Convert each of the following two's complement representations to its equivalent base ten representation.

Two's complement representation is used to represent both positive and negative numbers. In this course we study only representation of integers (not floating point numbers).

Let us take 4 bits. Using 4 bits we can represent 16 distinct numbers (refer to the previous assignment). If the 4 bits are just a binary representation of integers then we can represent all integers in the range 0-15 (both inclusive).

Suppose we want to represent both positive and negative integers using four bits. That is when we use two's complement representation. We know there are a total of 16 combinations. 8 of those combinations represent positive integers and the remaining 8 combinations represent negative integers.

Using two's complement notation and using 4 bits we can represent positive integers 0 to 7 (total of 8 positive numbers) and -8 to -1 (total of 8 negative numbers). Another way of looking at this is half of 2^4 is 2^3. The largest positive integer we can represent is 2^3-1 (note that 0 is a positive integer too). The smallest negative integer we can represent is -2^3.

Using the same logic, suppose we have 8 bits, using two's complement representation, the largest positive integer we can represent is 2^7 -1 and the smallest negative integer we can represent is -2^7

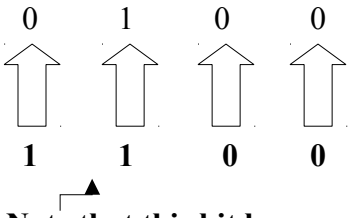
Again, suppose we have 6 bits, using two's complement representation, the largest positive integer we can represent is 2^5 -1 and the smallest negative integer we can represent is -2^5

Now let us see how to represent negative numbers. If we have 4 bits and we want to represent -4. We start off with positive 4 whose representation is 0100.

To represent -4, we use the copy and complement algorithm. We start at the right end of the bit pattern. Copy the bits one bit at a time until we copy a 1. After that we complement each bit (0 becomes a 1 and 1 becomes 0).

Copy and complement algorithm is used to find the negative of any number.

If we apply the algorithm to 0100, we have the following bit pattern.



So 1100 represents -4 in two's complement.

Refer to Figure 1.21, 1.22 page # 51-52 for two's complement representation of negative integers. Note that in two's complement representation, if the left most bit (MSB – most significant bit) is 1 then the number is negative.

a) 01111 → **15**

MSB is 0, so this represents a positive integer. And the number is (1+2+4+8) = 15

b) 10011 → **-13**

MSB is 1, so the number is negative. To find the number we use the copy and complement algorithm. **Note that –(-x) = x for any number x.**

If we use the copy and complement algorithm on 10011, we get the bit pattern 01101. This bit pattern represents (1+4+8) = 13. **So 10011 must represent -13**

c) 01101 → **13**

MSB is 0, so it represents a positive integer. And the number is (1+4+8) = 13

d) 10000 → **-16**

MSB is 1 so it represents a negative number. To negate this number, we use copy and complement algorithm. Applying this algorithm gives us the bit pattern 10000 which evaluates to 16. So 10000 must represent -16. (Note that using 5 bits, the largest positive integer that can be represented is 2^4-1 = 15 and the smallest negative integer that could be represented is -2^4 = -16)

e) 10111 → **-9**

MSB is 1 so it represents a negative integer**.** To negate this number, we use copy and complement algorithm. Applying this algorithm gives us the bit pattern 01001 which evaluates to 9. So 10111 must represent -9.

Convert each of the following base ten representations to its equivalent two's complement representation in which each value is represented in seven bits.

a) 12 → 000 1100

12 is a positive integer, so we do not have to use copy and complement algorithm.

b) -12 → 111 0100

-12 is a negative number. To find its representation, we start off with 12 whose bit pattern using 7 bits is 000 1100. If we use the copy and complement algorithm on this bit pattern we get the bit pattern 111 0100

c) -1 → 111 1111

-1 is a negative number. To find its representation, we start off with 1 whose bit pattern using 7 bits is 000 0001. If we use the copy and complement algorithm on this bit pattern we get the bit pattern 111 1111.

d) 0 → 000 0000

e) 8 → 000 1000

Realize that as far as the computer is concerned everything it deals with is a sequence of 0's and 1's. What the bit pattern represents is different depending on the application. A sequence of bits could represent part of a text file, part of Excel spreadsheet, part of a music file, or part of a jpeg file etc. We need to know what the bit pattern represents to make sense out of it.

For example, let us take the bit pattern 100 1101. If I tell you that it represents an ASCII character, you realize that each ASCII character requires 8 bits and you would look at the pattern as 0100 1101 (adding zeroes at the left end does not change the value).

You look at the ASCII table and realize that 0100 1101 represents character M

If you are told that this bit pattern represents a decimal integer in binary notation, then you interpret this sequence as integer (1+4+8+64) = 77.

If you are told that this bit pattern represents an integer in two's complement notation, then you interpret this as follows. Note the MSB is 1 so it represents a negative number. You use the copy and complement algorithm to negate the number obtaining 011 0011 which evaluates to (1+2+16+32) = 51. So the bit pattern 100 1101 represents -51 in two's complement notation.

So you can see, the same bit pattern could be interpreted differently depending on what it represents.

Perform each of the following additions assuming the bit strings represent values in two's complement notation. Identify each case in which the answer is incorrect because of overflow.

Recall that in two's complement representation, left most bit is the sign bit. You always look at the left most bit to determine whether the number is positive or negative. Left most bit of 1 denotes that the number is negative and left most bit of 0 denotes that the number is positive.
Overflow occurs when you add two positive integers and get a negative integer as the result or when you add two negative integers and get a positive integer as the result. Also, recall that the carry bit has nothing to do with overflow.

OVERFLOW CAN NEVER OCCUR WHEN YOU ADD A POSITIVE VALUE TO A NEGATIVE VALUE. OVERFLOW IS A POSSIBILITY ONLY WHEN TWO POSITIVE NUMBERS OR TWO NEGATIVE NUMBERS ARE ADDED.

Also, when you add two 5 bit numbers, the result should also fit in 5 bits. Sometimes we get a carry bit of 1, but that is not part of the answer. You have to ignore it since there are only 5 bits are allowed to store the result.

a)
```
  0 0 1 0 1
+ 0 1 0 0 0
  0 1 1 0 1
```
NO OVERFLOW. 00101 and 01000 are both positive numbers since their leftmost bit (which is the sign bit) is 0. The result of adding these two is 01101 and it is also a positive integer since its left most bit is 0.

b)
```
  1 1 1 1 1
+ 0 0 0 0 1
  0 0 0 0 0
```
NO OVERFLOW. It is not even a possibility in this case since 11111 is a negative number and 00001 is a positive number.

c)
```
  0 1 1 1 1
+ 0 0 0 0 1
  1 0 0 0 0
```
OVERFLOW – we are adding two positive integers here and the result is a negative number.

d)
```
  1 0 1 1 1
+ 1 1 0 1 0
  1 0 0 0 1
```
NO OVERFLOW. We are adding two negative numbers (sign bit of 10111 and 11010 are both 1) and the result is 10001 which is also negative. **Note that there is a carry bit in this case, but the carry bit plays no role in determining whether there is an overflow or not. The carry bit is not part of the answer either.**

e)
```
  0 0 1 1 1
+ 0 0 1 1 1
  0 1 1 1 0
```
NO OVERFLOW – both 00111 and 00111 are positive and so is 01110

f)
```
  0 0 1 1 1
+ 0 1 1 0 0
  1 0 0 1 1
```
OVERFLOW – 00111 and 01100 are both positive but their added result is 10011 which is negative

g)
```
  1 1 1 1 1
+ 1 1 1 1 1
  1 1 1 1 0
```
NO OVERFLOW - We are adding two negative numbers (sign bit of 11111 and 11111 are both 1) and the result is 11110 which is also negative. **Note that there is a carry bit in this case, but the carry bit plays no role in determining whether there is an overflow or not. The carry bit is not part of the answer either.**

h)
```
  0 1 0 1 0
+ 0 0 0 1 1
  0 1 1 0 1
```
NO OVERFLOW – both 01010 and 00011 are both positive (their sign bits are 0) and the added result 01101 is also positive.

i)
```
  0 1 0 0 0
+ 0 1 0 0 0
  1 0 0 0 0
```
OVERFLOW – 01000 and 01000 are both positive but their added result is 1000 which is negative.

j)
```
  0 1 0 1 0
+ 1 0 1 0 1
  1 1 1 1 1
```

NO OVERFLOW. It is not even a possibility in this case since 01010 is a positive number and 10101 is a negative number. You can never get an overflow when you add a positive and a negative number.