****

**Computer Organization and Assembly Language**

**Dr. Mohamed Shalan**

**Project 1 Report**

**RISC-V Simulator**

**Name: Ahmed Mohamed Ahmed**

**ID: 900141806**

**Name: Hossam El Samanody**

**ID: 900143231**

**11/13/2016**

**Objectives:**

A C++ program that:  
1. Read RV32i machine code file as a sequence of 4-byte instructions.  
2. Decode each binary instruction as an RV32i assembly language instruction.  
3. Execute all instructions according to the program’s flow.

4. Print all instructions representing the whole assembly code.

5. Simulator can be invoked from the command line.

**Tools:**

C++ programming language & IDE. also: iostream, fstream, stdlib.h, and iomanip libraries.

RISC-V open source manual.

**Inputs:**

Object file of RV32i machine code program.

**Outputs:**

RV32i program execution’s simulation.

RV32i program’s assembly code.

A list of all RV32i registers and their final contents.

**Procedures:**

1. Reading the requirements and rubrics.
2. Reading the first two chapters in RISC-V Manual to get the basic needed info and the online Github page to get the opcodes.
3. Trying the given skeleton and understanding it.
4. Adding to the program’s if statements and switch case the info for all instructions.
5. Managing input program’s execution.
6. Managing input program’s printed code.
7. Debugging the code.

**RISC-V’s Basic Data:**

**Registers**: X0 = 0 (always), X1 = ra, X10 = a0, and X17 = v0.

**Instructions’ formats:**

R: {7-bit opcode | 5-bit rd | 3-bit funct3 | 5-bit rs1 | 5-bit rs2 | 7-bit funct7}

I: {7-bit opcode | 5-bit rd | 3-bit funct3 | 5-bit rs1 | 12-bit imm [11:0]}

S: {7-bit opcode | 5-bit imm [4:0] | 3-bit funct3 | 5-bit rs1 | 5-bit rs2 | 7-bit imm [11:5]}

U: {7-bit opcode | 5-bit rd | 20-bit imm [31:12]}

where: rs1&rs2 are sources, rd is destination, opcode chooses instruction format, funct3 chooses function, and funct7 chooses a function from two similar ones.

**Immediate formats:**

I-Immediate: — inst[31] — inst[30:25] inst[24:21] inst[20]

S-Immediate: — inst[31] — inst[30:25] inst[11:8] inst[7]

B-Immediate: — inst[31] — inst[7] inst[30:25] inst[11:8] 0

U-Immediate: inst[31] inst[30:20] inst[19:12] — 0 —

J-Immediate: — inst[31] — inst[19:12] inst[20] inst[24:21] inst[11:8] 0

**Instructions List:**

R instructions: Add, Sub, Sll, Slt, Sltu, Xor, Srl, Sra, Or, and And.

I instructions: Addi, Slli, Slti, Sltiu, Xori, Srli, Srai, Ori, and Andi.

S instructions: Sb, Sh, and Sw.

I+L instructions: Lb, Lh, Lw, Lbu, and Lhu.

Sb instructions: Beq, Bne, Blt, Bge, Bltu, and Bgeu.

U instructions: Lui, and Auipc.

J instructions: Jal, and Jalr.

SCALL: print int, print string, read int, and terminate execution.

**Code’s Description:**

Language: C++, using Microsoft Visual Studio.

The code is modular by using functions and fully commented.

Includes five functions: emitError, printPrefix, instDecExec, instDecPrint, and main.

Global Data: integer 1D array of 32 registers, unsigned int program counter, and char 1D array as an 8KB memory.

**Main:**

Takes command line arguments. If arguments are not enough, calls emitError() to print arguments’ error message.

Opens input file as a binary input file, and sets iterator to the end of the file. If arguments are not enough, calls emitError() to print error opening message.

Calls instDecExec() and instDecPrint() to execute and print program.

Prints a list of all registersand their contents after execution.

**instDecExec:**

Boolean function that takes instruction word as input argument.

Sets values for: rd, rs1, rs2, funct3, funct7, opcode and I\_imm, S\_imm, B\_imm, U\_imm, J\_imm.

Uses a sequence of if-else statements, each containing of a switch case, to first define current instruction’s format depending on the opcode, the exact instruction’s function using funct3, and for some cases, chooses one of two instructions using funct7.

Last, it executes the specific chosen instruction, and if it’s an SCALL with v0 = 10, it returns true to terminate. Otherwise, it returns false.

**instDecPrint:**

Void function that takes instruction word as input argument.

It uses the same sequence of if-else statements containing switch cases as instDecExec to choose the given instruction.

Last, it prints the instruction’s prefix using printPrefix() and then prints the specific chosen instruction, and if it cannot define the instruction, it outputs “Unknown instruction”.

**printPrefix:**

Prints instructions’ prefix as a hexa-decimal program counter and hexa-decimal instruction representation.

**Challenges faced:**

1. SCALL: finding the opcode, and which registers are input and output.
2. Jumps and branches: whether it uses current of following instruction address for pc and also managing their immediates.
3. Command line arguments.
4. Mapping MIPS registers to RISC-V registers to validate simulation using MARS.
5. Signed and unsigned functions and signed/zero extensions.

**Limitation:**

We thought that the B immediates in printArray sample needs a special case, but by decoding the branch instructions, we found the B\_imm are right and fixed the code.