Lab 5 written

Question 1

3.6

The maximum value for an unsigned 8-bit integer is 255, and both 185 and 122 are below this, which makes them valid values. Calculating the final value of $185 - 122$ in decimal, we get 63, which is also an allowable 8-bit value. There is neither overflow nor underflow.

3.20

The bit pattern $0\text{x}0\text{C}000000$ converts to 201326592 in decimal, if it were either an unsigned or 2s complement value. This is because the sign bit is not used here as the value is positive, therefore it will be the same in both forms.

3.21

The bit pattern $0\text{x}0\text{C}000000$ converts to $0000\ 1100\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$. The 6 largest bits are our opcode, which in our case is equal to 000011, which corresponds to the hex value $0\text{x}3$, which itself corresponds to `jal`, a jump and link instruction.

3.22

The bit pattern $0\text{x}0\text{C}000000$ converts to $0000\ 1100\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$. Following the IEEE-754 standard, the below table is generated.

Sign	Fraction	Exponent
0	0...	00011000

To calculate this value in decimal, we can use the following formula:

$$val = (-1)^s * (1 + \text{fraction}) * 2^{\text{exponent} - \text{bias}}$$

This is a single precision value, therefore our bias is equal to 127. Our exponent value in decimal is equal to 24. Substituting our values, we get

$$val = (-1)^0 * (1 + 0) * 2^{24-127} = 1 * 1 * 2^{-103} = 2^{-103}$$

Question 2

a.

111111.01_2

1. To find the significand, shift bits to the right until there is only 1 digit of value 1 to the left of the decimal point
   $\text{significand} = 1.1111101$
2. The exponent is equal to the bias (127 because this is single-precision) + the number of bits shifted to the right (5)
   $\text{exponent} = 0\text{x}7\text{f} + 0\text{x}5 = 0\text{x}84$
3. Determine the sign by looking whether or not this number is negative
   $\text{sign} = 0$
4. Because the first bit is implied, we encode the fraction without it

Sign	Fraction	Exponent
0	1111101...0	010000100

b.

$$1000111110001010 11.01101_2$$

1. To find the significand, shift bits to the right until there is only 1 digit of value 1 to the left of the decimal point
   $1.00011111000101011 01101$
2. The exponent is equal to the bias (127 because this is single-precision) + the number of bits shifted to the right (17)
   $\text{exponent} = 0\text{x}7\text{f} + 0\text{x}11 = 0\text{x}90$
3. Determine the sign by looking whether or not this number is negative
   $\text{sign} = 0$
4. Because the first bit is implied, we encode the fraction without it

Sign	Fraction	Exponent
0	00001111100010101101101	10010000

Question 4

RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
0	1	0	1	0	0	0	0	1

No new control signals will be needed. `RegDst` will be set to 0, as this is an immediate instruction that will write data to a register, and we do not need to use the read register 2 bus on the registers. `ALUSrc` is set to 1 as we want to pass the immediate value that has been sign extended to the ALU. `MemtoReg` is set to 0 because we want the ALU result to go to the write data bus on the registers. `RegWrite` is set to 1 because we are writing the ALU result to a register. `MemRead` and `MemWrite` are set to 0 because this operation does not require the use of memory. `Branch` is set to 0 as we are not branching with this instruction. `ALUOp1` and `ALUOp0` are set to 0 and 1, respectively because we

want to perform a subtraction operation, but we do not have have a funct value due to this being an I-type instruction. Because of this, we mooch off of the subtraction operation performed for a `beq` instruction, which gives us the 2 values

Question 5

RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
X	0	X	0	0	0	1	0	1

In order to implement a `bne` instruction, we can piggyback off of the `beq` instruction and add a new control signal that flips the bit of the selecting signal going into the multiplexer (the one that determines the value that goes into the PC) if it is a `bne` instruction rather than an `beq` instruction. We will do this by adding an XOR gate in between the zero line of the ALU and the AND gate that compares `branch` and the zero line. The XOR gate can be implemented with NAND gates as detailed in the below image. The A line of the XOR gate will be connected to the zero line of the ALU and the B line will be connected to our new control signal, `ALUZero`. The out line will connect to the AND gate that the zero line of ALU was previously connected to.