EE2026 (Part 1)
Tutorial 2 - Solutions

1. First look at the sign from the MSB, then convert the remaining bits (magnitude) as usually done in conversion from decimal to binary (iterative divisions)

Decimal	Sign Mag.
+127	0111 1111
-0	1000 0000
-55	1011 0111

2. Look at the sign, just convert from decimal to binary if positive , or reverse all bits of magnitude if negative

Decimal	1's Comp.
+43	00 0010 1011
-1	11 1111 1110
-128	11 0111 1111

3. The objective is to derive sign and magnitude of the signed decimal number.
Look at the MSB to know about the sign. Regarding the magnitude, from the definition of 2'complement representation we proceed as follows:
   - if positive (MSB=0), the 2's complement representation is equal to the magnitude, hence simply convert the number from binary to decimal
   - if negative (MSB=1), the 2's complement representation is the 2's complement of the magnitude. To evaluate the magnitude from the 2's complement of a binary number, recall that the 2's complement A^* of a number with magnitude A is $A^*=2^n-A$, hence the magnitude A is immediately found to be $A=2^n- A^*$. More explicitly, the magnitude is found by simply performing the 2'complement of A^*. In this case

(a) $10000\,(2's) \xrightarrow{\text{2's Complement}} 10000\,(\text{magnitude}) \to -16$

(b) $10000001\,(2's) \xrightarrow{\text{2's Complement}} 01111111\,(\text{magnitude}) \to -127$

4. (a) $(250)_{10} = (1111\ 1010)_2$

(b) (i) 1111 1010 (signed magnitude) → -122
Sign is negative because MSB = 1,magnitude is simply expressed by the remaining bits 1111010.

4

(b) (ii) $11111010(1's) \xrightarrow{\text{Invert bits}} 00000101(\text{magnitude}) \longrightarrow -5$

Indeed, sign is negative since MSB = 1. Now, let us evaluate the magnitude. Being negative, the given number 11111010 represents the 1's complement A* of the magnitude A, by definition of 1's complement representation.

By definition of 1's complement, we have $A^* = 2^n - 1 - A$, hence the magnitude A is equal to $A = 2^n - 1 - A^*$ i.e., the magnitude is obtained from the 1's complement by simply evaluating the 1's complement of the latter). Hence, the magnitude results to the 1's complement of 11111010 , which is 00000101 .

(iii).$11111010(\mathbf{2's}) \xrightarrow{-1} 11111001(\mathbf{1's}) \xrightarrow{\text{Invert bits}} 00000110\ (\mathbf{magnitude}) \rightarrow -6$

Same considerations apply here. The only difference is that $A^* = 2^n - A$, hence $A = 2^n - A^*$. Again, this means that the magnitude of the 2's complement representation of a negative number is simply obtained by evaluating its 2's complement.

5 (a) $(0101)_{2's} = (+5)_{10} = (0000\ 0101)_{2's}$
(b) $(1010)_{2's} = (-6)_{10} = (1111\ 1010)_{2's}$

6 (a) (-1) + 45

$$\begin{array}{r} 11111111 \\ +\ \underline{00101101} \\ \mathbf{100101100} \longrightarrow 44 \end{array}$$

(Adding these two numbers causes a carry over into the 9th bit position, which is ignored in the 8-bit arithmetic system. Thus the addition computation produces a correct answer.)

(b) (-128) + (-60)

$$\begin{array}{r} 10000000 \\ +\ \underline{11000100} \\ 01000100 \xrightarrow{\quad} 68 \end{array}$$

This example is particularly interesting since it considers the case of an "overflow", i.e. the result is constrained to have the same number of bits (bit width) as the operands, and hence the result can be out of the range that is covered by the 2's complement representation with 8 bits ($-2^{8-1} \ldots 2^{8-1}-1$, i.e. -128...127).

Being -128 the minimum value that can be represented with 8-bit 2's complement representation, subtracting 60 clearly leads to a result that is beyond the range, and an overflow occurs.

Now, the question is how to detect an overflow in an addition in a computer, where usually the bit width of the result is the same as the operands. To answer the question, we first observe that the sum of a positive and a negative number in the above range is always within the same range. In other words, the overflow can occur only if the two operands have the same sign.
When the operands have the same sign, the result should clearly have the same sign. To better understand, let us assume the two operands are positive (same considerations hold for negative numbers). As long as the result is within the correct range, its MSB in 2's complement representation will be 0, being a positive number. If the result exceeds the maximum positive number that is within the range (i.e., 011...11), its MSB will become 1 and the result will hence represent a negative number (which is clearly incorrect).

Hence, overflow occurrence can be simply checked as follows:
- compare the sign (i.e., MSB) of operands
    - if it is different, no overflow occurs (OVERFLOW = 0)
    - if it is the same, compare the sign (MSB) of the result
        - if the MSB of the result is the same as the MSB of the operands, no overflow occurred (OVERFLOW = 0)
        - otherwise, overflow was occurred, which will be signaled by raising OVERFLOW = 1 (i.e., the computer performs the calculations, providing the result, as well as the OVERFLOW signal to confirm the correctness of the result or not).

7$(00100)_{SM} = (00100)_{2's}$ [the number is positive]

$(10100)_{2's} + (00100)_{SM} = (10100)_{2's} + (00100)_{2's} =$

$(11000)_{2's}$

Convert to integers and add to verify your result!