Notes for the upcoming exam on 11/20/98

Multiplexers
There will certainly be a question on multiplexers but you might see a bit of a twist. As you implement the function and push it back some of the reduced functions you end up with may be identical so you can save a multiplexer. Because of this you might be asked to solve the problem with a specific number of multiplexers. The result may differ depending on what variables you put into the different multiplexers. You will have to try all the possible combinations to see which comes out the least. To make things a bit easier the problem will be kept to a four variable function. One additional slight variation would be if you were given an extra gate such as an inverter and then asked to reduce the function farther. You’d then be looking for reduced functions that are complements of each other.

Decoder
Remember that you can be given either an AND (active high) or an OR (active low) decoder to work with. If given an AND decoder you will working with the function in a sum of minterms form (dealing with the 1’s of the function). If given an OR decoder you’ll want instead the product of maxterms form (handling the 0’s of the function).
You could be asked to build a function of many inputs from small decoders. For example you might need to implement a three input function using only two input multiplexers.

**ROM implementation**

You may be asked to implement some combinatorial circuit in a ROM. Recall that all you really have to do is figure out the truth table and then put it into the proper form. The problem is more one of understanding the word description, determining the inputs and outputs, and getting to the truth table.

**State diagram**

Your line of attack here should be to begin by specifying the number of states you need from the number of flip-flops in the circuit.

**Flip-flops**

There might be a problem involving flip-flops with more than two inputs. Note that you don’t really need to have the insides of the various flip-flops memorized. It’s more important that you understand how they work.

**Overview of Chapter 7**

So far we’ve been focusing on the ‘bit’ level of state machines. We are now going to move into larger components such as:

- Registers
- Shift Registers
- Counters
- Controllers
- Memory devices
These devices all contain a number of flip flops. In some cases they contain a very large number of them. Let’s try and get a feel for what some of these things are and where they might be used. A generic block diagram would contain some set of registers or memory elements and a computation element. That is:

![Block Diagram]

So what does this do? The computation element pulls in some set of the registers, performs some type of computation, and then stores the result back over in the registers again. Let’s refine this picture a little bit more. Imagine that we have a bank of registers as in the block diagram above. We’d like to select any two of them and perform one of a set of functions on these two registers (add, subtract, multiply, etc.) and then store the result into any one of the registers in our memory bank.
So let’s think for a bit- how would we select registers? Let’s see, selecting one from many- sounds like a multiplexer to me. What about storing it back. We have one input, the result, and many outputs, the registers. Sounds like a decoder/demultiplexer. Let’s look at this block diagram:
Recall for a minute the second extra-credit problem that was assigned. That was an example of a simple ALU. The function select lines determine what function will be performed on the register inputs to the ALU. The select inputs to the muxes choose which registers are those inputs while the select lines on the decoder/demultiplexer determine which register the result gets stored in.

For example if we wanted to do $R0 = R1 + R2$ we would select R1 on the top mux, R2 on the bottom mux, the ADD function for the ALU, and R0 on the decoder/demultiplexer.

The four items we need to select make up the machine instruction:
This is obviously a very different world that the one of simply flip-flops that we’ve been working in. However every single item is familiar to you.
One thing lacking in this block diagram is a control unit that issues the commands given in the machine instruction words in the correct sequence to the various devices such that the operation is performed.
Your program resides somewhere in memory as a string of instruction codes. The control unit goes out to that memory, grabs an instruction, implements it, goes back and grabs the next one, and so on.
As you might expect there is more to this picture. We haven’t discussed items such as the output flags from the datapath or how to perform branching instructions. Still this should hopefully pique your interest in the material.