

# ПОДСЧЁТ ЕДИНИЧНЫХ БИТОВ

Материал к видеолекции «Беседы о программировании 011»

Караваев Артём Михайлович, 13.02.2016

Текст является неотъемлемой частью видеозаписи

<http://zealcomputing.ru>

## Вступление

Для исследования производительности алгоритмов были написаны учебные программы на языке C++. Компиляция программ проверена на VISUAL C++, INTEL C++, GCC и CLANG последних версий (на момент видеозаписи).

Тестирование проводилось на компьютере с процессором Core 2 Duo E8400 @3GHz на 64-х битовой Windows 7.

Для тестирования использовался только компилятор VISUAL C++ 2015.

Демонстрация функций в презентации выполнена также на языке C++, при этом используются псевдонимы для типов данных

```
typedef unsigned char      u8;  
typedef unsigned short int u16;  
typedef unsigned int       u32;  
typedef unsigned long long u64;
```

## Метод 0

Наивный метод последовательно перечисляет биты числа.

```
u8 CountOnes0 (u8 n) {  
    u8 res = 0;  
    while (n) {  
        res += n&1;  
        n >>= 1;  
    }  
    return res;  
}
```

Легко обобщается на случай чисел любого размера.

Тестирование выполняется на потоке из  $2^{32}$  чисел в хаотичном порядке.  
Время в секундах. Погрешность порядка 0,1 с.

Функция	Тип n			
	u8	u16	u32	u64
x86				
CountOnes0	38,18	72,00	130,49	384,76
x64				
CountOnes0	37,72	71,51	131,47	227,46

Утилита измерения времени runexe

<http://code.google.com/p/runexe>

У пользователей Linux есть встроенные функции измерения времени.

## Метод 1

Основан на удалении младшего единичного бита.

Пусть  $n = 232$ .

$n$	1	1	1	0	1	0	0	0
$n - 1$	1	1	1	0	0	1	1	1
$n \& (n - 1)$	1	1	1	0	0	0	0	0

```
u8 CountOnes1 (u8 n) {  
    u8 res = 0;  
    while (n) {  
        res ++;  
        n &= n-1;  
    }  
    return res;  
}
```

Легко обобщается на случай чисел любого размера.

Функция	Тип n			
	u8	u16	u32	u64
x86				
CountOnes0	38,18	72,00	130,49	384,76
CountOnes1	44,73	55,88	72,02	300,78
x64				
CountOnes0	37,72	71,51	131,47	227,46
CountOnes1	40,96	69,16	79,13	126,72

## Метод 2

Предподсчёт в заранее подготовленной таблице.

```
u8 BitsSetTableFF[256];      // Здесь все ответы для одного байта
```

```
u8 BitsSetTableFFFF[65536]; // Здесь все ответы для двух байт
```

```
u8 CountOnes2_FF (u8 n) {  
    return BitsSetTableFF[n];  
}
```

```
u8 CountOnes2_FF (u32 n) {  
    u8 *p = (u8*)&n;  
    n = BitsSetTableFF[p[0]]  
        + BitsSetTableFF[p[1]]  
        + BitsSetTableFF[p[2]]  
        + BitsSetTableFF[p[3]];  
    return n;  
}
```

```
u8 CountOnes2_FFFF (u32 n) {  
    u16 *p = (u16*)&n;  
    n = BitsSetTableFFFF[p[0]]  
        + BitsSetTableFFFF[p[1]];  
    return n;  
}
```

Применять метод имеет смысл тогда, когда

- вам не жалко памяти,
- число обращений к функции многократно превышает число элементов в таблице  
(«отыграть» время на заполнение таблицы),
- обращение к таблице не вытесняет что-то важное из кэша.

Функция	Тип n			
	u8	u16	u32	u64
x86				
CountOnes0	38,18	72,00	130,49	384,76
CountOnes1	44,73	55,88	72,02	300,78
CountOnes2_FF	0,01	1,83	21,07	36,25
CountOnes2_FFFF	—	0,05	7,95	13,01
x64				
CountOnes0	37,72	71,51	131,47	227,46
CountOnes1	40,96	69,16	79,13	126,72
CountOnes2_FF	0,01	1,44	24,79	26,84
CountOnes2_FFFF	—	0,07	8,49	13,01



## Метод 3

Умножение и остаток от деления. Начнём с 1-го байта

$$n = \begin{bmatrix} a & b & c & d & e & f & g & h \end{bmatrix}$$

$$n' = n \cdot 0x010101 =$$

$$\begin{array}{r}
 \\
 + \qquad \qquad \qquad a \ b \ c \ d \ e \ f \ g \ h \\
 + \ a \ b \ c \ d \ e \ f \ g \ h \\
 \hline
 = \ a \ b \ c \ d \ e \ f \ g \ h \ a \ b \ c \ d \ e \ f \ g \ h \ a \ b \ c \ d \ e \ f \ g \ h
 \end{array}$$

Такой приём будем называть словом «тиражирование».

n · 0x010101	a b c	d e f	g h a	b c d	e f g	h a b	c d e	f g h
0x249249	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1
	2	4	9	2	4	9		
n' & 0x249249	0 0 c	0 0 f	0 0 a	0 0 d	0 0 g	0 0 b	0 0 e	0 0 h

$n \cdot 0x010101 \& 0x249249$	0 0 c	0 0 f	0 0 a	0 0 d	0 0 g	0 0 b	0 0 e	0 0 h
--------------------------------	-------	-------	-------	-------	-------	-------	-------	-------

Остаток от деления на  $7 = 2^3 - 1$  даёт сумму указанных блоков по модулю 7.  
(См. беседу 004).

В нашем случае:

$$(h \cdot 2^0 + e \cdot 2^3 + b \cdot 2^6 + g \cdot 2^9 + d \cdot 2^{12} + a \cdot 2^{15} + f \cdot 2^{18} + c \cdot 2^{21}) \bmod (2^3 - 1) = \\ (h + e + b + g + d + a + f + c) \bmod (2^3 - 1).$$

```
u8 CountOnes3 (u8 n) {
    if (n == 0)    return 0;
    if (n == 0xFF) return 8;
    n = (0x010101*n & 0x249249) % 7;
    if (n == 0)    return 7;
    return n;
}
```

```
u8 CountOnes3 (u16 n) {
    return CountOnes3 (u8(n&0xFF)) + CountOnes3 (u8(n>>8));
}
```

$$n' = n \cdot 0x08040201 =$$

0 a b c	d e f g	h 0 a b	c d e f	g h 0 a	b c d e	f g h 0	a b c d	e f g h
0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1

$n' \& 0x11111111$  даёт 9 4-битовых блоков, один из которых нулевой.  
Остаток от деления на 15 даст сумму 4-битовых блоков по модулю 15.

```
u8 CountOnes3_x64 (u8 n) {
    return ((u64)0x08040201*n & 0x11111111) % 15;
}
```

Проблема в том, что здесь требуется работать с числами размером 64 бита.

Здесь мы задействовали лишь  $9 \cdot 4 - 3 = 33$  бита.

Максимум, что мы можем себе позволить этим способом, это числа до 14 бит (больше нельзя из-за остатка от деления на 15).

```
u8 CountOnes3_x64 (u14 n) { // Это не опечатка (и не должно работать)!
    return (n*0x200040008001llu & 0x1111111111111111llu) % 15;
}
```

```

u8 CountOnes3_x64 (u16 n) {
    u8 leastBit = n&1;
    n >>= 1;
    if (n == 0)    return leastBit;
    if (n == 0x7FFF)    return leastBit + 15;
    return leastBit + (n*0x200040008001llu & 0x1111111111111111llu) % 15;
}

```

В варианте для 32-х бит удобно рассматривать поля по 12 бит.

```

u8 CountOnes3_x64 (u32 n) {
    if (n == 0)    return 0;
    if (n + 1 == 0)    return 32;
    u64 res = (n&0xFFFF)*0x1001001001001llu & 0x84210842108421llu;
    res += ((n&0xFFFF000)>>12)*0x1001001001001llu & 0x84210842108421llu;
    res += (n>>24)*0x1001001001001llu & 0x84210842108421llu;
    res %= 0x1F;
    if (res == 0)    return 31;
    return res;
}

```

Комбинируя разные множители и битовые маски, можно создать очень много подобных трюков.

Функция	Тип n			
	u8	u16	u32	u64
x86				
CountOnes0	38,18	72,00	130,49	384,76
CountOnes1	44,73	55,88	72,02	300,78
CountOnes2_FF	0,01	1,83	21,07	36,25
CountOnes2_FFFF	—	0,05	7,95	13,01
CountOnes3	12,42	30,57	—	—
CountOnes3_x64	39,78	60,48	146,78	—
x64				
CountOnes0	37,72	71,51	131,47	227,46
CountOnes1	40,96	69,16	79,13	126,72
CountOnes2_FF	0,01	1,44	24,79	26,84
CountOnes2_FFFF	—	0,07	8,49	13,01
CountOnes3	13,88	33,88	—	—
CountOnes3_x64	6,78	12,28	31,12	—

Вместо взятия остатка можно суммировать блоки умножением.  
 Вернёмся снова к числу  $m = n \cdot 0x010101 \& 0x249249$  (1 байт).

0 0 c	0 0 f	0 0 a	0 0 d	0 0 g	0 0 b	0 0 e	0 0 h
C	F	A	D	G	B	E	H

$m \cdot 0x249249 =$

					C	F	A	D	G	B	E	H
+					C	F	A	D	G	B	E	H
+				C	F	A	D	G	B	E	H	
+			C	F	A	D	G	B	E	H		
+		C	F	A	D	G	B	E	H			
+	C	F	A	D	G	B	E	H				
+		C	F	A	D	G	B	E	H			
+			C	F	A	D	G	B	E	H		

```
u8 CountOnes3_x64_m (u8 n) {
    if (n == 0xFF) return 8;
    return ((u64(0x010101*n & 0x249249) * 0x249249) >> 21) & 0x7;
}
```

```

u8 CountOnes3_x64_m (u16 n) {
    u8 leastBit = n&1;
    n >>= 1;
    return leastBit
    + (((n*0x200040008001llu & 0x1111111111111111llu)*0x1111111111111111llu
        >> 56) & 0xF);
}

```

```

u8 CountOnes3_x64_m ( u32 n ) {
    if (n+1 == 0) return 32;
    u64 res = (n&0xFFF)*0x1001001001001llu & 0x84210842108421llu;
    res += ((n&0xFFF000)>>12)*0x1001001001001llu & 0x84210842108421llu;
    res += (n>>24)*0x1001001001001llu & 0x84210842108421llu;
    return (res*0x84210842108421llu >> 55) & 0x1F;
}

```

Функция	Тип n			
	u8	u16	u32	u64
x86				
CountOnes0	38,18	72,00	130,49	384,76
CountOnes1	44,73	55,88	72,02	300,78
CountOnes2_FF	0,01	1,83	21,07	36,25
CountOnes2_FFFF	—	0,05	7,95	13,01
CountOnes3	12,42	30,57	—	—
CountOnes3_x64	39,78	60,48	146,78	—
CountOnes3_x64_m	12,66	42,37	99,90	—
x64				
CountOnes0	37,72	71,51	131,47	227,46
CountOnes1	40,96	69,16	79,13	126,72
CountOnes2_FF	0,01	1,44	24,79	26,84
CountOnes2_FFFF	—	0,07	8,49	13,01
CountOnes3	13,88	33,88	—	—
CountOnes3_x64	6,78	12,28	31,12	—
CountOnes3_x64_m	3,54	4,51	18,35	—



## Метод 4

Параллельное суммирование.

Сложим биты каждой пары между собой.

$$n = ((n \gg 1) \& 0x55) + (n \& 0x55);$$

n	a b	c d	e f	g h
$x = n \& 0x55$	0 b	0 d	0 f	0 h
$y = (n \gg 1) \& 0x55$	0 a	0 c	0 e	0 g
$x + y$	a + b	c + d	e + f	g + h

Сложим соседние пары между собой.

$$n = ((n \gg 2) \& 0x33) + (n \& 0x33);$$

n	a + b	c + d	e + f	g + h
$x = n \& 0x33$	0	c + d	0	g + h
$y = (n \gg 2) \& 0x33$	0	a + b	0	e + f
$x + y$	a + b + c + d		e + f + g + h	

Сложим четвёрки.

```
n = ((n>>4) & 0x0F) + (n & 0x0F);
```

n	a+b+c+d	e+f+g+h
x=n&0x0F	0	e+f+g+h
y=(n>>4)&0x0F	0	a+b+c+d
x+y	a+b+c+d+e+f+g+h	

```
u8 CountOnes4 (u8 n) {  
    n = ((n>>1) & 0x55) + (n & 0x55);  
    n = ((n>>2) & 0x33) + (n & 0x33);  
    n = ((n>>4) & 0x0F) + (n & 0x0F);  
    return n;  
}  
  
u8 CountOnes4 (u16 n) {  
    n = ((n>>1) & 0x5555) + (n & 0x5555);  
    n = ((n>>2) & 0x3333) + (n & 0x3333);  
    n = ((n>>4) & 0x0F0F) + (n & 0x0F0F);  
    n = ((n>>8) & 0x00FF) + (n & 0x00FF);  
    return n;  
}
```

```
u8 CountOnes4 (u32 n) {  
    n = ((n >> 1) & 0x55555555) + (n & 0x55555555);  
    n = ((n >> 2) & 0x33333333) + (n & 0x33333333);  
    n = ((n >> 4) & 0x0F0F0F0F) + (n & 0x0F0F0F0F);  
    n = ((n >> 8) & 0x00FF00FF) + (n & 0x00FF00FF);  
    n = ((n >> 16) & 0x0000FFFF) + (n & 0x0000FFFF);  
    return n;  
}
```

```
u8 CountOnes4 (u64 n) {  
    n = ((n>>1) & 0x5555555555555555llu) + (n & 0x5555555555555555llu);  
    n = ((n>>2) & 0x3333333333333333llu) + (n & 0x3333333333333333llu);  
    n = ((n>>4) & 0x0F0F0F0F0F0F0F0Fllu) + (n & 0x0F0F0F0F0F0F0F0Fllu);  
    n = ((n>>8) & 0x00FF00FF00FF00FFllu) + (n & 0x00FF00FF00FF00FFllu);  
    n = ((n>>16) & 0x0000FFFF0000FFFFllu) + (n & 0x0000FFFF0000FFFFllu);  
    n = ((n>>32) & 0x00000000FFFFFFFFllu) + (n & 0x00000000FFFFFFFFllu);  
    return n;  
}
```

Возможна небольшая оптимизация (на примере 32-х бит).

Было:

```
u8 CountOnes4 (u32 n) {  
    n = ((n >> 1) & 0x55555555) + (n & 0x55555555);  
    n = ((n >> 2) & 0x33333333) + (n & 0x33333333);  
    n = ((n >> 4) & 0x0F0F0F0F) + (n & 0x0F0F0F0F);  
    n = ((n >> 8) & 0x00FF00FF) + (n & 0x00FF00FF);  
    n = ((n >> 16) & 0x0000FFFF) + (n & 0x0000FFFF);  
    return n;  
}
```

Стало:

```
u8 CountOnes4_opt (u32 n) {  
    n -= (n>>1) & 0x55555555;  
    n = ((n>>2) & 0x33333333 ) + (n & 0x33333333);  
    n = ((n>>4) + n) & 0x0F0F0F0F;  
    n = ((n>>8) + n) & 0x00FF00FF;  
    n = ((n>>16) + n);  
    return n; // Здесь происходит неявное обрезание по 8 младшим битам.  
}
```

*Подсказка к вычитанию: двухбитовый блок ab имеет значение  $2a+b$ .*

Функция	Тип n			
	u8	u16	u32	u64
x86				
CountOnes0	38,18	72,00	130,49	384,76
CountOnes1	44,73	55,88	72,02	300,78
CountOnes2_FF	0,01	1,83	21,07	36,25
CountOnes2_FFFF	—	0,05	7,95	13,01
CountOnes3	12,42	30,57	—	—
CountOnes3_x64	39,78	60,48	146,78	—
CountOnes3_x64_m	12,66	42,37	99,90	—
CountOnes4	7,52	14,10	21,12	62,70
CountOnes4_opt	7,18	11,89	18,86	65,00

Функция	Тип n			
	u8	u16	u32	u64
x64				
CountOnes0	37,72	71,51	131,47	227,46
CountOnes1	40,96	69,16	79,13	126,72
CountOnes2_FF	0,01	1,44	24,79	26,84
CountOnes2_FFFF	—	0,07	8,49	13,01
CountOnes3	13,88	33,88	—	—
CountOnes3_x64	6,78	12,28	31,12	—
CountOnes3_x64_m	3,54	4,51	18,35	—
CountOnes4	8,06	11,89	21,30	22,59
CountOnes4_opt	8,09	10,27	19,20	19,20

## Метод 5

Комбинация параллельного сложения и умножения. На примере 32-х бит. Сначала параллельным методом получаем сумму бит в каждом байте.

```
n -= (n>>1) & 0x55555555;  
n = ((n>>2) & 0x33333333) + (n & 0x33333333);  
n = (((n>>4) + n) & 0x0F0F0F0F);
```

Теперь складываем 4 байта через умножение

$$n = \begin{array}{|c|c|c|c|} \hline A & B & C & D \\ \hline \end{array}$$

$n \cdot 0x01010101 =$

$$\begin{array}{rcccc} & & A & B & C & D \\ + & & A & B & C & D \\ + & A & B & C & D & \\ + & A & B & C & D & \end{array}$$

```
u8 CountOnes5 ( u32 n ) {  
    n -= (n>>1) & 0x55555555;  
    n = ((n>>2) & 0x33333333 ) + (n & 0x33333333);  
    n = (((n>>4) + n) & 0x0F0F0F0F) * 0x01010101 >> 24;  
    return n; // Здесь происходит неявное обрезание по 8 младшим битам.  
}
```

```
u8 CountOnes5 (u16 n) {  
    n -= (n>>1) & 0x5555;  
    n = ((n>>2) & 0x3333) + (n & 0x3333);  
    n = (((n>>4) + n) & 0x0F0F) * 0x0101) >> 8;  
    return n; // Здесь происходит неявное обрезание по 8 младшим битам.  
}
```

```
u8 CountOnes5 ( u64 n ) {  
    n -= (n>>1) & 0x5555555555555555llu;  
    n = ((n>>2) & 0x3333333333333333llu ) + (n & 0x3333333333333333llu);  
    n = (((n>>4) + n) & 0x0F0F0F0F0F0F0F0Fllu)  
        * 0x0101010101010101) >> 56;  
    return n; // Здесь происходит неявное обрезание по 8 младшим битам.  
}
```



## Итоговое сравнение (x86)

ВНИМАНИЕ! Обращение к таблице с предподсчётом хаотичное.

Функция	Тип n			
	u8	u16	u32	u64
Наивный метод	38,18	72,00	130,49	384,76
Удаление младшей единицы	44,73	55,88	72,02	300,78
Предподсчёт - 1 байт	0,01	1,83	21,07	36,25
Предподсчёт - 2 байта	—	0,05	7,95	13,01
Умножение и остаток (32 бита)	12,42	30,57	—	—
Умножение и остаток (64 бита)	39,78	60,48	146,78	—
Умножение и умножение (64 бита)	12,66	42,37	99,90	—
Параллельное суммирование	7,52	14,10	21,12	62,70
Параллельное суммирование (оптимизация)	7,18	11,89	18,86	65,00
Комбинированный метод	—	17,65	13,60	52,65

## Итоговое сравнение (x64)

ВНИМАНИЕ! Обращение к таблице с предподсчётом хаотичное.

Функция	Тип n			
	u8	u16	u32	u64
Наивный метод	37,72	71,51	131,47	227,46
Удаление младшей единицы	40,96	69,16	79,13	126,72
Предподсчёт - 1 байт	0,01	1,44	24,79	26,84
Предподсчёт - 2 байта	—	0,07	8,49	13,01
Умножение и остаток (32 бита)	13,88	33,88	—	—
Умножение и остаток (64 бита)	6,78	12,28	31,12	—
Умножение и умножение (64 бита)	3,54	4,51	18,35	—
Параллельное суммирование	8,06	11,89	21,30	22,59
Параллельное суммирование (оптимизация)	8,09	10,27	19,20	19,20
Комбинированный метод	—	10,53	13,60	9,41

## Замечание

Тестирование проводилось на компьютере с процессором Core 2 Duo E8400 @3GHz на 64-х битовой Windows 7.

Для тестирования использовался только компилятор Visual C++ 2015.

При использовании других компиляторов или процессоров (особенно с другой архитектурой), результаты могут ОЧЕНЬ СИЛЬНО измениться.

Были изложены лишь базовые приёмы, комбинируя которые можно получить много различных алгоритмов. Дальше творите сами!