Двоична бройна система

На най- ниско ниво компютрите и електрониката могат да представят състоянието на дадено нещо само по два начина: вкл. / изкл., 1 / 0, високо / ниско. Затова и почти цялата електроника разчита на двоичната бройна система, за да записва, манипулира и да прави изчисления с числа. Тук е разгледана двоичната система като бройна система и основните операции, които могат да се извършат.

Двоичната (binary) бройна система е от фундаментално значение за съвременната изчислителна техника, защото нейните две цифри 1 и 0 технически лесно могат да бъдат различени - по това дали в даден възел от електрическата/електронната верига протича или не протича ток, или е налице или не напрежение.

Приемаме, че при двоичната бройна система най- младшия символ е десният или най-слабо значимият (**least-significant**). Този символ или цифра е цифрата, която оказва най-малко влияние върху стойността на числото. Важно е да се отбележи, че едно бинарно число може да се представи като биг ендиън (**big-endian**), където най- слабо значимата позиция е най-лявата или литъл ендиън (**little-endian**), където най- слабо значимата позиция е най-дясната (обикновено двоичните числа са little-endian). Какво означава това нека разгледаме **преобразуването от двоична в десетична бройна система**:

$101011 = 1*2^0+1*2^1+0*2^2+1*2^3+0*2^4+1*2^5 = 1*1+1*2+0*4+1*8+0*16+1*32 = 1+2+8+32 = 43$

Както се вижда от примера започваме да умножаване по степени на двойката от лявата към дясната позиция, като първата степен на двойката е нула и постелено се покача +1

След това нека направим обратното числото 43 да го представим в двоична форма:

43 / 2 = 21 с остатък 1 (това е най- дясна 1ва позиция)

21 / 2 = 10 с остатък 1 (2ра позиция)

10 / 2 = 5 с остатък 0 (3та позиция)

5 / 2 = 2 с остатък 1

2 / 2 = 1 с остатък 0

1 / 2 = 0 с остатък 1

Като имаме в предвид, че работим с little-endian, започваме от дясно наляво или получаваме следното:

позиция	8(7)	7(6)	6(5)	5(4)	4(3)	3(2)	2(1)	1(0)
стойност	0	0	1	0	1	0	1	1

Обърнете внимание на това, че последните позиции (7ма и 8ма) са нули и не е задължително да се записват при представянето на число в двоична форма. Дали ще запишем числото 0**b**101011 или 0**b**00101011 няма никакво значение просто нулите на 7ма и 8ма позиция могат да не се записват. По този начин ще направим и връзката към дължината на двоичните числа, а те биват следните:

Дължина	Име	Пример
1	Bit	1
4 bits	Nibble	1101
8 bits	Byte	1100 0011
16 bits	Word (2 bytes)	1100 0011 100 0011
32 bits	dWord (4 bytes)	1100 0011 1100 0011 1100 0011 1100 0011

Побитови операции

Има редица начини за манипулиране с бинарните числа. Както при десетичните така и при двоичните могат да се извършват стандартните математически операции – събиране, изваждане, умножение и деление. Побитовите операции изпълняват функциите бит по бит. Те използват булевата логика върху група от битове. Следните побитови операции са широко използвани в електрониката и програмирането:

Отрицание (NOT)

Отрицанието гледа всеки един бит и го преобразува от 1 в 0 и от 0 в 1.

Пример:

NOT 10110101 (decimal 181)

\- - - - - - - -

= 01001010 (decimal 74)

*В програмирането побитово отрицание **NOT** се означава със знака **~** (shift + `):*

```
z = ~(0b10100110);
// z ще бъде 0b01011001
```

Или (OR)

Четирите възможни комбинации са следните:

0 OR 0 = 0

0 OR 1 = 1

1 OR 0 = 1

1 OT 1 = 1

Пример:

```
        10011010
OR      01000110
        - - - - - - - -
  =     11011110
```

*В програмирането побитово или **OR** се означава със знака **|** (shift+\):*

```
y = 0b10011010 | 0b01000110;
// y ще бъде 0b11011110
```

И (AND)

Четирите възможни комбинации са следните:

0 AND 0 = 0

0 AND 1 = 0

1 AND 0 = 0

1 AND 1 = 1

Пример:

```
        10011010
AND     01000110
        - - - - - - - -
  =     00000010
```

*В програмирането побитово и **AND** се означава със знака **&** (shift+7):*

```
x = 0b10011010 & 0b01000110;
// x ще бъде 0b00000010
```

Изключващо или (XOR)

Четирите възможни комбинации са следните:

0 XOR 0 = 0

0 XOR 1 = 1

1 XOR 0 = 1

1 XOR 1 = 0

Пример:

```
        10011010
XOR     01000110
        --------
  =     11011100
```

*В програмирането побитово изключващо или **XOR** се означава със знака **^** (shift+6):*

```
r = 0b10011010 ^ 0b01000110;
// r ще бъде 0b11011100
```

Шифтване (изместване) на битове

Шифтването на битовете не се води побитова операция като горе споменатите, но е доста полезен инструмент в манипулацията на двоични числа. Шифтването (изместването) само по себе си има два параметъра ***посока*** и ***брой*** на битовете които искаме да изместим. Можем да изместим една двоична стойност на ляво или на дясно, а също така с един бит или с повече.
Когато изместваме на дясно най- младшите битове просто биват заличени

```
RIGHT-SHIFT-2       10011010 (decimal 154)
                    -------- =
                    00100110 (decimal 38)
```

Когато изместваме на ляво избутваме всички битове към най- старшите. За всяко изместване се добавя нула към най- младшите битове

```
LEFT-SHIFT-1        10011010 (decimal 154)
                    -------- =
                    100110100 (decimal 308)
```

*В програмирането за да изместим едно двоично число наляво или надясно **n** позиции използваме операторите **<< n** или **>> n***

```
i = 0b10100101 << 4;    // измести i наляво 4 бита
// i ще бъде 0b101001010000
j = 0b10010010 >> 2;    // измести j надясно 2 бита
// j ще бъде 0b00100100
```

Маскиране на битове

Маската определя кои битове искаме да запазим и кои искаме да премахнем. Маскирането представлява прилагането на тази маска. То може да се осъществи с побитовите операции AND (&), OR (|), XOR (^). Най- често срещаното и прилагано маскиране е с използване на побитово и AND (&).

Например ако искаме от 16 битово число да запазим само първите 4 бита:

```
Num 1  :        1000 0001 1110 1011
     &  :       0000 0000 0000 1111
-----------------------------------------------------
резултат :      0000 0000 0000 1011
```

В програмирането маскирането може да бъде направено по долния пример, в който е разгледано голямо число и затова е в шестнайсетичен формат за по- голяма прегледност). Обърнете внимание, че разделяме 32 битово число на 4 отделни байта byte1, byte2, byte3, byte4, като за целта използваме и функцията изместване (шифтване shift >>):

```
void more_stuff(uint32_t value) {               // примерна стойност:0x01020304
    uint32_t byte1 = (value >> 24);             // 0x01020304 >> 24 е 0x01 така
                                                // че не е нужно маскиране
    uint32_t byte2 = (value >> 16) & 0xff;      // 0x01020304 >> 16 е 0x0102
                                                // нужно е маскиране, за да
                                                // получим 0x02
    uint32_t byte3 = (value >> 8)  & 0xff;      // 0x01020304 >> 8 е 0x010203
                                                // така че трябва да маскираме
                                                // за да получим 0x03
    uint32_t byte4 = value & 0xff;              // тук е достатъчно само
                                                // маскиране без
                                                // шифтване
    ...
}
```

Източници : learn.sparkfun.com, wikipedia.com, Arduino Reference / bitwise operators , stackoverflow.com