Data Representation

1. Unsigned integers

We have reviewed how to use express unsigned integers using one of the following format: `uint8_t`, `uint16_t`, and `uint32_t`. We have also discussed how to *see or express* the same number (between 0 to 15) using one of the following four ways:

Decimal	Binary	Octal	Hex
0	0000	00	0x0
1	0001	01	0x1
2	0010	02	0x2
3	0011	03	0x3
4	0100	04	0x4
5	0101	05	0x5
6	0110	06	0x6
7	0111	07	0x7
8	1000	010	0x8
9	1001	011	0x9
10	1010	012	0xA
11	1011	013	0xB
12	1100	014	0xC
13	1101	015	0xD
14	1110	016	0xE
15	1111	017	0xF

Now, we quickly review how to do the conversion between different expressions of the same number.

1.1. From binary, octal, or hexadecimal to decimal

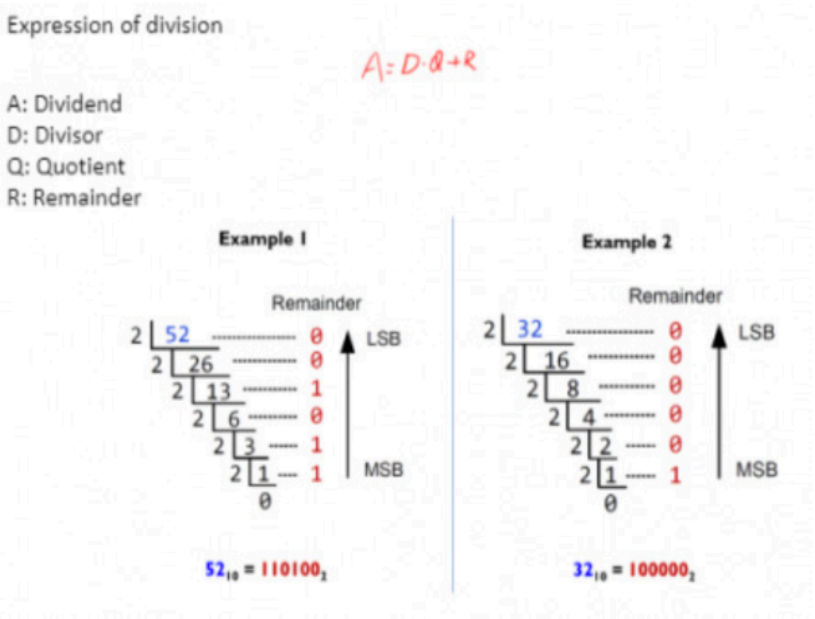
This conversion is easy, just use the following equation

$$(d_3 d_2 d_1 d_0)_b = d_3 \times b^3 + d_2 \times b^2 + d_1 \times b^1 + d_0 \times b^0,$$

where $b = 2, 8$, and 16 for binary, octal, and hexadecimal, respectively.

1.2. From decimal to binary

We can use the following algorithm to do the conversion from decimal to binary:



1.3. From binary to octal or hexadecimal

This can be done easily by regrouping the bits.

1.4. From decimal to hexadecimal

There are two approaches to do this. The first is to convert the decimal to a binary and then regroup the binary to hexadecimal. The second is to convert the decimal directly to hexadecimal, as illustrated below

$$2968 = 285 \times 16 + 8 \tag{1}$$
$$= ((11 \times 16) \times +9) \times 16 + 8. \tag{2}$$

Hence we have $2960_{10} = 0\text{xB98}_{16}$.

2. Signed integers

2.1. Three ways to represent signed integers

For many applications, we need to use signed integers. When a signed integer is positive, it can be expressed using the same way as the unsigned one. When it is negative, we can use one of the following three ways to express it (in principle).

- *Sign-and-magnitude* uses the most significant bit to represent the sign, and the rest of the bits to represent the magnitude.
- *One's complement* denotes a negative number by inverting every bit of its positive equivalent.
- *Two's complement* represents a negative number by adding one to the equivalent one's complement.

Binary Bit String	Sign-and-Magnitude	One's Complement	Two's Complement
0000	+0	+0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

Again: There is no change in the representation of the positive numbers—we only need to consider the representation of negative numbers following the above rules.

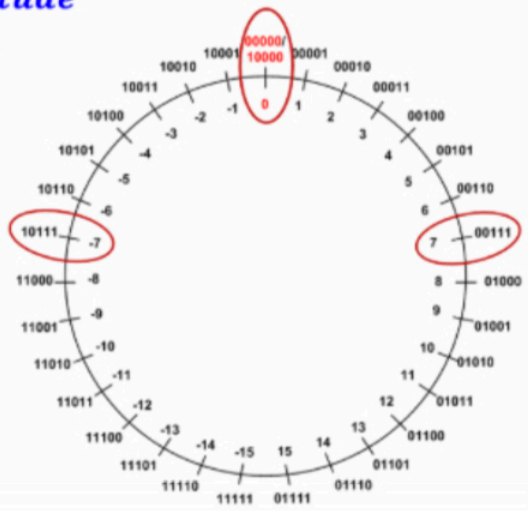
2.2. Sign and magnitude expression

Sign-and-Magnitude:

$$value = (-1)^{sign} \times Magnitude$$

- The most significant bit is the sign.
- The rest bits are magnitude.

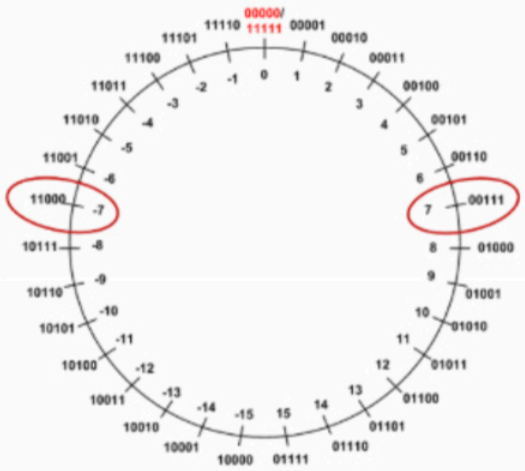
- Example: in a 5-bit system
  - $+7_{10} = 00111_2$
  - $-7_{10} = 10111_2$
- Two ways to represent zero
  - $+0_{10} = 00000_2$
  - $-0_{10} = 10000_2$
- Not used in modern systems
  - Hardware complexity
  - Two zeros



2.3. One's complement expression

One's Complement ($\tilde{\alpha}$):

$$\alpha + \tilde{\alpha} = 2^n - 1$$



The one's complement representation of a negative binary number is the bitwise NOT of its positive counterpart.

Example: in a 5-bit system

$+7_{10} = 00111_2$

$-7_{10} = 11000_2$

$$+7_{10} + (-7_{10}) = 00111_2 + 11000_2 = 11111_2 = 2^5 - 1$$

Note: $\tilde{\alpha}$ is the One's Complement of α. We use $\tilde{\alpha}$ to represent $-\alpha$

2.4. Two's complement expression

Two's Complement ($\overline{\alpha}$):

$$\alpha + \overline{\alpha} = 2^n$$

TC = Two's Complement

TC of a negative number can be obtained by the bitwise NOT of its positive counterpart plus one.

Example 1: ~~TC(3)~~ Convert -3 to TC:

	Binary	Decimal
Original number	0b00011	3
Step 1: Invert every bit	0b11100	
Step 2: Add 1	+ 0b00001	
Two's complement	0b11101	-3

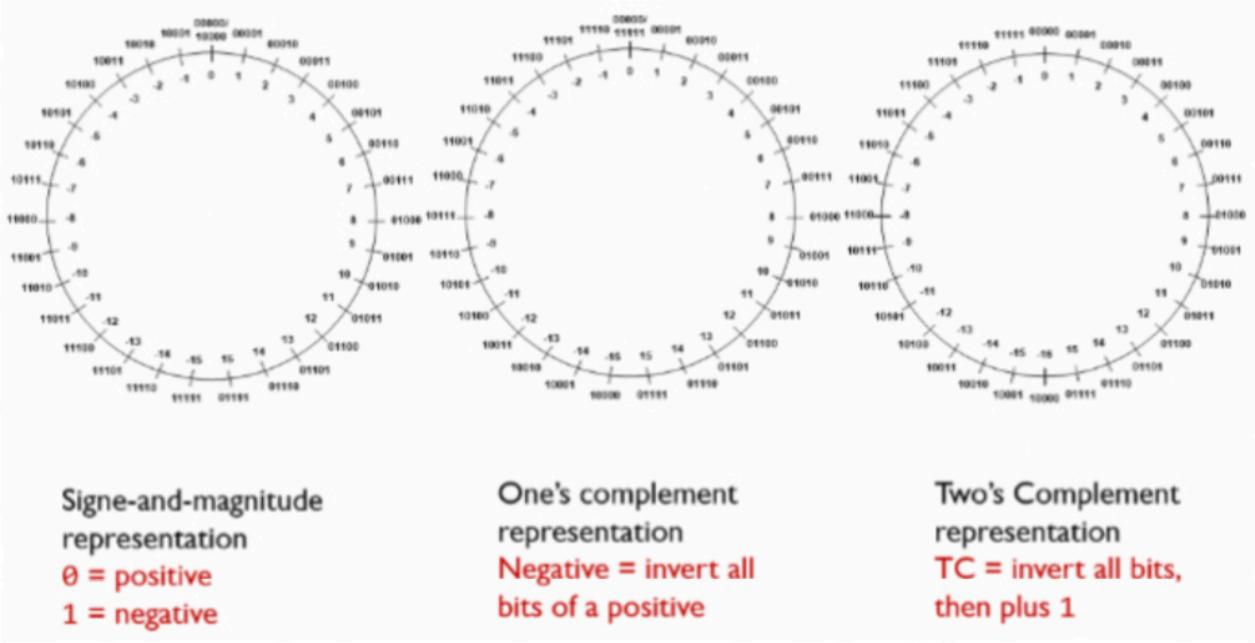
The following clarifies two confusing terms:

- a) ***Convert x to two's complement***: Find the two's complement representation of x without changing its value. This is to obtain the TC representation of x itself. Note that if x > 0, there is no change in the representation after the conversion.
- b) ***Calculate or take two's complement of x***: Compute the negative of x. This is to obtain the negative value of a number, which is better called "negate with TC." See the TC operation on p. 40, just below Fig. 2-11.

Example: Perform the following operations using a 5-bit system (give the solution in binary format). (Use the number circle to save time.)

1. Convert the following numbers to TC: 5, -6, -14, and 12.
2. Calculate (take) TC of the following numbers: 5, -7, -12, 3.

2.5. Comparison of the above three expressions



2.6. Advantages of two's complement expression

- Only a single 0 exists in the number system, which is easy for zero checking.
- Simplifies hardware implementations for the MCU:
    - The hardware implementation for two's complement subtraction is the same as that for two's complement addition.
    - The hardware implementation for addition, subtraction, and multiplication for signed integers are identical to those for unsigned integers.

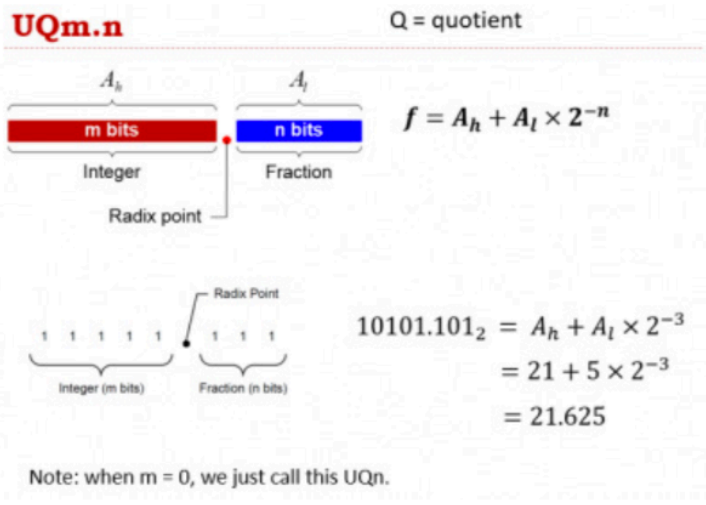
Please refer to Section 2.4.5.2 of the textbook for more detailed coverage.

3. Fixed-point real numbers

3.1. Expressin of real numbers by moving the point

- So far, we have dealt with integers only. In many applications, we need to express real numbers, such as 3.1415926. There are two ways to express real numbers:
    - Fixed-point real numbers
        - Very similar to the integers we have been using—put a *point* in an appropriate place.
        - Just a convention (an idea) to explain the integers as real numbers.
        - Very efficient computations using the instruction sets of integers to be discussed.
        - Limited range of number expressions.
    - Floating-point real numbers
        - Use special format to express the numbers, as we will see later.
        - Computations are very inefficient if there is no floating-point unit (FPU, a hardware unit) on the MCU.
        - Greater range of number expressions
- We will discuss the floating-point expression and arithmetic in the later part of the class.

3.2. Unsigned fixed-point representation



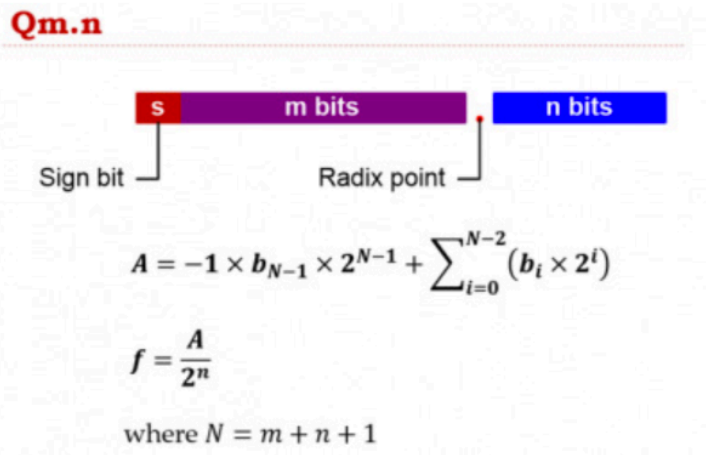
Note that the above calculation can be seen as $f = A \times 2^{-n}$.

Example: Convert $f = 3.141593$ to unsigned fixed-point UQ4.12 format.

- Calculate $f \times 2^{12} = 12867.964928$
- Round the result to an integer, $round(12867.964928) = 12868$
- Convert the integer to binary: 12868 = $\texttt{11\_0010\_0100\_0100}_2$
- Organize into UQ4.12: $\texttt{0011.0010\_0100\_0100}_2$
- Final result in Hex: `0x3244`
- Error: $\frac{12868}{2^{12}} - f = -8.5625 \times 10^{-6}$

- Class exercise: conver f = 1.125 to UQ2.6 format and find the error.

3.3. Signed fixed-point representation



Note that in digital signal processing, we often have $m = 0$. In this case, we just call the expression as Qn, such as Q15.

Example: Convert $f = -3.141593$ to signed fixed-point Q3.12 format.

- Calculate $f \times 2^{12} = -12867.964928$
- Round the result to an integer, $round(-12867.964928) = -12868$
- Convert the absolute integer to binary: $12868 = 11\_0010\_0100\_0100_2$
  (Note that the integer is represented in two's complement.)
- Make the result into 16 bits: $\mathbf{0011\_0010\_0100\_0100_2}$
- Find the two's complement: $\mathbf{1100\_1101\_1011\_1100_2}$
- Final result in Hex: **0xCDBC**
- Error: $-\frac{12868}{2^{12}} - f = 8.5625 \times 10^{-6}$

3.4. Fixed-point addition and subtraction in UQ16.16 or Q15.16

Assume Q16.16 $\quad f_C = f_A + f_B$

$$\begin{cases} I_A = f_A \times 2^{16} \\ I_B = f_B \times 2^{16} \\ I_C = f_C \times 2^{16} \end{cases} \Rightarrow \begin{cases} f_A = I_A \times 2^{-16} \\ f_B = I_B \times 2^{-16} \\ f_C = I_C \times 2^{-16} \end{cases}$$

$$\Rightarrow \quad \begin{aligned} f_C &= f_A + f_B \\ &= I_A \times 2^{-16} + I_B \times 2^{-16} \\ &= (I_A + I_B) \times 2^{-16} \end{aligned}$$

$$\Rightarrow \quad I_C \times 2^{-16} = (I_A + I_B) \times 2^{-16}$$

$$\Rightarrow \quad I_C = I_A + I_B$$

$$f_C = f_A + f_B \iff I_C = I_A + I_B$$

3.5. Fixed-point multiplication in UQ16.16 or Q15.16

Will be addressed later when we discuss the multiplication instructions.