Lebanese American University
School of Arts and Sciences
Department of Computer Science and Mathematics

CSC 320 – Computer Organization

A Simple RISC Processor

Design Phase: Due Nov. 14 at 09:30 a.m.
Implementation Phase: Due Dec. 19 at 09:30 a.m.

In this project, you are expected to design and simulate a simple RISC processor that supports a set of only four arithmetic instructions: add, subtract, multiply, and divide.

Assume there are four 16-bit registers: R1, R2… R8, in addition to the program counter (PC) register. The arithmetic operations involve three registers, two of which are source registers and the third is a destination register. Thus, before performing the operation, the source registers to be input to the ALU must be selected. The output of the operation will be saved in the destination register.

As for the memory configuration, there are two separate memories: 8-bit program memory and a 16-bit data memory. The program memory is where the instructions are saved; whereas the data memory holds the values of the data used in the program and accessed using load and store instructions. Since the latter instructions are not required, you do not need to implement data memory.

The control flow of the processor works in four stages:

1- Fetch: at this stage, instructions are read from the program memory according to the value of the PC register. PC is then incremented.

2- Decode: At this stage, the instruction's op-code is decoded. The processor will know what type of instruction will be executed, and the operands of the ALU (source registers for the arithmetic instructions) are selected.

3- Execute: The instruction decoded in the previous stage is executed. Arithmetic operations on the selected operands are executed by the ALU.

4- Write back: This is the last stage where data is written to the instruction's destination. For the arithmetic operations, the destination must always be a register.

Design

to guide you through your design, answer the following questions and justify your answer

1. Design the instruction structure by specifying the instruction fields and their sizes (i.e. number of bits required for each field).
2. For each given instruction, define its syntax and corresponding opcode.
3. Specify unique identifiers for each of the 4 registers.
4. Specify the increment performed to the program counter.
5. Sketch the datapath corresponding to each type of instruction identified in part 1. (this item is due with the implementation phase)

Implementation

In this phase, you need to simulate the designed processor. You are required to implement the given arithmetic instructions only.

To implement the desired processor, you must first implement its components depending on their behavior:

1- Register: Saves a binary value presented at its input when its load signal is activated.

2- Memory: The two types of memory are nothing but arrays of binary values. You only need to implement program memory. Do not implement data memory. The program memory is 256 bytes in size.

3- Program Counter PC: for simplicity, the program counter is a 16-bit register with an internal increment unit.

4- ALU: takes two 16-bit binary operands and performs one of the four arithmetic operations, depending on the operation select signal.

5- Datapath: Contains instances of:

   a. All the registers. The general-purpose registers have their inputs from the ALU output.

   b. An ALU.

6- A Control Unit: It is responsible for the control flow of the program, by guiding the data-path's control signals throughout the fetch, decode, execute, and write back stages. It basically chooses the registers to be read from, the operation to be executed in the ALU, and the registers to be saved to. It chooses all of these depending on the op-code it reads at the location pointed to by the value in the PC, and stops execution once a halt statement is encountered.

7- Microprocessor: It is the main component, which has an instance of the data-path, and an instance of the control unit.

Initialization

Before the microprocessor starts operation. You need to initialize the general purpose registers with random values to be displayed upon starting the microprocessor. In addition, you need to initialize the program memory with the required binary values of the instructions. You can convert the assembly code into op-code manually and initialize the program memory, or you can write a conversion function that reads the assembly code and initializes the memory automatically.

Output

After each instruction, your program should display the contents of all the registers.

Additional Feature

Write an assembler function that converts assembly code into machine code. The assembler can read the assembly language program from a given file.

Guidelines

- Form groups of three.
- Work on your processor design that conforms to the given system characteristics. Answering questions in the Design part guides you through the design details.
- Discuss the design details with the course instructor or the graduate assistant before you proceed with the implementation phase.
- Use JAVA programming language to simulate your processor.

Deliverables

1. Submit a hard copy of the progress report by the due date of Phase I. The progress report should answer all questions in the Design part (except for question 5) and explain the design details.
2. Prepare the final report that includes the final design and implementation details.
3. Prepare a well-documented source code of the processor.
4. Zip all files and upload to BB by the due date of Phase II.
5. Perform a demo of the final processor.