1.

a. 34 = **00100010**
b. -34 → flip all bits and add 1 → **11011110**
c. -128 = **10000000**
d. 128 —> cannot be represented using 8 bit 2s complement
e. 0 = **00000000**

2.

a. 01001010
+ 01100000
10101010 → 170

b. 54 = 00110110 → -54 = 11001010
00010011
+ 11001010
11011101 → -35

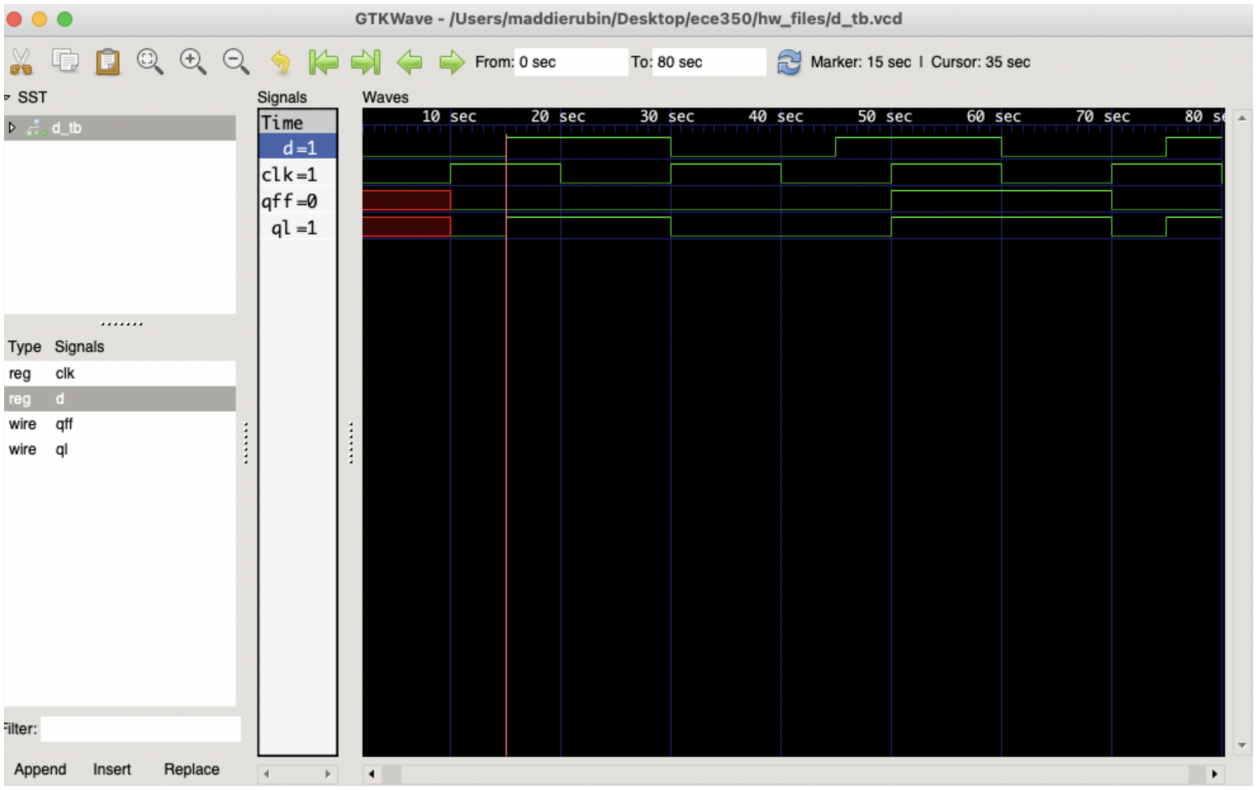
c. -55 = -54 -1 → 11001001
73 = 01001001→ -73 = 10110111
11001001
+ 10110111
1000001 →-127

d. 00110111
+ 01001001
01111111 →127

e. 00100000
+ 00101100
01001100 → 76

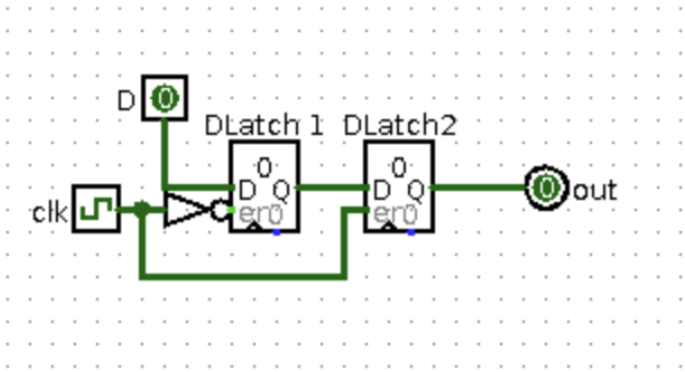
3.

a. In D-latches, q gets the value of d whenever the clock is high, meaning it is level sensitive. However, with D-flip-flops, it will capture and retain d when the clock is on its positive edge, meaning that it will ignore any changes to d when the clock stays high, until the next clock cycle. This difference is shown below on gtkwave:



As shown around time 15, the D-latch (represented by ql, will take the value of d as long as the clock is high. However, qff only takes the value of d when the clock is on its rising edge, meaning that it ignores the changes in d that occur when the clock is still high. This is shown at time 20, when the clock is low when d is high, so ql goes to 1, while qff remains at 0.

b. To build a DFF out of 2 D-latches, we need one D-latch to be clocked to store on the negative clock, where Q will get the value of D whenever the clock is not positive, and then another D-latch that will get the value of the first D-latch only when the clock is positive, thus allowing the second D-latch to only change when the Q of the previous latch has changed, and the clock has hit it's rising edge. This is shown below in logisim:



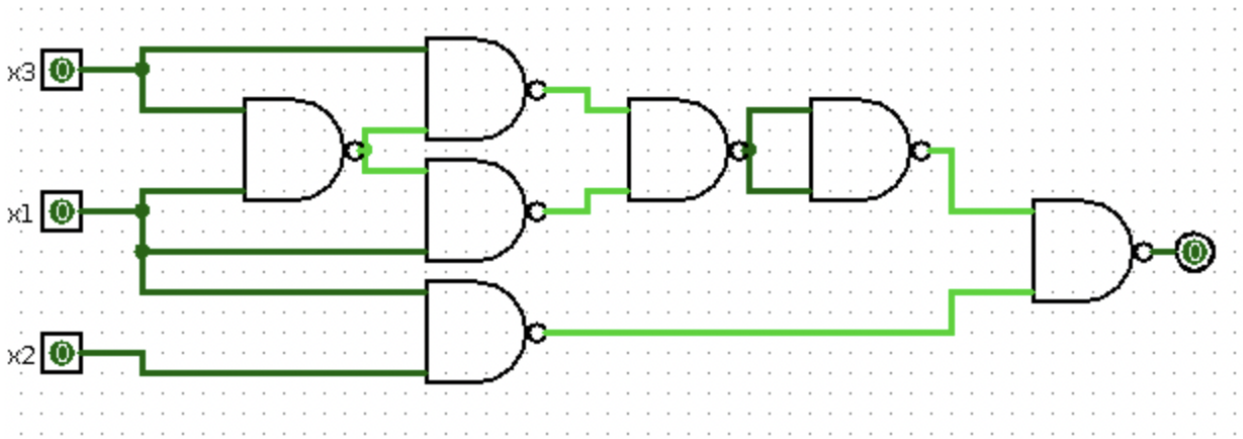
4. SUM(m(1,3,4,6,7)) = x1'x2'x3 + x1'x2x3 + x1x2'x3' + x1x2x3' + x1x2x3
   x1'x3 + x1x3' + x1x2

→ x1 xor x3 in nand gates: (!(x1x3) and x1) OR (!(x1x3) and x3)
→ convert OR and into NAND by complementing the outputs of both sides of OR and replacing OR with NAND

(((x1 NAND x3) and x1) or ((x1 NAND x3) and x3)) or (x1 and x2)
Get rid of ors by adding NANDS and nots.
Not gates are the same as a NAND gate where the two inputs are the same:

!(((x1 NAND x3) NAND x1) NAND ((x1 NAND x3) NAND x3)) NAND (x1 NAND x2)

Shown below in logisim:



5.
   11101111 * 11110110
   Negative multiplicand: 00010001

Product Before Shift	Product After Shift
00000000 1111011**0 0**	00000000 0111101**1 0**
00010001 0111101**1 0**	00001000 1011110**1 1**

00001000 10111101 1	00000100 01011110 1
11110011 01011110 1	11111001 10101111 0
00001010 10101111 0	00000101 01010111 1
00000101 01010111 1	00000010 10101011 1
00000010 10101011 1	00000001 01010101 1
00000001 01010101 1	00000000 10101010 1
Final Product:	0000000010101010

6. 11101111 * 11110110
   Negative multiplier: 00001010

Product Before Shift	Product After Shift
00000000 00001010	00000000 00000101
11101111 00000101	11110111 10000010
11110111 10000010	11111011 11000001
11101010 11000001	11110101 01100000
11110101 01100000	11111010 10110000
11111010 10110000	11111101 01011000
11111101 01011000	11111110 10101100
11111110 10101100	11111111 01010110
Final Product:	(negate) 0000000010101010 = 170