The American University in Cairo

School of Sciences and Engineering

Computer Organization and Assembly Language

Dr. Nourhan Zayed

RARS Simulator Project Report

By: Hamza Algobba 900202728 , Ramy Badras 900194248 , Zein Noureddin 900212591

Program Description:

Our program is able to execute 40 RISC-V instructions that are stated on page 130 in the RISC V instruction manual: unprivileged ISA. It is a terminal based interface. The user is required to provide a file path to an input.txt file featuring the program instructions to be executed as well as a file path to data.txt file featuring memory addresses and their initial values. The format of the data.txt file is as follows: address:value, where address and value are both integers. Our program is encapsulated in many functions including but not limited to

1- 37 functions that hold the names of the aforementioned instructions with the exception of ecall, ebreak, and fence. Those functions take registers, immediates, offsets, or labels and executes their appropriate functions as specified in the instruction manual.

2- Read data: This function reads the aforementioned data.txt file and parses it to get memory addresses and values. It pushes addresses in a map called "mem" as a key, and converts the value from decimal to binary and then pushes it to the same map as a value.

3- Put into vector: This function reads the input.txt file, pushes it to a vector (excluding empty lines). The index of the vector resembles the address of each instruction. It also searches for any label definitions and adds them to a map that maps the label names to the address of the instruction that succeeds the label definition.

4- Validate registers: It validates that the register names in each instruction are valid and that the user does not overwrite the zero register. It is important to mention that the program only accepts the registers' aliases which implies that register names like x0 and x1 are unreadable by the program.

5- Validate instructions: Checks that the instruction entered belongs to the 40 supported instructions, it calls the appropriate parsing function based on whether the instruction takes two or the operands. It also prints the memory map and register map after each instruction executed.

6- parsing functions: It parses all 40 instructions and any label, stores the instruction, rd, rs1, rs2, immediates, offsets, or labels.

7- signed binary to decimal: converts 2s complement to its equivalent decimal value.

The bonus features implemented were: Printing the binary, decimal and hexadecimal values of the registers' contents.

Design Choices and Assumption:

We assumed that the user will not branch or jump to any labels that do not exist in the code. As mentioned before, we relied heavily on the use of maps and vectors to save data. The program does not support use of user comments. We created a flag that is set to true whenever any of the validation functions meet syntactically invalid instructions. The program should exit in case of the flag being true. Our choice of programming language was C++, because of its familiarity to team members as well as its support of bit manipulation.

Program defects:

The program is not very lenient in the treatment of user input. If the registers in the input file are not separated by an appropriate amount of spaces, the program might throw an exception. Additionally, the program does not always specify the errors present in the RISC-V input code.

User guide:

The user should include the input.txt and data.txt in the same directory as the source code, the program will prompt the user to initialize the program counter.

Link to video on how to use the simulator:

https://drive.google.com/file/d/1fvCM_VgDGbRW2uV0p_eCRoGv0GORHIgo/view?usp=share_link