Computer Architecture - CS2323. Autumn 2024
Lab-6 (Double Floating Point Arithmetic in RV64I Assembly)

Double-precision floating point addition and multiplication in RV64I assembly code (F, D, M extensions strictly not allowed). **THIS ASSIGNMENT IS TO BE DONE INDIVIDUALLY.**

Problem Statement:
Write two assembly routines named **fp64add** and **fp64mul** (in the same **main.s** file) which take two double precision floating point numbers as arguments, perform floating point addition or multiplication on them, and return a double precision floating point number. Multiplication should be done as another routine named **dmult** and called within the fp64mul. **dmult** routine should take two numbers as parameters and return their product. All routines should follow standard calling conventions for register usage.
The floating point computations should take care of rounding to the nearest half.

A set of configurations and then input numbers are provided as dwords in the data segment starting from address 0x10000000.
Please be careful with the addresses given and use only these addresses in your code.

Input Format:
Floating point configuration starts from the address **0x10000000** (data segment base + 0)
The first dword in the data segment indicates the exponent bits in the floating point standard used (11-bits for double precision)
The second dword in the data segment indicates the mantissa bits in the floating point standard used. (52-bits for the double precision)

The input starts from the address **0x10000010** (data segment base + 0x010)
The first dword at the address 0x10000010 indicates the number of floating point pairs to be operated upon. The subsequent dwords indicate the floating point pairs (Refer to the starter code below for better understanding)

The double precision addition and multiplication computed from your code should be present starting from address **0x10000200** in the memory (data segment base + 0x200), as shown below.

Optional: You may use your own created simulator (Lab-4) to simulate the program and test out your assembly code as well as the simulator!

Note: For this assignment, it is sufficient to implement the IEEE floating point standard for double precision which is 11 bits for exponent and 52 bits for mantissa. The floating point computations should take care of rounding to the nearest half.

Regarding denormal/infinity/NaN numbers:

1. Any operation with NaN should result in NaN
2. Any operation with 0 (or) infinity should behave as shown:
    a. -infinity * -6 = infinity
    b. -infinity + infinity = 0
    c. infinity * 0 = NaN
3. When the exponent overflows~~/underflows~~, it should result in +infinity/-infinity
4. The number with zero exponent and non-zero fraction will not appear in any of the test cases

Extra Credit: Make your code generic to work for any values of exponent and mantissa being set in the first two dwords in the data section. Report error if the sum of exponent and mantissa bits exceeds 63.

Starter Code:

```
#starts at 0x10000000
.data
#fp_exponent_bits - number of exponent bits. It should be 11 for double precision
number
#fp_mantissa_bits - number of mantissa bits. It should be 52 for double precision
number
.dword 11, 52

#starts at 0x10000010
#count_of_floating_pairs - tells the number of floating point pairs to be operated
upon.
.dword count_of_floating_pairs, input11, input12, input21, input22, input31,
input32, ...

.text
#The following line initializes register x3 with 0x10000000
#so that you can use x3 for referencing various memory locations.
lui x3, 0x10000
#your code starts here
<<YOUR CODE>>

#The final result should be in memory starting from address 0x10000200
#The first dword location at 0x10000200 contains sum of input11, input12
#The second dword location at 0x10000200 contains product of input11, input12
#The third dword location from 0x10000200 contains sum of input21, input22,
#The fourth dword location from 0x10000200 contains product of input21, input22, and
so on.
```

Samples:

Input 1:

```
.data
.dword 11, 52
.dword 1
.dword 0x3fd2c00000000000, 0x40a22676f31205e7
```

Output 1:

Starting Memory Address	dword
0x10000200	0x40a2270cf31205e7
0x10000208	0x4085451364d91eeb

Input 2:

```
.data
.dword 11, 52
.dword 2
.dword 0x409fa0cf5c28f5c3, 0x4193aa8fc4f0a3d7
.dword 0x40e699d2003eea21, 0x420e674bcb5a1877
```

Output 2:

Starting Memory Address	dword
0x10000200	0x4193aaaf65c00000
0x10000208	0x4243700f85975d74
0x10000210	0x420e675171ce9887
0x10000218	0x43057929844f64ac

Submission instructions:

1. THIS ASSIGNMENT IS TO BE DONE **INDIVIDUALLY**.
2. File should be named Lab6_ROLLNUM.zip (Lab6_CSYYBTECHXXXXX.zip)
3. The zip file should include your assembly code **main.s** and a **report.pdf** explaining your design approach, verification approach, and any specific issues you encountered (if any).
4. If doing extra credit, clearly mention it in the report.
5. The code should work on RIPES (RV64I configuration) without any changes
6. The input and output format and locations should exactly match as specified
7. Verify your code properly with various input combinations.
   Suggestion/Hint: You can write a program in C/C++/python by using float data type to verify your generated results (this program does not need to be submitted). One example computation was also discussed in class and shared in slides.