### Overall design flow:[[1]](#footnote-1)

**Big Picture:** In this lab you will build a simplified MIPS single-cycle processor using VHDL. You will combine the ALU you previously created with the code for the rest of the processor. Then you will load a test program and confirm that the system works. Next, you will implement two new instructions, and then write a new test program that confirms the new instructions work as well. By the end of this lab, you should thoroughly understand the internal operation of the MIPS single-cycle processor.

# Background

Before starting this lab, you should be very familiar with the single-cycle implementation of the MIPS processor described in Section 7.3 of your text. The single-cycle processor schematic from the text is repeated at the end of this lab assignment for your convenience. This version of the MIPS single-cycle processor can execute the following instructions: add, sub, and, or, slt, lw, sw, beq, addi, and j.

Our model of the single-cycle MIPS processor divides the machine into two major units: the control and the datapath. Each unit is constructed from various functional blocks. For example, as shown in Figure 1, the datapath contains the 32-bit ALU that you designed in previously, the register file, the sign extension logic, and five multiplexers to choose appropriate operands.



Figure 1 – MIPS single-cycle processor schematic.

# MIPS Single-Cycle Processor

The VHDL single-cycle MIPS module is given in Section 7.6 of the text. Use the slightly-modified electronic version of all these files combined on the course website (mips\_combo.vhd). Copy it to your own Lab4 folder.

Study the file until you are familiar with its contents. Look at the MIPS top-level entity, which instantiates two sub-modules, controller and datapath. Then take a look at the controller module and its sub-modules, maindec and aludec. The maindec module produces all control signals except those for the ALU. The aludec module produces the control signal, alucontrol[2:0], for the ALU. Make sure you thoroughly understand the controller module. Correlate signal names in the VHDL code with the wires on the schematic. (Hint: your understanding of these modules and signals would make a great section in your lab report.)

After you thoroughly understand the controller module, take a look at the datapath VHDL module. The datapath has quite a few sub-modules. Make sure you understand why each submodule is there and where each is located on the MIPS single-cycle processor schematic. You’ll notice that the alu module is not defined. Copy your ALU into your working directory, or simply paste the library, entity and architecture statements into the labeled places in the mips\_combo.vhd file.

The datapath includes the instruction and data memories as well as the processors. Each of the memories is a 256-word × 32-bit array. The instruction memory needs to contain some initial values representing the program. The test program is given in Figure 7.60 of the textbook. Study the program until you understand what it does.

1. **Prelab:** The machine language code for the program is stored in memfile.dat. Create this plain-text file and store exactly one instruction per line. Make sure there are no blank lines!
2. **Prelab:** Modify the imem section of mips\_combo.vhd to point to your data file.

# Testing the Single-Cycle MIPS Processor

In this section, you will test the processor with your ALU.

In a complex system, if you don’t know what to expect the answer should be, you are unlikely to get the right answer. Begin by predicting what should happen on each cycle when running the program.

1. **Prelab:** Complete the chart in Table 1 at the end of the lab with your predictions. *What address will the final sw instruction write to and what value will it write?*

Create a top-level testbench that provides the clock and reset signals for the MIPS processor. Run the simulation. Look at the waveforms and check that they match your predictions in Table 1. If they don’t, the problem is likely in your ALU or because you edited another part of the file.

1. **Add all of the signals from Table 1 to your waveform window.** (Note that most are not at the top level; you’ll have to drill down into the appropriate part of the hierarchy – look in the “Instance and Process Name” and “Objects” panels – to find them.)

If you need to debug, you’ll likely want to view more internal signals. However, on the final waveform that you turn in, show ONLY the following signals in this order: clk, reset, pc, instr, alu\_result, write\_data, mem\_write, and read\_data. **All the values need to be output in hexadecimal and must be readable to get full credit.**

After you have fixed any bugs, take a snip of your final waveform for your lab notebook. Simulation signals should be in the following order: clk, reset, pc, instr, aluout, writedata, memwrite, and readdata. Is the correct value being written to the correct address? Indicate (with a drawing program) where these correct signals appear in your simulation.

# Modifying the MIPS Single-Cycle Processor

You now need to modify the MIPS single-cycle processor by adding the ori and bne instructions.

1. Modify the MIPS processor schematic at the end of this lab, Figure 3, to show what changes are necessary. You can neatly draw your changes directly onto the schematic with a computer program.
2. Modify the main decoder and ALU decoder as required. Show your changes in Table 2 and Table 3 at the end of the lab.
3. Modify the VHDL code as needed to include your modifications.

# Testing the Modified MIPS Single-Cycle Processor

Next, you’ll need a test program to verify that your modified processor works. The program should check that your new instructions work properly and that the old ones didn’t break. Use lab4\_test2.asm below.

# lab4\_test2.asm

# Test MIPS instructions.

#Assembly Code

main: ori $t0, $0, 0x8000

addi $t1, $0, -32768

ori $t2, $t0, 0x8001

beq $t0, $t1, there

slt $t3, $t1, $t0

bne $t3, $0, here

j there

here: sub $t2, $t2, $t0

ori $t0, $t0, 0xFF

there: add $t3, $t3, $t2

sub $t0, $t2, $t0

sw $t3, 82($t0)

Figure 2 – MIPS assembly program: lab4\_test2.asm

1. Convert the program to machine language and put it in a file named memfile2.dat. You may choose to use the MARS assembler to help assemble this file – however, be sure to double check that the assembler creates the correct jump instruction.
2. Modify imem to load this file.
3. Modify your testbench to check for the appropriate address and data value indicating that the simulation succeeded. Run the program and check your results. Debug if necessary.
4. When you are finished, take a snip of the waveforms as before and indicate (with a drawing program) the address and data value written by the sw instruction. (Be sure to label which is which.)

Simulation signals should be in the following order: clk, reset, pc, instr, aluout, writedata, memwrite, and readdata. *What address and data value are written by the sw instruction?*

# Implement in Hardware

The skills needed to implement this design in hardware and interface it with useful peripherals (e.g., switches, LEDs, buttons, 7-segment displays, USB, VGA port, etc.) are beyond the scope of this lab and course. You will learn some of these skills in ECE 382 and ECE 383.

**Lab Report**

A great deal of information should be included in your lab report. Inserting a photo of your hand-completed chart or schematic is inappropriate. However, if hand-drawn items are a part of your thinking process, feel free to include snapshots of those. Complete (or re-create) tables with a word-processing program; complete the schematic with an appropriate drawing program. **Clearly explaining in detail how you know your waveforms are correct is critical.** Screenshots of each instruction in the simulation, modified in a drawing program to show exactly which signals at a certain time show you the information you are discussing, will be very useful in your lab report. Ensure your simulation is readable in your README. Don’t forget to analyze both simulations. Simulations without any clear visual indication of the information that indicates your MIPS processor works properly will not receive full credit.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cycle | reset | pc | instr | branch | src\_a | src\_b | alu\_result | zero | pc\_src | write\_data | mem\_write | read\_data |
| 1 | 1 | 00 | addi $2,$0,5  20020005 | 0 | 0 | 5 | 5 | 0 | 0 | 0 | 0 | x |
| 2 | 0 | 04 | addi $3,$0,12  2003000c | 0 | 0 | C | C | 0 | 0 | 0 | 0 | x |
| 3 | 0 | 08 | addi $7,$3,-9  20067fff7 | 0 | C | F7 | 3 | 0 | 0 | 0 | 0 | x |
| 4 | 0 | 0C | or $4, $7, $2  00e22025 | 0 | 7 | 2 | 7 | 0 | 0 | 2 | 0 | x |
| 5 | 0 | 10 | and $5, $3, $4  00642824 | 0 | 3 | 4 | 0 | 1 | 0 | 4 | 0 | x |
| 6 | 0 | 14 | add $5, $5, $4  00a42820 | 0 | 5 | 4 | 9 | 0 | 0 | 4 | 0 | x |
| 7 | 0 | 18 | beq $5, $7,end  10a7000a | 1 | 5 | 7 | 1 | 0 | 0 | 7 | 0 | x |
| 8 | 0 | 1C | slt $4, $3, $4  0064202a | 0 | 3 | 4 | 1 | 0 | 0 | 4 | 0 | x |
| 9 | 0 | 20 | beq $4, $0, around  10800001 | 1 | 4 | 0 | 0 | 1 | 1 | 0 | 0 | x |
| 10 | 0 | 28 | slt $4, $7, $2  00e2202a | 0 | 7 | 2 | 0 | 1 | 0 | 2 | 0 | x |
| 11 | 0 | 2C | add $7, $4, $5  00853820 | 0 | 4 | 5 | 9 | 0 | 0 | 9 | 0 | x |
| 12 | 0 | 30 | sub $7, $7, $2  00e23822 | 0 | 4 | 5 | FF | 0 | 0 | FF | 0 | x |
| 13 | 0 | 34 | sw $7, 68($3)  ac670044 | 0 | 3 | 7 | 0 | 1 | 0 | 7 | 1 | x |
| 14 | 0 | 38 | lw $2, 80($0)  8c020050 | 0 | 0 | 50 | 0 | 1 | 0 | 2 | 0 | 7 |
| 15 | 0 | 3C | j end  08000011 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | x |
| 16 | 0 | 44 | sw $2, 84($0)  ac020054 | 0 | 0 | 2 | 0 | 1 | 0 | 2 | 1 | x |

Table 1 - First sixteen cycles of executing lab4\_mipstest.asm



Figure 3 – MIPS single-cycle processor schematic. Modify this diagram with your modifications needed to add the ori and bne instructions.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction | Op5:0 | reg\_write | reg\_dst | alu\_src | branch | mem\_write | mem\_to\_reg | alu\_op1:0 | jump | ori | bne |  |
| R-type | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 0 |  |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 1 | 00 | 0 | 0 | 0 |  |
| sw | 101011 | 0 | X | 1 | 0 | 1 | X | 00 | 0 | 0 | 0 |  |
| beq | 000100 | 0 | X | 0 | 1 | 0 | X | 01 | 0 | 0 | 0 |  |
| addi | 001000 | 1 | 0 | 1 | 0 | 0 | 0 | 00 | 0 | 0 | 0 |  |
| j | 000010 | 0 | X | X | X | 0 | X | XX | 1 | 0 | 0 |  |
| ori | 001101 | 1 | 0 | 1 | 0 | 0 | 0 | 11 | 0 | 1 | 0 |  |
| bne | 000101 | 0 | X | 0 | 1 | 0 | X | 01 | 0 | 0 | 1 |  |

Table 2 – Main Decoder table. Insert your expanded functionality logic directly into this table.

|  |  |
| --- | --- |
| alu\_op1:0 | Meaning |
| 00 | Add |
| 01 | Subtract |
| 10 | Look at funct field |
| 11 | Or |

Table 3 – ALU Decoder table. Insert your expanded functionality logic directly into this table.

|  |  |
| --- | --- |
| Lab 4 Cut Sheet | **Name:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Instructor:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Section:** \_\_\_\_\_\_\_\_\_\_\_ |

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Points** | **Out of** | **Late Date** |
| Prelab |  | 10 |  |
| Use of Git |  | 5 |  |
| Lab notebook |  | 20 |  |
| Simulation explanations/analysis |  | 15 |  |
| .vhd and .dat files |  | 10 |  |
| Simulation results w/ no mod |  | 10 |  |
| Simulation results w/ mod |  | 30 |  |
| **Total** |  | **100** |  |

**Number of hours spent on Lab 4:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (no points associated with this unless you leave it blank)

**Suggestions to improve Lab 4 in future years:** (use blank space below)

1. Modeled after a lab provided with instructor notes for Digital Design and Computer Architecture, David Money Harris & Sarah L. Harris, 2nd Edition [↑](#footnote-ref-1)