Computer Architecture

Exercise 1

Number Systems

Peter Stallinga
Universidade do Algarve 2024-2025

Remember that **numbers do not exist in reality**, also not in the computer, and everything is simply mathematical (imaginary) **conventions**! In most number systems we use the convention that the number *N* in base *x*

$$N(x) = abcd$$

has an increased weight of the digits from right to left. In fact, the digits are multiplied by 1, by the base *x*, by the base squared, by the base cubed, etc. As such, the above base-*x* number is equal to

$$N = a{\cdot}x^3 + b{\cdot}x^2 + c{\cdot}x^1 + d{\cdot}x^0$$

This also gives a way to convert from base *x* to base *y*. We use succesive multiplications and sum. As an example, 1101 in base 2 is equal to $2^3 + 2^2 + 2^0$. In base-10 that is 13, namely $1{\cdot}10^1 + 3{\cdot}10^0$.
If we want to convert from any two bases we can do that directly, for instance from base-6 to base-5. For that we have to know how to do calculations in base 5, which most people are not able to do. Instead of direct conversions we can then do it in two steps: In the first step we convert from base-6 to base-10 and in the next step we convert from base-10 to base-5. The latter we do by successive divisions and remembering the remainders. 53 (base-10) = 32 + 16 + 4 + 1 = $2^5 + 2^4 + 2^2 + 2^0$ = 110101 (base 2).

Convert to base 10: successive multiplications Convert from base 10: successive divisions

We can continue after the unitary digit by placing a floating point, after which the series of powers simply continues with negative powers:

$$N(x) = cd.ef$$

is

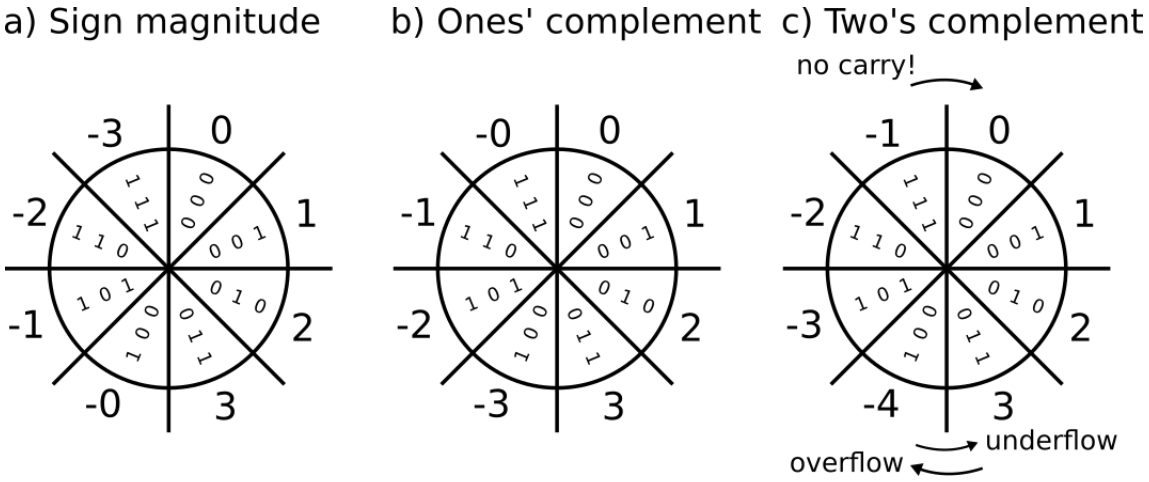
$$N = c{\cdot}x^1 + d{\cdot}x^0 + e{\cdot}x^{-1} + f{\cdot}x^{-2}$$

Example: 1.11 (2) = 1 + 0.5 + 0.25 = 1.75 (10).

For negative numbers we can use either of the three conventions. To change the sign of a number we use any of these conventions:

a) Sign-magnitude: The first digit is the sign (0 = positive, *N*–1 = negative), the rest of the digits are normal. Changing the sign is inverting the first digit (subtracted from *N*–1). b) (N–1)s'-complement: To change the sign all digits are inverted. c) N's-complement: The number (written as 0.xxxx...) is subtracted from 10.0000...

Examples:
  base 10, S/M: 0123 = +123, 9123 = −123.
  base 10, 9s'-complement: 0123 = +123, 9876 = −123
  base 10, 10's-complement: 0123 = +123, 9877 = −123
(Note: positive numbers are equal in all conventions).
(Note: N's complement is (N−1)s'-complement 'plus 1').
(Note: 2's complement puts weight $-2^{(n-1)}$ to first bit and regular positive weight to others)
(Note: subtraction in N's-complement is addition of the negated number).
(Note: The first digit can only be 0 or N−1).
(Note: S/M and (N−1)s'-complement have two 0s. N's complement is asymmetric).



1a,b,c: Convert 10000000 (2)(S/M, 1s'C, 2'sC) to decimal.
1d: Invert the signs of the results above.

2a: What is the 16-bit binary representation (S/M, 1s'-C, 2's-C) of +234(10)?
2b: What is the hexadecimal representation (S/M, 15s'-C, 16's-C) of +234(10)?
2c: What is the 16-bit binary representation (S/M, 1s'-C, 2's-C) of −234(10)?
2d: What is the hexadecimal representation (S/M, 15s'-C, 16's-C) of −234(10)?

3: How much is ASCII (convention) 'A'+'B'? What will be the ouput of this C code:

```
char a = 'a', b = 'B', c = 'C';
printf("%c\n", a-b+c);
```

(The % tells the compiler what **convention** to use in formating the output).

4a: In binary 8-bit 2's complement, add −120(10) + 23(10)
4b: In binary 4-bit 1's complement add −6(10) to 1(10)

5: Convert 87.21(10) to a number in base-8. (Do you understand the problem with floating point?)

6: Base 6. Convert 5444 (6)(S/M) to hexal 6's-complement.

7: Convert 123(5) to base-6. Can you do it directly from base-5 to base-6?