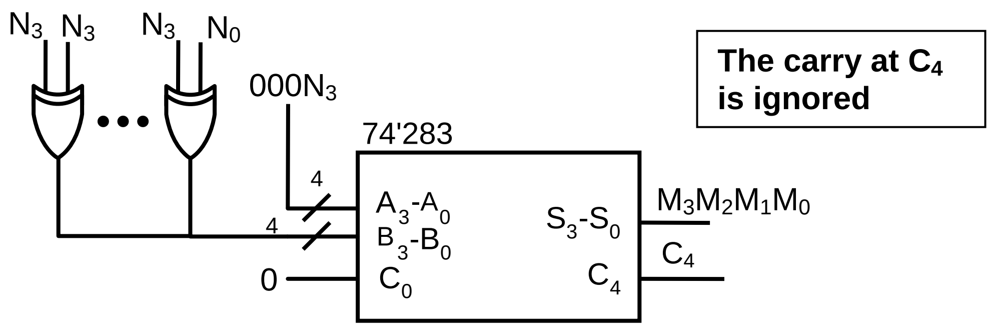
EE2020 Tutorial 5 - Solutions

1.
We are required to find the magnitude, *M*, of a 4-bit number, *N*, expressed in 2's complement notation. A 4-bit parallel adder (74'283) is used.
In a circuit implementation, the magnitude of a 2's complement is found by inverting all the bits of a number and adding 1 to it if the number is negative. We can equivalently write *M* = *N* if *N* is positive. Else *M* = complement(*N*) + 1 if *N* is negative (i.e., *B*=*N* if N_3=0, *B*=NOT(*N*) if N_3=1; both conditions are achieved by summing 000N_3).



2. A 3-to-8 decoder (74'183) is here used. NOT(E_1), NOT(E_2) and NOT(E_3) are the active-high enable signals.

$$A_2 = A,\ A_1 = B,\ A_0 = \bar{C},\ E_3 = H,\ E_2 = L,\ E_1 = D$$

Denote Z_1, Z_3, Z_4, Z_7 to be the inputs to NAND gate. Then,

$$\bar{Z}_1 = \bar{A}_2\ \bar{A}_1\ A_0 \cdot E_3\ \bar{E}_2 \bar{E}_1 = \bar{A}\bar{B}\bar{C} \cdot \bar{D}$$
$$\bar{Z}_3 = \bar{A}_2\ A_1 A_0 \cdot E_3\ \bar{E}_2 \bar{E}_1 = \bar{A} B \bar{C} \cdot \bar{D}$$
$$\bar{Z}_4 = A_2\ \bar{A}_1 \bar{A}_0 \cdot E_3\ \bar{E}_2 \bar{E}_1 = A \bar{B} C \cdot \bar{D}$$
$$\bar{Z}_7 = A_2\ A_1 A_0 \cdot E_3\ \bar{E}_2 \bar{E}_1 = A B \bar{C} \cdot \bar{D}$$
$$X.H = \overline{Z_1 . Z_3 . Z_4 . Z_7}$$
$$X.L = Z_1 . Z_3 . Z_4 . Z_7$$
$$= \overline{\bar{A}\bar{B}\bar{C}\bar{D}}.\overline{\bar{A}B\bar{C}\bar{D}}.\overline{A\bar{B}C\bar{D}}.\overline{AB\bar{C}\bar{D}}.$$
$$= \overline{\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + A\bar{B}C\bar{D} + AB\bar{C}\bar{D}}$$

The problem can also be done using viewing NAND as an OR with complimented inputs. That will do a bubble-to-bubble cancellation, and we can easily obtain X as

$$X.H = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + A\bar{B}C\bar{D} + AB\bar{C}\bar{D}$$
$$X.L = \overline{\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + A\bar{B}C\bar{D} + AB\bar{C}\bar{D}}$$

3. a) Table I

S_3	S_2	S_1	S_0	Value (decimal representation of 2's complement)
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

b)

Table II: Modifications to comparator output for signed comparison

Sign Bits		A > B Output	A = B Output	A< B Output
A_3	B_3			
0	0	No change	No change	No change
0	1	Invert output	No change	Invert output
1	0	Invert output	No change	Invert output
1	1	No change	No change	No change

OUTPUT(A=B)
output(A=B) always gives correct result

OUTPUT(A>B), OUTPUT(A<B)
0 0: + + => all outputs need to be kept as they are

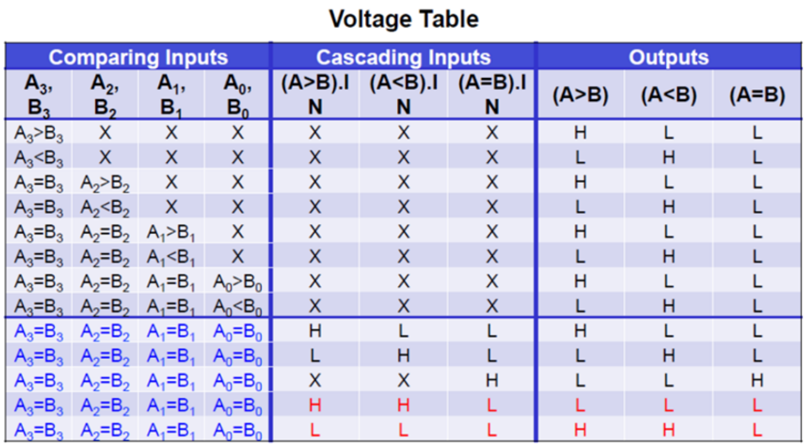
0 1: + - => output(A>B) should be 1, but in the above table it would be 0 (since neg. no. has always binary representation greater than pos. no.)
=> invert output(A>B) and output(A<B)

0 1: - + => output(A>B) should be 0, but in the above table it would be 1 (since a negative number always has binary representation greater than pos. no.)
=> invert output(A>B) and output(A<B)

1 1: - - => since 2's compl representation of a negative number is higher for larger (less negative) number
=> keep output(A>B) and output(A<B) as they are

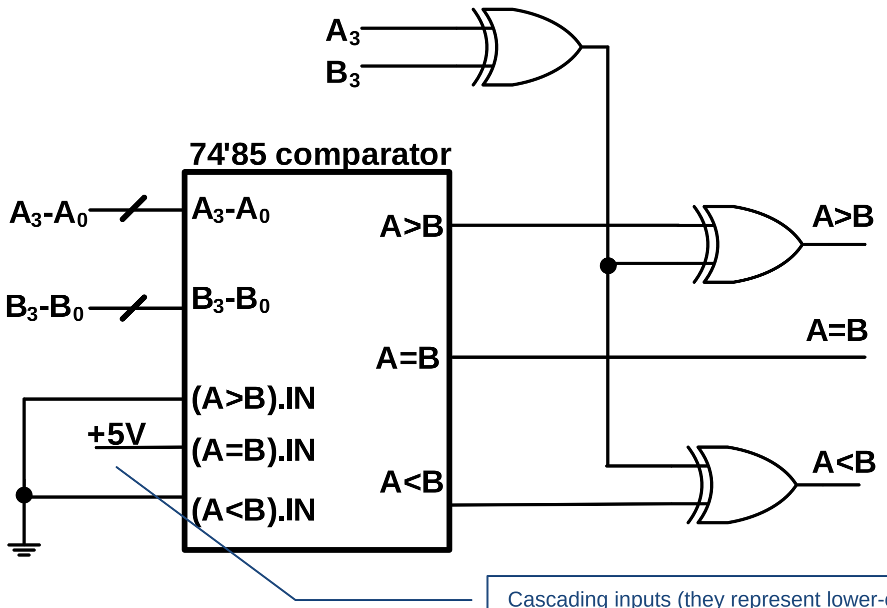
After we write down Table I, it is clear that if A and B are of the same sign, then the magnitude comparator will give the correct outputs. However if A and B are of opposite sign, then the magnitude comparator outputs will be incorrect. It is easy to see from Table II that if A and B are of opposite sign we need to invert the A>B and A<B outputs of the magnitude comparator. An EX-OR gate can be used to detect whether A and B are of opposite sign. The output of the EX-OR gate can be used to invert the comparator outputs as appropriate.

Voltage table of 74’85



Voltage Table

Comparing Inputs				Cascading Inputs			Outputs		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	(A>B).IN	(A<B).IN	(A=B).IN	(A>B)	(A<B)	(A=B)
$A_3>B_3$	X	X	X	X	X	X	H	L	L
$A_3<B_3$	X	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2>B_2$	X	X	X	X	X	H	L	L
$A_3=B_3$	$A_2<B_2$	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1>B_1$	X	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1<B_1$	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0>B_0$	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0<B_0$	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	L	L	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	H	L	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	X	X	H	L	L	H
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	H	L	L	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	L	L	H	H	L



Cascading inputs (they represent lower-order bits) are configured so that they do not affect any output.
Since they affect output only if inputs are equal, we want to generate output as if they are equal => (A>B).IN = 0, (A=B).IN = 1, (A<B).IN = 0,

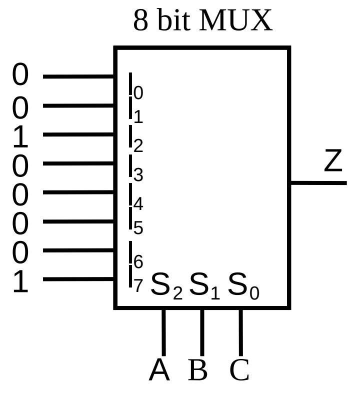
4.

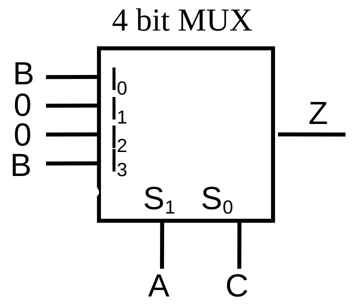
(a)

Let (A, B, C) be the 3-bit input. Then with connections as shown, Z = B when A=C else Z = 0.

(b)

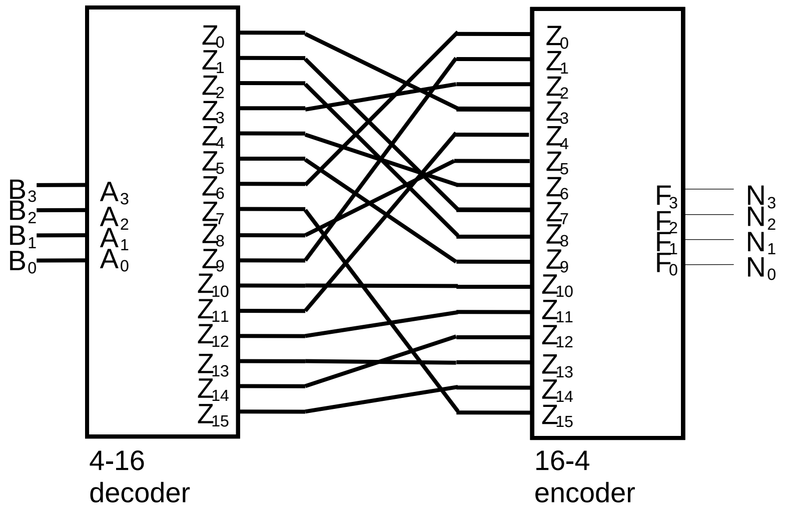
A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1





If we connect A and C to S_1 and S_0, respectively, in a 4-bit multiplexer then it is obvious from the problem statement (or the truth table) that if the connection to I_0-I_3 are as shown, the function can indeed be implemented with a 4-bit multiplexer.

5.



Decoder outputs the code number corresponding to the code that was present at its input. When these are appropriately connected to encoder inputs (as specified by table), the decoder outputs the corresponding N-code.