Computer Architecture - CS2323. Autumn 2025

Lab-3: Taylor Series Implementation using RISC-V F-extension

As a part of this lab assignment, you are required to implement several common scientific functions in assembly using the RISC-V F-extension. The functions to be implemented in this assignment are:

- exp(x) (Function Code: 0)
- sin(x) (Function Code: 1)
- cos(x) (Function Code: 2)
- ln(x) (Function Code: 3)
- 1/x (Function Code: 4)

Each function must be implemented using its Taylor Series up to the specified number of terms. If you are not aware, you may use this link as a reference for details on Taylor Series expansions of commonly used functions.

The set of input numbers are provided as 32-bit words in the data segment starting from address 0x10000000. The values computed from your code should be present starting from address 0x10000200 in the memory (data segment base + 0x200), as shown below. The first word in the data segment indicates the total group of input numbers (total number of function values to be computed) followed by the actual function specifications one-by-one. Floating point values (x) are in IEEE-754 FP32 (single float) format.

You can use any RISC-V F-extension instruction available (flw, fsw, fadd.s, fsub.s, fmul.s, fdiv.s etc.). The detailed official spec is available at this link (Ch-20) and another quick reference is here.

```
.data
.word N, func1_code, func1_x, func1_terms, func2_code, func2_x, func2_terms,
… funcN_code, funcN_x, funcN_terms

.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

   #The final result should be in memory starting from address 0x10000200
   #The first word location at 0x10000200 contains the Taylor Series
approximation of func1(func1_x) up to func1_terms in FP32 format.
   #The second word location from 0x10000200 contains the Taylor Series
approximation of func2(func2_x) up to func2_terms in FP32 format.
and so on.
```

Example: If the data section contains .word 2, 1, 0x3f800000, 5, 0, 0x40000000, 8 - it indicates that we have 2 function values to be calculated:
The first function is sin(1.0) up to 5 terms, and second is exp(2.0) up to 8 terms. Here 1.0 and 2.0 are equivalent to FP32 numbers 0x3f800000 and 0x40000000 respectively.

After executing your code, memory location from 0x10000200 should contain 0x3f576aa4 and 0x40ec30c4, which are the Taylor series approximations of sin(1.0f) and exp(2.0f) in FP32 format.

Note: You may assume that the input values are in the convergence domain of the Taylor Series. Further, the number of terms can be assumed to be such that it will not lead to any inherent overflow.

You should implement the code in a modular manner using subroutines for each supported function, and helpers like fact and pow. Ensure you follow the standard ABI convention whenever using function calls.

Your code must handle the following erroneous cases by returning NaN in FP32 format:

1. If function code provided in input is invalid (not in {0, 1, 2, 3, 4})
2. If number of terms provided in input is invalid (less than or equal to zero)
3. If the value x provided in input does not fall in the domain of the specified function (e.g. x equal to zero for 1/x etc.)

Verification: There can be many different approaches to verifying the code. One approach is to write a similar (need not be most efficient) program in Python or C to generate expected outputs for various kinds of inputs and match them with what is generated in assembly code. Ensure that the precision of floating point data type used matches the FP32 format.

Submission instructions:

1. Submit a zip file named YOUR_ROLLNUM.zip (e.g., CSYYBTECHXXXXX.zip)
2. 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.
3. The assignment should be done individually
4. Copying from others or any other source is strictly prohibited and subject to strict penalty
5. Assignments will be tested for similarity among each other and any violation will be reported appropriately

Alternative problem statement for interested students with hardware design background:
Implement an FPU in hardware (using HLS or in HDL), and show it to be working on a FPGA board (Zynq type or pure FPGA board). More details will be shared with interested students.