CprE 381 – Computer Organization and Assembly Level Programming, Fall 2018

Lab – 7, Part A

There are two parts to this lab (see Lab 7 Part B on BBL). In this part of the lab, you will simulate the main MIPS processor components to get an understanding of how they work. You will then build a simplified MIPS processor using a register file, ALU, and memory.

0) Before working on the lab:

(a) Create a new folder for lab 7a <user home folder>/cpre381/lab7a. Use this directory to save your work.

(b) Unzip the provided lab7a.zip in the lab7a folder. The zip contains three components:

a. A register file with two read ports and one write port. It contains 32, 32-bit registers.

b. An ALU supporting twelve arithmetic and logical operations.

c. A word-Addressable memory with one read port and one write port.

1) A **register file** is an addressable array of registers. In the MIPS-architecture there are 32, 32-bit registers in the register file. This means an address must contain 5 bits to address all 32 registers. There are different naming conventions, but for this lab, we will name the registers R0, R1… R30, R31 for simplicity. The MIPS register file has two read ports and one write port. To write to a register you must specify the 5-bit address and the data to be written. *Note: In MIPS, register zero (R0) always holds the constant value of zero. It is not writeable.*

A register file is provided in lab7a.zip. It requires VHDL dependencies, which are also included in the zip. Compile all files in the "Register File" folder from the lab7a.zip. The high-level entity name for the register file is "register_file". Using ModelSim simulate the register file and provide screenshots in your report demonstrating your knowledge of the component and how it behaves. Your screenshot should write values to at least two registers and afterwards read-back the same values showing the register file working correctly. *Note: The regs signal is an array of the registers' values in the register file. This is useful for debugging or forcing initial values.*

2) An **Arithmetic Logic Unit (ALU)** is the main execution unit within a processor. It takes parameters and calculates mathematical (addition, subtraction, etc.) or logical (AND, OR, etc.) operations on them. The type of operation is determined by the control value given to the ALU. An ALU is given in lab7a.zip under the "ALU" folder.

(a) Read through ALU.vhd and fill in the control table of operations. The first value is given. This ALU only supports 12 operations. With a 4-bit control value, it could in total support 16 operations.

Control Value	ALU Operation
0000	Addition
0001	
0010	
0011	
0100	
0101	
0110	
0111	
1000	
1001	
1010	
1011	

(b) Simulate the ALU using ModelSim and provide screenshots of (at least) three different operations. Label your screenshots with the input values and the operations performed. **You will lose points if you do not label your screenshots.**

3) The **Memory** component provided for this lab has one read port and one write port. This memory is word-addressable, which means each address contains an entire word (4 bytes) instead of a single byte like typical MIPS memory does. The write to this memory happens on the falling edge of a clock cycle. The details of the memory (**make sure you understand all of the below!**):

i) The *address* port is the 10-bit address value used for both reading and writing. By default, the memory is 1024 words in size. This means we need a 10-bit address to access all cells.

ii) The *data* port is the 32-bit input value to be written to the memory at *address* if the write is enabled.

iii) The *wren* port is the write enable. Set this to '1' to write *data* to *address*.

iv) The *byteena* port (byte enable) is used to control which bytes of *data* are written to *address*. For this class it's OK to set this value to all ones (i.e. enable all bytes of data to be written).

v) The "q" port is the output being read from the memory cell at "address".

vi) "mem" is an array of all values stored in the memory. It is an internal signal. You can use it to read what is stored in memory for debugging and screenshots. This signal is your "friend."

vii) "mif_filename" is a string for the name of the memory initialization file (.mif). By default, the string is "dmem.mif" but it can be changed. The memory initialization file allows you to specify what values are stored in memory at time zero.

(a) Read through the provided dmem.mif (memory initialization file). It contains a description of how to use it to initialize memory. Later in the semester, we can use it to store a program to execute on a processor. Modify the dmem.mif file to contain the values -1, 2, -3, 4, -5, 6, -7, 8, -9, 10 starting from address 0. Include the modified dmem.mif file in your submission.

(b) Simulate mem.vhd and verify that the memory is initialized to the new values you've written in the dmem.mif file. Provide a screenshot in your report for proof.

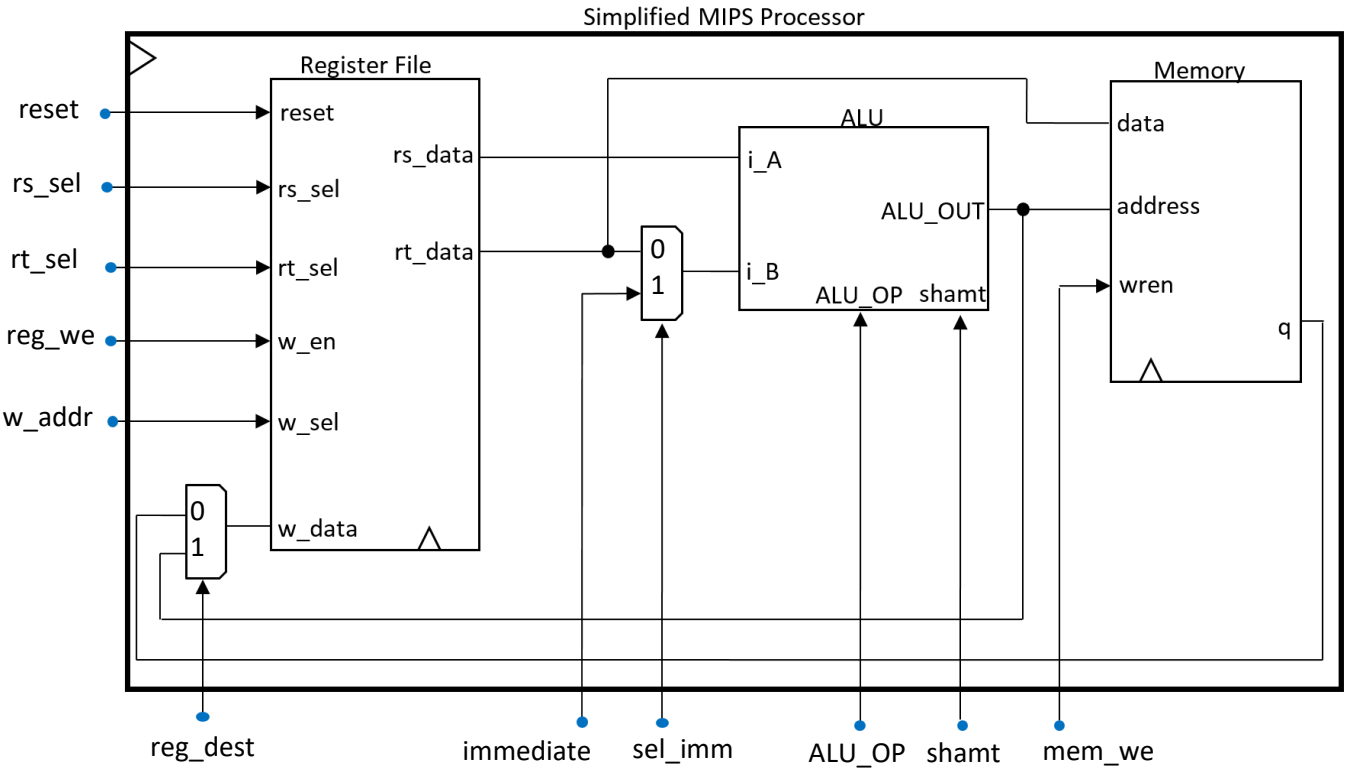
4) Now that we've gotten a basic understanding of the main components of the MIPS processor, we will add them together to create a simplified MIPS processor. If you want, you can use Quartus to build a block diagram version of this simplified processor, and then, use ModelSim to simulate it.

(a) (for Quartus users) Using Quartus create a simplified MIPS processor. You will need to create a project to do this. Name your project *simplifiedMipsProcessor*. Include the VHDL files for the register file, ALU, and memory components. You will need to create block symbol files for the three components. Refer to lab 4 section 4b if you've forgotten how to do this. *Note: For the register file just create a symbol file for register_file.vhd. Usually you only need to create symbol files for the top-level entities in a hierarchy.*

(b) A diagram of the simplified MIPS processor is given below. You will need to include two 32-bit 2:1 muxes. One is for selecting an immediate to the second input of the ALU (this allows for I-type instructions like ADDI). The other is for selecting to write the output of the ALU or the memory to the register file (i.e. for selecting between arithmetic or memory instructions). The ALU_OUT output port is 32 bits and is connected to the 10-bit address input of the memory. Take the lower 10-bits of the ALU_OUT for the address. *Note: The simplified processor does not have any outputs, only inputs. For testing you will set these inputs and internally inspect the register file, ALU, and memory contents to verify functionality.*

(c) You will be testing simplified versions of a few MIPS instructions. In order to do this, you need to consider control values for the register file, ALU, memory, and 2:1 muxes. An example for the R-type instruction ADD would be:

ADD $1, $2, $3 # R1 = R2 + R3
rs_sel = "00010" (read contents of R2 for rs_data)
rt_sel = "00011" (read contents of R3 for rt_data)
reg_we = '1' (write enable for register file; we are writing to R1)
w_addr = "00001" (selects to write register file input data to register 1)
reg_dest = '1' (selects to use ALU output as write data to register file; vs. mem output)
sel_imm = '0' (selects ALU second input as rt_data)
ALU_OP = "0000" (addition operation in ALU)
shamt = "00000" (the shift amount is a don't care because this is an add operation)
mem_we = '0' (do not write to memory, i.e. only set to '1' for store word instruction)
immediate = don't care (there is no immediate used in this instruction)



(d) (for Quartus users) Export the simplified MIPS processor to VHDL and simulate it using ModelSim.

(e) Include test cases for the following MIPS instructions: ADDI, SUB, SLL, LW and SW. Include a screenshot and description of what's happening for each test case. **Make sure you understand the format and operation of each of the test instructions.** They will give important clues for how to set the control values. Before you start testing instructions, run a reset (i.e. set reset = '1') for one clock cycle to zero out the register file. You may force register file contents before running instructions to get more interesting results.

SUBMISSION:

- Create a zip file (for only part a files), including the completed code and screenshots from the lab.
- The lab report that answers all questions from this document. You can include your screenshots in the report if you'd like.
- The file names in your submission should be self-explanatory.
- Submit the report as a pdf and the zip on Canvas along with the requirements for Lab 7 part B, under "Lab 7" assignment, before your lab time next week.