Note on Binary Numbers

Maitham Shams
Jan 2013

The most familiar number system to us is decimal number system. The number 10 is the base for the decimal system. That is why we express 8759 as,

$$8759 = 8 \times 10^3 + 7 \times 10^2 + 5 \times 10^1 + 9 \times 10^0$$

In the decimal system we use 10 digits (0 to 9) to express a number.

In general a number N in base b, i.e. N_b expressed as

$$N_b = \underbrace{a_{n-1}\, a_{n-2} \cdots a_1\, a_0}_{\text{n digits, integral part}} \cdot \underbrace{a_{-1}\, a_{-2} \cdots a_{-m}}_{\text{m digits, fractional part}}$$

can be written as a polynomial in b as follows.

$$N_b = a_{n-1} b^{n-1} + a_{n-2} b^{n-2} + \cdots + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} + \cdots + a_{-m} b^{-m}$$

The most important number system in digital electronics is the binary system. Its base is 2 and uses digits 0 and 1 to express a number, like 110101. A binary digit is called a bit. Other useful number systems are octal and hexadecimal.

Base 2 : binary system, uses only 0 and 1
Base 8 : octal system, uses 0 to 7
Base 16 : hexadecimal system, uses 0 to 9 and A to F

> Converting to the Decimal System

The above polynomial is used to convert a number in other systems to decimal.

$$(10101.11)_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 32 + 8 + 2 + 1 + 0.5 + 0.25$$
$$= 43.75$$

$(2034)_8 = 2\times8^3 + 0\times8^2 + 3\times8^1 + 4\times8^0$
$= 1024 + 24 + 4 = 1052$

$(A39E.8C)_{16} = 10\times16^3 + 3\times16^2 + 9\times16^1 + 14\times16^0 + 8\times16^{-1} + 12\times16^{-2}$
$= 41,886.54688$

Common Number Systems

Decimal (Base 10)	Binary (Base 2)	Octal (Base 8)	Hexadecimal (Base 16)
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

> Converting from Decimal to Binary

For integer part we divide the number by 2 recursively till it reduces to Zero. Then, we print the remainders in reverse order.

For the fraction part, multiplication by 2 is carried out repetitively. The integer part of the result is saved and the fraction part is multiplied by 2. The process is stopped when the fraction obtained is Zero or until the desired accuracy is achieved.

The same procedures can be used to convert from decimal to octal and hexadecimal systems.

Example: Convert 68.375 to binary

÷2	R
68	R
34	0
17	0
8	1
4	0
2	0
1	0
0 (stop)	1

x2 ↓	
0.375	0.75
0.75	1.5
0.5	1.0 ← stop

1000100.011
integer fraction

$68.375 = (1000100.011)_2$

> From Octal or Hexadecimal to Binary and Vice Versa

These are much easier, because each octal digit can be represented by 3 binary digits, and each hex digit is represented by 4 binary digits. Hence, a binary number can be converted to octal by partioning it into sets of 3 digits, and to hex by partioning it into sets of 4 digits.

$(10\ 110\ 111\ 011.101\ 1)_2 = (2673.54)_8$
2 6 7 3 . 5 4

$(356.07)_8 = (11\ 101\ 110.000\ 111)_2$

$(110\ 0101\ 1101.011)_2 = (65D.6)_{16}$
6 5 D . 6

$(3C9.E)_{16} = (11\ 1100\ 1001.1110)_2$

The procedure is reversed for converting from binary to octal and hex.

> Binary Numbers and Computers

Computers and digital electronic devices use binary numbers to represent and maipulate data. This is because a binary digit can take only two values (0 and 1) and it can be easily represented by an electric property.

Binary Digit (Bit)
- 0 voltage at a node is high / A capacitor is charged / switch open (current passing)
- 1 voltage at a node is low / A capacitor is not charged / switch closed (current not passing)

1 Bit $\longrightarrow$ 2^1 values: $0 - 2^0$ (1)
2 Bits $\longrightarrow$ 2^2 values: $0 - 2^3$ (3)
⋮
N Bits $\longrightarrow$ 2^N values: $0 - 2^{N-1}$

4 bits = a Nibble
8 bits = a Byte i.e. 8b = 1B
16 bits = a word
32 bits = a Double Word
* Byte is usually used as the unit of binary data.
KB = 2^{10}b = 1024b, MB = 2^{20}b, GB = 2^{30}b

> Basic Operations with Binary Numbers

Carried out exactly like decimal

Digits a_1, a_2	Addition $a_1 + a_2$		Subtraction $a_1 - a_2$		Multiplication $a_1 \cdot a_2$
	Sum	Carry	Difference	Barrow	Product
0, 0	0	0	0	0	0
0, 1	1	0	1	1	0
1, 0	1	0	1	0	0
1, 1	0	1	0	0	1

Addition:

Carry → 1 0 1 1 1.1

1 0 1 1.1 1
+ 1 0 0 1.0 1

1 0 1 0 1 . 0 0

Decimal version

11.75
+ 9.25

21.00

Subtraction:

```
          1 1   .1   ← Borrow
        0 1     0    ← changed bits
        1 0 0 1 1.0 1
      - 0 1 1 0 0.1 1
      -------------------
        0 0 1 1 0.1 0
```

Decimal version

```
   19.25
 - 12.75
 --------
    6.50
```

Multiplication:

```
          1 0 0 1 1
        x       1 0 1
        ---------------
          1 0 0 1 1
        0 0 0 0 0
      1 0 0 1 1
      ---------------
      1 0 1 1 1 1 1
```

Decimal Version

```
    19
  x  5
  ----
    95
```

> Signed Binary Numbers

In computers negative binary numbers are written in what is known as two's (2's) complement. The two's complement of a binary N with n digits is obtained by

$$N_{2c} = 2^n - N \qquad \text{e.g. } N = 110010,\ n = 6 \Rightarrow N_{2c} = \underbrace{1000000}_{2^6} - \underbrace{110010}_{N} = 001110$$

2's complement of a number can also be obtained from its 1's complement: $N_{2c} = N_{1c} + 1$. One's complement of N can be obtained by flipping all its digits from 1 to 0 and 0 to 1. For the above example we have

$N = 110010 \quad N_{1c} = 001101 \quad N_{2c} = 001101 + 1 = 001110$

In 2's complement system a negative number is recognized by its most significant bit (msb) being 1. If $n = 2$, we can represent the following.

$00 = 0 \quad 01 = 1 \quad 10 = -2 \quad 11 = -1$ i.e. from -2 to $+1$

In general, with n bits the representable numbers range from -2^{n-1} to $+2^{n-1}-1$

e.g. for $n = 8$, from $-2^7 (= -128)$ to $+2^7 - 1 (= +127)$.

The advantage of the 2's complement system is that the operation of subtraction is performed by the mechanism of addition. Instead of $A - B$ we perform $A + (-B)$ by taking 2's complement of B.

Let A = 25 and B = 17. If n = 6, we can represent A, -A, B, and -B.

$A = 0\,1\,1\,0\,0\,1$ $\quad$ $B = 0\,1\,0\,0\,0\,1$

$-A = 1\,0\,0\,1\,1\,1$ $\quad$ $-B = 1\,0\,1\,1\,1\,1$

Then, for A - B we perform A + (-B):

$$\begin{array}{r} 25 \\ -\,17 \\ \hline 8 \end{array} \qquad \begin{array}{r} \overset{\curvearrowleft}{0\,1\,1\,0\,0\,1} \\ +\;1\,0\,1\,1\,1\,1 \\ \hline 1\;0\,0\,1\,0\,0\,0 \end{array}$$

discard ↗ the leading 1

The 6-bit answer is 001000 = +8 as expected.

Now, we try A + B:

$$\begin{array}{r} 25 \\ +\,17 \\ \hline 42 \end{array} \qquad \begin{array}{r} 0\,1\,1\,0\,0\,1 \\ +\;0\,1\,0\,0\,0\,1 \\ \hline 1\,0\,1\,0\,1\,0 \end{array} = -22$$

According to 2's complement system, the answer is negative. This indicates an overflow, because 6 bits are not enough to represent 42.

Using 7 bits, we get

$$\begin{array}{r} 0\,0\,1\,1\,0\,0\,1 \\ +\;0\,0\,1\,0\,0\,0\,1 \\ \hline 0\,1\,0\,1\,0\,1\,0 \end{array} = +42$$

Similarly, for -A - B = (-A) + (-B) we have

$$\begin{array}{r} -25 \\ -\,17 \\ \hline -42 \end{array} \qquad \begin{array}{r} \overset{\curvearrowleft}{1\,0\,0\,1\,1\,1} \\ +\;1\,0\,1\,1\,1\,1 \\ \hline (1)\;0\,1\,0\,1\,1\,0 \end{array}$$

The 6-bit answer is positive (= 22), which shows overflow again. We must use 7 bits to get $1010110 = -42$.

with 7 bits, we have

$$\begin{array}{r} \overset{\curvearrowleft}{1\,1\,0\,0\,1\,1\,1} \\ +\;1\,1\,0\,1\,1\,1\,1 \\ \hline \text{discard} \rightarrow 1\;1\,0\,1\,0\,1\,1\,0 \end{array} = -42$$

Thus, an overflow occurs if we add two numbers of similar signs (- or +) and the answer has a different sign. In that case, we need more bits to hold the right answer.

* Part of the material is from: N. Balabanian and B. Carlson, Digital Logic Design Principles, Wiley, 2001.