ECS 154B Lab 2, Spring 2016
Due by 11:55 PM on April 22, 2016
Via SmartSite

Goals

- Build and test a single cycle MIPS CPU that implements a subset of the MIPS instruction set.
- Design a combinational logic control unit.

Description

In this lab, you will use Logisim to build a single cycle CPU to understand the MIPS control and datapath signals. To test your CPU, you will run an assembly language program given to you, and simulate the operations in Logisim. You will be given several functional blocks to help you out. You must implement the control signals as combinational logic.

Details

You will be given an empty project to start your lab, which includes an implementation of an ALU and a Register File. In creating your CPU, you are allowed to use all of the features and modules available in Logisim. The project is available in the Assignment section of the course's SmartSite page. Simply download the .zip file containing **Lab 2 Given.circ**. The blocks are described below:

ALU

- **A**, **B**: The 32 bit data inputs to the ALU.
- **ALUResult**: The 32 bit result of the ALU operation.
- **Overflow**: Not used in this lab.
- **Shamt**: 5 bit shift amount. Not used in this lab. Connect a constant zero to this input.
- **Zero**: A flag bit that is set when the ALU result equals zero (all bits are low).
- **ALUCtl**: The 4 bit control input, as described in the following table:

Instruction	ALUCtl3	ALUCtl2	ALUCtl1	ALUCtl0
`OR`	0	0	0	0
`SLTU`	0	0	0	1
`SLT`	0	0	1	0
`ADD`	0	0	1	1
`SUB`	0	1	0	0
`XOR`	0	1	0	1
`AND`	0	1	1	0

More instruction types to be supported will be added in Lab 3, thus the extra ALUCtl signal.

Register File

- **Clock**: The main clock signal.
- **RdAddr1**, **RdAddr2**: The 5 bit register addresses to read. MIPS has 32 registers in its register file.
- **RdData1**, **RdData2**: The 32 bit data values from the read registers.
- **WrAddr**: The 5 bit register address to write.
- **WrReg**: A control signal that causes the register file to be written to when high. If **WrReg** is low, no register values will change.
- **WrData**: The 32 bit value to store in the register specified by **WrAddr**, if **WrReg** is set.

You will need to add additional logic blocks to your diagram: a PC, an instruction memory, a data memory, and combinational logic circuits to decode the current instruction word and enable the respective datapaths.

PC

- Use a 32 bit Register as your PC.
- **Please note that MIPS is a byte addressable architecture, and therefore PC is incremented by 4, not 1.**

Instruction Memory - ROM

- Use an 8 bit address, 32 bit data ROM from the memory library as your instruction memory, to which you can load your hex code.
- A sample test program will be included. To load the instructions into your ROM, right-click the ROM and select *Edit Contents...* In the Hex Editor that pops up, click *Open* and select **mipsInstructions.lst**, which is provided with the assignment files.
- The corresponding MIPS instructions are in **mipsInstructions.mps**.
- You can assemble your own test programs using Sean Davis' ASIDE program, available at `http://csiflabs.cs.ucdavis.edu/~ssdavis/50/`. After compiling, you will need to edit the `.lst` file that corresponds to your code to match the input that Logisim expects.
- The MIPS architecture is byte addressable, so the PC is incremented by 4 to reach the next instruction. For example the first instruction will be at 0x00000000, the second at 0x00000004, the third at 0x00000008, and so on.

Data Memory - RAM

- Use an 8 bit address, 32 bit data RAM from the memory library as your data memory. The separate load and store ports option is easier to use but you can use the other two options if you'd like.
- **A**: the 8 bit address of the word you want to access.
- **D**, left side: the 32 bit word to be written or stored at the address specified by A.
- **str**: When high, the value on the left D is stored at address A.
- **sel**: When 0, the chip is disabled. Set to high or leave floating.
- ^: the main clock signal.
- **ld**: When high, the value on the memory at address A is placed on the right D.
- **clr**: When high, sets all of the values in the RAM to 0. Either set low or leave floating.
- **D**, right side: the 32 bit word to be read or loaded from the address specified by A.

Instructions to Implement

Your CPU must execute the following instructions:

`AND, ANDI, ADD, ADDI, OR, ORI, SUB, SLT, SLTU*, XOR, LW, SW, BEQ, J, JAL, JR`

Note that the instructions for SLTU are incorrect in the given MIPS PDF. See the section below.

MIPS Architecture

A PDF was included in the archive to help you with the lab. Note that the instructions for `SLTU` are incorrect. It should read:

> If rs < rt with an unsigned comparison, put 1 into rd. Otherwise put 0 into rd.

For this lab, you must use only combinational logic to implement the control signals. Do not use a MUX with constant inputs to generate your control signals.

`JAL` and `JR`

The book does not explain too much about these instructions or give any designs on how to implement them, so it will be up to you to figure out what you need to do.

- `JAL` (jump and link) is just like `J`, except it stores the contents of PC + 4 in register 31.
- Use: `jal offset`
- Effect: $31 = PC + 4. PC = PC + 4[31..28], Inst[25..0], 00
- Encoding: `0000 11ii iiii iiii iiii iiii iiii iiii`, where the `i`'s are the bits encoding the immediate value.
- `JR` (jump, register) sets the PC to the value contained in register s.
- Use: `jr $s`
- Effect: PC = $s
- Encoding: `0000 00ss sss0 0000 0000 0000 0000 1000`, where `s` is the address of the register.

The following probes are included to help you debug your lab.

Label Name	Radix	Description
PC	Unsigned Decimal	The current value of the PC.
WrReg	Binary	1 if writing to the register file, 0 otherwise.
WrAddr	Unsigned Decimal	The address of the register that is going to be written to.
WrData	Signed Decimal	The value that is going to be written to the register.
MemWr	Binary	1 if writing to memory, 0 otherwise.
MemData	Signed Decimal	The data to be written to memory.
Branch	Binary	1 if taking a branch, 0 otherwise.
Jump	Binary	1 if executing the jump instruction, 0 otherwise.
Jal	Binary	1 if executing the jump and link instruction, 0 otherwise.
Jr	Binary	1 if executing the jump register instruction, 0 otherwise.

Submission

README

Include a README file with the following:

- Name of Partner 1, and Student ID Number
- Name of Partner 2, and Student ID Number
- Any known issues or comments you would like the TAs to see.

Submit your circuit and README to the Lab 2 Assignment on SmartSite. As with the previous lab, you may use up to 2 slip days.

Only one partner should submit the assignment. Make sure to turn in the README with both names included so that the TAs know who is working with whom. You may re-submit as many times as you like.

Grading

To grade your assignment, the TAs will run your CPU with the instructions contained in the file **mipsInstructions.lst**, then look at the contents of your registers after the program is finished. If you look at the comments in **mipsInstructions.mps**, it will tell you what the final states of the registers (or memory location) should be. For each register or memory location with a correct value at the end of the run, you will receive 1 point. Any instructions that run that should not have will reduce your score. There are 14 total points.

- 90% Implementation
- 10% Interactive Grading

Hints

- You can use Logisim's Analyze Circuit tool, under the Project menu, to automatically construct combinational circuits for you. This can be an extremely time-saving tool, so make an effort to learn how to use it.
- Test and debug in steps. Start with a subset of the lab requirements, implement it, test it, and then add other requirements. Performing the testing and debugging in steps will ease your efforts. For example, you could implement the R-type and `ADDI` instructions, and verify that those work as intended. Then, add the branch instruction, and finally add the memory access instructions.
- Think about the hardware you are creating before trying it out. The text is necessarily vague and leaves out details, so do not simply copy the figures and expect your CPU to work.
- Remember that, though the PC and data addresses are 32 bits, the instruction memory and data memory addresses are only 8 bits. Be careful which bits you use to address the two memories.