



## ECS404U: COMPUTER SYSTEMS & NETWORKS

2020/21 – Semester 1

Prof. Edmund Robinson, Dr. Arman Khouzani

-----

### ***Lab 4: More on Two's Complement, Introduction to Transistors and Logical Gates***

October 12, 2020

-----

Deadline for submitting your proof of work: **Next week's Thursday, at 10:00 AM**

Student Name:

Student ID:

*Brief Feedback to student (commendations, areas for improvement):*

# 1 Digital Representation: Two's complement

1. Learning objective: understand key features of two's complement representation using the example of 8-bit two's complement.

Compare eight-bit unsigned integers with eight-bit two's complement:

	Unsigned Integers	Two's Complement
How many numbers can be represented?		
Largest number bit pattern?		
Largest number?		
Smallest number bit pattern?		
Smallest number?		

2. Learning objective: understand two's complement representation and able to translate positive or negative numbers given in decimal into it.

This question asks you to translate numbers from decimal into 8-bit two's complement. Solutions are given at the end of the lab sheet as we will be using these in further examples. It repeats and reinforces work from week 3.

**Example: -3**

-3 is negative so the first bit is 1. The remaining bits represent 125, since this is  $(-3) - (-128) = 128 - 3$ .

powers	-128	64	32	16	8	4	2	1
diff	125	61	29	13	5	1		0
digits	1	1	1	1	1	1	0	1

-3 is represented by: 11111101

**Example: 14**

14 is positive so the first bit is 0. The remaining bits represent 14.

powers	-128	64	32	16	8	4	2	1
diff					6	2	0	
digits	0	0	0	0	1	1	1	0

14 is represented by: 00001110

(a) 17

17 is , so the first bit is . The remaining bits represent .

powers	-128	64	32	16	8	4	2	1
diff								
digits								

17 is represented by:

(b)  $-6$

$-6$  is \_\_\_\_\_, so the first bit is \_\_\_\_\_. The remaining bits represent \_\_\_\_\_.

powers	-128	64	32	16	8	4	2	1
diff								
digits								

$-6$  is represented by:

(c) 126

126 is \_\_\_\_\_, so the first bit is \_\_\_\_\_. The remaining bits represent \_\_\_\_\_.

powers	-128	64	32	16	8	4	2	1
diff								
digits								

126 is represented by:

(d)  $-126$

$-126$  is \_\_\_\_\_, so the first bit is \_\_\_\_\_. The remaining bits represent \_\_\_\_\_.

powers	-128	64	32	16	8	4	2	1
diff								
digits								

$-126$  is represented by:

3. Learning objectives: to practice binary addition and to see examples of how ordinary (unsigned) binary addition works for both positive and negative numbers when we use two's complement representation.

(a)  $14 + 17 = 31$

From the previous question you should have that the 8-bit two's complement representation of 14 is 00001110 and the 8-bit two's complement representation of 17 is 00010001.

Carry out the binary addition of these two bit sequences:

$$\begin{array}{r}
 0000\ 1110 \\
 0001\ 0001\ + \\
 \hline
 \text{carries} \\
 \hline
 \text{solution}
 \end{array}$$

This should be a particularly simple example. Verify that your answer represents 31 in both unsigned representation and 8-bit two's complement.

(b)  $-126 + 17 = -109$ .

From the previous question you should have that the 8-bit two's complement representation of  $-126$  is 10000010 and the 8-bit two's complement representation of 17 is 00010001

Carry out the binary addition of these two bit sequences:

$$\begin{array}{r}
 1000\ 0010 \\
 0001\ 0001\ + \\
 \hline
 \text{carries} \\
 \hline
 \text{solution}
 \end{array}$$

- i. (**Unsigned interpretation**) Check that if we interpret the bit sequences as unsigned integers then 10000010 represents 130, and that the bit sequence you have just calculated as the sum correctly represents  $130 + 17 = 147$ .

- ii. (**Two's complement interpretation**) Check that the bit sequence you have just calculated as the sum correctly represents  $-126 + 17 = -109$ .

4. Learning objectives: as for previous question, but in more complex examples: to practice binary addition and to see examples of how ordinary (unsigned) binary addition works for both positive and negative numbers when we use two's complement representation.

(a)  $126 + (-3) = 123$

From before you should have that the 8-bit two's complement representation of 126 is 01111110 and the 8-bit two's complement representation of  $-3$  is 11111101.

Carry out the binary addition of these two bit sequences:

$$\begin{array}{r}
 0111\ 1110 \\
 1111\ 1101\ + \\
 \hline
 \text{carries} \\
 \hline
 \text{solution}
 \end{array}$$

- i. (**Unsigned interpretation**) If we interpret the bit sequences as unsigned integers then 0111 1110 represents 126. Check that 1111 1101 represents 253. The bit sequence you have just calculated as the sum should have nine bits. Check that it represents  $126 + 253 = 379$ .

- ii. (**Two's complement interpretation**) The sum you have just calculated is 1 nnnn nnnn. Ignore the initial 1 (this is because in an 8-bit system, there is literally no space to fit the leftmost bit, and it is all right, because, in fact, ignoring that and keeping only the rightmost 8 bits is what makes 8-bit two's complement work in the first place!). Check that the last eight bits: nnnn nnnn are the 8-bit two's complement representation for  $126 + (-3) = 123$ .

Note that  $123 + 256 = 379$ , the unsigned sum.

(b)  $(-6) + (-3) = -9$

From before you should have that the 8-bit two's complement representation of  $-6$  is 1111 1010 and the 8-bit two's complement representation of  $-3$  is 1111 1101.

Carry out the binary addition of these two bit sequences:

$$\begin{array}{r}
 1111\ 1010 \\
 1111\ 1101\ + \\
 \hline
 \text{carries} \\
 \hline
 \text{solution}
 \end{array}$$

- i. **(Unsigned interpretation)** If we interpret the bit sequences as unsigned integers, check that then 11111010 represents 250 and that 11111101 represents 253. The bit sequence you have just calculated as the sum should have nine bits. Check that it represents  $250 + 253 = 503$ .

- ii. **(Two's complement interpretation)** The sum you have just calculated is 1 nnnn nnnn. Ignore the initial 1. Check that the rightmost eight bits: nnnn nnnn are the 8-bit two's complement representation for  $(-6) + (-3) = -9$ .  
Note that  $-9 + 512 = 503$ , the unsigned sum.

(c)  $17 + 126 = ??$

This is an example of something apparently going wrong.  $17 + 126 = 143$ . Notice that 143 is larger than 127, and therefore cannot be represented in 8-bit two's complement. That means something strange *must* happen.

The 8-bit two's complement representation of 17 is 00010001 and the 8-bit two's complement representation of 126 is 01111110. Carry out the binary addition of these two bit sequences:

$$\begin{array}{r}
 0001\ 0001 \\
 0111\ 1110\ + \\
 \hline
 \text{carries} \\
 \hline
 \text{solution}
 \end{array}$$

- i. **(Unsigned interpretation)** The result you have just obtained should have eight bits. Check that it represents  $17 + 126 = 143$  as an unsigned integer.

- ii. **(Two's complement interpretation)** The result you have just calculated should have a leading 1. It therefore represents a negative number, and so we appear to have just added two

positive numbers to get a negative! This is an example of what is called **integer overflow** (or **wraparound**) error. Computers sometimes signal that this is an error and sometimes don't. There is a small example on QMPlus to show that C can actually behave like this. As an experiment, find out what Java does.

Also, if interested, read more about “integer overflow” on its wikipedia page:

[https://en.wikipedia.org/wiki/Integer\\_overflow](https://en.wikipedia.org/wiki/Integer_overflow). Integer overflows, if not taken care of, can seriously compromise reliability, or even security of your programmes (e.g. take a look here:

<https://cwe.mitre.org/data/definitions/190.html> and

[http://projects.webappsec.org/w/page/13246946/Integer Overflows](http://projects.webappsec.org/w/page/13246946/Integer%20Overflows) )

Check that the number you have just obtained represents  $-113$  in two's complement.

Note that  $-113 + 256 = 143$ , the unsigned sum.

(d)  $(-126) + (-3) = ??$

Note that  $(-126) + (-3) = -129$  and that  $-129$  is less than  $-128$ , so cannot be represented in 8-bit two's complement. Again something strange *must* happen when we try to do this addition in 8 bits.

From before you should have that the 8-bit two's complement representation of  $-126$  is 10000010 and the 8-bit two's complement representation of  $-3$  is 11111101. Carry out the binary addition of these two bit sequences:

$$\begin{array}{r}
 1000\ 0010 \\
 1111\ 1101\ + \\
 \hline
 \text{carries} \\
 \hline
 \text{solution}
 \end{array}$$

i. (**Unsigned interpretation**) If we interpret the bit sequences as unsigned integers, check that then 1000 0010 represents 130 and that 1111 1101 represents 253. The bit sequence you have just calculated as the sum should have nine bits. Check that it represents  $130 + 253 = 383$ .

ii. (**Two's complement interpretation**) The sum you have just calculated is 1 nnnn nnnn. Ignore the initial 1: as before, in 8-bit two's complement, we will only look at the last eight bits: nnnn nnnn. The eight bits should have a leading 0. They therefore represent a positive number, and so we appear to have just added two negative numbers to get a positive number! This is another example of “integer overflow” error. Once again, computers sometimes signal that this is an error (raise a “flag”, cause an “interrupt”, etc.) and sometimes they don't.

Check that the number you have just obtained represents 127 in 8-bit two's complement.

Note that  $127 + 256 = 383$ , the unsigned sum. Note also that  $127 - 256 = -129$ , the actual sum of  $-126$  and  $-3$ .

Notice that in each case the 8-bit two's complement calculation gives a result that differs from the unsigned calculation by a multiple of 256. Notice also that when the real value of the sum is out of range the 8-bit two's complement answer is different from the actual sum by a multiple of 256.

5. Learning objective: as for question 3, but with multiplication instead of addition.

(a)  $14 * 7 = 98$

The 8-bit two's complement representation of 14 is 0000 1110 and the 8-bit two's complement representation of 7 is 0000 0111.

Carry out the binary multiplication of these two bit sequences:

$$\begin{array}{r}
 0000\ 1110 \\
 0000\ 0111\ * \\
 \hline
 \\
 \\
 \\
 \hline
 \text{carries} \\
 \hline
 \text{solution}
 \end{array}$$

Check that the result represents  $98 = 14 * 7$  as both an unsigned integer and as a 8-bit two's complement integer.

(b)  $(-3) * 14 = -42$

The 8-bit two's complement representation of  $-3$  is 1111 1101 and the 8-bit two's complement representation of 14 is 0000 1110.

Carry out the binary multiplication of these two bit sequences:

$$\begin{array}{r}
 1111\ 1101 \\
 0000\ 1110\ * \\
 \hline
 \\
 \\
 \\
 \hline
 \text{carries (two ahead)} \\
 \text{carries (one ahead)} \\
 \hline
 \text{solution}
 \end{array}$$

Check that the rightmost eight bits of the result represent  $-42$

(c)  $(-6) * (-3) = 18$

The 8-bit two's complement representation of  $-6$  is 1111 1010 and the 8-bit two's complement representation of  $-3$  is 1111 1101.

Carry out the binary multiplication of these two bit sequence. You need only calculate the result for the rightmost eight bits.

$$\begin{array}{r}
 1111\ 1010 \\
 1111\ 1101 \quad * \\
 \hline
 \\
 \\
 \\
 \\
 \hline
 \end{array}$$

carries (two ahead)  
 carries (one ahead)  
 solution

Check that the rightmost eight bits of the result is indeed the 8-bit two's complement representation of  $18 = (-6) * (-3)$ .

As in the addition, sometimes the product of two numbers is out of the range of numbers that can be represented:

- $14 * 17 = 238$  which is too large.
- $(-14) * 17 = -238$  which is too small.

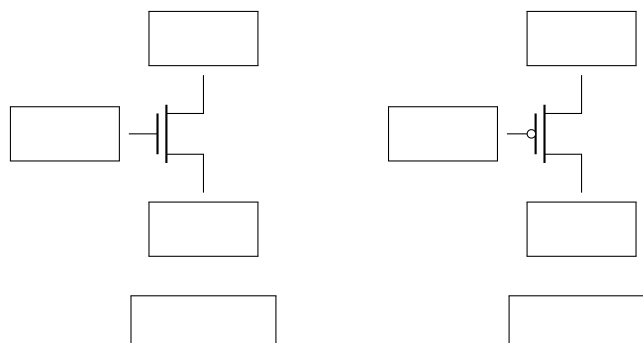
In these cases the number returned by the algorithm would be the correct answer “modulo” 256 (i.e., as the remainder when dividing by 256). So in the first instance it would be  $-12$  and in the second it would be  $+12$ .

## 2 Computer Architecture: transistors and gates

*Learning Objective:* the aim of this section is to make sure you are familiar with how the two types of transistor (nmos and pmos) operate as switches, how they can be used to construct the most basic logic gates.

Recall that standard cmos (CMOS) uses two basic types of transistor: *nmos* (NMOS) and *pmos* (PMOS).

6. (a) The symbols below are for the two types of transistor. Label them to show which is which. Also label the *Gate* with *G*, *Source* with *S* and *Drain* with *D*.

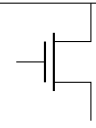
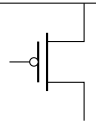




- (b) Recall that the gate  $G$  is used to control the transistor when it is used as a switch. Complete the following table to show when the two sorts of transistor switches are *closed* (i.e. there is a connection between source and drain), or *open* (no connection between source and drain).

Note: Pay close attention to the convention: in electronics, when we say a circuit is “open”, it means it is “disconnected”, i.e., there is no conductive path. This comes from the expression of “open-circuit”, which means disconnected. Similarly, when we say a circuit is closed, it means it is “connected”, i.e., there is a conductive path. So when a CMOS transistor is open, it means there is NO connection between its source and drain, and when it is closed, there is a connection.

In the past, this has been source of some unnecessary confusion, perhaps, due to the word “gate” in the CMOS transistor, and naively applying the idea of open/closed to the gate, as a port or a door! Don’t make that mistake!

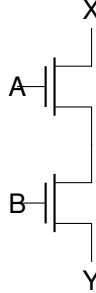
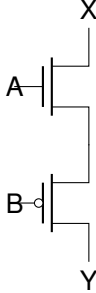
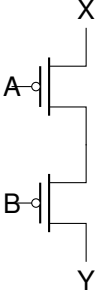
			
Type			nmos or pmos?
$G=1$ (high)			open (disconnected) or closed (connected)?
$G=0$ (low)			open (disconnected) or closed (connected)?

7. The following diagrams give simple circuit designs with two inputs, A and B. For each possible pair of values of the inputs A and B, say whether there is a connection from X to Y.

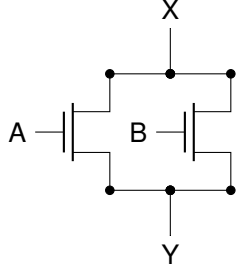
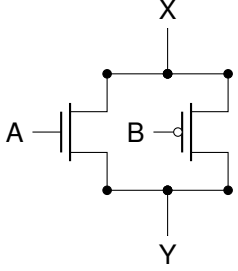
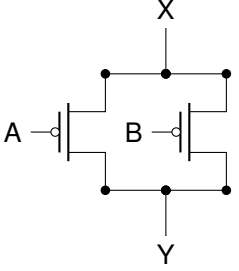
1 represents high (voltage), and 0 represents low (voltage).

Note that in the first set of diagrams, transistors in *series*, *both* transistors have to be on/closed (i.e., forming a connection), and in the second, transistors in *parallel*, it is enough for one of the two to be.

Transistors in series

				
A	B			
1	1	connection		no connection
1	0			
0	1			
0	0			

Transistors in Parallel

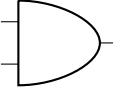
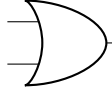
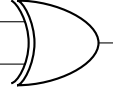
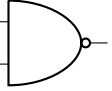
				
A	B			
1	1	connection		no connection
1	0			
0	1			
0	0			

### 3 Logic Gates

*Learning Objective:* the aim of this section is to ensure that you can write down the truth table of the logical operation that corresponds to a gate you are given. A more advanced skill is to design a gate that corresponds to a given logical operator.

Don't confuse a logic "gate", or simply a "gate" (like and, or, etc.) with the "gate" as one of the input pins of a cmos transistor!

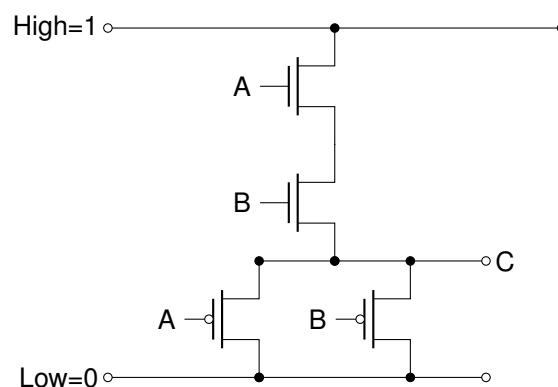
Truth tables for some binary logical connectives.

A	B	A and B	A or B	A implies B	B implies A	A xor B	A nand B
gate							
1	1	1	1	1	1	0	0
1	0	0	1	0	1	1	1
0	1	0	1	1	0	1	1
0	0	0	0	1	1	0	1

8. The gate below takes two inputs, A and B. The output is taken from C. For each possible set of inputs, work out which transistors are open or closed and hence whether C is connected to the top rail (High=1), or the bottom rail (Low=0). If C is connected to one of these, but not the other, then that is the output of the gate. If it is connected to both, then we have a short (bad), and if it is connected to neither, then the output is undetermined (also bad).

When you see the same letter written next to multiple wires, e.g. there are two wires in the figure with the label "A", it means they are connected to the same input wire. That is, they are connected to each other – and therefore, they have the same potential/voltage. The reason we do not draw the connection is to avoid cluttering the picture so it is easier to analyse!

You determine the output based on its "voltage" (High=1, Low=0). Do not analyse based on currents! In computing electronics, there is negligible current and we want to eliminate current altogether if we can (this is because electric current creates heat, and we hate heat!)



A	B	C connected to High=1	C connected to Low=0	Output on C
0/1	0/1	yes/no	yes/no	0/1/undef/short
1	1			
1	0			
0	1			
0	0			

A good gate design will produce an output for any possible combination of inputs. It will never produce a short, or be undefined.

Is this a good gate design?

Answer:

What is the logical connective being computed by this gate?

Answer:

9. Construct a gate corresponding to the logical connective *or*. You will need to design the gate in two parts: a top half and a bottom half. The top half controls the connection to High=1. There should be a connection when either  $A=1$  or when  $B=1$  (but not when  $A=B=0$ ). The bottom half controls the connection to Low=0. There should be a connection when both  $A=0$  and  $B=0$  (but not when either is 1).

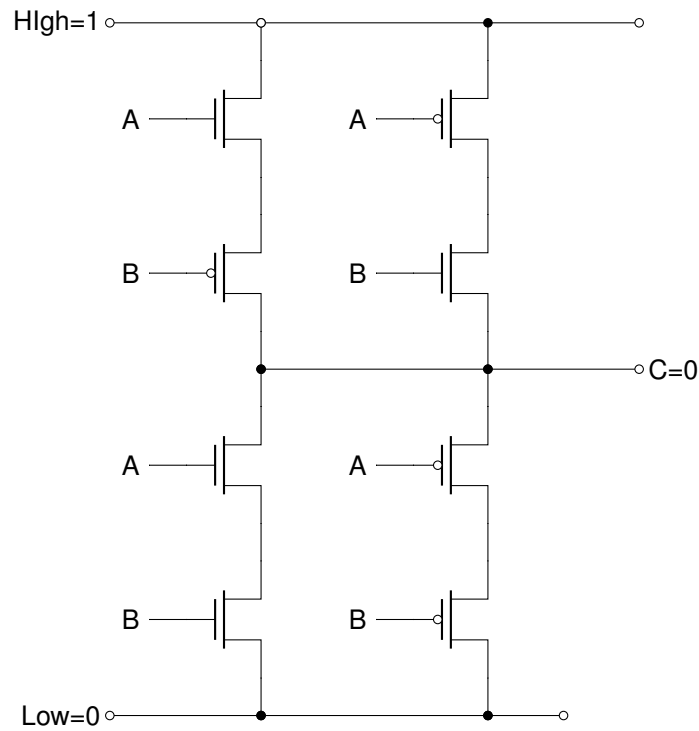
High=1 ○————○

●————○ C

Low=0 ○————○

**Optional** As you can guess, there is no “unique” implementation of the same logical gate. The truth table describes the functionality of the item, while the above design provides a recipe to actually (physically) build a device that can carry out the functional specification (this is what we mean by “implementation”). Try to see if you can reduce the number of cmos transistors in your design while still being a valid *or* gate, and get a bonus from your chip-manufacturing company! (See if you can beat the record of the last year students!)

10. The gate below is more complex. Once again, it takes two inputs, A and B. The output is taken from C. For each possible set of inputs, work out which transistors are open or closed and hence whether C is connected to the top rail (High=1), or the bottom rail (Low=0). If C is connected to one of these, but not the other, then that is the output of the gate. If it is connected to both, then we have a short (bad), and if it is connected to neither, then the output is undetermined.



A	B	C connected to High=1	C connected to Low=0	Output on C
0/1	0/1	yes/no	yes/no	0/1/undef/short
1	1			
1	0			
0	1			
0	0			

Is this a good gate design?

Answer:

What is the logical connective being computed by this gate?

Answer:

---

End of questions