COMP 303 Term Project Fall 2017
Single Cycle Processor Design

Deadline for Part I: Sunday, midnight, Dec 10, 2017
Deadline for Part II and Part III: Sunday, midnight, Dec 24, 2017

Instructor: Didem Unat
Corresponding TAs: Najeeb Ahmad (ENG110), Doga Dikbayir (ENG 230)

Notes:

- The project has two parts and each part has a different deadline. You have to submit Part I by Dec 10th and submit the whole project including Part II and Part III by Dec 24th.
- You may do the project as a team of 2.
- No late submissions will be accepted.
- Submit a SOFT copy of your project to the blackboard
- The assignment is 10% of your total grade
- You will be asked to demonstrate your project design and implementation to the TAs during the demo. All project partners have to be present at the demo.
- TAs will hold two PS hours on Dec 4 and Dec 18 to help you with the project. Make sure to get familiar with the processor simulator before coming to the PS hour.
- Read the whole project description before starting, **particularly grading section.**

Academic Integrity: You may consult any of the MIPS architecture documentation available to you in order to learn about the instruction set, what each instruction does, etc. But we expect your design to be entirely your own. If you are unsure if it is okay to borrow from some other source, just ask the TAs, and give credit in your final write-up. If you are unsure about asking the TAs, then it is probably not okay. Plagiarism in any form will not be tolerated.

Project Description: In this project, you will design, implement and test a 16-bit single cycle processor using *Logisim*. Logisim is an educational tool for designing and simulating digital circuit that provides basic components to design a simple processor. You will be first designing ALU and register file, and then use these to design a fully functioning processor.

Part 0: Preamble

You first need to setup the environment in order to run the simulator

- Download Logisim .tar.gz file (Unix-based OS) or .exe file (Windows) from http://www.cburch.com/logisim/. If you choose to install .jar file, make sure you have latest distribution of Java Runtime Environment (JRE).

Part 1: ALU and Register File Design

The first step towards designing a processor is to design and implement ALU and the Register file. You will be then using the ALU and Register file for the single processor design in the next part. Both ALU and Register file support 16-bit.

1. The ALU

Your ALU will support the instructions listed in the Table 1. Your instructions will have 32-bit instructions and use the same instruction format as MIPS. For instance, R-type instructions will have 6 bits opcode, 5 bits each for rs, rd and rt registers, 5 bits for shamt and 6 bits for funct. You can use built-in arithmetic library to build your ALU **EXCEPT for adder which you must design by yourself.**

Table 1. ALU Instructions

Instruction	Opcode	Type	Operation	Description
add rd, rs, rt	100000	R	rd = rs + rt	rd = destination, rs, rt=source
sub rd, rs, rt	100010	R	rd = rs - rt	rd = destination, rs, rt=source
mult rs, rt	011000	R	hi;lo = rs*rt	hi, lo: two 16-bit registers in multiplier unit to store 32-bit multiplication result
and rd, rs, rt	100100	R	rd = rs & rt	rd = destination, rs, rt=source
or rd, rs, rt	100101	R	rd = rs | rt	rd = destination, rs, rt=source
addi rd, rs, I	001000	I	rd = rs + I	rd = destination, rs, rt=source, I = 16-bit sign extended immediate value
sll rd, rs, shamt	000000	R	rd = rs << shamt	rd = destination, rs, rt=source, shamt = shift amount
slt rd, rs, rt	101010	R	rd = (rs < rt)	rd = 1 if rs < rt, otherwise rd = 0
mfhi rd	010000	R	rd = hi	Load hi from multiplier unit into register rd
mflo rd	010010	R	rd = lo	Load lo from multiplier unit into register rd

2. Register File

You will design a register file with eight 16-bit registers. The register file should support same I/O signals as the MIPS register file with two read register address ports, two read register output ports, one write register address port, one write register data port, a register write control signal and a clock input.

Part 2: Single Cycle Processor Design

In this part, you will use the ALU and the register file designed in Part 1 and components provided by Logisim. Apart from the instructions for the ALU, your processor should also support the load/store, branch and jump instructions listed in Table 2.

Table 2. Load/Store, Branch and Jump Instructions

Instruction	Opcode	Type	Operation	Description
lw rd, i(rs)	100011	I	rd = rs[i]	rd = destination, rs=base address, i=offset One word refers to 2 bytes (16-bits)
sw rs, i(rd)	101011	I	rd[i] = rs	rd = destination, rs, rt=source One word refers to 2 bytes (16-bits)
beq rs, rt, label	000100	I	if(rs == rt) jump to label	rs, rt=registers to compare Label= label to jump to
blez rs, label	000110	I	if(rs <= 0) jump to label	rs=register to compare Label=label to jump to
j label	000010	J	Jump to label	Label=label to jump to
myIns	111111	-	Define your own operation	Use your imagination to design this instruction

Memory: For the memory part of your design, please note that the instruction memory can be built using either a ROM or a RAM module in Logisim. The data memory, however, has to be a RAM module.

Part 3: Testing your processor

To test your processor, you will need to

- Write an assembly code in a file named *test_program_asm.txt* (pseudocode is given in Listing 1)

- Provide corresponding machine instructions for your assembly program in a separate file named *test_program_machine_code.txt.* (You may write a simple assembler in C to convert your instructions to machine code OR you can convert the assembly code manually)

We are providing you with example test_program_asm.txt and test_program_code.txt files to help you write you own. Please note that you cannot use these files directly to test your implementation as the opcodes for your implementation may differ from the given ones. We are also providing you test_program_data.txt file that you will use to test your implementation. Below is the brief description of the example files.

test_program_asm.txt contains assembly instructions and corresponding machine codes for a simple bank transactions simulator program. For instance, the first assembly instruction in the program is addi $5, $0, 1 followed by the line 0: 20050001, where 0: shows address of the instruction in memory and 20050001 shows instruction machine code. The purpose of this file show correspondence between assembly instructions and their machine code while also showing their address in memory. To create this file, you will follow these steps:

- Write assembly instructions for your program as per the pseudocode given in Listing 1
- Generate hexadecimal machine code for the assembly instructions and write below each assembly instruction along with the memory address where the instruction will be placed (as per the sample test_program_asm.txt file provided). As you don't have an assembler for your CPU (assembler converts assembly instructions to machine code), you need to manually convert assembly instructions to machine code. The reason why haven't provided you with this file is because your machine instructions may differ depending upon the register addresses you choose in your implementation.

test_program_code.txt contains only the machine code instructions of your program. You can create this file by simply placing machine code (without address part) part of your test_program_asm.txt file into test_program_code.txt as per the given format. This file will be used to load instructions into instruction memory of your processor for testing. As you will see, Logisim allows instructions and data to be loaded into memory by right clicking on memory module and assigning a file to the memory using Load Image option. We will use this option to load test_program_code.txt file into instruction memory of your CPU.

test_program_data.txt file contains data for your program. This file will be loaded into data memory of the processor using the Load Image option described above to test your implementation. Please note that the first data element will loaded at data memory address 0, the second at address 4 and so on.

We will use the data and code files to test your processor implementation in the demo.

- We ask you to implement the following program to test your implementation.

● In addition you should device a simple program to test your custom instruction you developed in Part II. Make sure to provide its code and data files as well in your submission.

Listing 1. Vector-Vector Multiplication Pseudocode

```
Define vector a[0..3] as [5, 7, -2, 40]
Define vector b[0..3] as [65, -23, 17, 1024]
Initialize sum = 0

While i <- 0 to 3
      sum = sum + a[i] * b[i]
      i = i + 1
End While
```

Listing 2. Vector-Vector Multiplication C Implementation

```
void main()
{
    int a[4]={5, 7, -2, 40}; /* Initialize vector a */
    int b[4]={65, -23, 17, 1024}; /* Initialize vector b */
    int i = 0, sum = 0;

    for(i = 0; i < 4; i++)
    {
        sum = sum + a[i] * b[i];
    }
}
```

Submission:

You are supposed to submit the following files for Part 1.

1. All components and any intermediate files made while implementing the ALU and register file or its components (these files should be in .circ format)
2. The final ALU should be in one single file (.circ format)

You are supposed to submit the following files for Part 2.

1. The final processor design file (.circ format)
2. Report containing a circuit schematic
3. Table summarizing control signals
4. Description of your design and custom instruction.

You are supposed to submit the following files for Part 3.

1. Test programs: their assembly codes and instruction files
2. Trace of the test programs, demonstrating that correct result was obtained.

Grading:

Here is how grading will work.

- Overall, this project contributes to 10% of your final COMP 303 grade.
- 70% of your grade will come from the implementation of Part I, II, III and report. Part I (30%), Part II (30%), Part III and report (10%).
    - Please document what parts of your code works.
- The remaining 30% of your grade will come from Project-Related Questions in the final exam. Note that these are additional questions to final exam questions and doesn't affect the final exam score but only your project score.
- Those who receive unsatisfactory score from the project-related questions will be invited to perform a demo, which will affect their implementation score even if their processor is fully functioning. We can also request a demo for those whose project is not working properly.
- Demos will take place on January 8th and 9th.
- If you miss Part I deadline, you will get zero points for Part I. But you can still submit the full project by the second deadline and get evaluated out of 70%.

References:

- https://www.cise.ufl.edu/~mssz/CompOrg/CDA-proc.html

GOOD LUCK!