# Lab# 8 FLOATING POINT

**Instructor**: I Putu Danu Raharja.

**Objectives**:
Learn to carry out arithmetic operations using a floating-point representation of real numbers. Learn to use logical operations to mask fields within a word.

**Method**:

Write assembly code to implement a function for floating-point multiplication.

**Preparation**:

Read the chapter 3 of lecture textbook.

**File To Use**: float2.asm

## 8.1 FPU REGISTERS

The floating-point unit has 32 floating-point registers. These registers are numbered like the CPU registers. In the floating-point instructions we refer to these registers as $f0, $f1, and so on. Each of these registers is 32 bits wide. Thus, each register can hold one single-precision floating-point number. How can we use these registers to store double precision floating-point numbers? Because these numbers require 64 bits, register pairs are used to store them. This strategy is implemented by storing double-precision numbers in even-numbered registers. For example, when we store a double-precision number in $f2, it is actually stored in registers $f2 and $f3.

Even though each floating-point register can hold a single-precision number, the numbers are often stored in *even* registers so that they can be easily upgraded to double-precision values.

## 8.2 FLOATING-POINT REPRESENTATION

Single precision floating point number (32-bit):

| Sign (1) | Exponent (8) | Fraction (23) |
|---|---|---|

Value = $(-1)^{\text{Sign}} * (1.F)_{\text{two}} * 2^{\text{Exp} - 127}$

Double precision floating point number (64-bit):

| Sign (1) | Exponent (11) | Fraction (52) |
|---|---|---|

$\text{Value} = (-1)^{\text{Sign}} * (1.F)_{\text{two}} * 2^{\text{Exp} - 1023}$

## 8.3 FLOATING-POINT INSTRUCTIONS

The FPU supports several floating-point instructions including the standard four arithmetic operations. Furthermore, as are the processor instructions, several pseudo-instructions are derived from these instructions. We start this section with the data movement instructions.

### A. Move instructions

| Instruction | Example | Meaning |
|---|---|---|
| mov.s Fdst, Fsrc | mov.s $f0, $f1 | to move data between two floating-point registers (single). |
| mov.d Fdst, Fsrc | mov.d $f0, $f2 | to move data between two floating-point registers (double). |
| mfc1 Rdest, FRsrc | mfc1 $t0, $f2 | to move data from the FRsrc floating-point register to the Rdest CPU register (single). |
| mfc1.d Rdest, FRsrc | mfc1.d $t0, $f2 | to move data the two floating-point registers (FRsrc and FRsrc+1) to two CPU registers (Rdest and Rdest+1). |
| mtc1 FRdest, Rsrc | mtc1 $f2, $t0 | to move data from the Rdest CPU register to the FRsrc floating-point register. |

### B. Load and Store Instructions

| Instruction | Example | Meaning |
|---|---|---|
| lwc1 FRdst, address | lwc1 $f0, 0($sp) | load a word from memory to an FPU register. |
| swc1 FRdst, address | swc1 $f0, 0($sp) | stores the contents of FRdest in memory at address. |
| l.s FRdest,address | l.s $f2,0($sp) | Pseudo-instructions to load and store data from/to memory. |
| l.d FRdest,address | l.d $f2,0($sp) | |
| s.s FRdest,address | s.s $f2,0($sp) | |
| s.d FRdest,address | s.d $f2,0($sp) | |

## C. Comparison Instructions

Three basic comparison instructions are available to compare floating-point numbers to establish <, =, and ≤ relationships. All three instructions have the same format. We use the following to illustrate their format.

```
c.lt.s FRsrc1,FRsrc2        # for single-precision values
c.lt.d FRsrc1,FRsrc2        # for double-precision values
```

It compares the two floating-point values in FRsrc1 and FRsrc2 and sets the floating point condition flag if FRsrc1 < FRsrc2.

To establish the "equal to" relationship, we use **`c.eq.s`** or **`c.eq.d`**. For the ≤ relationship, we use either **`c.le.s`** or **`c.le.d`** depending on the precision of the values being compared.

Once the floating-point condition flag is set to reflect the relationship, this flag value can be tested by the CPU using **`bc1t`** or **`bc1f`** instructions. The format of these instructions is the same. For example, the instruction

```
bc1t target
```

transfers control to target if the floating-point condition flag is true. Here is an example that compares the values in $f0 and $f2 and transfers control to skip1 if $f0 < $f2.

```
c.lt.s $f0,$f2      # $f0 < $f2?
bc1t skip1          # if yes, jump to skip1
```

We don’t really need instructions for the missing relationships >, □ =, or ≥. For example, the code

```
c.le.s $f0,$f2      # $f0 ≤ $f2?
bc1f skip1          # if not, jump to skip1
```

transfers control to skip1 if $f0 > $f2.

### D. Arithmetic Instructions

| Instruction | Example | Meaning |
|---|---|---|
| sub.s FRdest,FRsrc1,FRsrc2 | sub.s $f0, $f2, $f4 | $f0 = $f2 - $f4 |
| sub.d FRdest,FRsrc1,FRsrc2 | sub.s $f0, $f2, $f4 | |
| add.s FRdest,FRsrc1,FRsrc2 | add.s $f0, $f2, $f4 | $f0 = $f2 + $f4 |
| add.d FRdest,FRsrc1,FRsrc2 | add.d $f0, $f2, $f4 | |
| div.s FRdest,FRsrc1,FRsrc2 | div.s $f0, $f2, $f4 | $f0 = $f2 / $f4 |
| div.d FRdest,FRsrc1,FRsrc2 | div.d $f0, $f2, $f4 | |
| mul.s FRdest,FRsrc1,FRsrc2 | mul.s $f0, $f2, $f4 | $f0 = $f2 * $f4 |
| mul.d FRdest,FRsrc1,FRsrc2 | mul.d $f0, $f2, $f4 | |
| abs.s FRdest,FRsrc | abs.s $f0, $f4 | $f0 = abs($f4) |
| abs.d FRdest,FRsrc | abs.d $f0, $f4 | |
| neg.s FRdest,FRsrc | neg.s $f0, $f4 | Sf0 = –$f4 |
| neg.d FRdest,FRsrc | neg.d $f0, $f4 | |

### E. Conversion Instructions

| Instruction | Meaning |
|---|---|
| cvt.s.w FRdest, FRsrc | Convert integer to single-precision floating-point value. |
| cvt.d.w FRdest, FRsrc | Convert integer to double-precision floating-point value. |
| cvt.w.s FRdest, FRsrc | Convert single-precision floating-point number to integer. |
| cvt.d.s FRdest, FRsrc | Convert single-precision floating-point number to double precision floating-point number. |
| cvt.w.d FRdest, FRsrc | Convert double-precision floating-point number to integer. |
| cvt.s.d FRdest, FRsrc | Convert double-precision floating-point number to single precision floating-point number. |

### F. System I/O

| Service | Code in $v0 | Argument(s) | Result(s) |
|---|---|---|---|
| Print float | 2 | $f12 = number to be printed | |
| Print double | 3 | $f12-13 = number to be printed. | |
| Read float | 6 | | Number returned in $f0. |
| Read double | 7 | | Number returned in $f0-1. |

## 8.4 EXERCISE:

1. The file *float2.asm* contains an outline of assembly code for a procedure that multiplies two numbers in IEEE 754 single precision floating point format. The file also contains a program for testing the multiplication function. Complete the

assembly code for the *fmult* procedure. Ignore the possibility of overflow and underflow, and do not round the result.

2. Write an interactive program that will convert input temperatures in Fahrenheit to Celcius. The program should prompt the user for a temperature in Fahrenheit and then display the corresponding temperature in Celcius (C = 5/9*[F-32]).