Binary Arithmetic

© Luis Entrena, Celia López, Mario García,
Enrique San Millán

Universidad Carlos III de Madrid

Outline

- Representing Signed Numbers
  - Sign-magnitude, 1s-complement and 2s-complement systems
  - 2s-complement system properties
- Binary arithmetic
  - Addition and subtraction
  - Multiplication and division
- Representing Real Numbers

Representing Signed Numbers

- Signed numbers have two parts: sign and magnitude
- The **sign** of the number has to be encoded to represent it using only 0s and 1s
  - Usually the sign is encoded as 0 for positive numbers and 1 for negative numbers.
- There are different encoding systems depending on how the **magnitude** is encoded:
  - Sign-magnitude
  - 1s-complement
  - 2s-complement

Sign-magnitude system

- Numbers in sign-magnitude system are encoded:
  - The MSB is used for the sign (0 if positive, 1 if negative)
  - The remaining bits are the magnitude of the number to represent, encoded in natural binary
- Examples:

$$25_{10} = 11001_{BIN}$$

$$+25_{10} = 011001_{SM}$$

$$-25_{10} = 111001_{SM}$$

1s-complement

- Numbers in 1s-complement are encoded:
  - If it is a positive number:
    - MSB is 0 (sign)
    - The remaining bits are the magnitude in natural binary
  - If it is a negative number:
    - MSB is 1 (sign)
    - The remaining bits are the complement (1s-complement) of the magnitude
- Examples:

$+25_{10} = 011001_{1C}$

$-25_{10} = 100110_{1C}$

2s-complement

- Numbers in 2-s complement are encoded:
  - If it is a positive number:
    - MSB is 0 (sign)
    - The remaining bits are the magnitude in natural binary
  - If it is a negative number:
    - MSB is 1 (sign)
    - The remaining bits are the 2s-complement of the magnitude.
      - 2s-complement of a number is its complement + 1

$$2C(A) = \overline{A} + 1$$

      - Equivalently, it can be defined as (if n is the number of bits):

$$Ca2(A) = 2^n - A$$

$$(2^n - A = 2^n - 1 + 1 - A = 11....11 - A + 1 = \overline{A} + 1)$$

- Examples:

$+25_{10} = 011001_{2C}$

$-25_{10} = 100111_{2C}$

Complemento a 2

- Do not confuse these concepts: ”2s-complement” operation and “2s-complement” representation of a number:
  - 2s-complement **operation** of a number:

$$2C(A) = \overline{A} + 1$$

  - 2s-complement **representation** of a number:
    - It is needed to distinguish if the number is positive or negative. Only if the number is negative the 2s-complement representation is obtained by the 2s-complement operation.
- 2s-complement is the most used representation for signed numbers in digital systems

Sign extension

- A number can be represented equivalently with different number of bits:

$$+25_{10} = 011001_{SM} = 0000011001_{SM} = 0000000000011001_{SM}$$

- Number of bits extension:
  - In SM:
    - Add zeros after the sign
  - In 1C and 2C:
    - If it is positive add zeros at the left
    - If it is negative add ones at the left

$$+25_{10} = 011001_{2C} = 0000011001_{2C} = 0000000000011001_{2C}$$

$$-25_{10} = 100111_{2C} = 1111100111_{2C} = 1111111111100111_{2C}$$

2s-Complement

Encoding			SM	1C	2C
0	0	0	0	0	0
0	0	1	1	1	1
0	1	0	2	2	2
0	1	1	3	3	3
1	0	0	-0	-3	-4
1	0	1	-1	-2	-3
1	1	0	-2	-1	-2
1	1	1	-3	-0	-1

- In SM and C1 systems there are 2 different encodings for number zero
- In 2C the zero representation is unique

2s-Complement

- Another property of the 2C system is that the inverse function of the Ca2 is the Ca2:

$$2C(2C(A_{2C})) = A_{2C}$$

Proof: $2C(2C(A_{2C})) = 2^n - (2C(A_{2C})) = 2^n - (2^n - A_{2C}) = A_{2C}$

- And with that property it can be deduced that :

$$-A_{2C} = 2C(A_{2C})$$

Dem: If A_{2C} is positive, then because of the definition itself of the 2C

If A_{2C} is negative, then $-A_{2C}$ can be obtained performing the inverse operation of the 2C, but since

$Ca2(Ca2(A_{Ca2})) = A_{Ca2}$ then $-A_{Ca2} = Ca2(A_{Ca2})$

- Example:

Starting with a positive number:	00001001	(+9)
Applying 2C operation:	11110111	(-9)
Applying 2C operation:	00001001	(+9)

2s-Complement

- There is an easy way to obtain the 2C of a number:
  - Starting from the LSB, let untouched all bits up to the first 1, and invert all the others:
  - Ca2(11100100) =00011100

    Ca2(11100100) = 00011011 + 1 = 00011100

- There is other easy way to convert from Ca2 to decimal:
  - Consider the sign weight as negative
  - Examples:
    - $1110_2 = 1*(-2^3) + 1*2^2 + 1*2^1 + 0*2^0 = -8 + 4 + 2 = -2_{10}$
    - $0110_2 = 0*(-2^3) + 1*2^2 + 1*2^1 + 0*2^0 = 4 + 2 = 6_{10}$
  - Proof:
    - If A is positive, the weight of the sign is zero, and therefore the number is obtained just as the sum of the other weights.
    - If A is negative the weight is -2n, and therefore the number that will be obtained is:

      -2n + A = -Ca2(A) = -(-A) = A

Binary Addition

- Additions of natural numbers are performed the same manner as in decimal system:

```
  1001   (9)
+ 1101  (13)
 10110  (22)
```

- Using 2s-complement system, this method is also valid for signed numbers, with the following rules:
  - Addends with same number of bits
  - Last carry is discarded
  - If both addends have the same sign, and the result of the operation has different sign to them, then the result is not valid. In this case it is said that there is "**overflow**" in the operation.
    - This is due to the need of an additional bit to represent the result.

Binary Addition using 2C: Examples

- Two positive numbers: (+9) + (+4)

$$\begin{array}{r} 0\ 1001_{2C}\ (+9) \\ +\ 0\ 0100_{2C}\ (+4) \\ \hline 0\ 1101_{2C}\ (+13) \end{array}$$

- Two negative numbers: (-9) + (-4)

$$\begin{array}{r} 1\ 0111_{2C}\ (-9) \\ +\ 1\ 1100_{2C}\ (-4) \\ \hline \cancel{1}\ 1\ 0011_{2C}\ (-13) \end{array}$$

- Big positive number and smaller negative number pequeño: (+9) + (-4)

$$\begin{array}{r} 0\ 1001_{2C}\ (+9) \\ +\ 1\ 1100_{2C}\ (-4) \\ \hline \cancel{1}\ 0\ 0101_{2C}\ (5) \end{array}$$

- Small positive number and larger negative number negativo grande: (-9) + (+4)

$$\begin{array}{r} 1\ 0111_{2C}\ (-9) \\ +\ 0\ 0100_{2C}\ (+4) \\ \hline 1\ 1011_{2C}\ (-5) \end{array}$$

- Equal and Opposite numbers:(+9) + (-9)

$$\begin{array}{r} 0\ 1001_{2C}\ (+9) \\ +\ 1\ 0111_{2C}\ (-9) \\ \hline \cancel{1}\ 0\ 0000_{2C}\ (0) \end{array}$$

- Overflow: (+1) + (+1)

$$\begin{array}{r} 0\ 1_{2C}\ (+1) \\ +\ 0\ 1_{2C}\ (+1) \\ \hline 1\ 0_{2C}\ (-2) \end{array}$$

Binary Subtraction

- Binary subtractions can be performed using the previously studied property of 2s-complement system:

$$-A_{Ca2} = Ca2(A_{Ca2})$$

Therefore:

$$A_{Ca2} - B_{Ca2} = A_{Ca2} + Ca2(B_{Ca2})$$

Or equivalently:

$$A_{Ca2} - B_{Ca2} = A_{Ca2} + \overline{B_{Ca2}} + 1$$

- In digital systems signed numbers are usually represented using 2s-complement, and substracters circuits are not needed (substractions can be performed using only adders and inverters).

Binary Multiplication

- Binary multiplications can be performed in the same manner as decimal multiplications:

```
    1001    (9)
    1101   (13)
    1001
   0000
  1001
 1001
1110101 (117)
```

- For signed numbers, 2s-complement representation is used:
  - If both numbers are positive, the multiplication can be performed directly. A zero bit for the sigh is added.
  - If both numbers are negative, then the 2s-complements of the numbers are obtained first, then they are multiplied, and finally a zero bit is added for the sign
  - If one of them is positive and the other one is negative, then the 2s-complement of the negative number is obtained, then the numbers are multiplied and finally a 1 bit for the sign is added.

Binary Division

- Binary division can be performed in the same manner as decimal division as well:

```
1001  /  11
 011     0011
 0011
   11
    0
```

- In case of signed numbers, 2s-complement representation is used, and the same rules shown previously for multiplication are applied.

Real Numbers Representation

- Real numbers can be represented in two different forms:
  - Fixed point
  - Floating Point
- Fixed Point:
  - The radix point is considered fixed
    - Example: 32 bits data, 20 bits used for the integer part and 12 bits for the fractional part
  - the algorithms shown in previous slides can be easily be extended to perform additions, subtractions, multiplications and divisions
  - Notation: Qm,n
    - m: number of bits of the integer part (optional)
    - n: number of bits of the fractional part
    - An additional bit is added for the sign (m+n+1 bits are needed)
    - Examples: Q16.16, Q.32, etc.

Real Numbers Representation

- Floating point:
  - The radix point is “floating”
  - The number is decomposed into two parts: significand (or mantissa) and exponent:

    $$N = M \times b^E$$

    Examples:
    - $2547{,}35_{10} = 2{,}54735 * 10^3$
    - $0{,}0035_{10} = 3{,}5 * 10^{-3}$
    - $111{,}0110_2 = 1{,}11011_2 * 2^2$
    - $0{,}001101_2 = 1{,}101_2 * 2^{-3}$
  - There is a fixed number of bits to represent the significand, another for the exponent, and an additional bit for the sign.
  - Normalization: It is common to select a fixed position for the radix point in the decomposition (so that the representation is unique)

Real Numbers Representation

- Standard IEEE 754: Single Precision (32 bits)
  - 1 bit for the sign
  - 8 bits for the exponent (E)
  - 23 bits for the significand (M)
  - Normalized numbers: $N= (-1)^s * 2^{E-127} * 1.M$
    - E may take values from 1 to 254 (the exponent is biased, shifted by -127)
  - Zero is represented with all zeros in E and M (it is an exception)
  - Other exceptions: E = 255 denotes infinite (used for operations with overflow, for example)
  - Not normalized numbers can be represented as well (E=0) in this case. The decomposition is different, the significand is 0.
- Standard IEEE 754: Double Precision (64 bits)
  - Similar convention to single precision, but extended to 64 bits
  - 1 bit for the sign, 11 bits for the exponent and 52 bits for the significand

Real Numbers Representation

- Example: Represent -7.625_{10} using single precision

$$-7.625_{10} = -111.101_2$$

Decomposing in the form $N= (-1)^s * 2^{E-127} * 1.M$:

$$-111.101_2 = (-1)^1 * 1.11101 * 2^2$$

$$S = 1$$

$$M = 11101$$

$$2 = E - 127 \rightarrow E = 129_{10} = 10000001_2$$

Therefore, the number represented with single precision:

1 10000001 11101000000000000000

Real Numbers Representation

- Arithmetic operations with floating point numbers: more complicated
  - Addition:
    - Modify one of the numbers so that both of them have the same exponent (denormalization)
    - Normalize the result once performed the operation
  - Subtraction:
    - Analogous to addition. Numbers are converted to 2s-complement
  - Multiplication:
    - Add exponents
    - Multiply significands
    - Round and normalize
  - Division:
    - Analogous to multiplication
- Due to the complexity of operations in floating point, many digital electronic circuits have dedicated blocks or chips to perform this kind of operations

Referencias

- Digital Systems Fundamentals. Thomas L. Floyd. Pearson Prentice Hall
- Introduction to Digital Logic Design. John P. Hayes. Addison-Wesley
- Digital Systems. Principles and Applications. Ronald J. Tocci. Pearson Prentice Hall.