CIS 240: **MICROCOMPUTER ARCHITECTURE & PROGRAMMING**

HW3: KMaps (I thought these problems were already posted so due 2/17). Maybe consider this midterm review?

1. KMaps: Find the minimal logic (biggest circles).

a.

Inputs			Output
A	B	C	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

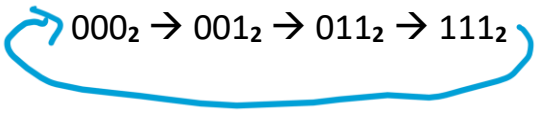
b.

	Inputs			Output
A	B	C	D	Out
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

c. Use KMaps to find minimal logic for the truth table for the thermometer code counter from last week's homework.

Here's the table for the thermometer code question from last week.

Make a truth table that converts 3-bit thermometer code to binary. Thermometer code counts up:



In thermometer code, $000_2=0_{10}$, $001_2=1_{10}$, $011_2=2_{10}$, and $111_2=3_{10}$. HINT: In the truth table, any row not used can have an output of X (for don't care).

2. Fill in the blanks (Midterm review part 1)

A computer is a digital system. Digital systems are made of circuits created by putting together _____s(1). If you split open a (1), you would see _____s(2). (2)s are the thing that the (1)s are actually made from. In order to make a circuit from (1)s, you start by _____ing(3) the problem. Hardware is very unforgiving and if you don't figure out all the details at the beginning, before you start to build, you may have to start over when, later in the design process, you discover you missed something. Once you (3) the problem, you fill in a _____ (4). A (4) is used to design the circuit but also completely describes the behavior of the circuit. They are good to include in documentation because they are a great guide to anyone that may want to use your circuit and wants to know exactly what the circuit does in a single table. The size of the (4) is determined by the number of _____s(5) to your circuit and the (4) will have _____(6 *an equation*) rows. In our class, we used SOP (Sum Of Products) to create _____(7) directly from the (4). This method, officially, is called the "canonical form" and usually makes the least efficient circuits in terms of _____(8), _____(9), and _____(10). It has the most terms and each term contains one of every (5). Because canonical form can be wasteful, you can reduce the size of your (7) by using _____s(11). (11)s use the _____(12) theorem to get rid of terms and (5)s in your (7) equation.

Numbers inside a computer are made up of one or more _____s(13 *a digit*) and _____s(14 *a digit*). In hardware, they are usually expressed as a low voltage and

a high voltage. Because there just _____(15) values a digit can have, the number representation is a base-_____(16) number system. Base (16) number representations are also called _____(17) number systems. When you consider all (17) numbers positive, the number representation is called _____(18). There are multiple ways to express negative numbers but the one we used (and that most every computer out there uses) is called _____(19). If a number is positive, the number in (19) is **EXACTLY** the same as in (18). The range of values you can express in (18) and (19) aren't the same. If you have a 4-bit word system in (18), the numbers range from _____ (20) to _____(21). If you express that as an equation with the number of bits expressed as "N", the range would be _____(22) to _____(23). But, if you have a 4-bit word system in (19), the numbers range from _____ (24) to _____(25). If you express that as an equation with the number of bits expressed as "N", the range would be _____(26) to _____(27). If a number is negative, it **can / can't** (28 *Select one answer*) be expressed in (18).

When you express a number in (17), there can be A LOT of digits (bits). Humans have trouble memorizing a lot of digits so expressing a number in base-16 or using _____(29) numbers can reduce the number of digits in your number. For example, if you have a base-(16) number like 1011010110.11001_2 you first group bits into groups of _____(30) going left and right starting at the _____(31) point. If you go to the left from the (31) point, and you end up with less than (30) bits, if your number is positive, you add _____s(32) to the _____(33) of the bits you have. Adding (32)s to the (33) of your bits doesn't change the value of your number but allows you to get a group of (30) bits and getting a group of (30) bits allows you to convert the bits to a (29) value. If you go to the right from the (31) point, and you end up with less than (30) bits, if your number is positive, you add _____s(32b) to the _____(33b) of the bits you have. Adding (32b)s to the (33b) of your bits doesn't change the value of your number but allows you to get a group of (30) bits and getting a group of (30) bits allows you to convert the bits to a (29) value. If you go to the left from the (31) point, and you end up with less than (30) bits, if your number is negative, you add _____s(32b) to the _____(33b) of the bits you have. Adding (32b)s to the (33b) of your bits doesn't change the value of your number but allows you to get

a group of (30) bits and getting a group of (30) bits allows you to convert the bits to a (29) value. If you go to the right from the (31) point, and you end up with less than (30) bits, if your number is negative, you add _____s(32c) to the _____(33c) of the bits you have. Adding (32c)s to the (33c) of your bits doesn't change the value of your number but allows you to get a group of (30) bits and getting a group of (30) bits allows you to convert the bits to a (29) value. Base-16 counts up the same as base-10 (decimal) until you get to the value _____(34). When you get to (34) there is no single digit we have that equals it so we call it _____(35). The next 5 digits after (34) we call __, __, __, __, and __(36) in base-16.

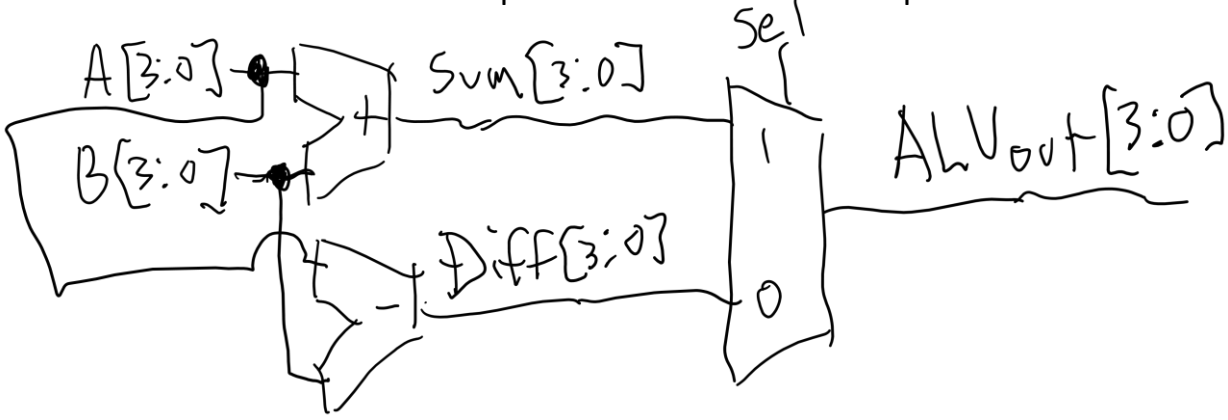
Once we saw how data is represented inside a computer, we went on to see how to add and subtract numbers. In class, you built a ripple carry adder. That means the carry is passed from column to column from the LAB to the MSB. In a computer, subtraction, for example, A-B is done by adding the _____(37) of B to A. When addition or subtraction is done, A and B come from a block called a _____(38). The (38) can supply the inputs to the ALU (adder and subtractor) quickly and any data just used in a calculation, presently being used in a calculation, or just used in a calculation will be in the (38). When data hasn't been used in a calculation recently, it is stored in the _____(39) memory. The (39) memory is slow so the whole computer would be slowed down if every calculation had to get its values from the (39) memory and not the (38). The way the ALU knows what to do is by looking at an _____(40). The (40)s reside in the _____(41) memory. The PC keeps track of which (39) is presently being executed. In more complicated computers, there will be a special block that looks at the (40) and generates all the signals to the blocks to tell them what to do. This block is called the "controller". In our system, bits from the (40) were directly taken from the (40) and used to tell blocks what to do.

I may have forgotten a topic or two but if you can fill in these blanks you are in pretty good shape for the midterm.

3. Build a FSM (finite state machine) that counts:
   000 → 001 →111 → 100

If you'd like more practice with LTSpice: (Not due)

4. If you can't use LTSpice at home, please tell me. You can use LTSpice on a Mac but it's harder (a lot of shortcuts to remember: https://www.analog.com/media/en/news-marketing-collateral/solutions-bulletins-brochures/ltspiceshortcutsformacosx.pdf – This list is old…. I don't have a Mac to test them… Here's another guide that's newer: https://www.woolseyworkshop.com/wp-content/uploads/WoolseyWorkshop_Cheatsheet_LTspiceForMac_v1.0.pdf)
   a. Create a subtractor that uses the fact that A-B = A+(-B). This design does not require you to use truth tables. HINT: You did the hardest part when you built your adder in class. If this problem seems complicated, please ask me or a classmate for hints.
   b. Create a symbol for your subtractor.
   c. Put the adder and the subtractor into a single schematic and put a MUX in (I created two versions of a 4-input MUX for you) so you can select between the adder output and the subtractor output:



   d. Create a symbol for your ALU. I know this ALU's abilities are pretty limited but this will show you how computer's work in general.
   e. Test your ALU.

f. If you do this, you might as well add it to your report!Again, not part of the homework. No need to turn it in with your homework. A reminder: If you get strange error messages or the simulation isn't what you expect, send me all of your *.asc files, all of your *.asy files, and the mode2025 file and tell me what you are seeing that bothers you. Don't fight for hours with it if you are getting something strange or it's a tool issue.