CS281, Computer Systems

Hardware Lab 9: Y86 Control

1 Overview

The purpose of this laboratory is to design the control circuitry necessary to run the Y86 logisim computer.

This is an individual lab. You should complete this lab entirely on your own using help only from the course instructor or the course TA. You should not consult with other students or other external resources to complete this lab.

The Logisim circuits provided now consist of three files.

1. Y86Memory.circ – This file has not changed and defines the Instruction Memory and a handful of subcircuits exactly as provided in the last lab.
2. Y86.circ – This is the modified datapath with connections to the Y86Control circuit that you will build in this lab. In addition to placing the Y86Control circuit interface on the datapath, we have added the Stat register along with control signals for statWrite and statValue. Error signals from instruction and data memory have also been tunneled to provide input to the Y86Control circuit.
3. Y86Control.circ – This file currently only has one circuit defined in it, and is where you should be doing all of your work in this lab. The main circuit in this file is called Y86Control, just like its filename, and the circuit simply consists of the input pins and output pins so that the interface can be used in the parent circuit, Y86.circ.

***** IMPORTANT *****
You must open the Y86Control.circ file separately (first) and not as part of the Y86.circ file. Otherwise your work seems to be saved in the Y86.circ instead of the Y86Control. You will submit your Y86Control.circ file for this lab.

2 Logistics

You will modify only the `Y86Control.circ` file and submit only this file. Do not change any of the other files. Do not change (add, remove, alter the position of) any inputs or outputs to the Y86Control file. It must have exactly the same interface so that it plugs into the Y86 file correctly. You can verify this by using your module (reload circuit library from Y86.circ).

Your task is to add combinational logic for all the control outputs of the Y86 circuit. The inputs to the circuit include `icode`, `ifun`, `CC`, `Stat`, `imemError`, and `dmemError`. Your goal is to build a subcircuit

for each one or two control signals. These subcircuits (all defined in Y86Control.circ) should implement the correct functionality based on the current instruction and other inputs to Y86Control, but not all subcircuits will require *all* the inputs.

Use your DataPath Lab datasheet as the key piece of information for deciding how to set each of these control signals based on the icode and ifun.

You may design this control as you like. For instance, you may build function tables and use K-maps to devise control signals for the different control lines. Or you could build a bunch of multiplexor ("Mux Method") units. Or mix and match.

You are, however, required to use subcircuits, so the top level of Y86Control should be your collection of subcircuits along with appropriate wiring of inputs into the subcircuits, and wiring from the outputs of the subcircuits to the output pins of Y86Control.

3 Evaluation

Your grade for this assignment is determined by the following factors:

1. Correct operation. Your circuit should execute all the instruction types correctly. You should fully test the operation of your control logic by running example programs that execute the full range of Y86 instructions.

2. Testing. I will provide you a full program which tests most of the instructions. You are to write other programs to make sure your circuit works COMPLETELY for EVERY possible instruction.

3. Design. You should have a clean, simple design. Make good use of sub-blocks to keep the logic simple and reduce wiring clutter.

4. Mux or Minimal Logic: Most of these control signals can be completed using a "mux design" where the icode is used as the select line on a multiplexor and constants are fed into the mux inputs. This is a straight forward way to build most of the design logic. We will demonstrate the mux design process in class. An alternative is to design minimal circuit logic using function tables and K-maps. This can result in a faster/smaller circuit design (the mux looks simpler, but remember that it is built from a lot of pieces). If you use the easier mux-design approach, your maximum grade is B+ (88). If you use the better minimal K-map design process, you can earn up to the full 100.

4 Hints and Advice

1. Notice the outputs all have constants attached to them at the start. This way your circuit will not have colored-line errors when you first load it. You will want to remove these constants as you build each subcircuit.

2. Start by hardcoding `continue` to 1, and set up "sensible" defaults for most of the control signals. Then work a single control signal at a time (consistent with the rest of the defaults). I basically started with PCIncrSrc and then went left-to-right for each control signal. I saved the new things (condition codes and status register) for the end.

3. Build your set of test programs as you go, so that you don't have to do the same work twice.

4. I would begin with getting `irmovl` to work, since you need a mechanism to load registers before you can test much else.

5. Save the integrated program tests for *after* you have worked through the individual instruction tests.

6. Figure out how the condition codes are interpreted for jump and conditional move instructions. We skipped over this part in the DataPathLab assuming that a branch was always taken. Now you must reconcile with the condition codes to decide if the branch is taken or not. Some of the control lines will be different depending upon a taken or not taken jump instruction.

7. Be careful with the `rrmovl`, because you must provide for the general case of `cmovXX` that `rrmovl` is a specific instance of.

8. Look at the table below and figure out what the status register is used for.

9. Figure out how to "turn off" the clock with the status register and control logic.

10. Figure out the two new memErr lines. You will find that sometimes the dataMemErr line is activated with an instruction that does not use data memory. You should ignore the error on this line for such an instruction that does not read or write from/to data memory.

The Table below gives the meanings for each of the control signals on the Y86 datapath. For each, you should first determine the input(s) for determining the correct control signal value. Then design the control logic for each.

Control	Width	Description
continue	1	When 1, allows execution to continue since this control is ANDed with the clock before the clock continues to the rest of the datapath. So this value should be 1 as long as Stat is 00, and should be 0 otherwise.
statWrite	1	Determines whether or not the Stat register should be written.
statVal	2	Value to be stored in the Stat register when statWrite is 1. 00 means execution is OK, 01 means an invalid instruction was encountered, 10 means instruction memory received an invalid address, and 11 means data memory received an invalid address.
PCIncSrc	2	Determines value to add to PC to get to next instruction. 00 - 1, 01 - 2, 10 - 5, 11 - 6.
valCsrc	1	Determine value for valC. 0 means valC ←Dest (M4[PC+1]), 1 means valC ←V/D (M4[PC+2]).
valAsrc	1	Determine read register file output for valA. 0 means valA ←R[rA], 1 means valA ←R[%esp].
valBsrc	1	Determine read register file output for valB. 0 means valB ←R[rB], 1 means valB ←R[%esp].
dstEsrc	2	Determine destination register for write of valE. 00 means R[rB] ←valE, 01 means R[%esp] ←valE, 10 and 11 mean send 0xf to dstE, indicating no write.
dstMsrc	1	Determine destination register for write of valM. 0 means R[rA] ←valM, 1 means send 0xf to dstM, indicating no write.
aluAsrc	2	Determine value to send to ALU_A. 00 means ALU_A ←valA, 01 means ALU_A ←valC, 10 means ALU_A ←4, 11 means ALU_A ←-4.
aluBsrc	1	Determine value to send to ALU_B. 0 means ALU_B ←valB, 1 means ALU_B ←0.
setCC	1	Determine whether or not to update the CC register on the next clock. 0 indicates do not update, 1 indicates update.
aluOp	1	Determine operation to route to ALUfun. 0 means 0000 (add), 1 means use ifun.
dmemAddr	1	Determine address routed to data memory address line. 0 means dAddr ←valE. 1 means dAddr ←valA.
dmemData	1	Determine value routed to data memory D input. 0 means D ←valA, 1 means D ←valP
dmemWrite	1	Determine whether or not to store D at M4[dAddr] on the next clock. 0 indicates do not write memory, 1 indicates write memory.

newPC	2	Determine source of next Program Counter to be routed to input of PC register. 00 means newPC ←valP, 01 means newPC ←valC, 10 means newPC ←valM, and 11 is undefined.