Assignment : 02

Md. Sabbir Akon
22301242
Sec: 12

Ans to the Q.S No. 1

Multiplicand = $(13)_{10}$ = $(01101)_2$

Multiplien = $(17)_{10}$ = $(10001)_2$

Itenation	Multiplicand	Product
0	01101	00000 10001
1	01101	00110 11000
2	01101	00011 01100
3	01101	00001 10110
4	01101	00000 11011
5	01101	00110 11101

Ans to the Q.S No. 2

0x ABB9609 = 1010 1011 1011 1001 0110 0000 1001

Bias = $2^5 - 1$ = 31 , exp bias = 21

∴ Actual exponent = - 21 - 31 = - 10

X = - 0.0018184226694

Ans to the QS No. 3

a) $50.7869 + 79.83 - 29.58$

$$\begin{aligned} 50.7869 &= 0.1100101100 1 \times 2^6 \\ (+)\ 79.83 &= 1.0011111101 0 \times 2^6 \end{aligned}$$

$$= 10.00001010011 \times 2^6$$

Now,

$$\begin{aligned} & 1.000001010011 \times 2^7 \\ (-)\ & 0.001110110010 \times 2^7 \quad (29.58) \end{aligned}$$

$$0.110010100001 \times 2^7$$

b) 64.2486×49.1832

$64.2486 = 1.00000000111 \times 2^6$

$49.1832 = 1.10001001 01 \times 2^5$

$\therefore (64.2486 \times 49.1832) = 1.10001 \times 2^{11}$

Ans to the Q.S No. 4

$28.4810 - (-4.0210)$

$= 28.4810 + 4.0210$

$$\begin{aligned} 28.4810 &= 1.1100011111 \times 2^4 \\ (+)\ 4.0210 &= 0.0100000001 \times 2^4 \end{aligned}$$

$$= 1.00000 \times 2^5$$

∴ actual exponent = 5

bias = $2^{8-1} - 1 = 127$

biased exp = 132

There is no underflow or overflow.

Ans to the Q.S No. 5

a) A bias is added to the actual exponent in the IEEE754 representation to allow both positive and negative exponents to be represented as unsigned integers. This simplifies hardware implementation since unsigned integers are easier to handle in binary. The bias ensures that the smallest possible exponent corresponds to 0, and the longest corresponds to the maximum unsigned value.

b) Optimized multiplications, for example: Karatsuba method, reduces complexity from $O(n^2)$ in traditional multiplication to $O(n^{\log_2 3})$ or $O(n\log n)$. This improves speeds by minimizing operations and leveraging efficient problem decomposition.

Ans to the Q.S No. 6

1. fadd.s

→ fadd.s instruction performs single-precision floating point addition

→ Syntax: ~~fadds~~ fadd.s $\underset{\text{dest}}{f_3}, \underset{\text{src1}}{f_1}, \underset{\text{src2}}{f_2}$

→ This instruction adds two single-precision floating point values.

→ fadd.s f_3, f_1, f_2

2. fsub.s

→ This instruction performs single precision floating point subtraction.

→ Syntax: fsub.s $\underset{\text{dest}}{f_3}, \underset{\text{src1}}{f_1}, \underset{\text{src2}}{f_2}$

→ It substracts src2 register value from source1 register value and then store the result

→ fsub.s f_3, f_1, f_2

3. fmul.s

→ This instruction performing single-precision floating-point multiplication.

→ syntax: fmul.s $\underset{\text{dest}}{f_3}, \underset{\text{src1}}{f_1}, \underset{\text{src2}}{f_2}$

→ It multiply src1 register's value with source 2 registers value and store the result

~~f~~

→ fmul.s f_3, f_2, f_1

4. fdiv.s

→ This instruction performs single precision floating point division.

→ Syntax: fdiv.s $\underset{\text{dest}}{\underline{f_3}}, \underset{\text{src1}}{\underline{f_1}}, \underset{\text{src2}}{\underline{f_2}}$

→ It divides src1 register value by source 2 registers value and store the output.

→ fdiv.s f_3, f_2, f_1

5. fsqrt.s

→ This instruction calculates the square root of a single precision floating point number.

→ Syntax: fsqrt.s $\underset{\text{dest}}{\underline{f_3}}, \underset{\text{src1}}{\underline{f_1}}$

→ Performs square root on the src1 register value and store the output.

→ fsqrt.s f_2, f_1

6. feq.d

→ This instruction checks whether the two double precision floating point values are equal or not.

→ syntax feq.d $\frac{f_3}{dest}$, $\frac{f_2}{src1}$, $\frac{f_1}{src2}$

→ It compares the values between src1 and src2, if its equal then stores 1 in the dest.

→ feq.d x_{25}, f_2, f_1

7. fle.d

→ This instuction checks whether one double-precision floating number is less than or equal to another double-precision number.

→ Syntax: fle.d $\frac{x_{25}}{dest}$, $\frac{f_1}{src1}$, $\frac{f_2}{src2}$

→ If src1 registers value is less than or equal to src2 registers value than it stores 1 in the dest.

→ fle.d x_{25}, f_1, f_2

8. flt.s

→ This instruction checks whether one single precision floating number is less than another single precision floating number.

→ Syntax : flt.s $\underset{dest}{x25}$, $\underset{src1}{f_1}$, $\underset{src2}{f_2}$

→ If src1 values is less than src2 than it stores 1 in the dest.

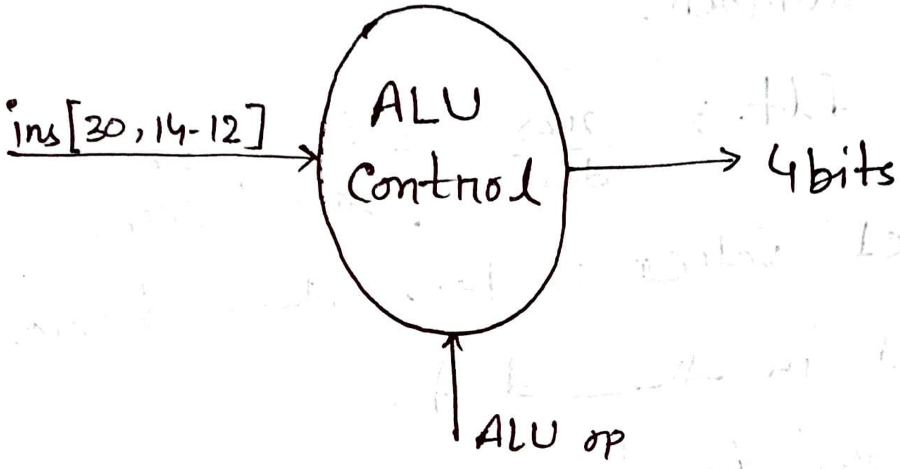
→ flt.s x_{23}, f_1, f_2

Ans to the Q.s No. 7

```
if  a==d :      a = f1
  jumpEqual     b = f2
else:
  jumpNotEqual

code:
    feq.d  x25, f1, f2
    beq   x0, x25, else
     jumpEqual
     Exit:
     else:
         jumpNotEqual
         beq x0, x0, Exit.
```

Ans to the Q.S No.8



ALU control takes input from ins[30,14-12] and ALU op from control unit to perfonms tasks such as add, sub etc.

a) No, the ALU control does not utilize Instruction bits 30 and 14-12 for the LD instruction. The LD instruction perfonms memory address calculation using addition, which is determined directly by the ALUop signal. Bits 30 and 14-12 are not required as they are relevant only for specifics R-type operations.

b) The ALU control utilizes instuetions bit 30 and 14-12 for R-type instauction (add, sub, AND, or) I-type instruction (SRAI), Branch instruction (Beq, Bne, Blt, Bge)

For R-type instruction, func7 [30] use to differentiate between operation like sub and add. func3 [14-12] use to specify the type of operation.

Ans to the Q.S No. 9

add x_{21}, x_{22}, x_{23} → R-type