# Introduction to Computer Engineering (E114)

## Lab 8: MIPS Components

## Introduction

In this lab, you will begin developing a schematic model of the MIPS processor. In the next lab, you will assemble the components into a fully functional microprocessor and will run programs on the model. By the end of this lab, you should thoroughly understand the internal operation of the MIPS instruction and execution units.

Please read and follow the instructions in this lab carefully. You will need the results of this lab to complete the next lab, so any errors you make will require correction later. Also, in the past, many students have lost points for silly errors like not printing all the signals requested.

The lab will involve several steps. First, you will design and test basic blocks including a multiplexer and sign extension logic. Then you will complete the instruction unit. Before starting this lab, you should be very familiar with the single cycle implementation of the MIPS processor described in Section 5.3 of Patterson & Hennessy.

Our model of the MIPS processor divides the machine into four major units: the execution unit (ebox), instruction unit (ibox), data memory unit (dbox), and control unit (cbox). The partition is shown in Figure 1 on the next page. You have already designed the control unit in Lab 2.

Each unit in turn is constructed from various functional blocks. For example, as shown in the figure, the ebox contains the ALU (that you designed in Lab 5), the register file, the sign extension logic, and three multiplexers to select the register to write, the data to write, and the input to the second source of the ALU.

You will use several techniques in your design. You will draw new cells with the schematic editor. You will add logic to partially completed designs provided for you. You will use Xilinx's LogiBLOX feature to automatically create special structures like RAMs, ROMs, and multiplexers. And in Lab 9, you will import your existing controller and ALU as symbols.

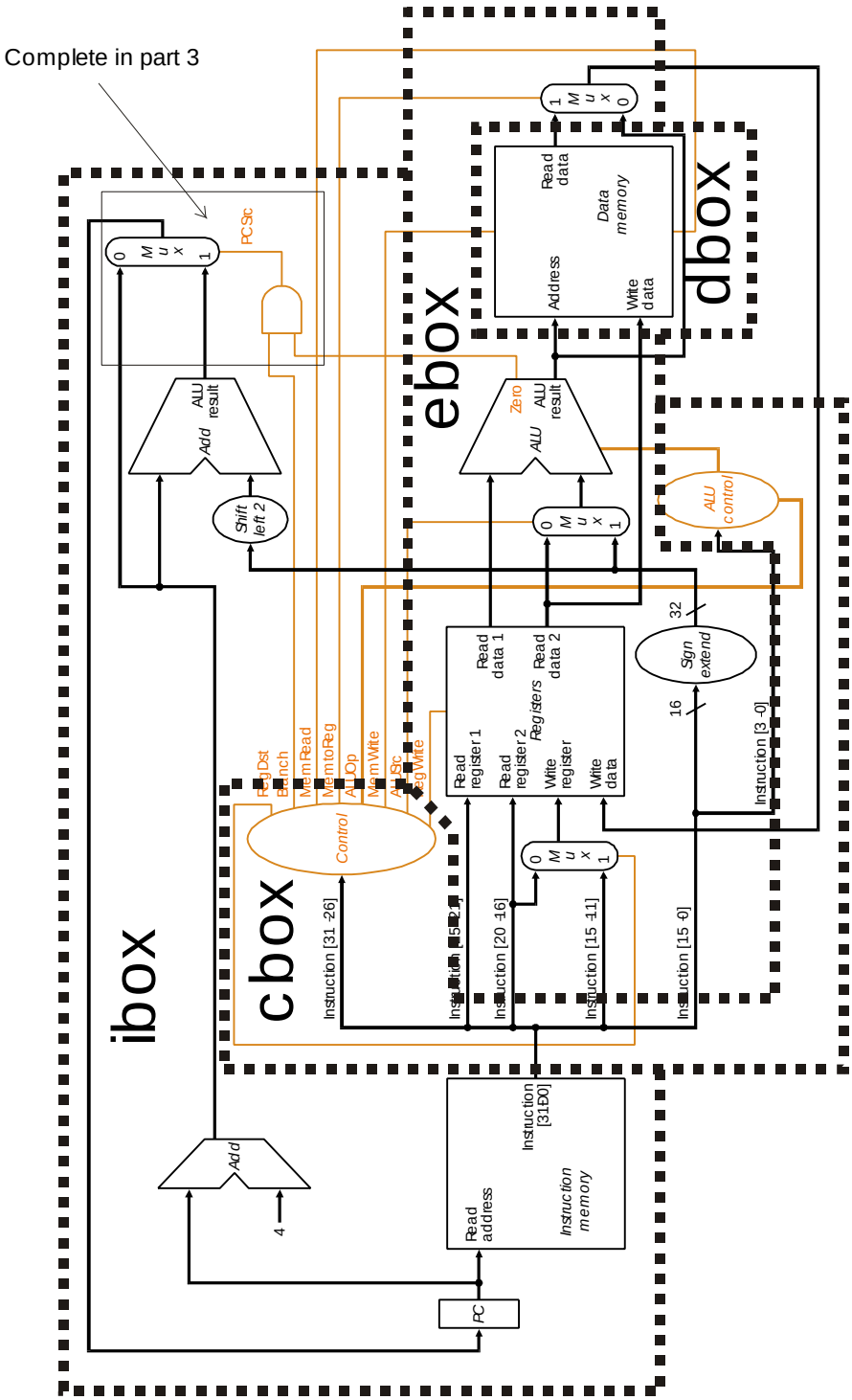



Figure 1: MIPS Processor Block Diagram

## 1. Multiplexer

Your first LogiBLOX design will be a 5-bit multiplexer used to extract the write register from either Instruction[20:16] or Instruction[15:11] depending whether the instruction uses the R format or I format.

We have already created a project with some of the blocks for you. Use the File•Copy Project command in the Project Manager to copy it to your directory. With Windows Explorer, make sure the EngServe1 \E114 directory is mapped to the G drive (or some other drive letter if the G drive is already in use). If not, map it yourself. Then return to the Project Manager and browse to find the source on g:\labs\lab8\lab8base.pdf and copy it to your directory under the name lab8_xx (where xx are your initials). You will get some errors about being unable to copy a project with a different name; ignore these. Then open your lab8_xx in the Project Manager.

Create a new schematic. Then you will use the LogiBLOX feature to create a 2-input 5-bit multiplexer. Choose Tools•LogiBLOX Module Generator from the schematic editor menu. In the LogiBLOX Module Selector window, select a Module Type of Multiplexer. Specify a data bus width of 5 bits. Make the Module Name mux2_5. In the Details panel, click Type 2 and be sure Input Busses is 2 to indicate a 2-input 5-bit wide multiplexer. Finally choose OK to generate the mux.

Now you will simulate your 5-bit multiplexer to make sure it works. Create a simple schematic with your mux, two 5-bit bus inputs A and B, a 5-bit output bus Y, and a select SEL. Use the integrity test after each time you draw a schematic to make sure there are no errors. You will get warnings about hierarchy connectors on the top level schematic; this is ok. Click the Simulation button on the Project Manager to simulate the mux. You may get warnings that the LogiBLOX are out of date; you can safely ignore these. Use the Signal•Add Signals command to add the A, B, Y, and SEL signals. Some of these signals are busses, so they will appear as (A4, A0), (B4, B0), etc.

Connect SEL to the B0 clock and A and B to the following formulae, respectively, as you learned in Lab 5:

F0: [1F]12[00]8 F1: [15]10[0A]10

Simulate the multiplexer for 20 ns and check that Y matches your expectations. When your design works correctly, print the simulation waveforms.

Finally, repeat this process to generate a mux2_32 with LogiBLOX. You need not simulate this mux if you are confident you've repeated the correct design process.

## 2. Sign Extension

Your next task is to complete the sign extension functional block, *sgnext*. This block converts a 16-bit signed 2's complement number to a 32-bit signed 2's complement number by copying the sign of the 16 bit number into the upper 16 bits of the 32-bit number, as described on page 216 of your book. You'll use sgnext later in the ebox.

Create another new schematic. Place a 16-bit input bus D and a 32-bit output bus EXT. You cannot connect inputs directly to outputs, so you may find buffers useful. Then simulate the sign extension cell with the following stimulus for D:

[DADE]10[9999]10[FEDD]10[1234]10

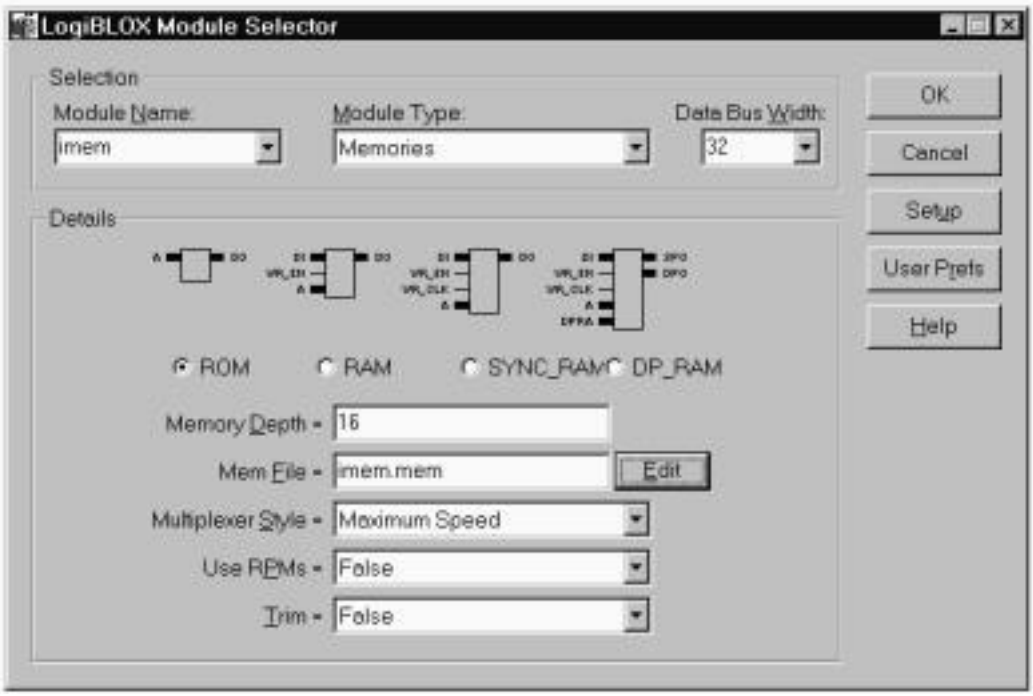
When your module works properly, print the simulation waveforms showing the correct output. Also print your schematic containing your sgnext. Finally, create a macro for sgnext, as you did with the fulladder and ALU_1 in Lab 5, by selecting Hierarchy•Create Macro Symbol from Current Sheet. You will use sgnext in the ebox in Lab 9.

## 3. Instruction Box

The Instruction Box contains all of the Program Counter (PC) logic and the Instruction memory. After each instruction, it either increments the PC by 4 or adds the branch offset when a branch should be taken.

In the project manager, choose Project•Add Source Files and add ibox.sch, a partially completed schematic of the ibox. Look at the ibox schematic. It is missing the branch selection logic and the instruction memory. There is some funny syntax used to combine bits of busses. For example, the branch offset needs to be computed as the 32-bit constant ConstExtended shifted left by 2. This is done with the ConstExtended[31:0] bus on the left and the Gnd terminal in the middle. The input bus to the branch adder is named ConstExtended[29:0],Gnd,Gnd, which represents a 32-bit number consisting of the lower 30 bits of ConstExtended followed by two 0's, as a left shift by 2 would produce. This is done correctly for you in the schematic; observe how the signals are connected by name and that you should not connect the busses with a line yourself.

The instruction memory (imem) will be constructed as a LogiBLOX ROM holding the program to execute. Bring up the LogiBLOX Module Selector dialog and create a module type Memories. Set the module name to imem (for Instruction Memory), the Data Bus Width to 32, the Memory Depth to 16 and the type to ROM. This will produce a 16 word by 32-bit ROM. Enter imem.mem as the name of the memory file and click Edit.




This will bring up a window in which you can enter data stored in the ROM. Immediately after the line DATA, enter the following hexadecimal numbers which represent the Fibonacci program from Lab 5:

```
20080008
2009ffff
200A0001
```

```
11000005
012a5820
21490000
216a0000
2108ffff
1000fffa
1000ffff
```

This .mem file corresponds to the following program:

```
# fib, by Travis Furrer, January 1999
#
# t0: loop index (n)
# t1: fib(n-2) (starts with -1)
# t2: fib(n-1) (starts with 1)
# t3: fib(n) (will contain result after execution)
                              # Address   Machine Instruction
fib:  addi $t0,$0,8           # 00        20080008
      addi $t1,$0,-1          # 04        2009ffff
      addi $t2,$0,1           # 08        200A0001
loop: beq  $t0,$0,done        # 0C        11000005
      add  $t3,$t1,$t2        # 10        012a5820
      addi $t1,$t2,0          # 14        21490000
      addi $t2,$t3,0          # 18        216a0000
      addi $t0,$t0,-1         # 1C        2108ffff
      beq  $0,$0,loop         # 20        1000fffa
done: beq  $0,$0,done         # 24        1000ffff
```

Save and close the .mem file, then click OK on the Module Selector to create the instruction memory ROM. You should obtain a message that the LogiBLOX symbol imem was successfully place in the library.

Add the imem to the schematic. Connect the address input to the program counter register output and the data output to the instruction output of the schematic. Then complete the logic to compute the next program counter from Branch, Zero, PC + 4 (i.e. PCPLUSFOUR), and PC + 4 + the branch offset (i.e. PCPP), as is shown in Figure 1. Double-click on the multiplexer select signal to name it PCSrc, as it is named in the block diagram. Hint: you will probably need your mux2_32 from part 1 of this lab. When you integrity test, you will also get warnings that ConstExtended31 and 30 are loadless nets. This is ok; we didn't intend to connect anything to those signals.

We will simulate the ibox as if it were running the Fibonacci program by providing appropriate inputs and observing the PC and Instruction outputs. Watching the internal signals such as PCPlusFour may also be helpful when debugging. To understand our expected outputs and determine if the ibox is operating correctly, we must think through the sequence of signals we should expect as we run the program. On the first cycle, we will reset the machine to clear the program counter to zero. On subsequent cycles, the program counter will either advance by 4 or be changed by a jump.

The ibox requires Reset, Branch, Zero, and ConstExtended inputs. We will reset the ibox for the first cycle to initialize the PC to 0. The ConstExtended input is a 32 bit sign-extended version of the 16 least significant bits of the instruction. Branch is asserted by the cbox, which you designed in Lab 2 whenever the instruction is a branch. To

determine the Zero input value, we must imagine running the program and computing the output of the ALU. As you designed in Lab 5, Zero is true whenever the ALUResult signal is all zeros. Remember that beq causes the ALU to subtract the two operands. Complete the Table 1 at the end of the lab showing the first twelve cycles of operation of the machine to help understand the values you should expect to see on each cycle. If you use PCSpim to help you with the assembly language to machine language translation, recall that it has a bug that produces branch offsets one greater than they really should be. **It is important to complete Table 1 before proceeding so you can match simulation against expectations!**

Now, add the following stimulus to the ibox inputs to simulate the first twelve cycles:

| | |
|---|---|
| CLK: | B0 |
| Reset: | F0 = [1]10[0]1000 |
| Branch: | F1 = [0]36[1]10[0]40[1]20[0]20 |
| Zero: | F2 = [0]46[1]10[0]10[1]10[0]10[1]10[0]20[1]10 |
| ConstExtended | F3 = [00000008]16[FFFFFFFF]10[00000001]10[00000005]10 [00005820]10[00000000]20[FFFFFFFF]10[FFFFFFFA]10 [00000005]10[00005820]10[00000000]10 |

Simulate the ibox for 125 ns. Check that the PC and Instruction and PCSrc values match your expectations from Table 1 on each cycle. If they do not match, debug your design; you will need it in the next lab. Turn in a printout of your waveforms showing all of these inputs and outputs. Also turn in a printout of your completed ibox schematic.

## 4. Controller and ALU

In the next lab when you complete the single-cycle processor, you will need your controller and ALU from labs 2 and 5, respectively. At this point, you should check that you completed these labs correctly. If you did not turn in the lab or had errors in your lab, correct them now. You may refer to the solutions, but may not copy files from other people. Be sure to simulate the labs; in the past, many students have not and have spent many more hours than necessary trying to find the bugs when they put all the pieces together in the next lab. If your schematics are fully functional, congratulations: this is a short lab.

## What to Turn In

Please turn in each of the following items:

1. Please indicate how many hours you spent on this lab. This will not affect your grade, but will be helpful for calibrating the workload for next semester's labs.
2. A printout of your schematics from:
   - sgnext
   - ibox
3. Simulation waveforms of:
   - mux2_5
   - sgnext
   - ibox

   Be sure the waveforms match your expectations. Check that the waveforms are zoomed out enough that the grader can read your bus values. Use several pages as necessary.
4. A completed version of Table 1.
5. Don't turn in Labs 2 or 5, but be sure they are functional. The next lab is long enough that you will not want to redo those labs before completing Lab 9.

| Cycle | Reset | PC | Instruction | Branch | ALUResult | Zero | PCSrc |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 00 | `addi $t0,$0,8`<br>`20080008` | 0 | 8 | 0 | 0 |
| 2 | 0 | 04 | `addi $t1,$0,-1`<br>`2009ffff` | 0 | -1 | 0 | 0 |
| 3 | 0 | 08 | `addi $t2,$0,1`<br>`200A0001` | | 1 | 0 | 0 |
| 4 | | 0C | | 1 | | | 0 |
| 5 | | | | | 0 | 1 | 0 |
| 6 | | | | | | | |
| 7 | | | | | | | |
| 8 | | | | | | | |
| 9 | | | | | | | |
| 10 | | 0C | `beq $t0, $0, done`<br>`11000005` | | | | |
| 11 | | | | | | | |
| 12 | | | | | | | |

Table 1: First twelve cycles of executing Fib.