IEEE 754 encoding of floating point numbers.

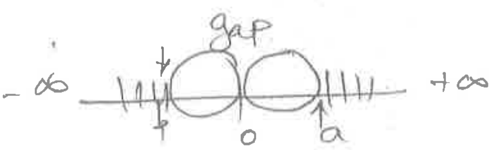
Single Precision		Double Precision		Object represented
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	Non Zero	0	Non Zero	± denormalized number
1-254	Anything	1-2046	Anything	± floating point number
255	0	2047	0	± infinity
255	Non Zero	2047	Non Zero	NAN (Not a Number)

NAN → implies 0/0 error or subtracting an infinity from infinity.

Infinity → 255 has if sign bit =1 ⇒ $-\infty$,
if sign bit =0 ⇒ $+\infty$.

<u>Denormalized numbers.</u> → These number are developed to remedy the problem of gaps among floating point number near 0.

Consider 2 nos, $a = 1.00\cdots 2^{-127}$ and b is $1.001_2 \cdot 2^{-150}$, this implies the gap between 0 and a is 2^{-127} and that gap between 0 and a is 2^{-150}



This can be remedied or solved by omitting the leading one from significand thus denormalize the floating point representation.

Floating point representation :- (biased Notation)

The desirable notation must therefore represent the most negative exponent as $00\cdots00_2$ and most positive as $11\cdots11_2$. This convention is called biased notation.

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

eg:-

IEEE 754 binary representation of number -0.75_{10} in single & double precision.

i) -0.75_{10}

Convert to binary,

$$-0.11_2 \times 2^0$$

$.75 \times 2 = 1.50$
$.50 \times 2 = 1.00$

(ii) Normalize scientific Notation

$$1.1_2 \times 2^{-1}$$

(iii) General representation is

$$(-1)^S \times (1+\text{fraction}) \times 2^{(\text{Exponent}-127)}$$

we get

$$(-1)^1 \times (1 + .1000\ 0000\ 0000\ 00000000000_2) \times 2^{(126-127)}$$

Single precision is

-ve → binary of 126

1	0 1111110	1 000 0000000 0000000 0000000
1 bit	8 bits	23 bits

double precision :-

$$(-1)^1 \times (1 + .10000\ 0000\ 0000\ 0000\ 0000\ 000 \cdots\cdots_2) \times 2^{(1022-1023)}$$

→ 1022

1	01111111110	100000 · · · · · · · · · · · ·
1 bit	11 bit	20 bits.

0 0000 · · · · · · · · · · · ·
32 bits

Floating point Addition.

Floating point addition involves 4 main steps to be followed.

Steps:-

1. Compare the exponents of two numbers, shift smaller number to right until its exponent would match the larger.

2. Perform Addition of the significands

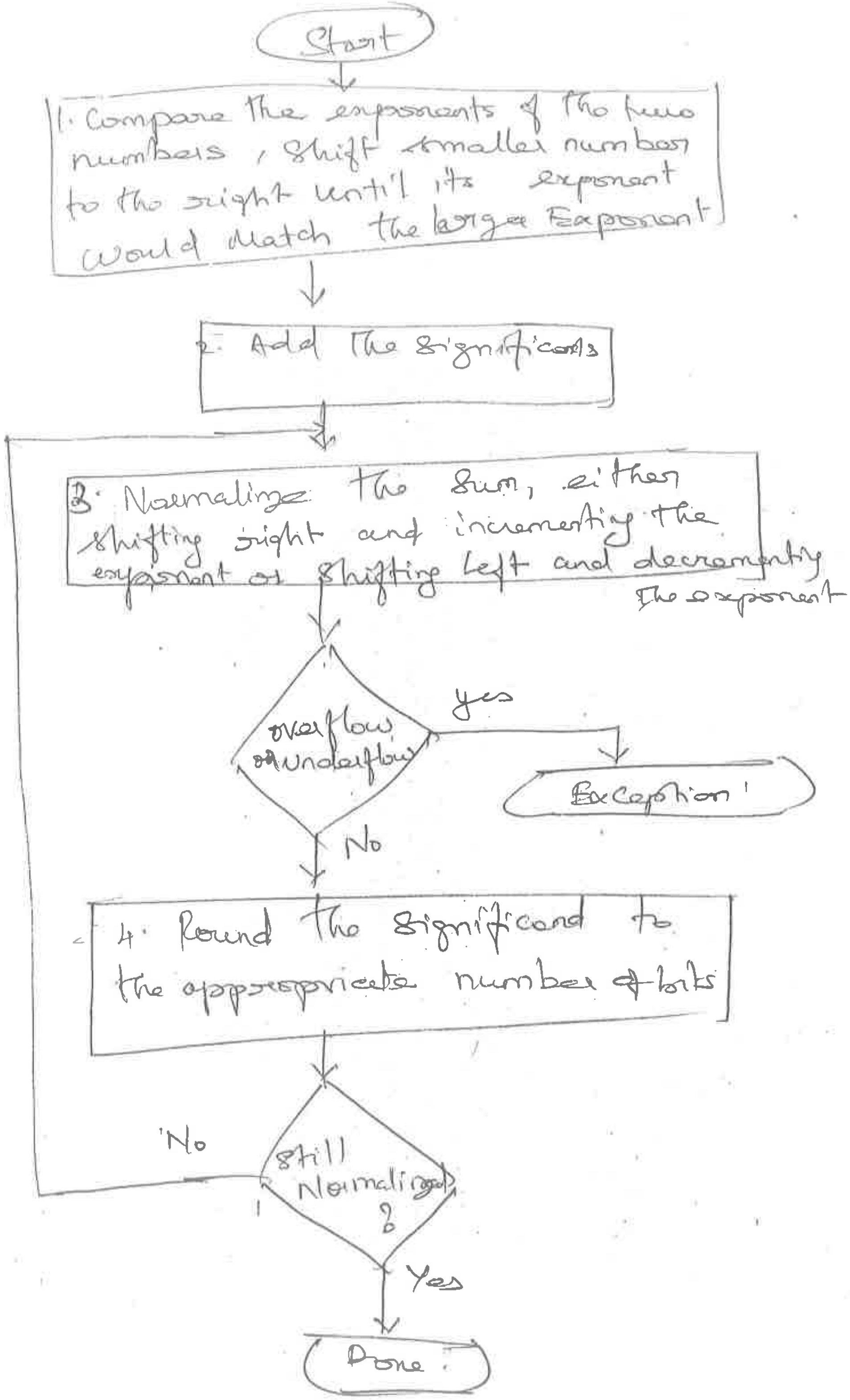
3. Normalize the sum, either shift right and increment the exponent or shift left and decrement the exponent.

3a) Check overflow / underflow if exist through exception otherse proceed.

4. Round the significand to appropriate no of bits

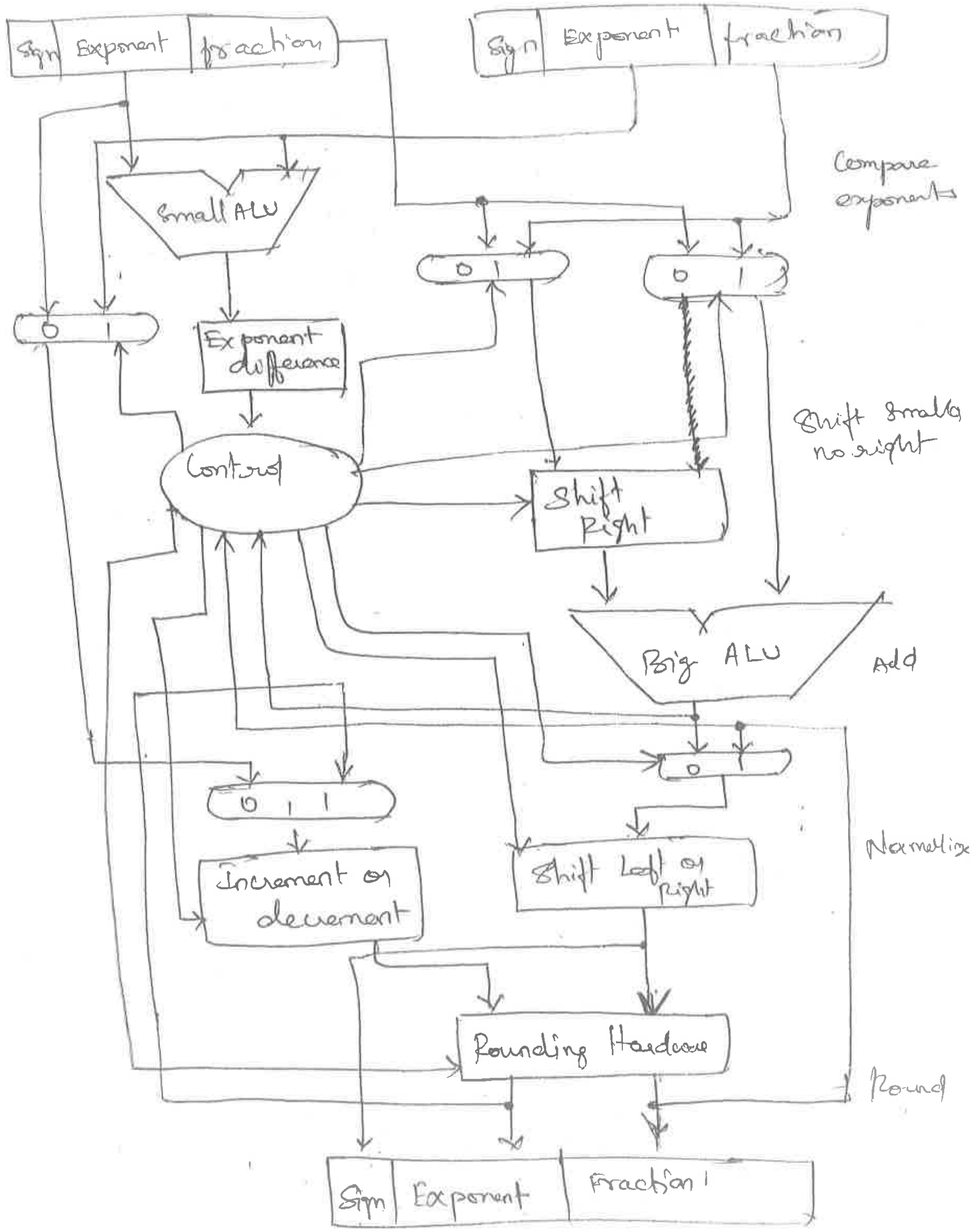
4a) Check again it is normalized if needed go to step 3 orelse finish.

Floating Point Addition Algorithm:-



Floating point Addition Algorithm.

Block diagram of an arithmetic Unit to floating point addition



Explanation:-

First the exponent of one operand is subtracted from the other using the small ALU to determine which is larger and by how much. This difference controls 3 Multiplexers. The smaller significand shifted right and added together using large ALU. Normalization shifts the sum left or right and increments or Decrements the exponent.

eg: Floating point addition : $9.999_{10} \times 10^{1} + 1.610 \times 10^{-1}$.

Solution:-

Assume that we can store only 4 decimal digit of significand, and two decimal digit of exponent.

Step 1: ~~...~~ Align the decimal point of the number that has smaller exponent.

1.610×10^{-1} is smaller exponent

$.01610 \times 10^{-1} \times 10^{2}$

$\Rightarrow 0.01610_{10} \times 10^{1}$

represent in 4 digits $= 0.016 \times 10^{1}$ //

Step 2 :- Perform addition of significands

$$\begin{array}{r} 9.999_{10} \\ 0.016 \\ \hline 10.015_{10} \\ \hline \end{array}$$

The sum is $10.015_{10} \times 10^1$.

Step 3 :- Normalized Scientific Notation

$10.015_{10} \times 10^1$

1.0015×10^2

& check overflow or underflow.

Step 4 :- Round the digits to 4-digits long.

1.0015×10^2

$\Downarrow$

1.002×10^2

Check again for Normalized and repeat step 3.

eg 2 :- Binary Floating point Addition.

Add : 0.5_{10} and -0.4375_{10}

Solution Convert decimal to binary & Normalize

36

Step:-

$0.5_{10} \xrightarrow{\text{binary}} 0.1_2 \times 2^0$ $\qquad 0.5 \times 2 \Rightarrow 1.0$

$= 1.000 \times 2^{-1}$

$-0.4375_{10} \xrightarrow{\text{binary}} -0.0111_2 \times 2^0$

$= 1.110 \times 2^{-2}$

$.4375 \times 2 = 0.8750$
$.8750 \times 2 = 1.7500$
$.7500 \times 2 = 1.5000$
$.5 \times 2 = 1.0$

Then follow the algorithm

Step 1: Shift the significant of smaller exponent

$1.000 \times 2^{-1} + (\underline{-1.110 \times 2^{-2}})$

↳ smaller Exponent

$-1.110 \times 2^{-2} \times 2^{1}$

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

Step 2:

Add the significand

$$\begin{array}{r} 1.000 \times 2^{-1} \\ -0.111 \times 2^{-1} \\ \hline 0.001 \times 2^{-1} \end{array}$$

Step 3:- Normalize

$0.001 \times 2^{-1} \times 2^{-3}$

$\boxed{1.000 \times 2^{-4}}$

Check for overflow $127 \geq -4 \geq -126$ So no overflow

($-4+127 \Rightarrow 123$ which is between 1 & 254).

Step 4: Round the sum

$$\boxed{1.000 \times 2^{-4}}$$

Convert binary to decimal.

$$.0001.000 \times 2^{-4} \times 2^{4}$$

$$0.0001_2$$

1×2^{-4}
0×2^{-3}
0×2^{-2}
0×2^{-1}

$$1 \times 2^{-4}$$

$$= \frac{1}{2^4} = \frac{1}{16_{10}} \Rightarrow 0.0625_{10}$$ //

Float Point Multiplication:

~~1) Calculate the product by adding the operands together.~~

~~New Exponent = 7~~

Floating point Multiplication

There are several steps involved in floating point Multiplication

Step 1: Perform Addition of the exponents and obtain the new biased exponent.

eg: $1.2 \times 10^{1} \times 1.2 \times 10^{1} \Rightarrow$ Product $\times 10^{2}$

Step 2: Perform Multiplication of the significand

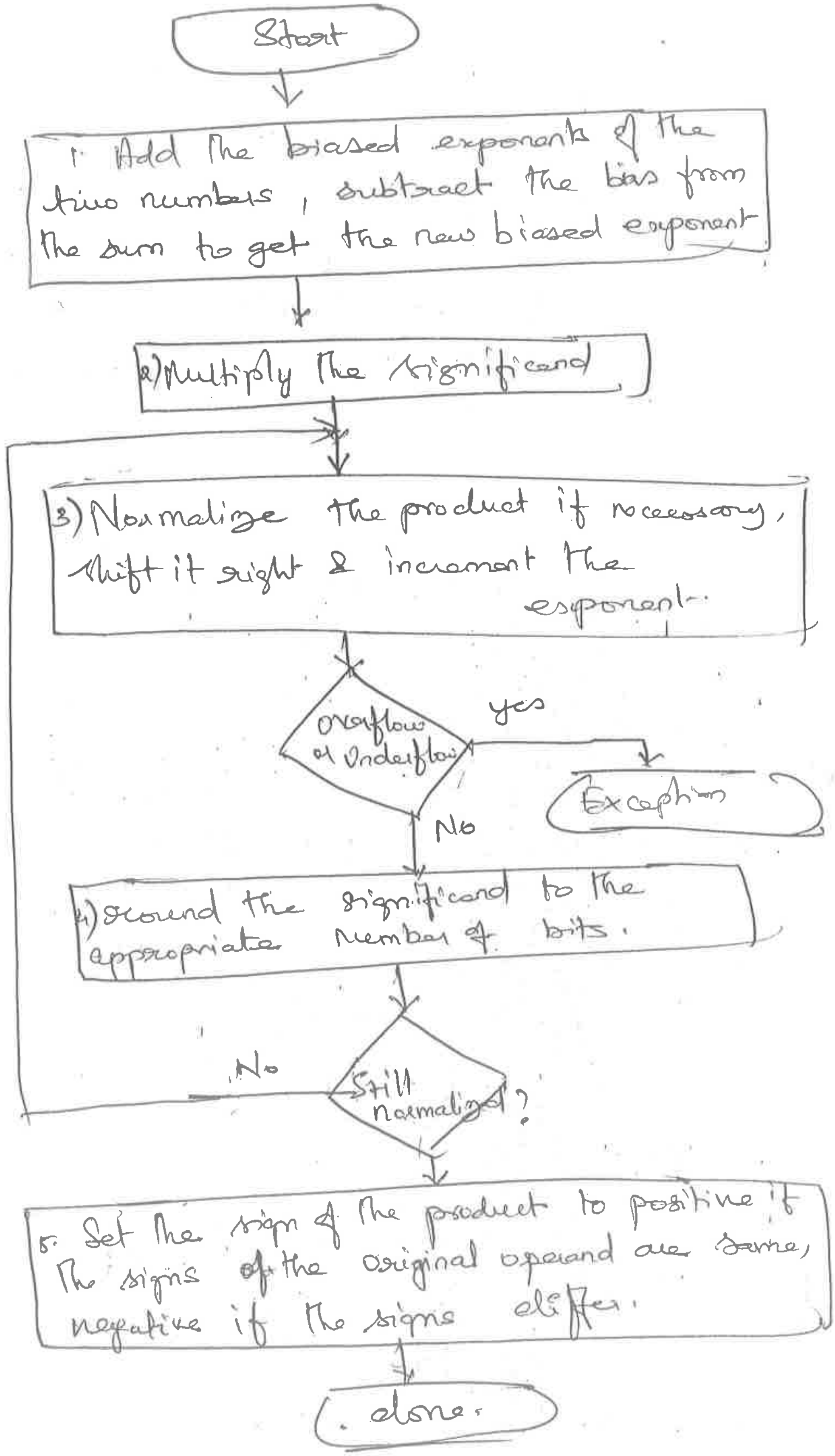
Step 3: Normalize the product if necessary by shifting right and incrementing the exponent.

3a) Check overflow or Underflow if occur then raise exception otherwise goto next step.

Step 4: Round the significand to the appropriate no of bits. If still not normalized normalize them.

Step 5: Set the sign of the product to positive if signs of operands are same, and negative if they differs in sign of operand.

Floating Point Multiplication Algorithm



eg:-

Multiply $1.110_{10} \times 10^{10} \times 9.200 \times 10^{-5}$

Assume we can store only 4 digits of significand.

Step 1:- Calculate the exponent of the product by simply adding exponents.

$$1.110 \times 10^{10} \times 9.200 \times 10^{-5}$$

New exponent is 5 (ie) $10^{10+(-5)}$
$= 10^{5}$

Step 2: Multiply the significand

$$\begin{array}{r} 1.110_{10} \\ 9.200_{10} \\ \hline 0000 \\ 0000 \\ 2220 \\ 9990 \\ \hline 10212000_{10} \\ \hline \end{array}$$

Place the decimal point 6 digits from right

$$10.212000_{10}$$

Assume only 4 digits can be kept (ie) 3 digits after decimal.

$$10.212_{10}$$

Step 3:- The product is not normalized Normalize them.

$10.212_{10} \times 10^5 \Rightarrow 1.0212 \times 10^5 \times 10^1$

$\Rightarrow 1.0212 \times 10^6$

Step 4:- round the number to 4 digits (ie) 1 digit before decimal and 3 digit after decimal.

$1.0212 \times 10^6 = 1.021 \times 10^6$

Step 5:- Obtain the sign of the product based on the sign of operand.

since the operands are both positive so product is also positive

$\boxed{Ans = +1.021_{10} \times 10^6}$

check it is normalize, if not again perform Normalization.

Floating point in MIPS :-

The MIPS floating point architecture uses seperate floating point instructions for IEEE 754 single and double precision.

(i) Floating point Addition Instruction

add.s (Single Addition)
add.d (double addition)

eg: add.s $\underbrace{\$f2\ \$f4\ \$f6}_{\text{floating point Registers}}$

(2) Floating point Subtraction

sub.s \$f2, \$f4, \$f6 → single subtraction
sub.d \$f2, \$f4, \$f6 → double subtraction

(3) Floating point Multiplication

mul.s , mul.d → double multiplication.
↳ single Multiplication

(4) Floating point division

div.s (single division)
div.d (double division)

* Floating point Comparison

eg ~~c.gt.s~~

eg:- (i) c.gt.s $f2, $f4 [Compare greater than single]

(ii) c.eq.s $f2, $f4
c.eq.d $f2, $f4 } - compare equal to for single and double.

* Data transfer Instructions :-

Lwc1 $f1, 100($s2)
Swc1 $f1, 100($s2).

SUBWORD PARALLELISM

Definition: The process of slicing the ALU (ie 128 bit ALU can be sliced into 4-32 bit or 8-16 bit or 2-64 bit etc) so that each sliced ALU can be used for executing instruction simultaneously so that parallelism can be achieved. It is called subword because A word is sliced into different smaller slices these smaller slices are called as "Subword".

→ Every processor has its own graphic display.

→ Many graphic system uses 8-bit to represent the 3 primary colors and 8-bit for locating the pixel.

→ The graphic display, speaker, Microphone for teleconferencing & video game supports sound also simultaneously. (ie) They are performed in parallel.

→ Audio sample requires more than 8-bit so 16 bit is sufficient for audio.

→ The rising popularity of these multimedia application led to arithmetic instruction that supports narrow operations that can be operated in parallel.

Two general Enhancements are needed that are identified ~~in~~ ~~multimedia~~ in adapting programmable processors for Multimedia application

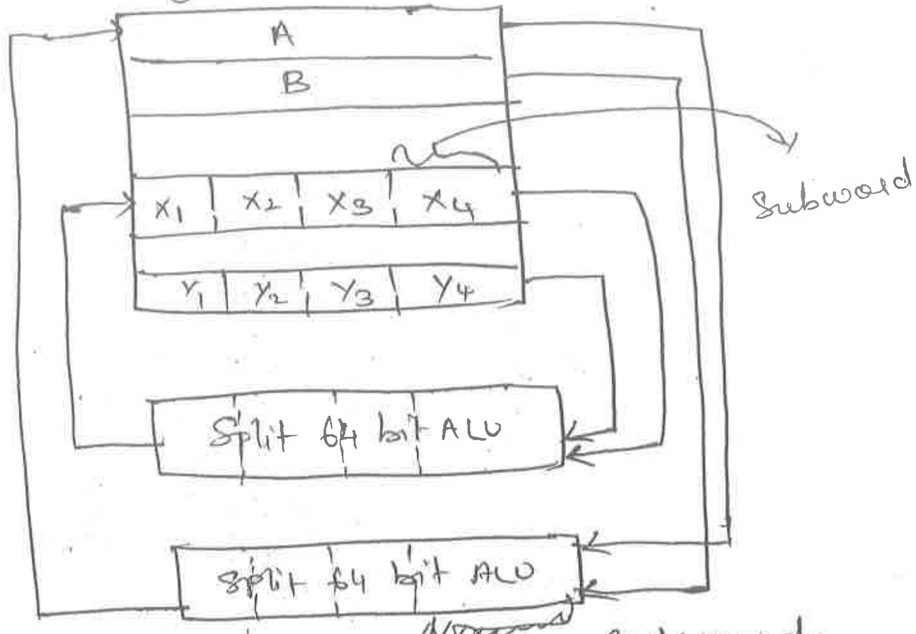
~~① To exploit instruction or data to achieve a significant increase in Computation power~~

① To exploit instruction or data level parallelism, called Vector SMID (Single Instruction Multiple Data) in order to achieve a increase in Computational power.

② To Introduce specialized instruction and integrated dedicated Hardware Modules.

Subword parallelism

general Registers



Data level parallelism can be achieved by partitioning 128-bit ALU into narrow slices enabling simultaneous arithmetic or logic operation on short vectors of 16 - 8bit operands, 8-16bit operands, 4-32bit operands or 2-64 bit operands.

→ Each subword can operate independently on independent data. The operations are all controlled by same opcode.

→ Cost of partitioning ALU was small.

ARM NEON Instruction for Subword parallelism :-

ARM (Advance Risk Machine), ARM 7, ARM v8 (Version 8) added More than 100 instruction in NEON — It is a Multimedia Instruction Extension supports Subword parallelism.

→ ~~Neon~~ NEON supports all the subword data types, such as 8-bit, 16 bit, 32 bit, 64 bit signed and unsigned integers.

→ 32 bit floating point number.

→ The MMX (Multimedia eXtension) and SSE (Streaming SIMD Extension) instruction for X86 (Architecture) include similar operation found in ARM NEON.

42

ARM NEON Instruction for Subword Parallelism.

Data transfer	Arithmetic	Logical / Compare
1) VLDR F32	VADD, F32	VAND 64
2) VSTR F32	VSUB, F32	VORR 64
3) VLD {1,2,3,4} {I8, I16, I32}	VMUL, F32	VAND, 128
4) VMOV {I64, I28}	VMIN	VORR, 128
	VMAX	VEOR 64 } EXOR
		VEOR 128 } EXOR