CSCE 230 Project Part III

Memory and instruction implementation — Due November 20th

In general, you are allowed to use block diagrams, VHDL, or a mix of both to complete any of these assignments. Each will come with its own advantages/disadvantages. Block diagrams are easier to realize but there may be more simple mistakes, i.e. wire connections. VHDL may be harder to implement but there will be less simple mistakes. If you copy any code from other sources, Internet/books, you must cite your source or receive a zero for the project. Each team member must work together and should only turn in one assignment/report/check off.

Overview

Your objective for this third part of the project is to add to the basic processor, completed in Part II, the following instructions: jr, cmp, lw, sw, addi, b, and bal. For further explanation of the instructions, refer to the **project overview document**.

Note that, technically, I/O implementation is covered by load and store operations (because of memory-mapped I/O). However, to keep this part manageable the I/O implementation is done in Part IV. In other words, lw and sw are required for this part, but only the data memory needs to be connected; you can add the I/O memory later. More help will be given in Part IV for I/O implementation.

The design flow will consist of five steps:

1. Block-level datapath design (or VHDL design)
2. Design of individual Components
3. Datapath integration
4. Control Unit Design
5. System integration and validation

Steps 1-3

Now that you already have a datapath, the first three steps will be incremental. For each new instruction to be added, ask yourself the question: Does the instruction require any modifications to the existing datapath? If the answer is no, the task is simple and is summarized in the few steps below.

1. Change the control unit (follow the scheme for the already-implemented instructions to add code for the new instructions)
2. Compile
3. Create a program to test your design (see Testing)

If the answer is yes and the datapath needs to be modified, the above steps will remain the same after making changes to the datapath. For figuring out the necessary changes to the datapath, it might be helpful to print out a few copies of the datapath from Part II so that you can quickly make changes or throw away mistakes. For each new instruction, try to answer the following questions:

1. Which register(s) need updating? Which do not?
2. Where will the data come from: one source or is a MUX needed?
3. Does the instruction need the ALU? Which function of the ALU? Does it need a separate add unit?
4. **Do I need to add more muxes/registers/wires?**

Finally, add the needed MUXs, buses, and mega-wizard functions to the datapath and proceed with the three steps described earlier.

Step 4

Add to the control unit following the fashion of the already-implemented instructions. When adding an instruction, think about what each control signals need to change at each stage of execution. It should only be a few at most. Remember, **stages 1 and 2 should not need to change** since they are the same for every instruction.

For this part you should break down each instruction and determine what flags need to be set at each stage. Start with a load instruction. Then determine what flags need to be set during the third stage for a load. Do this for the last two stages. Repeat this process for the rest of the instructions.
You can also print out multiple copies of the datapath and follow what each instruction would do to determine the correct flags to set. **NOTE: make sure to turn off flags in the next stage that should not carry over. This goes for enable flags and writes.**

Step 5

A **MAJOR** component of each part in the project is the last step. **Make sure you reserve sufficient time for this step** as your validation results will help us understand the success of your implementation.

The integration of the datapath and control circuitry requires that you understand the interactions between the components well and use the correct and consistent signal names between the components. The instruction encoding and the names of the control signals are **NOT** the same for our project as they are for the book. Refer to the project overview for correct instruction encoding and refer to your control unit for the correct names of the signals.

After you have integrated the datapath and the control circuitry, you will need to test your implementation thoroughly for correctness and timing performance.

Assignment

Modify your datapath and control unit files as described above. Then integrate the memory interface and instruction address generator into your existing processor.

Tasks you must perform

- Download the provided component(s) from Piazza
- Look over and develop an understanding any provided component(s)
- Add the memory interface and instruction address generator to your datapath
- Add to the control unit to implement additional instructions
- Hand-assemble an instruction of each type and test your design
- Create a test script(s) (.do file(s)) and run a simulation(s) of the processor to make sure you created everything. The .do file should only control the clock and the reset. Everything else should just be outputs.

One modification that you must change in your original processor design is that the clock going into your register file should be inverted.

Testing and simulation

The basic processor design should be able to carry out the instructions (jr, cmp, lw, sw, addi, b and bal) correctly. The processor should also be fetching the program instructions from memory.

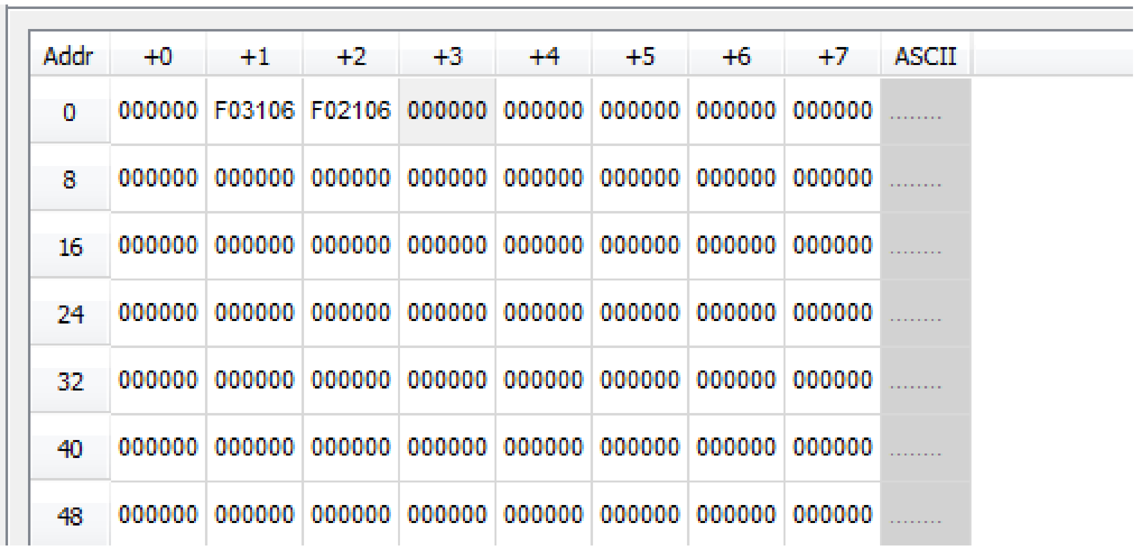
Demonstrate the correctness of your design by hand-assembling an instruction of each type. To this end, initialize the memory initialization file (.mif) with the instruction, cycle through the five stages of execution, and verify that everything executes correctly. In the components provided, MemoryInitialization.mif and MemoryInterface are used together. You can write the instructions in Hex format into*.mif. Figure 1 shows an example, the address is increasing from left to right.

Make sure you try each instruction with different choices of the registers for more comprehensive testing. By now, you should already be familiar with how to use ModelSim for functional simulation.

IMPORTANT TIPS:

- Implement and test instructions one at a time
- Be sure that load word works before testing store word
- In the .mif file, dont have any instructions in the first address location (leave it as 0x000000)

In your timing simulation, strive to test your processor for correct operation at the highest possible clock speed. As you processor grows more complex, it is very likely that you will have to slow down this clock speed of the system to produce the correct results by giving signals enough time to propagate. The test cases for the timing simulation should



Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	000000	F03106	F02106	000000	000000	000000	000000	000000	....
8	000000	000000	000000	000000	000000	000000	000000	000000	....
16	000000	000000	000000	000000	000000	000000	000000	000000	....
24	000000	000000	000000	000000	000000	000000	000000	000000	....
32	000000	000000	000000	000000	000000	000000	000000	000000	....
40	000000	000000	000000	000000	000000	000000	000000	000000	....
48	000000	000000	000000	000000	000000	000000	000000	000000	....

Figure 1: An example mif file

be the same as in the functional simulation.

Report

You must create a two page (minimum, not including images) technical report detailing the processor you have created so far. Include its current abilities (which instructions is can perform), a speed overview (how fast it can run), and the components inside of it. You should also briefly talk about your groups experiences with connecting everything. This should follow much the same as the report you did for Part 2. However, you do **NOT** have to include components of your processor that have not changed. This means that you do not have to discuss the original 5 instructions, nor the components integrated in that part (RegFile, ALU, RA-RZ, MUXES). **NOTE: However, If you changed the way one of these devices works you must talk about that modification (Control Unit, Immediate block, etc..)** Use 1" margins, size 11, Times New Roman, and images should not be large than 1/4 of the page. This report will be mostly a check of your teams understanding of what they have done so far. **Make sure to include a title page.**

Grading

The grading breaks down from the table below:

Points	Part
15	Design File (.qar)
10	Simulation
20	Check off and demonstration
30	Technical report

The design file, simulation, and report are due by 11:59PM of April 8th-9th depending on your lab. The Check off is due by the time Lab ends. If your project is not checked off by the end of the week it is due, April 10th by the time the last TA leaves, you will not receive any points for the check off. If you are unable to finish this part in time, **still hand in what you have done so far for partial points**. Late submissions will get zero points so make sure to meet the deadlines. Each part of this project is incremental so being late on a deadline will make you late on the next one.

Check off

The check off for this section will include the demonstration of the following instructions: ldw, stw, addi, bal, jr. An easy way to get the check off is to set up the following program in your .mif file and demonstrate it to a lab TA.:

```
addi  r2, r0, 5
addi  r3, r0, 25
stw   r2, 0(r3)      //Storing 5 to Memory address 25
ldw   r4, 0(r3)      //Loading the 5 back
bal   5              //Branch unconditionally forward 5
add   r2, r2, r4     //To be executed upon return
nop
nop                  //Nop's to avoid
nop
nop
```

```
jr      r15              //Return to add after the bal
```

It will make the check off much smoother and easier if you include the following outputs from your processor: RA, RB, RM, RZ, RY, PC, and IR.

Submit

You must submit the files electronically to webhandin by 11:59PM on April 8th-9th. The naming convention should be as follows:

- Project"team-number"_Part3.qar for the project archive file. The simulation files should be located inside of the archive.
- Project"team-number"_Part3.pdf for the project report. This report should follow the best practices of writing technical reports.

Hints

A few hints to help you along.

1. You should add a muxC, and a muxMA. muxMA should choose between the pc and RZ. Its selector should be ma_select. muxC should choose what Rd should be. Rd is in different places for D type and R type and the **link register is a constant 15.**
2. Make sure to **invert the clock** going into your register file in order to avoid any delay issues
3. Make sure to leave the **first instruction** in the .mif file empty. Placing an instruction there tends to cause problems.
4. You should only have two inputs at this stage of you processor, the Clock and the reset. Your IR should be fed instructions via Memory data out of your Memory device.
5. All images for this part are given in the Project Overview file.
6. Start you clock speed off at a 2000ps period. Then work you way to a faster period until your processor breaks.

Assembler

Now would be the time to start building an assembler for your processor if your group chooses to. This assembler would need to take assembly instructions, as specified by the project overview, and convert them to machine code, as specified by the project overview as well.

Now that you will have a memory device in your processor, you can store programs to be run by your processor on startup/simulation. This memory device is set with a .mif (Memory Initializtion File). Because of this, you can set up your assembler to output the machine code directly to a .mif file. This means you would not have to spend the time hand assembling and then placing these instructions in your .mif file.

This assembler can also count towards extra credit as specified in the project overview. The value of the extra credit depends on its completeness (data sections, labels, etc..) and the actual design of the assembler (two-pass or other methods) with better designs and more features receiving more points.