# Lab 11: MIPS Multicycle Processor 2

TA: ariana.villegas, Prof.: jgonzalez @utec.edu.pe

## Multicycle Processor Completion

In this lab, you will complete your own multicycle MIPS processor! Please refer to your Lab 10 handout for an overview of the processor. In Lab 10, you created the control unit. In this lab you will design the `datapath` and `mem` units and test your completed MIPS multicycle processor.

This lab uses the same testbench and generic parts as in Lab 9. Copy your `mipstest.v`, `mipsparts.v`, and `alu_xx.sv` files from Lab 9 to your Lab11_xx directory. Also copy your `mipsmulti_xx.v` file containing your controller from Lab 10.

Finally, copy and migrate to Verilog the `topmulti.v` and `mipsmem.v` files containing the top-level module and the memory from Verilog source folder. Look over the files and see how they work.

## Test Program

Use the same test program and `memfile.dat` from the first part of Lab 9. Copy it to your lab11_xx directory.

As in Lab 9, it is very helpful to first predict the results of a test program before running the program so that you know what to expect and can discover and track down discrepancies. **Table 1**, which is partially completed, lists the expected instruction trace while running the test program. Complete the remainder of the table. Do this before you run simulations so you have a set of expectations to check your results against; otherwise, it is easy to fool yourself into believing that erroneous simulations are correct.

Notice that the instruction (`instr`) is fetched during state 0 and therefore not updated until state 1 of each instruction.

When the ALUResult will not be used (e.g. in the Decode state of a nonbranch instruction, or the Writeback state of any instruction), you may indicate an 'x' for don't care rather than predicting the useless value that the processor will actually compute.

## Datapath Design

Refer to Figure 1 in Lab 10 for the hardware modules you need to set up your datapath. Design the datapath unit in Verilog.

Remember that you may reuse hardware from earlier labs (such as the ALU, multiplexers, registers, sign-extension hardware modules, register file, etc.) wherever possible.

All of your registers should take a *Reset* input to reset the initial value to a known state (0). The Instruction Register and PC also require enable inputs. Pay careful attention to bus connections; they are an easy place to make mistakes.

Simulate your processor using the testbench given (`mipstest.v`). The Reset signal is set high at first. Display, at a minimum, the *PC*, *Instr*, FSM *state* (from within your `controller` module), *SrcA* and *SrcB* (from within your datapath), *ALUResult*, and *Zero*, and the control word. You will likely want to add other signals to help debug. Check that your results match the expectations from Table 1. If there are any mismatches, debug your design and fix the errors.

**When you are finished – congratulations! You have built a microprocessor by yourself and have proven your mastery of microarchitecture, Verilog, FSMs, and logic design!**

## Debugging

Hopefully your lab will have at least one error so you will get to hone your debugging skills! Here are some hints:

- Be sure you thoroughly understand how the MIPS multicycle processor is supposed to work. This system is too complex to debug by trial and error. You should be able to predict what value every signal should be at every point in time while debugging.
- In general, trace problems by finding the first point in a simulation where a signal has an incorrect value. Don't worry about later problems because they could have been caused by the first error. Identify which circuit element is producing the bad output and add all its inputs to the simulation. Repeat until you have isolated the problem.

## What to Turn In

## For up to 17 pts:

Include the following elements **in the following order** in your final submission. Clearly label each part by number. Poorly organized submissions will lose points.

1. A completed copy of Table 1 indicating the expected outcome of running the test program.
2. All Verilog code of the datapath (implementation and testing).
3. Simulation waveforms of the processor showing *CLK*, *Reset*, *state*, *PC*, *instr*, and *ALUResult* in this order while running the test program. As always, output the values in hex (or decimal if that is more readable) and make sure they are readable. Do the results match your expectations? Does the program indicate Simulation Succeeded?

**For up to 20 pts: See HP and HH book for documentation.**

**Option A:** Add Floating Point Units to your design: ADD, MUL, LOAD, STORE.

**Option B:** Implement a Pipeline (in VHDL at the HP book ;)).

| Cycle | Reset | PC | Instr | (FSM) state | SrcA | SrcB | ALUResult | Zero | Control Word |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 00 | 0 | 0 | 00 | 04 | 04 | 0 | 5010 |
| 2 | 0 | 04 | addi 20020005 | 1 | 04 | x | x | 0 | 0030 |
| 3 | 0 | 04 | addi 20020005 | 9 | 00 | 05 | 05 | 0 | 0420 |
| 4 | 0 | 04 | addi 20020005 | 10 | x | x | x | 0 | 0800 |
| 5 | 0 | 04 | addi 20020005 | 0 | 04 | 04 | 08 | 0 | 5010 |
| 6 | 0 | 08 | addi 2003000c | 1 | 08 | x | x | 0 | 0030 |
| 7 | 0 | 08 | addi 2003000c | 9 | 00 | 0c | 0c | 0 | 0420 |
| 8 | 0 | 08 | addi 2003000c | 10 | x | x | x | 0 | 0800 |
| 9 | 0 | | | | | | | | |
| 10 | 0 | | | | | | | | |
| 11 | 0 | | | | | | | | |
| 12 | 0 | | | | | | | | |
| 13 | 0 | | | | | | | | |
| 14 | 0 | 10 | or 00e22025 | 1 | 10 | x | x | 0 | 0030 |
| 15 | 0 | 10 | or 00e22025 | 6 | 03 | 05 | 07 | 0 | 0402 |
| 16 | 0 | 10 | or 00e22025 | 7 | x | x | x | 0 | 0840 |
| 17 | 0 | 10 | or 00e22025 | 0 | 10 | 04 | 14 | 0 | 5010 |
| 18 | 0 | 14 | and 00642824 | 1 | 14 | x | x | 0 | 0030 |
| 19 | 0 | 14 | and 00642824 | 6 | 0c | 07 | 04 | 0 | 0402 |
| 20 | 0 | 14 | and 00642824 | 7 | x | x | x | 0 | 0840 |
| 21 | 0 | 14 | and 00642824 | 0 | 14 | 04 | 18 | 0 | 5010 |
| 22 | 0 | 18 | add 00a42820 | 1 | 18 | x | x | 0 | 0030 |
| 23 | 0 | 18 | add 00a42820 | 6 | 04 | 07 | 0b | 0 | 0402 |
| 24 | 0 | 18 | add 00a42820 | 7 | x | x | x | 0 | 0840 |
| 25 | 0 | 18 | add 00a42820 | 0 | 18 | 04 | 1c | 0 | 5010 |
| 26 | 0 | | | | | | | | |
| 27 | 0 | | | | | | | | |
| 28 | 0 | | | | | | | | |
| 29 | 0 | | | | | | | | |
| 30 | 0 | | | | | | | | |
| 31 | 0 | | | | | | | | |
| 32 | 0 | | | | | | | | |
| 33 | 0 | | | | | | | | |
| 34 | 0 | | | | | | | | |
| 35 | 0 | | | | | | | | |
| 36 | 0 | | | | | | | | |
| 37 | 0 | | | | | | | | |
| 38 | 0 | | | | | | | | |
| 39 | 0 | | | | | | | | |
| 40 | 0 | 30 | add 00853820 | 1 | 30 | x | x | 0 | 0030 |
| 41 | 0 | 30 | add 00853820 | 6 | 01 | 0b | 0c | 0 | 0402 |
| 42 | 0 | 30 | add 00853820 | 7 | x | x | x | 0 | 0840 |
| 43 | 0 | 30 | add 00853820 | 0 | 30 | 04 | 34 | 0 | 5010 |
| 44 | 0 | 34 | sub 00e23822 | 1 | 34 | x | x | 0 | 0030 |
| 45 | 0 | 34 | sub 00e23822 | 6 | 0c | 05 | 07 | 0 | 0402 |
| 46 | 0 | 34 | sub 00e23822 | 7 | x | x | x | 0 | 0840 |
| 47 | 0 | 34 | sub 00e23822 | 0 | 34 | 04 | 38 | 0 | 5010 |
| 48 | 0 | 38 | sw ac670044 | 1 | 38 | x | x | 0 | 0030 |
| 49 | 0 | 38 | sw ac670044 | 2 | 0c | 44 | 50 | 0 | 0420 |
| 50 | 0 | 38 | sw ac670044 | 5 | x | x | x | 0 | 2100 |
| 51 | 0 | 38 | sw ac670044 | 0 | 38 | 04 | 3c | 0 | 5010 |
| 52 | 0 | 3c | lw 8c020050 | 1 | 3c | x | x | 0 | 0030 |
| 53 | 0 | 3c | lw 8c020050 | 2 | 00 | 50 | 50 | 0 | 0420 |
| 54 | 0 | 3c | lw 8c020050 | 3 | x | x | x | 0 | 0100 |
| 55 | 0 | 3c | lw 8c020050 | 4 | x | x | x | 0 | 0880 |
| 56 | 0 | 3c | lw 8c020050 | 0 | 3c | 04 | 40 | 0 | 5010 |
| 57 | 0 | | | | | | | | |
| 58 | 0 | | | | | | | | |
| 59 | 0 | | | | | | | | |
| 60 | 0 | | | | | | | | |
| 61 | 0 | | | | | | | | |
| 62 | 0 | | | | | | | | |

**Table 1.** Expected Instruction Trace