Computer Architecture - CS2323
Lab 6
Report
Double Floating Point Arithmetic in RV64I Assembly

Devraj ES22BTECH11011

Introduction

This lab focuses on the implementation of a RISC-V Double Precision floating point adder/multiplier using assembly code. The program adds and multiplies the pairs of floating point values provided and stores the addition and multiplication results in memory.

Initialization

The program first initializes the addresses for the inputs and outputs, and stores the specifications like the number of pairs and the float numbers. Then it isolates the sign, exponent and fraction values from the given numbers. The isolated values are kept in their respective places without any shifting.

Rounding Policy

The standard rounding policy of **round to nearest, ties to even** is followed here.

Addition

The code does the following to add two numbers:

First, it identifies the larger of the two numbers using their exponent values and stores the difference between their exponents.

If the exponents are equal, the fraction parts are compared.

The sign bit and exponent are kept according to the larger number.

Now, the smaller number's fraction part is shifted by the difference in the exponents (with the bias kept in mid). Then, the fraction parts are added and the result is normalized.

While normalizing the exponent is changed according to the shifting of the result and rounding of values is done in case of shifting right.

The final result is computed by simply adding the resulting sign, exponent and fraction bits and is stored in memory.

Multiplication

For multiplication, the following steps are taken:

The sign bit of the result is computed by XOR-ing the the sign bits of the two values.

The exponent values are added and the value of the bias (1023) is added to the result.

To multiply is mantissae of the two numbers the subroutine dmul is called.

Inside dmul, multiplication is done through the shift right and add process as discussed in class.

While shifting, the discarded bit is kept track of using a register and the OR'd value of the previous discarded bits is maintained for the purpose of rounding.

The multiplied result of dmul is then shifted to fit in 52 bits, while changing the exponent accordingly.

The resultant sign, exponent and fraction parts are then combined to give the multiplied result and is stored in memory.

Special Cases

Special cases like where the value of the exponent is 11111111111 (i.e. infinities and NAN's) and overflows/underflows are taken care of according to the lab problem statement.

To do this, the isolated parts of the numbers were compared with the sign, exponent and fraction values for the special cases.