Lab Sheet 4 for CS F342 Computer Architecture

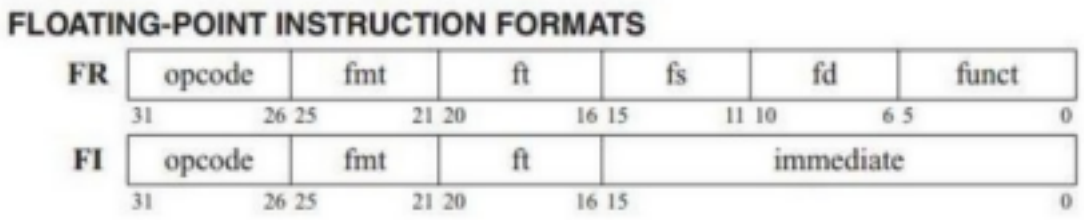
Semester 1 – 2023-2024

Explanation

Goals for the Lab: We build up on Lab Sheet 2 and explore floating point instructions. We also use some new registers (especially those for floating point operations and remainder for integer arithmetic – HI / LO) and convert across numeric data types.

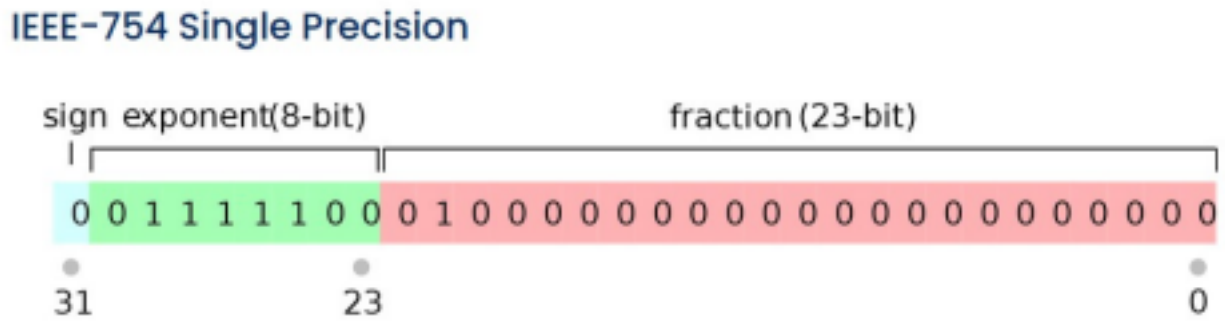
Reference for Floating point Instructions – refer to the **MIPS Reference Data sheet ("Green sheet" -** ARITHMETIC CORE INSTRUCTION SET – Page 1, column 2).

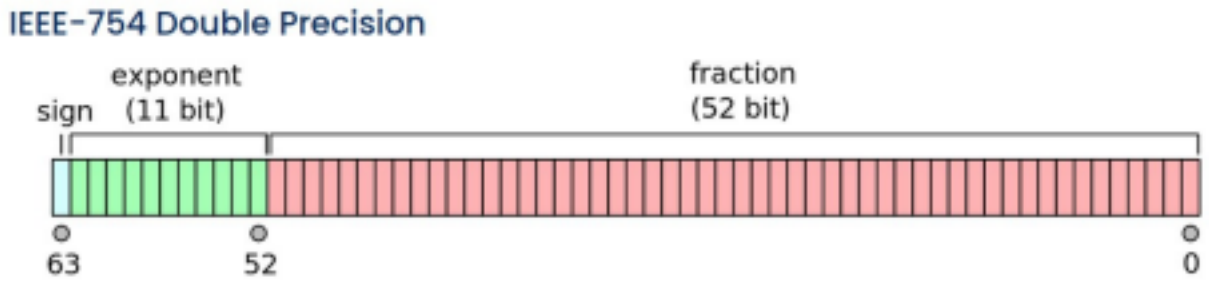
Some of the new instructions (and pseudo instructions) are l.s, mov.s, l.d, mov.d, cvt.d.s, cvt.s.d and similarly single precision and double precision flavours of add, sub, neg, mul, div.



There are 32 single precision floating point registers / 16 double precision values. In QtSpim they are labelled as FG0, FG1, FG2 .. FG31 for single precision and FP0, FP2, FP4 .. FP30 for double precision.

Floating point representation





Class Exercises

Exercise 1: Print float value by taking the float value from data segment and from the user at run time.

```
.data

val1: .float 0.001


.text

main:

l.s $f12, val1
li $v0,2

syscall

li $v0, 6
syscall

mov.s $f12, $f0
li $v0,2 syscall

li $v0,10
syscall
```

Exercise 2: Use double precision using l.d and mov.d instead of l.s and mov.s. Also use

cvt.d.s and cvt.s.d to convert across precession types.

Exercise 3: Add/Sub of float: use instructions like add.d and add.s; sub.s, neg.s etc. Use addition along with negation to implement subtraction.

Hint: Read float value, the value will be store in $f0

move it to any single precision floating registers $f1 and $f2

add.s $f12, $f1, $f2

sub.s $f12, $f1, $f2

neg.s $f12, $f1

print the float value using li $v0, 2 syscall

Exercise 4: Using the below example for Integer Multiply / Divide. Extend the example to perform the mul/divide for floating number and also find remainder in division operation ?

```
.data
 str0: .asciiz "\nMul:"
 str1: .asciiz "\nDiv:"
 w1: .word 300
 w2: .word 7

  .text

 main:

 lw $t0, w1
 lw $t1, w2

 la $a0, str0

 li $v0, 4
 syscall
 mul $a0, $t0, $t1
```

```
li $v0,1 # print from $a0 syscall
la $a0, str1
li $v0, 4
syscall

div $a0, $t0, $t1

li $v0,1 # print from $a0 syscall
# Modify to also print the modulo / remainder

li $v0,10
syscall
```

Exercise 5: Calculate the result of the expression Ax + B, where the values of A and B are provided in the program's data segment as a word. The input value x should be obtained as a single precision floating point number from the user. Utilise instructions such as mtc1 (move to floating-point register), cvt (convert), and mul (multiply) to perform the necessary calculations. And print the result on the console.

Hint: Convert A and B from word to single precision floating point, and solve the polynomial

Exercise 6: Explore disassembly for the new instructions.

1. 44880000
2. 46800060
3. 460208c2
4. 3c041001
5. 44042000
6. 0000000c

Practice Questions

Convert Integer to Single precision float and vice versa. Also compute the conversion errors for mathematical operations – e.g. rounding off during divide, precision loss during convert to float and multiple etc. Some of the new instructions are cvt.s.w, cvt.w.s, mtc1/ mtcz, mfc1/mfcz ... Note that cvt.x.w or cvt.w.x (where we convert from / to integers)

can only be done with registers in CP1. So if the value is in the main registers you first need to use mtc1. Similarly after converting to integer, you need to use mfc1 to get the values in regular registers. Skeletal algorithm:

A. Define two integers with relatively large values – but ensure that no overflow in multiplication

B. multiply and print output – save integer output to a spare register

C. Convert both integers to single precision float; multiply and print output using float [make sure to use mtc1 first and then cvt.s.w or similar

D. Convert the output to integer and compare with saved values from step B – first convert to integer in co-processor 1 register and then move the value to a0 register of main processor.

E. Convert both integers to double precision float; multiply and print using double precision float
F. Convert the output to integer and compare with saved values from step B //G. Print difference in output of D and E – (may happen only if large integer values are used)

Hint: Take two larger integers such that the multiplication should not create arithmetic overflow i.e., with in the limit 2^31-1 and greater than 2^23

2) Convert both the integer to single precision floating point and double precision floating point

3) The multiplication of the two floating point number will give precision error with respect to multiplication of two integer in single precsion as single precision contains only 23 bits after decimal (mentissa is 23 bit long), where as in double the difference of multiplication of integer and double precision is 0