 9 using the optimized multiplication approach?

⇒

Iteration	Multiplicand	Product
	1000	0000 1001
1	1000	1000 1001 0100 0100
2	1000	0010 0010
3	1000	0001 0001
4	1000	1001 0001 0100 1000

Throughout the iteration multiplicand এ কোনো change হবে না.

efficient কোনো way use করতে পারলে তাকেই বলি করবে optimized.

32 bit arch এর জন্য product এর size 64 bit.

▨ Perform Multiplication between 0110 (Multiplicand) and 110 (Multiplier) using optimized multiplication approach.

Iteration	Multiplicand (0110)	Product (0000 0110)
1	0110	0000 0011
2	0110	0110 0011 0011 0001
3	0110	1001 0001 0100 1000
4	0110	0010 0100

Faster Multiplication approach multiplication গুলো parallely handle করতে পারে।

MIPs Multiplication in MIPs Instruction :

64 bit register এ রাখা যায় না, তাই, দুটো register combine করে রাখছি, 2টা register হলো,

- high register
- low register

] Product reg. এ 2টা reg হলো,

HI → High Register → MSB 32 bit hold করে,

LO → Low Register → LSB 32 bit hold করে,

Instruction :

- mult rs, rt বা multu rs, rt
  - → 64 bit product in HI | LO
    - 2 টা 32 bit reg multiply করছি.
  - mult → multiplication signed
  - multu → multiplication unsigned.

- mfhi rd
  - → mfhi = move from high register.
  - → product এর MSB 32 bit টা rd তে store করে.

- mflo rd.
  - → move from low register = mflo
  - → product এর LSB 32 bit rd তে store করে.

$\boxed{\text{mul rd, rs, rt}}$ → এটা exist করে না।

↳ but এটা করার LSB 32 bit rd তে store হবে.

Floating Point Representation

• Normalized form

- Conversion from decimal to floating point and vice - versa.

- Floating point arithmetic operation.

Floating Point

- -2.34×10^{-56} → Floating Point in Normalized form
- $+0.002 \times 10^{-4}$
- $+987.02 \times 10^{9}$

} Floating point but not normalized

In binary

$1.xxxxxx \times 2^{yy}$

Normalized form এ decimal point এর আগে একটা non zero element থাকতে হবে

অনেক বড় বা অনেক ছোট ; binary number represent করার জন্য IEEE 754 format use হয়,

Normalization.

- Numbers with only one non-zero number befo. decimal point

→ 5.64×10^{33} → Normalize

→ 109.64×10^{33} → Not normalize

↳ Turning it into it's corresponding normalize form

decimal point কে 2 bit left shift করো

$$1.0964 \times 10^{33+2} = 1.0964 \times 10^{35}$$

যত bit left shift করছি base এর power এর সাথে তত add করে দিব।

→ 0.00578×10^{3} → Not normalize

↳ Transforming to Normalized

decimal point কে 3 bit right shift করতে হবে।

So, 5.78×10^{0}

যত bit right shift করছি base এর power থেকে তত minus.

right থেকে left এ গেলে left shift.

left থেকে right এ গেলে right shift.

Single precision → 32 bit ;
double precision → 64 bit

Decimal to Floating Point Representation.

sequentially follow করতে হবে।

- Convert Decimal to Binary
- Convert Binary to Normalized Binary
- Find out biased exponent.
- Find out sign bit and fraction

IEEE Floating Point Representation:

IEEE floating point এ কিভাবে যেকোনো number এর floating point এ representation করতে পারি,

২ ধরনের floating point representation আছে।

- 32 bit এর জন্য
  → এর Representation Single Precision

- 64 bit এর জন্য
  → এর Representation Double precision

64 bit এ আমরা বেশি range এর number represent করতে পারব।

↳ অনেক ছোট বা অনেক বড় number 64 bit এ represent করতে পারব যা হয়তো 32 bit দ্বারা possible হতো না।

↳ bit length যত বেশি তত বেশি range of number cover করতে পারব।

S	Exponent	Fraction

↳ S হলো Sign bit
- 0 for positive number
- (-1) for negative number

Exponent:

• Single Precision
⇒ 8 bit (exponent)
⇒ 23 bits (Fraction)

• Double Precision
⇒ 11 bit (exponent)
⇒ 52 bits (Fraction)

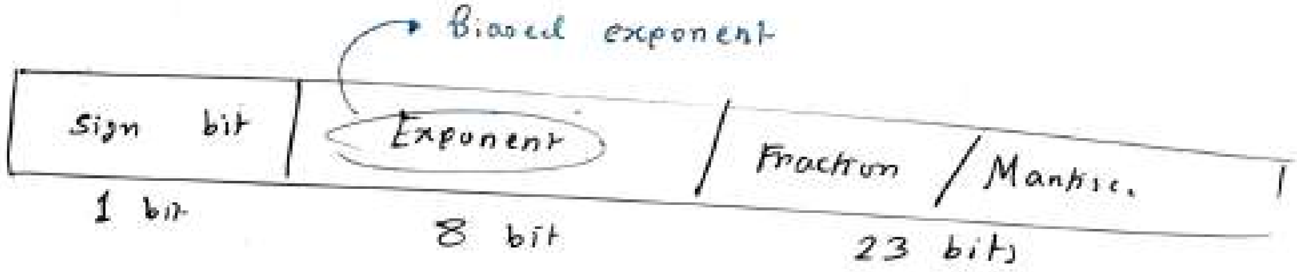
Floating point এ exponent টা মূলত biased exponent

biased exponent unsigned

$$X = (-1)^{s} \times (1 + \text{Fraction}) \times 2^{(\text{exponent} - \text{Bias})}$$

exponent:
↳ actual exponent + Bias.
- Single precision Bias = 127
- Double precision Bias = ~~1023~~ 1023

Single Precision (32 bit)



- Sign bit 0 হলে positive number; 1 হলে negative number.

- Exponent:
  - Base এর যে power তাই exponent
  - এই exponent টোকে Unsigned binary number হিসেবে 8 bit রাখবে.

$1\cdot11101 \times 2^{35}$
$1\cdot11101 \times 2^{8}$

8 bit unsigned binary range = 0 to $2^8 - 1$ = 0 to 255

কিন্তু, আমাদের এই exponent এর 00000000 and 1111 1111 is reserved..

$\therefore$ Range for biased exponent = 1 to 254

↳ Exponent এ 1 থেকে 254 পর্যন্ত এর 8 bit bin representation এই exponent এর 8 bit এর range.

✓ if biased exponent = n bits then,

$$\boxed{\text{bias} = 2^{(n-1)} - 1}$$

∴ ~~Bias~~ =

∴ For 8 bit biased exponent $= 2^{(8-1)} - 1 = 2^7 - 1 = 127$

✓ $\boxed{\text{Biased exponent} = \text{actual exponent} + \text{bias.}}$

যদি আমরা কোনো ও number কে 1 to 254 range এ রাখতে চাই। তাহলে শুধু 8 bit representation করব,

যদি bias এর value 1 থেকে ছোট বা 254 থেকে বড় হয় তাহলে তা single precision দ্বারা প্রকাশ না। তাহলে বেশি bitlength লাগবে,

Σ এর lowest power -126 হতে পারে,

as $-126 + 127 = 1$

এবং highest power $= 127$ হতে পারে,

as $127 + 127 = 254$

∴ range for exponent : 2^{-126} to 2^{127}

biased exp এর range হবে হবে 1 to 254

☑ Convert 50.6749 to 32 bits IEEE-754 Floating Point Representation

⇒ 50.6749

Binary of 50 = 110010

Binary of 0.6749 = 1010110011

∴ part এর পরের part minimum 10 বার multiply করবে.

Binary of

$.6749 \times 2 = 1.3498 \quad 1$

$.3498 \times 2 = 0.6996 \quad 0$

$.6996 \times 2 = 1.3992 \quad 1$

⋮

minimum 10 বার

∴ Binary of 50.6749 = 110010.1010110011

∴ Normalized Binary value := $1.\underbrace{10010\ 1010110011}_{\text{fraction / mantissa}} \times 2^5$

Here, actual exponent = 5.

∴ Bias = $2^{(8-1)} - 1$

∴ exponent bias = 5 + 127 = 132 = 10000100

∴ As positive number so sign bit = 0.

∴

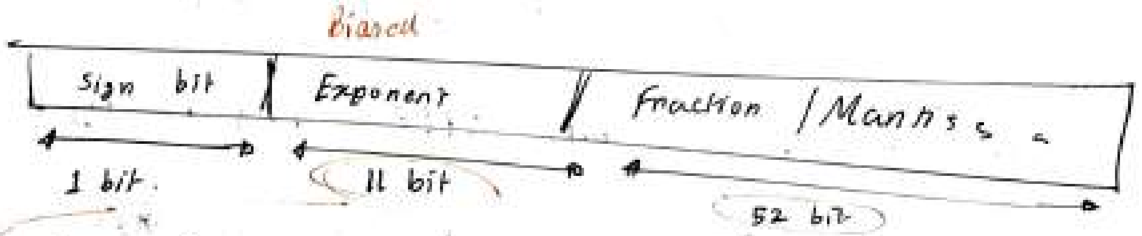
0	1000 0100	0000 0000 10010 1010 1100 11

IEEE-754 Floating Point Rep of 50.6749

= 0 1000 0100 0000 0000 10010 1010 1100 11

= 0X 424AB300

Double Precision (64 bit)



As fraction এবং exponent এর জন্য যত ~~বেশি~~ bit বেশি থাকবে তাই অনেক range of number নেওয়া যায়।

- as fraction 52 bit so অনেক বড় number represent করতে পারে।
- Exponent 11 bit তাই অনেক বড় number of represent করতে পারে।

- Sign bit 0 হলে positive 1 হলে negative
- Exponent
  - 11 bit unsigned binary range = 0 to $2^{11}-1 = 2047$

$\Rightarrow$ exponents 0000 0000 000 এবং 1111 1111 111 is reserved.

$\therefore$ range for biased exponent is = 1 to 2046

$\therefore$ Bias = $2^{(11-1)} - 1 = 2^{10} - 1 = 1023$

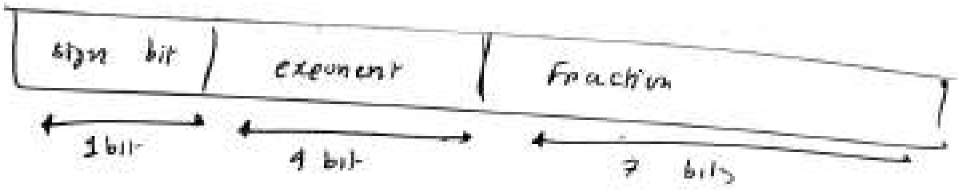
For double precision :

- Biased exponent = Actual exponent + Bias (1023)

∴ actual exponent का minimum value −1022 तथा maximum value 1023 .

▨ Convert − 0·232 to 12 bit IEEE −754 bit Floating Point Representation, where biased exponent is 4 bits

→



0010 0100

Binary of 0·232 = 0·00111 0 11

Normalized binary of 0·232 = $1\cdot 11\,011 \times 2^{-3}$

exponent:

actual exponent = −3

∴ Bias for 4 bits = $2^{(4-1)} - 1 = 7$,

∴ Biased exponent = −3 + 7 = 4 = 0100

Sign bit = 1

Fraction = 1101100

1	0100	1101100

= 0 × A6C.

Floating Point (Single Point) to Decimal Conversion

■ 0x F2400120

→ Hex to Binary.

1111 0010 0100 0000 0000 ~~0000~~ 0001 0010 0000

Binary according to Format.

1 11100100 100 0000 0000 0001 0010 0000

Find out exponent and fraction.

Biased exponent = 111 00 100

Biased exponent in Decimal = 228

∴ Actual exponent = 228 − 127

= 101

Fraction = 0.100 0000 0000 0001 0010 0000

$= 2^{-1} + 2^{-15} + 2^{-18}$

= 0.5000343323

∴ Decimal value = $(-1)^{\text{sign bit}} \times (1 + \text{Fraction}) \times 2^{\text{exponent}}$

$= (-1)^1 \times (1 + 0.5000343323) \times 2^{101}$

= ~~2.38~~ $3.8030388 43 \times 10^{30}$

∴ Upto 5 decimal point with rounding

$= 3.80304 \times 10^{30}$

Floating Point Addition / Substraction

■ $35.23142 + 0.00053$

$\Rightarrow$

$x = 35.23142$ $\qquad$ $Y = 0.00053$

$x_{bin} = 100011.0011101111$ $\qquad$ $Y_{bin} = 0.00000000001000 1011$

X (Binary Normalized)

$= 1.00011\ 00111\ 0\ 1111 \times 2^5$

Y (Binary Normalized)

$= 1.000\ 1011 \times 2^{-11}$

এটাকে 5 bit left Shift করে করতে পারি;

∴ Now updated y in Binary normalized form

$= 0.\underline{0000}0000\ 0000000\ 1000\ 1011 \times 2^5$

যাহার মান ছোট তাকে তাহার মান যার power সমান, তাকে [illegible] তার সাথে [illegible] করতে হবে।

Rule:
Match the lower exponent with higher exponent

$\therefore X+Y = (1.000\ 11\ 00\ 111\ 0\ 1111 \times 2^5) + (0.0000000000000000 1000\ 1011 \times 2^5)$

$= (1.000\ 11\ 00\ 111\ 0\ 1111 + 0.0000\ 0000\ 00\ 00\ 000\ 1000\ 1011) \times 2^5$

$= 1.000\ 11\ 00\ 111\ 0\ 1111\underline{1000\ 1011} \times 2^5$

$= 100011.\ 00\ 111\ 0\ 111\ 11\ 000\ 1011$

$= 35.239\ 222\ 4121$ (Decimal)

Floating Point Multiplication

☐ $5.239 \times (-0.003)$

• [illegible] [illegible]! [illegible] [illegible] just last [illegible] result [illegible] [illegible] [illegible] [illegible]

$\Rightarrow$ $X = 5.239$

$Y = 0.003$

X in Binary = 101.0011101111; Y in Binary

$= 0.00000000\ 11000100\ 101$

$= 1.1000\cdot 100101 \times 2^{-9}$.

(normalized)

X (Binary normalized)

$= 1.01\ 00111\ 0\ 1111 \times 2^{2}$

Multiplication [illegible] [illegible] base [illegible] power same [illegible] [illegible] [illegible]

• [illegible] [illegible] and negative [illegible] [illegible]

$X * Y = -(1.0100111\ 0\ 1111 \times 2^{2}) \times (1.100010\ 0101 \times 2^{-9})$

$= -(1.0100\ 111\ 0\ 1111 \times 1.1000\ 100\ 101) \times 2^{2-9}$

$= -(1.0100\ 111\ 0\ 1111 \times 1.1000\ 100\ 101) \times 2^{-7}$

$= -10.000000\ 101 \times 2^{-7}$

$= -0.00000\ 10\ 000000\ 101 \times 2^{0}$

$= -0.01570\ 12939$ (Decimal)

Floating Point Arithmetic

51500000 − BA10A000

→ X = 51500000

Sign bit 1 bit	exponent 8 bit	Mantissa 23 bit

X (Binary)

$= \underbrace{0|101\ 0001\ 0}101\ 0000\ 0000\ 0000\ 0000\ 0000$

Find out fraction and exponent

Biased exponent = 101 00010 = $(162)_{10}$

∴ For exponent = ~~127 − 162~~ 162 − 127 = 3.5

∴ Fraction or Mantissa = 0.101 0000 0000 0000 0000 0000

∴ X in Binary Normalized = $1.\text{Fraction} \times 2^{\text{exponent}}$

∴ X in Binary Normalized = 1.101 0000 0000 0000 0000 0000 $\times 2^{+35}$

Similarly,

Y = BA10A000

Y (Binary) = $\underbrace{1011\ 1010\ 0}001$ 0000 1010 0000 0000 0000

∴ Biased exponent = 011 10100 = 116

∴ exponent = 116 − 127 = −11

∴ Fraction = 0.001 0000 1010 0000 0000 0000

∴ Y in Binary normalized

= −1.001 0000 1010 0000 0000 0000 $\times 2^{-11}$

[As sign bit = −1]

Again,

X (Binary Normalized)

$= 1.101\ 0000\ 0000\ 0000\ 0000\ 0000 \times 2^{35}$

Y (Binary Normalized)

$= -1.001\ 0000\ 1010\ 0000\ 0000\ 0000 \times 2^{\boxed{-11}}$

$= -0.\ [45\ 0\text{'s}]\ 1001\ 0000\ 1010\ 0000\ 0000\ 0000 \times 2^{35}$

$X - (-Y) = X + Y$

$= (1.101\ 0000\ 0000\ 0000\ 0000\ 0000\ +$

$0.\ [45\ 0\text{'s} \ldots]\ 1001\ 0000\ 1010\ 0000\ 0000\ 0000) \times 2^{35}$

$= 1.101\ [42\ 0\text{'s}]\ 1001\ 0000\ 1010\ 0000\ 0000\ 0000 \times 2^{35}$

■ 7ACD 0000 + 5BCA 0000

⇒ X = 7ACD 0000

X (Binary)

= 0111 1010 1100 1101 0000 0000 0000 0000

$= 0\ \underbrace{111\ 1010\ 1}\ 100\ 1101\ 0000\ 0000\ 0000\ 0000$

∴ Biased exponent = 11110101 = 245.

∴ Exponent = 245 - 127 = 118. For exponent being 8 bit,

$\text{Bias} = 2^{(8-1)} - 1 = 127$

∴ Fraction / Mantissa = 0.1001101 0000 0000 0000 0000

∴ X (Binary normalized)

$= 1.100\ 1101\ 0000\ 0000\ 0000\ 0000 \wedge 2^{118}$

Y = 5BCA 0000

Y (Binary) = 0101 1011 1100 1010 0000 0000 0000 0000

∴ Biased exponent = 10110111 ~ 183.

∴ exponent = 183 - 127 = 56

∴ Fraction or Mantissa = 100 1010 0000 0000 ,0000 0000

∴ Y (Binary Normalized)

$= 1.100\ 1010\ 0000\ 0000\ 0000\ 0000 \times 2^{56}$

Y in Binary normalized = 0. [61 0s] 100 1010 00000 0000 0000 0000 $\times 2^{118}$

∴ X+Y

$=$(1.1001101 0000 0000 0000 0000 + 0.[61 0s] 100 1010 0000 0000 0000 0000) $\times 2^{118}$

$=$ 1.1001101 [54 0's] 100 1010 0000 0000 0000 0000 $\times 2^{118}$

Suppose, X = 19.454 and Y = 3.0124, perform $x * y$ using IEEE floating-point representation.

⇒ X = 19.454

X (Binary) = 10011.01110100

X (Normalized) = 1.0011 0111 0100 $\times 2^4$.

Y = 3.0124

Y (Bin) = 11.000000 1100

Y (Binary Normalized) = 1.1000000 1100 $\times 2^1$

$X * Y = (1.0011011101100 \times 1.1000000001100) \times 2^5$

$= 1.1101\,0100\,1011\,0010111\,0000 \times 2^5$

$= 111010.001011\,0010111\,0000$

$= 58.58734$

```
     1.00110 111 0 100
   x 1.1000 0001 100
   ------------------------
     0 0000000000 000
                        X
   1 0 0 1101110100 x X
 1 0 0 1 1 0 1 1 1 0 1 0 0 x x  x
-------------------------------------
 1.11010100 1 011 00 1 0111 0000
```

(P.T.O)

MIPS Division:

- Use HI / LO register for result.
  - HI : → 32 bit remainder (ভাগশেষ)
  - LO : → 32 bit quotient (ভাগফল)

- Instructions
  - div rs, rt / divu rs, rt.
  - No overflow on divide by 0 checking
    - Software must perform check if required
  - Use mfhi, mflo to access result.

div rs, rt মানে, rs register এর valle divided by rt.

div $t0, $t1

■ কোনো register এর value even নাকি odd তা আমরা easily div instruction দিয়ে বের করতে পারি।

আর যদি remainder টা বের করতে হয় বা কোনো register এ নিতে হয় তাহলে mfhi use করবো।

example $\boxed{\text{mfhi \$t0}}$ → remainder এর value $t0 তে নিয়ে আসছি।

এখন যদি $t0 এর value 0 হয় তাহলে even অন্যথায় odd

Floating Point Registers and Instruction.

Register এ যে data রাখতে পারি তা সবসময় int হয়,

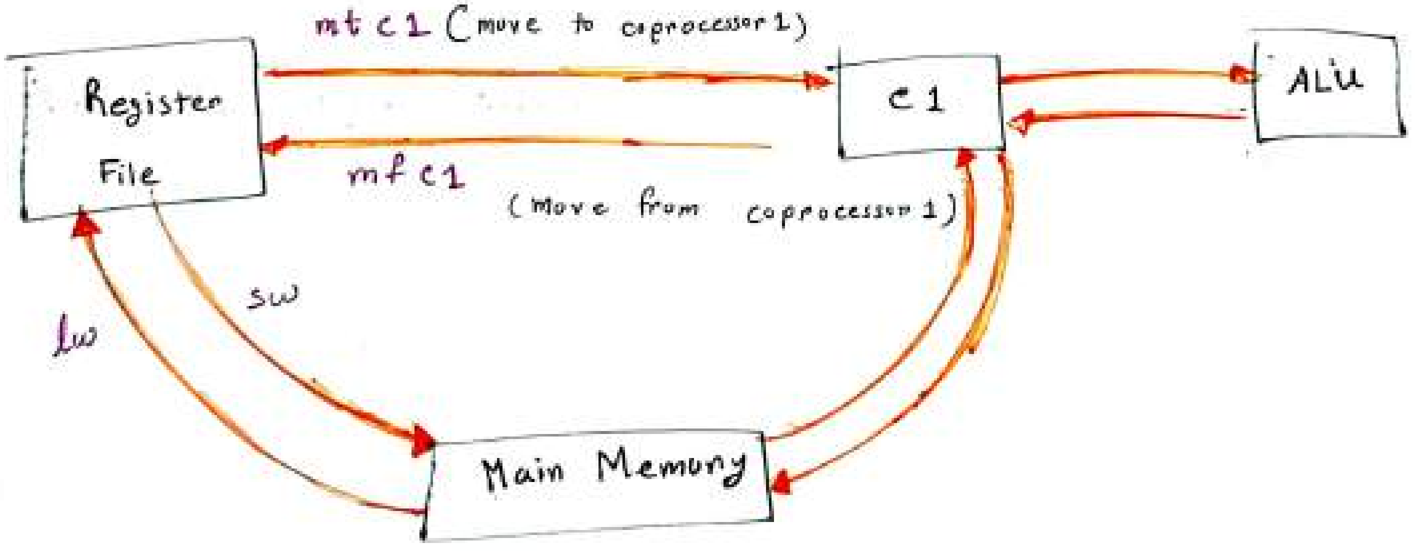
Floating Point তাদের Register এ রাখতে পারি না

Floating Point এর জন্য আলাদা একটি register file আছে,

Floating Point Register গুলো $f0, $f1, ..., এভাবে.

Named Co-processor 1.

→ এটা both int এবং float type data hold করে,



Coprocessor 1 এর 32 bit register থাকে এবং 32 টি register থাকে.

Coprocessor 1 এর register file হলো:

$f0 থেকে $f31

▨ A [3] = (float) x

Where x is in $f0 and base address of A in in $s1

⇒ swc1 $f0, 12 ($s1)

▨ Float (a) = B [2];

Where a is in $f0 and base address of B in $s1.

⇒ lwc1 $f0, 8 ($s1)

Co-processor এ register overwrite করার option নাই,

• MIPS Instruction For Single Precision:

⇒ Single Precision 32 bit

$\$f0 = \$f1 + \$f2$

$\xrightarrow{}$ add.s $f0, $f1, $f2

$\$f0 = \$f1 - \$f2$

sub.s $f0, $f1, $f2

"add.s আর sub.s indicate করছে single Precision এর জন্য,"

MIPS Instruction For Double Precision:

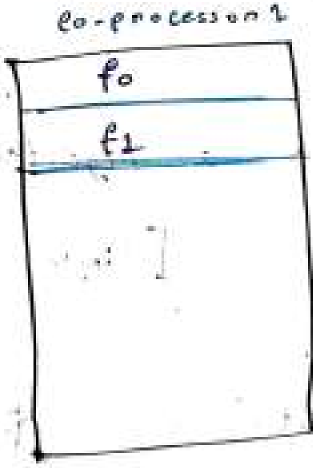
⇒ Double Precision 64 bit.

$\$f0 = \$f1 + \$f2$

$\xrightarrow{}$ ~~add.d~~ Valid না, as Coprocessor এ 32 bit register আছে যেটা 32 bit hold করতে পারে,

$\$f0 = \$f1 - \$f2$

but double precision আছে 64 bit, যেমন : এই example এ $f0, $f1, $f2 ; 32 bit hold করছে but আমাদের লাগবে 64 bit. So, Source & Destination 64 bit.



যদি $f0 তে রাখি এর 64 bit রাখতে so 2 টা slot লাগবে, So, f0 তে store [illegible] করা আছে f0 এর f1।
So, f0 and f1 combination করলে 64 bit পাবো।

So, Valid example.

$f0 = $f2 + $f4

add.d $f0, $f2, $f4

$f0 = $f2 − $f4

sub.s $f0, $f2, $f4

" add.d আর sub.d indicate করছে double precision."

Moving Values from registers:

$$\begin{tikzcd} \text{mtc1 } \$s1, \arrow[r, bend left] & \$f0 \end{tikzcd}$$

$$\begin{tikzcd} \text{mfc1 } \$s1, & \$f0 \arrow[l, bend left] \end{tikzcd}$$

Co processor থেকেই right hand side এ থাকবে,

Convert a Register value from float to int [Single Precision]

$$\begin{tikzcd} \text{cvt.w.s } \$f0, & \$f1 \arrow[l, bend right] \end{tikzcd}$$

Convert করে Coprocessor এই থাকে,

float value টা r.h.s এ,
So, r.h.s. থেকে convert হয়ে l.h.s এ assign.

Convert a register's value from int to float [Single precision]

$$\begin{tikzcd} \text{cvt.s.w } \$f0, & \$f1 \arrow[l, bend right] \end{tikzcd}$$

left এ converted value

Double Precision to Single Precision

$$\begin{tikzcd}\text{cvt . s.d.} & \text{\$f0,} & \text{\$f1} \arrow[l, bend right]\end{tikzcd}$$

Single Precision to Double Precision

$$\begin{tikzcd}\text{cvt . d.s.} & \text{\$ f0,} & \text{\$ f1.} \arrow[l, bend right]\end{tikzcd}$$

▨ Float (x) = (int) y + float (z)
int (y) = (float) x − (float) z

Suppose x, y, z are in $f1, $ s1 and $ f2 correspondingly.

⇒ MIPS Code:

```
mtc1     $s1, $f0   # reg থেকে mem এ (int) y যাবে,
cvt.s.w. $f0, $f0   # int y থেকে float (y), এ convert
add.s $ f1, $f0, $f2   # float(x) = float (y) + float(z)
sub.s $ f3 , $f1, $f2  # float (y) = float (x) - float (z)
cvt.w.s  $f3, $f3   # float(y) থেকে int (y) এ convert
mfc1     $ s1, $f3  # int (y) = float (x) - float(z)
```

and int(y) is value in again pushed in the saved register (register file)

Chapter - 3 Formula and Procedure.

- n bit system's unsigned number range

"$-2^{(n-1)}$ to $+2^{(n-1)}-1$"

- n bit systems signed number range

0 to $2^n - 1$

- Overflow: [কত bit arch. follow করছি সেখানে প্রযোজ্য হবে]

Int Addition

- Overflow if out of range.
- If sign bit is 1 while adding two positive number.
- If sign bit is 0 while adding two negative number.
- No overflow while adding a positive number and a negative number.

Int Substraction

- Overflow if out of range.
- If sign bit is 0 while substracting +ve number from -ve number.
- If result sign is 1 while substracting a -ve number from a positive number.
- No overflow while substracting two positive number or while substracting two negative number.

Sign bit = 0 হলে positive

Sign bit = 1 হলে negative.

- Long Multiplication
  - যদি n bit arch. হয় then, ~~[illegible]~~
    - multiplier n bit
    - multiplicand আর product 2n bit.
  - Flowchart for procedure ★

- Optimized Multiplication
  - যদি n bit arch হয় তাহলে;
    - multiplier and multiplicand n bit
    - product 2n bit.
  - But there's a catch,
    for example in 32bit arch; optimized multiplication এ.
    - multiplier 32 bit, multiplicand 32 bit
    - Product 64 bit
      ↳ product এর 64 bit 2 ভাগে দিবে.

HI $\leftarrow$ MSB 32 BIT $\xrightarrow{\text{denotes}}$

LO $\leftarrow$ LSB 32 BIT $\longrightarrow$ Multiplier

$\rightarrow$ MSB 32 bit এ multiplicand এর মাঝে ALU এর ভেতরে কাজটা হয় তার মধ্যে

- MIPS Multiplication Instruction

⇒ mult rs, rt ; multu rs, rt

⇒ mfhi rd :

↳ Product এর MSB 32 bit এরে rd তে store করে.

⇒ mflo rd ;

↳ Product এর LSB 32 bit এরে rd তে store করে,

- Decimal Point

  - left shift করলে power বাড়ে-
  - right shift " " কমে.

- Conversion From Decimal to Floating Point

① Convert Decimal to Binary

② Convert Binary to Normalized Binary.

→ Normalized form: $1.xxxxxx \times 2^{yy}$

③ Find out the Biased exponent.

④ Find out sign bit and fraction.

$$\boxed{X = (-1)^{\text{sign bit}} \times (1 + \text{Fraction}) \times 2^{(\text{Biased exponent} - \text{Bias})}}$$

Floating Point Format:

- Single Point Precision

⇒ Sign bit = 1 bit

⇒ exponent = 8 bit
(biased)

⇒ Fraction / Mantissa = 23 bit

∴ Total 32 bits

⇒ Bias = 127

- Double precision

⇒ Sign bit = 1 bit

⇒ exponent = 11 bit
(biased)

⇒ Fraction / Mantissa = 52 bit

∴ Total 64 bit

⇒ Bias = 1023

Sign bit	exponent	Fraction

Formula:

- If exponent field length = n bit, then,

$$\therefore \text{Bias} = 2^{(n-1)} - 1$$

- Biased exponent = Actual exponent + Bias

Single Precision exponent Range = −127 to 127.

Double Precision exponent Range = −1022 to 1023

⋮

(বাকিগুলো লিখি নি; আগামীতে
পরে আমাকে বুঝানো হবে)