CS 3432 – Computer Organization

Lab 2.2 – RISC-V 32-bit Interpreter Author: Robert Alvarez

**Instructions:**

Assignments must be submitted with the same person as in the previous lab, unless the other person drops the course, through Blackboard. Only assistance from your teammates, instructor, TA, or IA will be permitted. If working with some other team, only share ideas but not implementation because we are checking potential plagiarism.

**Introduction:**

In this lab, you will build a 32-bit RISC-V (RV32) interpreter in C that will receive, parse, and execute RISC-V instructions as strings. Your interpreter will not be a simulator as it does not mimic the hardware operation (e.g., utilizing clock cycles), but the resulting registers and memory should reflect what you would find after executing real RV32 instructions.

**Motivation:**

After completing this lab, you will:

1. Be able to effectively parse strings in C.
2. Understand how programs use stack memory.
3. Understand RISC-V instructions at a deeper level.
4. Become comfortable reading assembly instructions and programs.

**General documentation:**

Data components will be represented as follows:

* Program counter 🡪 A 32-bit integer is used to know which instruction to read.
* Registers 🡪 An array of size 32 where each entry is a 32-bit integer.
* This would be a global variable set to zeros before reading any assembly instruction.
* Memory 🡪 An unsigned char mem[MEM\_SIZE] variable would be used to simulate the stack.
* At the beginning of the program, the sp register would be set to &mem[MEM\_SIZE].

**RISC-V Instruction to implement:**

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Loads: 2. Load Byte 3. Load Word 4. Stores: 5. Store Byte 6. Store Word 7. Arithmetic: 8. Add 9. Add imm 10. Sub 11. Bitwise operators: 12. Exclusive OR 13. Exclusive OR imm 14. Shift left imm 15. Shift right imm 16. Pseudo-instructions: 17. Move 18. Load Immediate 19. Negate 20. Ones complement: 21. Jump offset: 22. Jump And Link 23. Jump 24. Jump offset + reg: 25. Jump And Link Reg 26. Jump Register | RISC-V Instruction  LB RD,RS1,IMM  LW RD,RS1,IMM  SB RD,RS1,IMM  SW RD,RS1,IMM  ADD RD,RS1,RS2  ADDI RD,RS1,IMM  SUB RD,RS1,RS2  XOR RD,RS1,RS2  XORI RD,RS1,IMM  SLLI RD,RS1,IMM  SRLI RD,RS1,IMM  MV RD,RS  LI RD,IMM  NED RD,RS  NOT RD,RS  JAL RD,IMM  J offset  JALR RD,RS1,IMM  JR RS | Example String  LB X7,1000(X5)  LW RA,28(SP)  SB A0,0(SP)  SW SP,RA,28  ADD A4,A0,A4  ADDI SP,SP,32  SUB A3,A3,A2  XOR A4,A4,A5  XORI T0,T1,3  SLLI A3,A3,2  SRLI A0,A0,1  MV A2,X0  LI A1,6  NED A2,A0  NOT T0,T1  JAL X0, 300  J 300  JALR RA,RA,240  JR RA | Order of Operations  RS <- M[RS1+IMM][0:7]  RS <- M[RS1+IMM][0:31]  M[RS1+IMM][0:15] <- RS2[0:7]  M[RS1+IMM][0:31] <- RS2[0:31]  RD <- RS1 + RS2  RD <- RS1 + IMM  RD <- RS1 - RS2  RD <- RS1 ^ RS1  RD <- RS1 ^ IMM  RD <- RS1 << imm[0:4]  RD <- RS1 >> imm[0:4]  ADDI RD,RS,IMM  ADDI RD,X0,IMM  SUB RD,X0,RS  XORI RD, RS, -1  RD <- PC+4 then PC += IMM  JAL x0, offset  RD <- PC+4 then PC = RS1 + IMM  JALR X0, RS, 0 |

You may assume the following:

* Input will be in all uppercase.
* Single space between the instruction and parameters, and a comma between parameters.
* Addresses would be given in base 10.
* Input will be a valid instruction.
* A sequence of instructions makes sense and is correct.

Input: File with a sequence of lines where each line has a single assembly instruction.

Example:

ADDI SP,SP,-64

SW RA,60(SP)

SW S0,56(SP)

LW RA,60(SP)

LW FP,56(SP)

ADDI SP,SP,64

JR RA

**Summary of already implemented files:**

* my\_string.c and my\_string.h:
* Some string.h standard functions implemented.
* Tokenizer.c and tokenizer.h:
* Same tokenizer as from Lab2.1.
* Given a string and some delimeters, return an array of pointers pointing to the beginning of each token in the original string.
* Process\_file.c and process\_file.h:
* This set of files convey the way to read the instruction using the PC from “memory”.
* Process\_file(const char \*file): Given the file name that would be use to read the assembly instruction, a seekable file is generated to be use for open\_file().
* Open\_file(void): This function is call after process\_file() to open the seekable file created so we can start reading lines from the file.
* Get\_line(char \*buffer, size\_t i): Given a buffer to store an assembly line from the file and the number i for the line of interest, the buffer is populated with a trim an null terminated string for line i in the file.
* NOTE: Each instruction takes 4 bytes, so if PC is set to 12 you want to read line 3.
* Close\_file(void): Free all necessary memory used to open the file given.

File/Libraries given (do not modify):

1. my\_string.h and my\_string.c.
2. tokenizer.h and tokenizer.c.
3. process\_files.h and process\_files.c
4. Makefile.

File/Libraries given (to implement):

1. riscv.c

riscv.c file:

* File processing, PC, and register set up are already implemented.
* Your job is to read through the file using the get\_line() function, until PC = -1, and interpret the line returned by it.
* To interpret the instruction, you must implement the following function: “int interpret(char \*instr)” that will parse “instr” to get an RV32 instruction. Once parsed the program must execute the instruction and return true if the instruction was properly executed and false otherwise.
* NOTE: DO NOT import the bool library, remember that 0 is false and everything is true.

**Notes about RISC-V instructions:**

* We would simulate calling a function by using jump instructions. Meaning, you don’t have to randomly set the SP to a new location whenever a new function call. So, we would use an old convention of making the SP to follow right after the previous SP.

**Deadline (Blackboard):** March 21th, 2024, by 11:59 pm.

1. Source code (Only .c and .h files, and 1 Makefile)

**Grading:** (Total of 70 points)

Each set group of instructions to implement counts for 10 points. E.g., LW and LH are 10 points total, SW and SH are 10 points total, ADD, ADDI, and SUB are 10 points total, MV, LI, NEG are 10 points total…

***NOTES:***

1. Laboratories 2.1 and 2.2 are a single assignment. Meaning, Lab 2.1 had 73 points and 2.2 70 points so the total possible is 143 points where your final grade is out of 100 points.
2. No late submissions.