cs281: Introduction to Computer Systems

CPUlab – Y86 Hardwired Control

Nov. 10

1 Specification

The objective of this lab is to build the hardwired Control Logic that drives automatic instruction execution on our SEQ datapath. Most of what you need is already in place, but we've updated and provided you with a new version of the Datapath. The Logisim circuits provided now consist of the following files.

1. Y86B.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.

2. Y86Memory.circ – This file has not changed and defines the Instruction Memory and a handful of subcircuits exactly as provided in the datapath assignment.

3. ALU.circ – This file has not changed and defines the top level ALU32 circuit.

4. alulibrary.circ – This file has the starter template as given in the last assignment. When you are ready to run/test OPl instructions, you should substitute your own alulibrary.circ in place of this one, to enable ALU operations other than addition.

5. Y86Control.circ – This file currently only has one circuit defined in it. 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, Y86B.circ.

6. controllibrary.circ – This file has subcircuit interfaces for all the combinational circuits you will build in this lab. I have created implementations of `continue` and `stat` to get you started. All the components defined in this `.circ` file are already wired into the given top level control circuit, `Y86Control`.

Your task is to add combinational logic for all the control outputs of the Y86 circuit to autmatically drive the control in place of the manual manipulation you performed in the last assignment. All the circuits you need to design/modify are in `controllibrary.circ`, and their interface is already given for you. All the control subcircuits use the 4 bits of `icode` as input. In addition, the `dstEsrc` and `newPC` circuits have the 1 bit `Cnd` as an input. There is also a subcircuit called `Cond` whose input is the 3-bit `CC` and the 4-bit `ifun` and whose purpose is to compute the single bit output of `Cnd`.

I am relaxing all restrictions on what kinds of devices you may employ in implementing these circuits. In particular, you can use a Decoder on icode if it aids your implementation. In general, you will use the control line outputs required per instruction, generated for the datapath assignment, as a specification for each of your circuits. You may also use the Logisim "Analyze Circuit" functionality if you wish ... but your interface to the Y86Control circuit cannot change as a result of using this tool.

Once you have your combinational circuits in place, your job is to thoroughly test the Y86 instruction set and your implementation of each instruction. Use yo2mem to generate memory images for our CPU and test instruction sequences. You have been given the images for example.yo and sum2.yo, but these are not sufficient to test well the entire instruction set.

Your lab report and submission, for this assignment, will be evaluated primarily on how well you convey how you have tested your CPU and how well you convince me that your CPU can execute the fullness of the Y86 instruction set – jumps taken and not taken, conditional moves taken and not taken, jumps forward and backward, etc. You should only need to send me the `controllibrary.circ` file, unless you needed to modify one of the "higher level" files, and, if so, you should provide justification.

The Table below gives the meanings for each of the control signals on the Y86 datapath, and have not changed since phase 1, other than the additions mentioned above. For each, you should first determine the input(s) for determining the correct control signal value. Then build a truth table for each, employing "Don't Care" values to enable minimal circuits.

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 an invalid address (instruction address *or* data address), and 11 means a halt instruction was encountered. Note that these differ slightly from the textbook values of Stat.
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].
aluAsrc	1	Determine value to send to ALU_A. 0 means ALU_A ←valB, 1 means ALU_A ←0.
aluBsrc	2	Determine value to send to ALU_B. 00 means ALU_B ←valA, 01 means ALU_B ←valC, 10 means ALU_B ←4, 11 means ALU_B ←-4.
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	2	Determine address routed to data memory address line. 00 means dAddr ←valE. 01 means dAddr ←valA. 10 and 11 mean to send address 0.
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.
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.
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.

2 Hints

1. After your first implementation of your controllibrary circuits, I would begin with getting `irmovl` to work, since your test programs will need a mechanism to load registers before you can test much else.

2. Be careful with `dmemError`. This signal is asserted by the data memory unit anytime `dmemAddr` has either too high an address or anytime the data address is not word aligned. If your answers for the control signals have `1X` for dmemAddr whenever the data memory unit is not used for an instruction, this will not be an issue. But if you specified `XX`, then the CPU may halt because of a dmemError on an instruction that is not using data memory.

3. Save the integrated program tests for *after* you have worked through the individual instruction tests. This is analagous to doing unit testing on methods of an object before moving to integrated tests.

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