EECS 2021 Lab 04 Report

By : Rayhaan Yaser Mohammed
Student ID: 219739887

Introduction:

In this lab we are exploring dealing with multiplication, division annd remainder. This labs also introduces the concept of Float Points and the various Intructions used for calculating various values related to FP.

1) d1a.asm

```
addi x5,x0,11
addi x6,x0,12
mul  x7,x5,x6
```

This code multiplies the values stored in the registers x5 and x6 and stores the value in registers up to a value of 64 bits.

2) d1b.asm

```
v1:     DD -2
v2:     DD 16
        ld x5, v1(x0)
        ld x6, v2(x0)
        mul x7,x5,x6        ;lower 64 bits
        mulh x8,x5,x6       ;upper 64 bits (signed, signed)
        mulhu x9,x5,x6      ;upper 64 bits (unsigned, unsigned)
        mulhsu x10,x5,x6    ;upper 64 bits (signed, unsigned)
        mulhsu x11,x6,x5    ;upper 64 bits (signed, unsigned)
```

This code is used for multiplication which produces a result more than 64 bits the mulh, mulh,mulhsu are instructions provided for calculations. In the above example we have used -2 as a multiplicand with a hex reperesntaion of 0xfffffffffffffe we then multiply by 16 which shifts the hexadecimal digits by 1 position (4 bits) to the left.

3) d1c.asm

```
s0: DC "Rectangle area Calculation.\0"
s1: DC "Enter a:\0"
s2: DC "Enter b:\0"
s3: DC "Rectangle area =\0"
```

```
        addi x5,x0,s0
        ecall x0,x5,4    ;out info
        addi x5,x0,s1
        ecall x1,x5,4    ;prompt a
        ecall x6,x0,5    ;input a
        addi x5,x0,s2
        ecall x1,x5,4    ;prompt b
        ecall x7,x0,5    ;input b
        mul x6,x6,x7     ; area = a*b
        addi x5,x0,s3
        ecall x1,x5,4    ;output area
        ecall x1,x6,0    ;output result
```

This code uses the ecall and mul to get inputs of the parameters of a rectangle and then calculates the area by mul (the multiplication instruction) and then outputs the result using the ecall command.

4) d1d.asm

```
addi x5,x0,1000
addi x6,x0,20
div x7,x5,x6
```

This code uses the div instruction to divide two values stored in the registers. Then stores the output in x7.

5) d1e.asm

```
addi x5,x0,67
addi x6,x0,16
rem x7,x5,x6
```

This code uses the rem instruction to find the reminder in the above case the code does 67%16 to obtain the remainder 3 which is subsequently stored in the register x7.

6) d1f.asm

```
s0: DC "Greatest Common divisor.\0"
s1: DC "Enter a:\0"
s2: DC "Enter b:\0"
s3: DC "GCD(a,b)=\0"
        addi x5,x0,s0
        ecall x0,x5,4    ;out info
```

```
        addi x5,x0,s1
        ecall x1,x5,4    ;prompt a
        ecall x6,x0,5    ;input a
        addi x5,x0,s2
        ecall x1,x5,4    ;prompt b
        ecall x7,x0,5    ;input b
loop:   rem x8, x6, x7
        add x6, x0,x7
        add x7, x0,x8
        bne x8,x0,loop
        addi x5,x0,s3
        ecall x1,x5,4    ;output gcd
        ecall x0,x6,0    ;out result
```

This code uses the code takes in 2 inputs positive integers a and b then it uses the rem and loop to do the euclidean algorithm to find the GCD and then out putts the result usign the ecall instruction.

7) d2a.asm

```
d1:     DF 2.0
d2:     DF 3.0
        fld f1, d1(x0)
        fld f2, d2(x0)
```

This code first defines two float points and then uses the fld (FP load Double) instruction to load the FP values into the memory into FP registers.

8) d2b.asm

```
d1:     DF 2.0
d2:     DF 3.0
        fld f1, d1(x0)
        fld f2, d2(x0)
        fadd.d f3,f1,f2  ;2.0 + 3.0 = 5.0
        fsub.d f4,f1,f2 ;2.0-3.0 = -1.0
        fmul.d f5,f1,f2 ;2.0*3.0 = 6.0
        fdiv.d f6,f1,f2 ;2.0 / 3.0 = 0.(6)
```

This code introduces the FP addition, subtraction, multiplication and division instructions. In the above code we take the defined float point stored in f1 and f2 and then use the instruction to perform simple arithmetic operations.

9) d2c.asm

```
v1:     DF -1.1e17
v2:     DF 1.1e17
v3:     DF 3.0
        fld f1, v1(x0)
        fld f2, v2(x0)
        fld f3, v3(x0)
        fadd.d f4,f1,f2
        fadd.d f4,f4,f3
        fadd.d f5,f2,f3
        fadd.d f5,f1,f5
```

The above code takes the defined float points and then uses the fld instruction to load into float point register and then uses the fadd.d Intructions to perform sums of f4 = (v1 +v2) +v3 and F5 = v1 + (v2 +v3). But there is a loss of precision due to the bumber being greater than 52 bits.

10) d2d.asm

```
v1:     DF -1.1e15
v2:     DF 1.1e15
v3:     DF 3.0
        fld f1, v1(x0)
        fld f2, v2(x0)
        fld f3, v3(x0)
        fadd.d f4,f1,f2
        fadd.d f4,f4,f3
        fadd.d f5,f2,f3
        fadd.d f5,f1,f5
```

This code perform the same task as d2c.asm but is more precision in calculation as float point values are more precise only until 52 bits or 15 decimal digits without rounding and uses precise calculation until 10^15 .

11) d2e.asm

```
d:      DF 3.0
        fld f1, d(x0)
        fadd.d f2,f1,f1
        fmul.d f3,f1,f1
        fdiv.d f4,f2,f3
        fsd f4,m(x9)
m:      DM 1
```

The code above uses the fsd (FP store double) instruction to store the calculated FP values from the code into the memory registers as seen int the code we load the calculated FP values into the memory register x9.

12) d2f.asm

```
fc1:    DF 1.0
fc2:    DF 2.0
        fld f1, fc1(x0)
        fld f2, fc2(x0)
        fmin.d f3,f1,f2
        fmax.d f4,f1,f2
```

This code uses the fmin.d and fmax.d which takes the 2 registers given and stores the minimum or the maximum in the register. In the above code we see The fmin.d is used to find the minimum between f1 and f2 and the result is stored in f3, same for fmax.d the maximum value between f1 and f2 and stores it in f4.

13) d2g.asm

```
d:      DF 2.0
        fld f1,d(x0)
        fsqrt.d f2,f1
```

This code uses the input float point values and Uses the fsqrt.d to calculate the square root of the input values and then stores the value in f2.

14) d2h.asm

```
fc1:    DF -2.0
fc2:    DF 3.0
        fld f1, fc1(x0)
        fld f2, fc2(x0)
        fsgnj.d f3,f1,f1
        fsgnjn.d f4,f1,f1
        fsgnjx.d f5,f1,f1
```

In this source we use fsgnj.d (FP sign inject), fsgnjn.d(FP sign inject NOT), fsgnjx.d (Sign Inject XOR) to inject sign into the values. In the above example we use fsgnj.d to make f3 = f1 and then fsgnjn.d to make f4 = - (f1) and then the fsgnjx.d to make f5 = abs(f1) .

15) d2i.asm

```
d:      DF 2.0
        fld f0, d(x0)
        fmv.x.d x30, f0
        addi x31, x0, -1
        fmv.d.x f1,x31
```

This code above uses the fmv.x.d (FP move double to long) and fmv.d.x (FP move long to double) This instruction moves data between the FP and the int register without the conversion. Here the int from x30 is moved from x30 to FP f0 and the FP in f1 is moved to int register x31

16) d2j.asm

```
addi x5,x0,22
addi x6,x0,7
fcvt.d.l f5,x5
fcvt.d.l f6,x6
fdiv.d f7,f5,f6
fcvt.l.d x7, f7
```

The fcvt.1.d (FP convert double to long) and fcvt.d.1 (FP convert long to double) instructions conduct FP to integer and vise versa conversions. In the above code we calculate the approximate value of by an FP division (22.0/7.0) and then convert the obtained result to an integer.

17) d2k.asm

```
s1:     DC "Enter x:\0"
s2:     DC "Result :\0"
c0:     DF 0.0
a:      DF 3.0
b:      DF 5.0
        fld f0,c0(x0)
        fld f1,a(x0)
        fld f2,b(x0)
loop:   addi x5, x0, s1
        ecall x1, x5, 4
        ecall f3, x0, 6
        flt.d x1,f3,f1
        beq x1,x0,cont
        fadd.d f3,f0,f2
        beq x0,x0,done
```

```
cont:   fle.d x1,f2,f3
        beq x1,x0,done
        fadd.d f3,f0,f1
done:   addi x5,x0,s2
        ecall x1,x5,4
        ecall x0,f3,1
        beq x0,x0,loop
```

This code uses the flt.d (FP less than) and fle.d (FP less or equal) instructions, to Compare double precision FP values. The code above uses the value of x provided by the user at run time. Then the program will prompt the user to enter a value for x, then it will calculate the new value of x as per the pseudo-code
(if (x<a) x-b else if (b<=x) x=a),

and finally output the result all this is contained with a loop which is looped until it is cancelled

Conclusion:

From this lab i have learned more assembly instruction which can be used to perform operations.
How to use Assembly instructions related to Float Point (FP). There are number of compands all performing different functions and different process to manipulate a float point value. We can add, subtract, divide, changes sign, or even move register and data types for example using fmv.x.d we can move it into from a long to double or vice versa.