ECS 154B, Winter 2015, Lab 2
Due by 11:55 p.m. on January 30, 2015
Via Smartsite

Objectives

- 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 assembly language programs that you write on it and simulate the operation in Logisim. You will be given several functional blocks to help you out and an assembler that will generate a file to initialize your program memory. You must implement the control signals as combinational logic.

Details

You will be given an empty project to start your lab that 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 SingleCycleMipsGiven.circ file. The blocks are described below:

- **ALU**
  - A,B: The 32-bit data inputs to the ALU.
  - ALUCtl: The 3-bit control input as described in the following table
  - ALUResult: The 32-bit result of the ALU operation.
  - 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).
  - Overflow: Not used in this lab.

Instruction	ALUCtl2	ALUCtl1	ALUCtl0
Sub	0	0	0
And	0	0	1
Add	0	1	0
Or	0	1	1
Slt	1	0	0

- **Register File**
  - 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.
  - RdAddr1, RdAddr2: The 5-bit register addresses to read because MIPS has 32 registers in its register file.
  - RdData1, RdData2: The 32-bit data values from the read registers.
  - Clock: The main clock signal.

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 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 (and not 1).
- **ROM** - Instruction Memory
  - 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, click the ROM and next to Content click (click to edit) on the pane on the left. In the new window that pops up click open and select mipsInstructions.txt.
  - The corresponding MIPS instructions are in mipsInstructions.rb
  - You can assemble your own test programs using /home/cs154b/bin/mips2mif utility found on the csif machines. You can then enter the hex values in the ROM manually.
  - MIPS architecture is byte addressable so the PC is incremented by 4 to reach the next instruction. For example the 1st instruction will be at 0x00000000, the 2nd at 0x00000004, the 3rd at 0x00000008, and so on.
- **RAM** - Data Memory
  - 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 2 options if you want to.
  - A : the 8 bit address of the word you want to access

- left D : the 32 bit word to be written or stored at the address specified by A.
- right D : the 32 bit word to be read or loaded from the address specified by A.
- str : When high the value on the left D is stored at address A.
- ld : When high the value on the memory at address A is placed on the right D.
- sel : When 0 the chip is disabled. Set to high or leave floating.
- clr : When high sets all of the values in the RAM to 0. Either set low or leave floating.
- $\dot{}themainclocksignal$.

Instructions to Implement

Your CPU must execute the following instructions:

ADD, SUB, ADDI, LW, SW, AND, OR, ANDI, ORI, SLT, SLTI, BEQ, J, JAL, and JR

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 is just like jump except it stores the contents of PC + 4 in register 31.
- Use: jal
- 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 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

Submission

In order to facilitate the grading of your lab, include probes with following labels and radixes so that the TA can identify any of the following signals in your assignment. (some of them are already present in the given circuit)

Label Name	Radix	Description
PC	Unsigned Decimal	The current value of the PC
WrReg	Binary	1 if writing the register file, 0 otherwise
WrAddr	Unsigned Decimal	The address of the register that is going to be written.
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.

You may add pins in order to facilitate debugging, but don't change the existing ones.

Include a README file with the following:

- name1, SID
- name2, SID
- any known issues or comments you'd like me to see

Submit your circuit and README to the Lab 2 Assignment on SmartSite.

Only one partner should submit the assignment. Make extra sure to turn in the README with your names so that I know who is working with whom. You can re-submit as many times as you would like.

Grading

To grade your assignment I will run your CPU with the instructions in mipsInstructions.rb file and look at the contents of your registers after the program is finished. If you look at the comments in mipsInstructions.rb it will tell you what the final states of the registers should be. For each register with a correct value at the end of the run you will receive 1 point. There are 12 total points.

- 90% Implementation
- 10% Interactive Grading

Note: You can have up to 2 slip days

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, 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 memories.