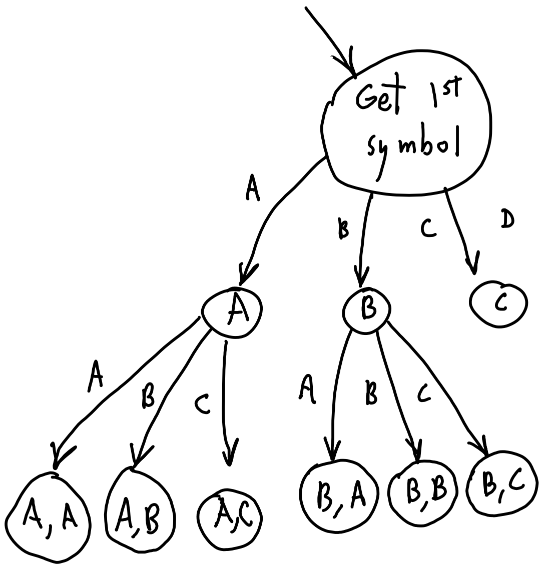
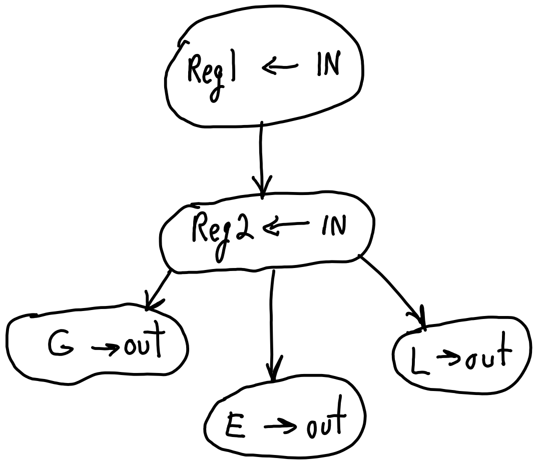
Lec-5-HW-4-regFsm

(Problem) -------------------------------
Suppose we have physical hardware devices that have three possible states, {HI, 0, LOW}. Suppose we want to record, i.e., register, an input symbol from the symbol set {A, B, C}. We start by getting the first symbol from input. Next, now that we know what symbol we have, we want to get another symbol. After we have both symbols, we next decide whether the letters we got from input are,
(1) both the same symbol, then go back to the start state;
(2) the first comes before the second in alphabetic order, then output "A";
(3) the second comes before the first, then output "C".

(a) Implement this as a Moore machine: draw the total state transition diagram. You can ignore output for states for which no output was indicated. You may also have transitions that do no input. How many total states are needed, assuming you used the minimum number of states. How many of our 3-state physical state devices would be required to physically implement the state for this machine?

(b) Implement this as a finite-state-machine controller with a data path consisting of two registers. That is, draw the state transition diagram for the controller, using RTL to indicate datapath register operations. Use "IN" for the source of input and "OUT" as the output. Label your controller's conditional branching with boolean expressions such as, for example, "Reg1 < Reg2" to mean the the symbol in Reg1 preceeds that in Reg2. How many total 3-state physical state devices would be required to implement this machine?





(Problem)---------------
Implement the even-odd parity machine described in lecture,
(a) as a Mealy machine (outputs on state transistion arcs)
(b) convert your Mealy machine to a Moore machine using the conversion shortcut mentioned in lecture. Draw a timing diagram to explain the timing relationships between inputs and and outputs.

(Problem)-----------------------
Suppose we have a TM, M, with a built-in even-parity machine as part of its FSM. Let's suppose its symbol set includes "e": whenever it sees "e" as input, it enters the start state of the parity machine. Suppose it also has a register that serves to record which state it should enter after exiting the parity machine. (We'll think of it in our controller/datapath model.) Using the controller/datapath model, and using RTL for controller states, show two different states, X and Y, of M that cause a transfer to the parity machine. Also show the return from that hardware sub-routine. You can suppose there are only two states it can return to, states A and B, A if it enters from state X or B if it enters from state Y. Show how M sets up the return, and how the return branching is done. You will need more states than just X, Y, A, and B.

(Problem)----------------------
In the previous problem, M was a TM (even though we described it in the controller/datapath model.) Certainly, we can describe M appropriately for some UTM and simulate it. Suppose we want to describe M, but we do not want to describe the parity machine. Instead, we will run a translator that will insert the parity machine into M's description for us. Briefly explain what mechanism the translator can use to,
(1) detect that the parity machine description should be inserted
(2) make the proper state connections so it will work correctly.

(Problem)------------------
Suppose we want a description language that works for TMs conceptualized in the controller/datapath model. In that model there is a controller FSM to describe, registers and RTL operations to describe, and functional data operations to describe. It was easy to devise a language for describing TMs in the Turing sense of description: it was just a rule table. Suggest a language approriate for composing descriptions in the controller/datapath model. In particular, suggest how to describe RTL operations and controller conditional branching.

(Problem)-------------------
Assume the previous problem defines a language, L, suitable for describing machines in the controller/datapath model, and that there exists a simulator, S, appropriate for that language. Using that language, could we describe a simulator, U, that reads descriptions written in another language, K? Could we use S to simulate U using its description in L? Suppose that were possible, what language would we use to write descriptions for U to simulate?

(Problem)----------------------
A flip-flop, FF, has the feature that its output is isolated from its input so that its output can only change once per clock tick. When the output changes, we say the FF's state changed. Early electronic computers used liquid (mercury) as storage elements: a ripple was sent along a horizontal tube partially filled with liquid. The ripple would bounce and return when it got the end of the tube. When it arrived back at the beginning, it could be detected as a stored value and regenerated. Is this device a FF? Explain. In particular, could it be used to implement controller state?

(Problem)--------------------
Consider two FSMs. The first, M1, sends its output to the second, M2, where it is used as input. M2's output likewise goes to M1. It there any problem with the timing of valid data in this arrangement? Illustrate with a timing diagram.

(Problem)-------------
Here is the rule table for a FSM, M (a Moore machine, ignore output):

current-state	input	next-state
A	0	B
A	1	C
A	2	A
B	0	C
B	1	A
B	2	C
C	0	B
C	1	C
C	2	B

(1) Pick a 1-hot state encoding.
(2) Show the next state function, F, as a table, using that encoding.
(2) Implement M: show state elements, indicate the next state function as a circle.
Note: The next-state function will actually consist of several functions that implement control branching. Indicate each as a circle with a description of its branching logic.