Увод в програмирането

Бройни системи. Побитови операции.

Бройни системи

В основата на числените пресмятания

- Десетична бройна система
- Двоична бройна система
- Шестнадесетична бройна система

Десетична бройна система

Азбука : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Числото 10 играе съществена роля в десетична бройна система и се нарича основа на бройната система.

Как се представя 21 639 ?

$21\,639 = 2*10^4 + 1*10^3 + 6*10^2 + 3*10^1 + 9*10^0$

Двоична бройна система

Азбука: 0, 1

Числото 2 играе съществена роля в двоичната бройна система и се нарича основа на бройната система.

$1*2^7 + 1*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$ = ???

= 237

Когато едно число се нарича е записано в двоична бройна система, се нарича двоично число.

От десетична в двоична бройна система

Чрез делене на 2.
Колко е 74 в двоична бройна система?

Делене на числото на 2	Получен остатък(b_i)
74 : 2 = 37	$b_0 = 0$
37 : 2 = 18 + 1/2	$b_1 = 1$
18 : 2 = 9	$b_2 = 0$
9 : 2 = 4 + 1/2	$b_3 = 1$
4 : 2 = 2	$b_4 = 0$
2 : 2 = 1	$b_5 = 0$
1 : 2 = 0 + 1/2	$b_6 = 1$

От десетична в двоична бройна система

$$74 = \Sigma_{i=0}^{6} b_i * 2^i =$$
$$= b_0*2^0 + b_1*2^1 + b_2*2^2 + b_3*2^3 + b_4*2^4 + b_5*2^5 + b_6*2^6 =$$
$$= 0*2^0 + 1*2^1 + 0*2^2 + 1*2^3 + 0*2^4 + 0*2^5 + 1*2^6 =$$
$$= 0*1 + 1*2 + 0*4 + 1*8 + 0*16 + 0*32 + 1*64 =$$
$$= 2 + 8 + 64 = 74$$

От десетична в двоична бройна система (неофициална версия)

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
1	2	4	8	16	32	64	128	256

74 = ? в двоична бройна система

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

От десетична в двоична бройна система (неофициална версия)

$74 = 64 + 8 + 2 = 2^6 + 2^3 + 2^1$

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	0	0	1	0	1	0

74 = 001001010 =
1001010

Двоична аритметика

Събиране

+	0	1
0	0	1
1	1	10

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

Примери:

$$\begin{array}{r} 100101 \\ + \quad 10010 \\ \hline 110111 \end{array} \qquad \begin{array}{r} 100101 \\ + \quad 10110 \\ \hline 111011 \end{array} \qquad \begin{array}{r} 1011101 \\ + \quad 110101 \\ \hline 10010010 \end{array}$$

Изваждане

-	10	11
0	10	11
1	1	10

Примери:

$$\begin{array}{r} 110011 \\ - \quad 10010 \\ \hline 100001 \end{array} \qquad \begin{array}{r} 100101 \\ - \quad 10010 \\ \hline 10011 \end{array} \qquad \begin{array}{r} 1111101 \\ - \quad 101111 \\ \hline 1001110 \end{array}$$

Умножение

*	0	1
0	0	0
1	0	1

1011011 * 1011

1011011

+ 1011011

+ 0000000

+1011011

1111101001

Деление

Същото като при десетична бройна система. Използват се за помощни операции умножение и изваждане.

```
10010 : 110 = 011
-
  000
 1001
-
  110
 0110
-
  110
 0000
```

От десетична в двоична бройна система (неофициална версия)

231 = ? в двоична бройна система

231 = 256 - 25 =

256 - (16 + 9) = 256 - (16 + 8 + 1)

	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
255	1	0	0	0	0	0	0	0	0
16	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	1

От десетична в двоична бройна система (неофициална версия)

	16	0	0	0	0	1	0	0	0	0
+	8	0	0	0	0	0	1	0	0	0
+	1	0	0	0	0	0	0	0	0	1
=										
	25	0	0	0	0	1	1	0	0	1

От десетична в двоична бройна система (неофициална версия)

	256	1	0	0	0	0	0	0	0
-	1	0	0	0	0	0	0	0	1
=									
	255	0	1	1	1	1	1	1	1

	255	0	1	1	1	1	1	1	1
-	25	0	0	0	0	1	1	0	0	1
=									
	230	0	1	1	1	0	0	1	1	0

	230	0	1	1	1	0	0	1	1	0
+	1	0	0	0	0	0	0	0	0	1
=										
	231	0	1	1	1	0	0	1	1	1

Шестнадесетична бройна система

Азбука : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Десетична	Двоична	Шестнадесетична	Десетична	Двоична	Шестнадесетична
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

От десетична в шестнадесетична

Чрез делене на 16.

Колко е 2590 в шестнадесетична бройна система?

Делене на числото на 16	Получен остатък(b_i)
2590 : 16 = 161	b_0 = E (14)
161 : 16 = 10	b_1 = 1
10 : 16 = 0	b_2 = A (10)

От двоична в шестнадесетична

$2^4 = 16$

За да променим бройната система просто пресмятаме последователно по 4тири символа от двоичната бройна система и записваме полученото в шестнадесетичен вид.

Побитови операции

Защо?

Побитовите операции са най-бързите операции, които процесорът може да изпълнява. Понякога с тяхна помощ можем да подобрим значително бързодействието на програмата си, или пък да спестим памет.

Какво са побитови операции

Побитови операции са прости операции, изпълнявани върху цели числа (`int`, `unsigned`, `long long`, etc.). При тях имаме две променливи (константи) `A` и `B` с *еднаква дължина откъм брой битове* и за всяка позиция се изпълнява операцията между съответния бит в `A` и този на същата позиция в `B`. Тъй като те са относително прости, те са имплементирани като инструкции в процесора, което ги прави много бързи (съизмерими по скорост със събиране и изваждане, около 3-4 пъти по-бързи от умножение и над 50 пъти по-бързи от деление и модул).

Побитови оператори

&	AND
|	OR
~	NOT
^	XOR
<<	изместване наляво
>>	изместване надясно

<< изместване наляво

променлива, която ще обработваме << брой битове, които да се запълнят с нули

- Извършва преместване наляво върху левия си операнд, с толкова на брой позиции, колкото са зададени от десния операнд, който винаги трябва да бъде положителен. Отместените позиции се запълват с нули.

```
int x = 17 ; //00000000 00000000 00000000 00010001
int mask = x  <<  8;
```

>> изместване надясно

Променлива, която ще обработваме >> брой битове, които да се запълнят с нули

- Извършва преместване надясно върху левия си операнд, с толкова на брой позиции, колкото са зададени от десния операнд, който винаги трябва да бъде положителен. Отместените позиции се запълват със стойността на знаковия бит.

```
int x = -17 ;
   //11111111111111111111111111111111111111111111111111111101111
int moveX = x  >>  2;
   //11111111111111111111111111111111111111111111111111111111101
int y = 17; //00000000 00000000 00000000 00010001
Int moveY = y  >> 2;  // 0010000000 00000000 00000000 000100
```

Маска

- Използваме, за да извлечен само информацията, която ни трябва, без да се интересуваме от останалата, която имаме
- При нея единствените вдигнати битове тези на позициите, които ни интересуват

& = AND = Побитово И

&	0	1
0	0	0
1	0	1

- Най-често се прилага за маски – т.е. да се извлекът само определени позиции от числото, което имаме.

```
int number = 15; // 00000000 00000000 00000000 00001111
int mask = 1; // 00000000 00000000 00000000 00000001
int c = number & mask; // информацията, която ни трябва т.е. само последната цифра
```

| = OR = Побитово ИЛИ

|	0	1
0	0	1
1	1	1

- Използва се включване на битове

```
int x = 17; // 00000000 00000000 00000000 00010001
int  SET_ON = 15; // 00000000 00000000 00000000
   00001111
x = x | SET_ON; // 00000000 00000000 00000000
   00011111
```

- Прави единици всички битове във x, които са били единици в SET_ON

~ = NOT

~	0	1
	1	0

- Унарен оператор, т.е. прилага се върху един елемент
- Предизвиква инверсия на цялото число
- Преобразува нулите в единици, а единиците в нули

```
int x = 17; //00000000 00000000 00000000 00010001
int y = ~x; //11111111 11111111 11111111 11101110
```

^ = XOR

^	0	1
0	0	1
1	1	0

- Използва се основно, когато искаме да обърнем стойностите на определени битове
- Задава единица на всяка битова позиция, където операндите имат различни стойности и нула на всяка позиция, където те имат еднакви битове.

```
int x = 21; // 00000000 00000000 00000000 00010101
int  SET_ON = 15; // 00000000 00000000 00000000 00001111
x = x ^ SET_ON; // 00000000 00000000 00000000 00011010
```

Машинна аритметика

За да се представят числата в компютъра минаваме през няколко стъпки: прав код, обратен код и допълнителен код. Нека ни се въведат 2 числа : 15 и -15

Правия код е като не се взема знака под внимание, т.е записваме числото в познатия ни двоичен вид и

пазим код за неговия знак

15 = 0| 00000000 00000000 00000000 00001111

-15 = 1| 00000000 00000000 00000000 00001111

Нищо не очаквано за момента. Но след това се минава през още няколко стъпки.

След като вече имаме правия код, той трябва да се преобразува до обратен. Правия код на положителните числа съвпада с обратния им код. Но при отрицателните числа не е така.

При тях имаме преобразуване. На всяко място, където в правия запис имаме 0 пишем единица, а на всяко място, където имаме единица пишем нула.

Обратния запис на -15 е 11111111 11111111 11111111 11110000

Машинна аритметика

Но разбира се работа не приключва до тук. Образува се и допълнителен код. При положителните числа допълнителния код съвпада с правия код, но разбира се при отрицателните числа отново настъпват промени. Към техния обратен запис се добавя единица.

Например, за да получим допълнителния запис на -15 от обратния /* 11111111 11111111 11111111 11110000 */, като добавим 1 получаваме 11111111 11111111 11111111 11110001, което е реално записа, които нашия компютър използва за числото -15.

Нека имаме числото -12

Прав код : 1| 00000000 00000000 00000000 00001100

Подълнителен код: 1111111 11111111 11111111 11110011

Допълнителен код: 1111111 11111111 11111111 11110100

Забележка :