CSE340

Assignment 02

Name: Umme Abira Azmary
ID : 20101539
section : 04

Ans. to the ques. no - 01:

Multiplicand $= 13_{10} = (1101)_2$

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

Iteration	Multiplicand	Product
1	1101	0000 10001. 1101 10001 01 10 11000
2	1101	0110 11000 0011 011 00
3	1101	0011 011 00 0001 10110
4	1101	0001 10110 0000 11011
5	1101.	0000 11011 1101 11011. 0110 11101.

Ans. to the ques. no-02:

0x ABB9609

A → 1010, B → 1011, B → 1011, 9 → 1001, 6 → 0110, 0 → 0000, 9 → 1001

$$\underbrace{1}_{\text{sign bit}}\underbrace{010101}_{\text{Biased expo}}\underbrace{110111001011000001001}_{\text{Fraction}}$$

sign = −

Biased exp. = 010101 = 21.

Bias = $2^{6-1}-1 = 31$.

∴ Exponent = 21 − 31.

= −10

Fraction = 110111001011000001001

$= 0.110111001011000001001$

$= 1\times2^{-1}+1\times2^{-2}+1\times2^{-4}+1\times2^{-5}+1\times2^{-6}+1\times2^{-9}+1\times2^{-11}+1\times2^{-12}+1\times2^{-18}+1\times2^{-21}$

= 0.8620648384

Decimal value $= (-1)^1 \times (1+0.8620648384)\times 2^{-10}$

$= -1.8620648384\times2^{-10}$

$= -1.818422694\times10^{-3}$

$= -1.81842\times10^{-3}$

Ans. to the ques. no - 03:

ⓐ $50.7869 + 79.83 - 29.58$

$= 110010.11001 + 1001111.11010 - 11101.10000$

$= 1.1001011001 \times 2^5 + 1.0011111010 \times 2^6 - 1.110110000 \times 2^4$

$= 1.10010 \times 2^5 + 1.00111 \times 2^6 - 1.110110 \times 2^4$

$= 0.110010 \times 2^6 + 1.00111 \times 2^6 - 1.110110 \times 2^4$

$= 10.00000 \times 2^6 - 1.110110 \times 2^4$

$= 1.00000 \times 2^7 - 1.110110 \times 2^4$

$= 1.00000 \times 2^7 - 0.001110 \times 2^7$

$= 0.11001 \times 2^7 \Rightarrow 1.10010 \times 2^6$

ⓑ $64.2486 * 49.1832$

$= 1000000.00111 * 110001.00101$

$= 1.00000 \times 2^6 * 1.10001 \times 2^5$

$= 1.00000 * 1.10001 \times 2^{(6+5)}$

$= 1.1000100000 \times 2^{11}$

$= 1.10001 \times 2^{11}$

Ans. to the ques. no-04:

$28.4810 - (-4.0210)$

$= 28.4810 + 4.0210$

$= 11100.01111 + 100.00110$

$= 1.11000 \times 2^4 + 1.00001 \times 2^2$

$= 1.11000 \times 2^4 + 0.01000 \times 2^4$

$= 10.00000 \times 2^4$

$= 1.00000 \times 2^5$

Bias $= 2^{8-1} - 1$.

$= 127$

Biased Exp. $= 127 + 5$

$= 132$

Range $= 0$ to $2^8 - 1$

$= 0$ to 255.

$= 1$ to 254 [reserved 0 and 255].

Here, $0 < 132 < 254$, so the result is none.

Ans. to the ques. no-05:

ⓐ Bias is added to the actual exponent to represent both positive and negative exponents using only unsigned, non-negative values. The reason is, it simplifies the hardware implementation. This affects the encoding of both positive and negative exponents. For example:-

For positive exponent while working with single precision:

An arbitary value 3 becomes $3+127 = 130$.

For negative exponent:

An arbitary value -3 becomes $-3+127 = 124$.

For zero exponent:

Zero becomes $= 0+127=127$.

As a result, it ~~vr~~ is visible that, both positive and negative exponents along with zero as encoded as non-negative ~~A~~ exponents.

ⓑ Optimized multiplication improves efficiency and performance compared to traditional long multiplication, by reducing the number of required ~~operations~~ components for instance: reducing multiplier by joining it with the upper half bits of the product, which leads to faster computation.

for larger numbers. Moreover, ~~it~~ the calculation is completed by one shift which increases effeciency. Also, ~~t~~ here the multiplicand value ~~d~~ need not to be double which reduces half of the calculations, because of all these, optimized multiplication improves efficiency and performance.

Ans. to the ques. no-06:

① fadd.s

(i) Do floating point addition for single-precision.

(ii) fadd.s des, ~~spc1~~ src1, src2

(iii) The single precision floating number of ~~se~~ src1 and src2 are performed addition and stores in des registration.

(iv) fadd.s f3, f4, f5

② fsub.s

(i) Do floating point substraction for single-precision.

(ii) fsub.s des, src1, src2

(iii) The value of src1 is substracted ~~from~~ by src2 and result ~~a~~ is stored in des.

(iv) fsub f4, f3, f2.

③ fmul.s

(i) Do floating point multiplication for single-precision.

(ii) fmul.s des, src1, src2

(iii) The values of single precision src1, src2 are multiplied and stored in des register.

(iv) fmul.s f4, f3, f2.

④ fdiv.s

(i) Do floating-point division for single precision number.

(ii) fdiv.s des, ~~rs~~ src1, src2

In floating
(iii) Values of single precision src1, src2, the value of src1 are divided by src2 and result is stored in des.

(iv) fdiv.s f2, f3, f4.

⑤ fsqrt.s

(i) ~~f~~ Do floating-point square root for a single precision number.

(ii) fsqrt.s des, src.

(iii) single-precision ~~fo~~ floating number of src is stored in square root and the result is stored in des.

(iv) fsqrt.s f20, f13.

⑥ feq.d

(i) Compares and evaluates whether two double precision numbers are equal.

(ii) feq.d des, src1, src2

(iii) if src1 == src2, the value 1 is stored in ~~reo~~ reg, otherwise 0

(iv) feq.d X1, ~~x~~ f2, f3.

(7) fle.d

(i) Shows comparison between two double precision numbers if one is less than or equal to othero number.

(ii) fle.d des, src1, src2.

(iii) If src1 is less than or equal to src2 then it ~~comes~~ stores 1 in des, otherwise 0.

(iv) fle.d x3, ~~f1~~, f2.

(8) flt.s

(i) checks whether a single precision number is less than other.

(ii) flt.s des, src1, src2

(iii) If src1 value is less than src1, the 1 is stored in des, otherwise 0.

(iv) flt.s x12, ~~x13~~ f3, f4

Ans. to the ques. no-7:

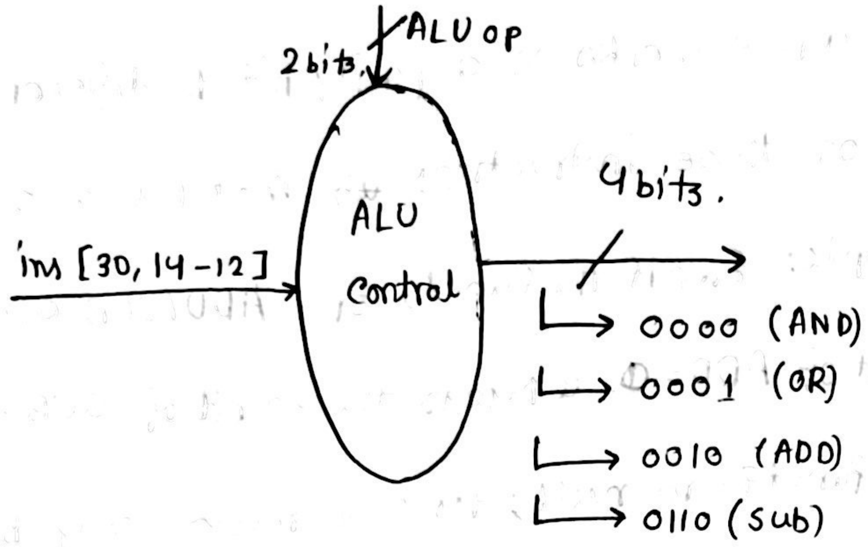
```
feq.d   x19, f1, f2

beq  x19, x0, JumpNotEqual

JumpEqual:
// codes for Jump Equal
beq x0, x0, exit

JumpNotEqual:
// codes for Jump not Equal.
Exit:
```

Ans. to the ques. no-08:

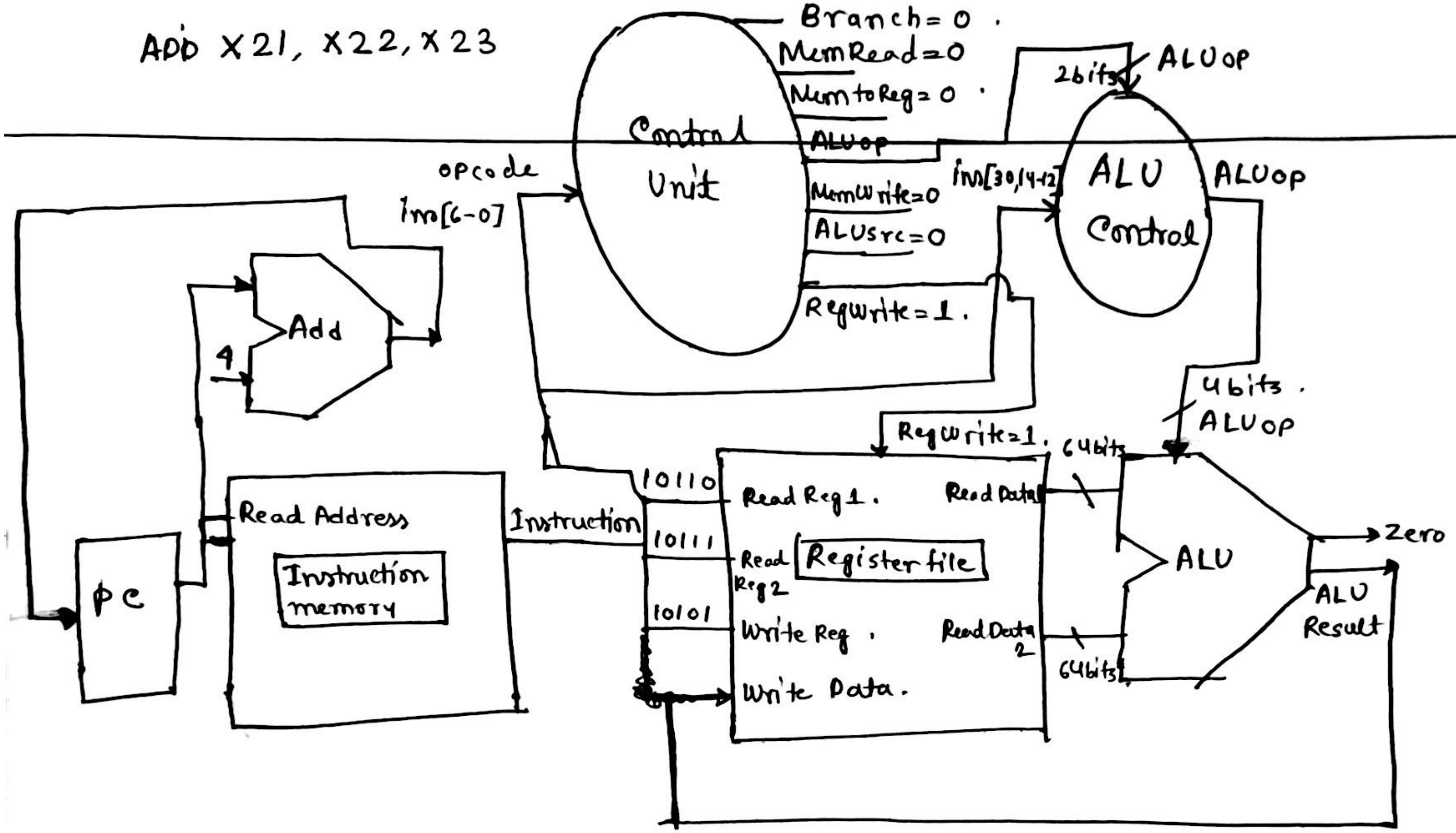


ⓐ The ALU control does not utilize instruction bits 30 and 14-12 to generate the output for the LD instruction. The reason is, the ALU operation for LD instruction is typically ~~as~~ (base + offset) addition which is determined by the control unit's ALU op's (2 bits) ~~bits~~ signal. Bit 30 typically determines the operation of R-type instruction ~~as~~ which is ~~irrel~~ irrevalent to this operation also the funct3 is used to distinguish the load type. So, the ALU control's operation on LD instructi--on is determined by Control Unit's ALUop signal.

ⓑ The ALU control utilizes instruction bits 30 and 14-12 in the case of handling R-type instruction's arithmatic and logical operations. The 14-12 bits are typically named as funt3 which specifies the general types of operations. For example: 000 is parsed for performing the ADD/SUB operation. The 30 bit is a ~~par~~ value of funct7 which works as a parity Bit to differentiate the operation of those instructions ~~ut~~ that has a same funct3. For example: Even if the funct3 of ADD/SUB are same, the 30 bit of ADD = 0, whereas the 30 bit of SUB = ~~0~~1, other values of funct7 are null; for these reasons only bit 14-12 and 30 ~~is~~ are used in R-type operations.

Ans. to the ques. no-09:

ADD X21, X22, X23



Here, all the control signal values are:

Branch = 0

MemRead = 0

MemtoReg = 0.

ALUop = XX.

MemWrite = 0

ALU Src = 0

Reg Write = 1.