Computer Architecture - CS2323
Lab 4
Report : RISC-V Simulator Implementation

Devraj ES22BTECH11011

Introduction

This project focuses on the implementation of a RISC-V simulator on top of the RISC-V assembler implemented in the previous lab assignment, using C++. The simulator converts assembly instructions into their corresponding machine code in hexadecimal format and executes these instructions using the registers and memory.

Implementation

This implementation builds on top of the assembler.
It uses the following data structures to simulate an assembly code:

Registers

The registers are implemented as a C++ ordered map, which maps register names to 64-bit unsigned integer values.

Memory

The memory is implemented as an array of bytes (unsigned 8-bit integers) of size 0x50001.

Call Stack

The call stack of functions is implemented by a vector containing the function names and line numbers as pair structures and a stack of strings containing the current function's name.

Other than these, some more data structures were used for underlying procedures like storing breakpoints, the instructions in the input file, etc.

Loading the input file

When the input file is loaded, the data part of the file is appropriately stored in addresses starting from 0x10000 and the text part is first encoded into hexadecimal machine code and stored into the memory starting from 0x0.

When a new input file is loaded the registers are reinitialized to zero, the call stack is emptied, all the global variables (like pc, addresses of text and data sections, etc) are given their globally defined values but the memory remains unchanged.

Executing the instructions

The instruction to be executed next is kept track of using a program counter. The program goes to the line to be executed next, checks its encoding format and assigns the appropriate executing function to do the desired operation on the registers and the memory.
The program can execute step by step, or in one go.

Breakpoints in the code can be set, which halt the execution of the program before executing the specified line number. A breakpoint can only be passed by stepping over it.
The breakpoint line numbers are stored in a set data structure, and whenever the line number corresponding to the program counter is inside the breakpoint set, the execution is halted.
The breakpoints are deleted by removing them from the set.

Number System Conversions

Several helper functions are used for number system conversions:

- Conversion of decimal values to binary strings.
- Conversion of binary strings to hexadecimal.
- Conversion of numbers into an array of bytes, for storing into memory.

These functions handle the various valid input formats of the code.

Error Handling

Error handling is implemented during the loading of the input file to ensure that only valid registers and memory are being accessed and the immediate values are within the valid range of the encoding format. This was done as part of the previous lab problem.

Testing

The program was tested with assembly code given in the Lab problem statement, as well as other instructions, and the results were verified with the RIPES simulator.

Specific cases such as handling signed and unsigned values in the instructions, multiple function calls, recursive function calls were also checked.

Conclusion

This C++ implementation of a RISC-V simulator successfully translates RISC-V assembly instructions into machine code and stores them into memory and executes these instructions using efficient data structures.