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.

**General documentation:**

Data components will be represented as follows:

* Registers 🡪 An array of size 32 where each entry is a 32-bit integer.

TODO: Modify ArrayList\_t to be an array of bytes.

* Memory 🡪 ArrayList\_t where the index is the memory address, and the value is the word (32-bit) stored at that location.

TODO: Delete the next two points.

* Remember that more than one item could be stored in a 32-bit address (single entry of the array), for example, 2 short variables, 4 characters, etc.
* So, you may want to save in each pointer entry a pointer to an array of constant size 4 (worst case) made of structs with two fields, one for the type (use an enum variable) and one for a void variable which would be typecast based on the other variable field.

You are to design a function “int interpret(char \*instr)” that will parse “instr” to get a RV32 instruction. Once parsed the program must execute the instruction and return true if the instruction was properly executed and false otherwise.

|  |  |  |
| --- | --- | --- |
| 1. Loads: 2. Load Half: 3. Load Word: 4. Stores: 5. Store Half: 6. Store Word: 7. Arithmetic: 8. Add: 9. Add imm: 10. Shifts: 11. Shift Left Log Imm: 12. Shift Right Log Imm: 13. Logical: 14. AND: 15. OR: 16. XOR: | RISC-V Instruction  LH RD,RS1,IMM  LW RD,RS1,IMM  SH RD,RS1,IMM  SW RD,RS1,IMM  ADD RD,RS1,RS2  ADD1 RD,RS1,IMM  SLLI RD,RS1,IMM  SRLI RD,RS1,IMM  AND RD,RS1,RS2  OR RD,RS1,RS2  XOR RD,RS1,RS2 | Example String  LH X7,1000(X5)  LW RA,28(SP)  SH A0,0(SP)  SW SP,RA,28  ADD A4,A0,A4  ADDI SP,SP,32  SLLI X13,X13,2  SRLI A0,A0,1  AND X5,X6,X7  OR X6,X28,X0  XOR X7,X5,X6 |

You may assume the following:

* [Simplify reading] Input will be in all uppercase.
* [Simplify reading] Single space between the instruction and parameters, and a comma between parameters.
* [Standardize solutions] Addresses would be given in base 10.
* [Not looking for robustness] Input will be a valid instruction.
* [Remove address offsetting] Memory addresses start at the address 0x0 so there is no need to count for offsets.

Notes:

TODO: Change how ArrayList\_t is used in this lab.

* Be vigilant on the additions and subtractions to SP because that would modify your Memory (ArrayList\_t) size and it must be exactly that amount.
* Because of this new behavior, you may want to add a new function to increase size to hold at least a new fixed size of memory entries. Mainly, “int AL\_increase\_size\_to(ArrayList\_t \*AL, size\_t n)”
* Also, to decrease the size to an exact new value. Mainly, “int AL\_decrease\_size\_to(ArrayList\_t \* AL, size\_t n)”

TODO: Set everything in registers and memory to 0 before starting.

* Initially, everything that is in the registers and Memory is “trash” so you must not set it to a default value. Each instruction is a valid instruction, but a series of instructions may be nonsense and we expect the code to fail as the scenarios indicate.

**Notes in reading and writing to memory:**

* Create a new header and c file name “read\_file” with the function “char \*\*read\_next\_line(FILE \*fp)”.

File/Libraries given:

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

TODO:

File/Libraries expected (new):

Permitted standard libraries:

* stdlib.h - size\_t, malloc(), calloc(), realloc(), reallocf(), free()

No standard C library can be included if you want to use any other library or other variables/functions from stdlib.h, contact the AI or TA, and we can help you figure out how to do the same thing without that library or function.

Deadline (Blackboard): March 5th, 2024, by 11:59 pm.

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

Grading: (Total of 47 points)

???

[-5 points]

***NOTE:***

Once the deadline is passed, answers will be given the next day of class. This means that every submission after the deadline would not be considered and if that was your only submission, you get a zero.