Представление данных

Как мы уже говорили ранее, большинство современных компьютеров работают с данными, представленными в двоичной системе счисления. То есть любые данные в памяти компьтера представленны определенными последовательностями из 0 и 1. И обработка данных — это преобразование этих последовательностей в другие последовательности. Сегодня мы рассмотрим, как в памяти представляются простейшие типы данных — числа.

Целые числа без знака хранятся в виде последовательности 0 и 1, получаемой при переводе этого числа из десятичной системы в двоичную. Понятно, что выделяя k бит для задания каждого числа, мы можем сохранить число от 0 до 2^k-1.

Для представления знаковых целых чисел существует несколько вариантов — прямой код, обратный и дополнительный. Во всех них самый левых бит числа используется для хранения знака — 0 для положительных чисел и 1 для отрицательных.

Положительные числа в прямом, обратном и дополнительном коде представляются одинаково — просто двоичное представление числа.

Очевидно, т.к. один бит ушел на знак, диапазон чисел в таком случае будет уже от 0 до 2^{k-1}-1.

А вот отрицательные числа в прямом, обратном и дополнительном кодах выглядят по-разному.

Прямой код отрицательных чисел

В знаковый разряд помещается 1, все остальное — как у положительного числа.

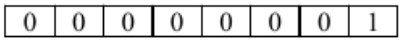
Диапазон — от - (2^{k-1}-1) до 0.

Обратный код отрицательных чисел

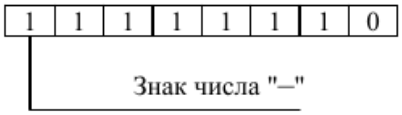
Обратный код получается инвертированием битов двоичного представления соответствующего положительного числа, включая бит знака.

Число: –1

Код модуля числа:

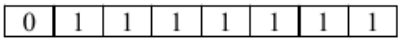


Обратный код числа:

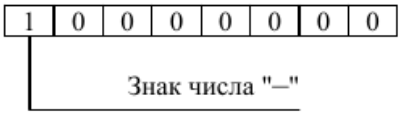


Число: –127

Код модуля числа:



Обратный код числа:

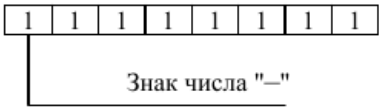


Диапазон — от - $(2^{k-1}-1)$ до 0.

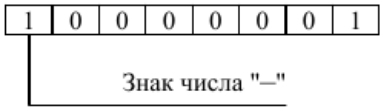
Дополнительный код отрицательных чисел

Дополнительный код получается прибавлением 1 к младшему разряду представления числа в обратном коде.

Дополнительный код числа –1:



Дополнительный код числа –127:



Диапазон — от - 2^{k-1} до -1.

Как видно из примеров, ноль имеет по два представления в прямом и обратном коде, но только одно в дополнительном.

Прямой код самый простой, однако обратный и дополнительный активно используются в технике, так как позволяют сильно упростить конструкцию вычислительного устройства путем замены некоторых арифметических операций сложением. Рассмотрим, как это происходит.

Сложение и вычитание

Сложение и вычисление чисел без знака тривиально и неинтересно:

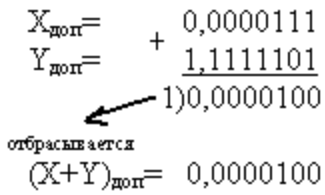
$001_2 + 100_2 = 101_2$
$101_2 - 010_2 = 011_2$

Сложение в обратном коде начинается с обычного сложения всех разрядов, включая знаковый. Если в результате получили переполнение разряда, то эту 1 прибавляем к младшему разряду результата.

$$
\begin{array}{ll}
X_{обр}= & 0,0000111 \\
+ & \\
Y_{обр}= & \underline{1,1111100} \\
 & 1\ 0,0000011 \\
 & \quad \longrightarrow +1 \\
(X+Y)_{обр}= & 0,0000100
\end{array}
$$

Вычитание чисел в обратном коде “x - y” сводится к сложению “x + (-y)“.

Сложение в дополнительном коде также начинается с обычного сложения всех разрядов, но если возникает единица переноса, она просто отбрасывается.



Вычитание происходит аналогично вычитанию в обратном коде, но по модулю 2^k (т.е. если уменьшаемое меньше вычитаемого, ему как бы добавляется 1 в дополнительном разряде слева).

Умножение во многих компьютерах умножение производится как последовательность сложений и сдвигов. Для этого имеется специальный регистр, называемый накапливающим сумматором, который до начала выполнения операции содержит число ноль. В процессе выполнения операции в нем поочередно размещаются множимое и результаты промежуточных сложений, а по завершении операции — окончательный результат. Другой регистр, участвующий в выполнении этой операции, вначале содержит множитель. Затем по мере выполнения сложений содержащееся в нем число уменьшается, пока не достигнет нулевого значения.

Деление для компьютера является трудной операцией. Обычно оно реализуется путем многократного прибавления к делимому дополнительного кода делителя.

Знаковые и беззнаковые числа

Один и тот же набор битов заданной длины можно рассматривать по-разному — как представление целого числа без знака, как представление целого числа со знаком или чего-то еще. Компьютер не знает, что у него лежит в определенном участке памяти, да и плевать он хотел, это ему говорите вы, когда заводите свои переменные. Это легко увидеть на следующем примере:

```
int x = -10;
printf("x: %d, y: %u", x, x);
```

Формат функции printf() такой — %d трактует аргумент как целое число со знаком, %u — как целое число без знака. Вывод этого всего будет такой:

```
x: -10, y: 4294967286
```

И вообще этот кусок памяти может быть частью картинки или видео-файла. И никто никогда не узнает.

Порядок хранения байтов

Разберем еще один пример:

```
int x = 0x12345678;
unsigned char * b = (unsigned char*)&x;
printf("%02X %02X %02X %02X", (unsigned)b[0], (unsigned)b[1], (unsigned)b[2], (unsigned)b[3]);
```

Результат будет такой:

78 56 34 12

Внезапно байты лежат в порядке, обратном ожидаемому! Действительно, бывают разные способы хранения байт многобайтовых значений в памяти.

Порядок от старшего к младшему (big-endian) — “слева направо”, запись начинается со старшего байта и заканчивается младшим. Этот порядок является стандартным для сетевого протокола TCP/IP, он используется в заголовках пакетов данных и во многих протоколах более высокого уровня, разработанных для использования поверх TCP/IP. Поэтому, порядок байтов от старшего к младшему часто называют сетевым порядком байтов. Так что когда будете писать код для посылки данных по сети на низком уровне, придется переворачивать байты у всех базовых типов данных. Этот порядок байтов также используется процессорами IBM 360/370/390, Motorola 68000, SPARC.

Порядок от младшего к старшему (little-endian) — “справа налево”, запись начинается с младшего байта и заканчивается старшим. Этот порядок записи принят в памяти персональных компьютеров с x86-процессорами, в связи с чем иногда его называют интеловский порядок байтов. Именно поэтому наш пример так “странно” и отработал.

Есть еще другие порядки — переключаемый, смешанный. Кому интересно, почитайте сами.

Кстати, порядок байт на компьютере можно узнать следующим образом:

```
unsigned short x = 1;
printf("%s\n", *((unsigned char *) &x) == 0 ? "big-endian" : "little-endian");
```

Представление вещественных чисел

Для представления вещественных чисел в современных компьютерах принят способ представления с плавающей запятой. Этот способ представления опирается на экспоненциальную запись действительных чисел — представление числа в виде x = $\pm$m * p^q, где p — основание системы счисления, q — целое число, а m — правильная p-ичная дробь, у которой первая цифра после запятой не равна нулю. Т.е. m принадлежит диапазону [$\frac{1}{p}$, 1). При этом m называется мантиссой числа, q — порядком числа.

Часто в компьютерной технике удобно использовать запись, когда первая цифра мантиссы как раз ненулевая, т.е. m из [1, p). Например, в случае двоичной системы счисления это позволяет экономить целый бит (он всегда равен 1). Также такой вид записи гарантирует уникальность представления числа. Такое представление числа называют нормализованным.

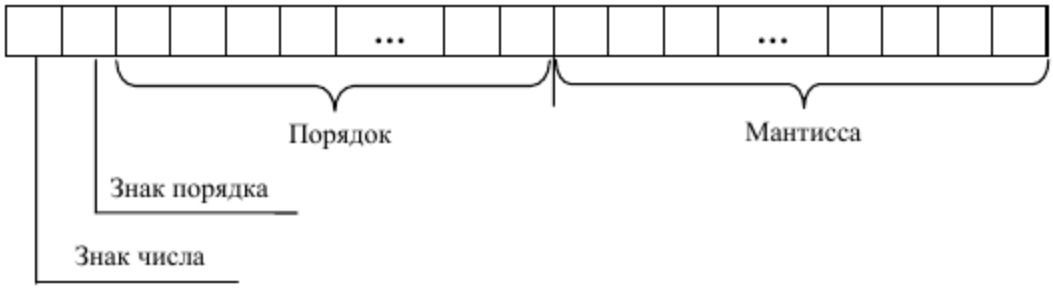
Примеры (порядок записан в 10-й системе):

- 3,1415926 = 0, 31415926 * 10^1
- 1000=0,1 * 10^4

- 0,123456789 = 0,123456789 * 10^0
- 0,00001078 = 0,1078 * 8^-4;
- 1000,00012 = 0, 100000012 * 2^4.

Так как число ноль не может быть записано в нормализованной форме в том виде, в каком она была определена, то считаем, что нормализованная запись нуля в десятичной системе будет такой: 0 = 0,0 * 10^0.

Числа с плавающей точкой представляются в виде битовых наборов, в которых отводяся разряды для мантиссы, порядка, знака числа и знака порядка:



Чем больше разрядов отводится под запись мантиссы, тем выше точность представления числа. Чем больше разрядов занимает порядок, тем шире диапазон от наименьшего отличного от нуля числа до наибольшего числа, представимого в памяти при заданном формате.

Для того, чтобы не хранить знак порядка, был придуман так называемый смещённый порядок, который рассчитывается по формуле 2^(a-1)+ИП, где а — количество разрядов, отводимых под порядок, а ИП — исходное значение порядка.

Пример

Если истинный порядок равен -5, тогда смещённый порядок для 4-байтового числа будет равен 127-5=122.

Алгоритм представления числа с плавающей запятой

1. Перевести число из р-ичной системы счисления в двоичную;
2. представить двоичное число в нормализованной экспоненциальной форме;
3. рассчитать смещённый порядок числа;
4. разместить знак, порядок и мантиссу в соответствующие разряды сетки.

Пример

Представить число -25,625 в машинном виде с использованием 4 байтового представления (где 1 бит отводится под знак числа, 8 бит — под смещённый порядок, остальные биты — под мантиссу).

1. $25_{10}=11001_2$
   $0,625_{10}=0,101_2$
   $-25,625_{10}=-11001,101_2$
2. $-100011,101_2 = -1,1001101_2 * 2^4$
3. смещенный порядок = 127+4=131
4.

Можно заметить, что представление действительного числа не очень удобно изображать в двоичной системе, поэтому часто используют шестнадцатеричное представление: 0xC1CD0000 = 11000001 11001101 00000000 00000000

Работа с битами

Для того, чтобы в копаться во внутреннем представлении, вам потребуются побитовые операции: &, | и ~. Обратите внимание на отличие логических и побитовых операций — логические операции могут не считать все свои аргументы. И ещё 1 & 2 == false, а 1 && 2 == true, так что внимательнее.

Ещё бывает битовый сдвиг — << и >>.

Ещё есть sizeof — размер типа в байтах.

Еще бывают битовые маски, они применяются, чтобы оперировать шаблонами из последовательностей битов.

Пример

```
char x = 5;

int bit = 0b10000000;
for (int j = 0; j < 8; j++)
{
        printf((x & bit) ? "1" : "0");
        bit = bit >> 1;
}
```

Вывод, как и следовало ожидать:

```
00000101
```

Полезные ссылки

1. http://ru.wikibooks.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F
2. http://habrahabr.ru/post/112953/
3. http://www.binaryconvert.com/