ECS 154B – Winter 2014 – Lab 2
Due by 11:55 PM on February 2, 2014

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 the MIPS ALU and a Register File. The project is available in the Resources section of the course's SmartSite page. Simply download the "MIPS-given.circ" file. The blocks are described below:

- ALU
  - `A`,`B`: The 32-bit data inputs to the ALU.
  - `ALUCtl`: The 4-bit control input as described on page 316 of the text. `ALUCtl[3]` is the left most bit in the table.
  - `ALUResult`: The 32-bit result of the ALU operation.
  - `Zero`: A flag bit that is set when the ALU result equals zero (all bits are low).
  - `Overflow:`: Not used in this lab.
- Register File
  - `RdReg1`, `RdReg2`: 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.
  - `WrReg`: The 5-bit register address to write.
  - `WrData`: The 32-bit data to store in the register specified by `WriteReg` if `RegWrite` is set.
  - `RegWr`: A control signal that causes the register file to be written to when set. If `RegWrite` is low, no register values will change.
  - `Clock`: The main clock signal.

You will need to add additional logic blocks. A PC, an instruction Memory, a Data Memory, and Combinational logic circuits which will essentially decode the instruction word, and enable the respective datapaths.

- PC
  - You may use a 32 bit Register as your PC.
  - Please note that MIPS is a byte addressable architecture therefore PC is incremented by 4 (and not 1).
- ROM - Instruction Memory
  - You can simply use a 8 bit address, 32 bit data ROM from memory library as your instruction memory to which you can load your hex code.
  - You can assemble your test programs using /home/cs154b/bin/mips2mif utility found on the csif machines. Now you only need the hex words from the output file. Manually enter the hex words in the ROM using the edit tool of ROM.
  - Since MIPS architecture is byte addressable so PC is incremented by 4 therefore, next instruction will be at address PC + 4. For example, in your ROM, first instruction will be at 0x00000000, 2nd at 0x00000004, 3rd at 0x00000008 and so on.
- RAM - Data Memory
  - You can simply use a 8 bit address, 32 bit data RAM from memory library as your data memory.
  - `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` : It is the MemWrite, A control signal that causes the memory to be written when set.
  - `ld` : It is the MemRead, A control signal that causes the memory to be read when set.
  - `sel` : Chip select, set it to 1.
  - `clr` : Not used in this lab. Can set to 0 if you want.
  - `^` : the main clock signal.

Your CPU must execute the following instructions:

- add, sub, addi, lw, sw, and, or, nor, andi, ori, beq, slt, j

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. You may optimize the functions to only work for the required instructions. You are free to use the Logisim Circuit Analyser to build it.

Grading and Submission

In order to facilitate the grading of your lab, "label" the following signals in your circuit so that the TA can identify any of the following signals in your assignment. (some of them are already present in the base file MIPS.circ)

- ALUA, ALUB: The ALU data inputs just before they enter the ALU.
- ALUCtl: The ALU Control signals from Page 316 of the textbook.
- ALUSrc: Control signal of mux at lower ALU input.
- Branch: Control signal indicating a branch.

- Zero: The zero output from the ALU.
- Jump: Control signal indicating a jump.
- MemRead, MemWrite: Data memory control signals.
- MemtoReg: Control signal for mux that selects data written to the register file.
- RegWriteData: The write data input to the register file.

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
- the boolean equations for your various control signals, along with intelligible descriptions of what your variable names are and which equations correspond to what signals in your CPU.

Submit a `tar` file containing your MIPS.circ and a README to the Lab 2 Assignment on SmartSite. Only one partner needs to actually submit the assignment. Just make sure the `tar` file contains the README file with your names so that I know who is working with whom. You can re-submit as many times as you would like. You may use up to two slip days per lab. I suggest very strongly that before submission you confirm that what you have in your `tar` file is actually a working MIPS single cycle CPU.

Hints

- 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. You will have to derive logic circuit from the truth tables given in chapter 4.4 of the textbook.
- Use the library utilities whenever you need more than a simple logic gate. Some blocks you may find useful in this lab include: `add_sub`,`mux`, `and`, and `or`.
- Use Bit extender block whenever you need to change 16 bit to 32 bit or 32 to 8 bit(because we are using 8 bit address space, you will require this when implementing the PC logic, or when you need to generate 8 bit RAM address)