Машинное представление чисел

Целые числа

Беззнаковые числа

В современных ЭВМ для представления чисел используется двоичная система счисления.

При использовании для представления положительных целых чисел позиционной системы счисления по основанию N и ограничении количества разрядов числом k максимальное число, которое возможно представить, составляет $N^k - 1$. Таким образом, для десятичной системы счисления при использовании 5 разрядов максимальное число $10^5 - 1 = 99999$. В ячейку из 8 бит в двоичной системе счисления максимальное представимое число $2^8 - 1 = 11111111$.

Для представления отрицательных чисел необходим другой способ интерпретации двоичных разрядов.

Код со сдвигом

Код со сдвигом (*offset binary*) — способ представления чисел со знаком, при котором 0 интерпретируется как минимально возможное отрицательное число, а ячейка, состоящая из всех единиц — как максимальное положительное число.

Для кодирования числа нужно к числу прибавить величину сдвига K, для декодирования — вычесть K.

Величина сдвига K для ячейки из n бит определяется по формуле: $K = 2^{n-1}$

Число 0 при $n = 8$ после кодирования примет вид: $0 + 2^{8-1} = 10000000$.

Число 11111111 после декодирования:
$11111111_2 - 128_{10} = 255_{10} - 128_{10} = 127_{10}$.

Число 00000000 после декодирования: $00000000_2 - 128_{10} = -128_{10}$.

Дополнительный код

Дополнительный код (*two's complement*) — наиболее распространенный способ представления знаковых целых чисел в компьютерах.

Алгоритм кодирования отрицательного числа:

1. Инвертировать все биты в двоичном представлении модуля числа.
2. Прибавить 1.

Алгоритм декодирования:

1. Если старший разряд равен 0, то в остальных битах записано двоичное представление числа, совпадающее с его прямым кодом.
2. Иначе инвертировать все биты двоичного представления числа.
3. Прибавить 1.
4. Записать перед получившимся числом минус.

Преимущество дополнительного кода: операция вычитания реализуется с использованием схемы сложения.

Пример: $69 - 12 = 69 + (-12)$. Представление числа -12 в дополнительном коде:

```
       00001100
---------------
(inv)  11110011
+             1
---------------
       11110100
```

Затем вычитание можно заменить сложением:

```
  01000101   (69)
+ 11110100  (-12)
----------
  00111001
```

Легко проверить, что схема работает и в случаях, когда результат является отрицательным числом.

```
  00001100   (12)
+ 10111011  (-69)
----------
  11000111  (-57)
```

Физический смысл

Представим себе механический счетчик с пятью десятичными разрядами и двумя кнопками: «вперед» и «назад». Если счетчик находится в состоянии 00000, то нажатие кнопки «вперед» переводит его в состояние 00001. Последующее нажатие кнопки «назад» возвращает счетчик в состояние 00000. Еще одно нажание «назад» переводит счетчик в состояние 99999. Поскольку в этом случае был выполнен один шаг назад от нуля, то числом 99999 можно договориться кодировать значение -1.

Аналогично можно поступить и с двоичными разрядами:

```
Код          Число
00000000      0
11111111     -1
11111110     -2
```

Однако таким образом можно дойти до 00000000 и предположить, что это код числа -255. Возникает вопрос: когда стоит остановиться? Если набор двоичных разрядов рассматривается как пердставление знакового числа, то отрицательными считаются комбинации, старший бит которых равен 1.

Объяснение алгоритма кодирования

Последовательность действий для получения дополнительного кода числа может показаться неочевидной. Инверсия бит и прибавление единицы — мнемоника для достаточно простых вычислений. Для ее объяснения сначала вспомним алгоритм вычитания в столбик.

```
  93702
- 58358
-------
```

Очевидно, нельзя вычесть 8 из 2. В этом случае мы занимаем единицу из старшего разряда:

```
     1
  93702
- 58358
-------
      4
```

На следующем шаге нужно из 0 вычесть 5 и еще занятую 1. Для этого нужно снова занять единицу:

```
    11
  93702
- 58358
-------
     44
```

Последующие шаги выпоняются аналогично:

```
    11
  93702
- 58358
-------
    344
```

```
  1 11
  93702
- 58358
-------
   5344
```

```
  1 11
  93702
- 58358
-------
  35344
```

Как это связано с представлением отрицательных чисел?

Для того, чтобы получить отрицательное число, нужно вычесть его модуль из нуля. Так, $-3 = 0 - 3$.

```
 000000
 -    3
 ------

    1
```

```
000000
-    3
------
     7

   11
000000
-    3
------
    97

  111
000000
-    3
------
   997

 1111
000000
-    3
------
  9997
```

Аналогично можно поступить в двоичной системе счисления. Пусть нужно представить число -75 в ячейке размером 8 бит.

```
  00000000
- 01001011
----------

        1
  00000000
- 01001011
----------
         1

       11
  00000000
- 01001011
----------
```

```
        01

       111
  00000000
- 01001011
----------
       101

     1111
  00000000
- 01001011
----------
      0101

    11111
  00000000
- 01001011
----------
     10101

   111111
  00000000
- 01001011
----------
    110101

  1111111
  00000000
- 01001011
----------
   0110101

 11111111
  00000000
- 01001011
----------
  10110101
```

Процесс можно продолжать до бесконечности, но в этом нет смысла, поскольку в данном примере принят размер ячейки = 8 бит. Учитывая тот факт, что разряды,

не умещающиеся в ячейке, игнорируются, при использовании дополнительного (несуществующего) старшего разряда результат не изменится:

```
  11111111
  100000000
- 01001011
----------
 010110101
```

Число 100000000 не помещается в 8 бит, но его можно представить как сумму: $11111111 + 1$. Таким образом:
$100000000 - 01001011 = 11111111 + 1 - 01001011.$

```
          1
+ 11111111
- 01001011
----------
```

Результат операции вычитания в последнем примере есть инверсия бит вычитаемого (первый шаг алгоритма кодирования), после чего остается прибавить единицу, которая была использована для разложения числа 100000000 на слагаемые (второй шаг алгоритма).

Вещественные числа

Представление вещественных чисел с плавающей точкой

Числа с плавающей точкой в современных ЭВМ представлены в формате, определенным стандартом IEEE 754.

```
S E     M
0 00000 0000000000
```

S — знак, E — порядок, M — мантиса.

Рассмотрим на примере перевод десячичного числа 6.125. Будем использовать число одинарной точности.

1. Перевод целой части числа в двоичную систему счисления.

$$6_{10} = 110_2$$

2. Перевод дробной части числа в двоичную систему счисления:

```
.125 * 2 = 0 .25
.25  * 2 = 0 .5
.5   * 2 = 1 .0
```

$$6.125_{10} = 110.001_2$$

3. Представление числа в нормализованной форме: в мантиссе слева от точки находится ровно один знак.

$$110.001 = 1.10001 \cdot 2^2$$

4. Порядок представляется в коде со сдвигом:
$2 + (2^{8-1} - 1) = 129 = 10000001_2$

5. Мантиса записывается без целой части, поскольку в нормализованной форме она всегда равна единице.

```
S Exponent Mantissa
0 10000001 100 0100 0000 0000 0000 0000
```

Выполним обратное преобразование по формуле: $(-1)^S \cdot 2^{(E-(2^{n-1}-1))} \cdot 1.M$

Где n — количество бит, отведенных под хранение порядка.

$$(-1)^0 \cdot 2^{(129-127)} \cdot 1.10001 = 2^2 \cdot 1.10001 = 110.001 =$$
$$= 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 6.125$$

Размерность компонентов формата

- Число половинной точности (16 бит): S: 1, E: 5, M: 10.
- Число одинарной точности (32 бита): S: 1, E: 8, M: 23.
- Число двойной точности (64 бита): S: 1, E: 11, M: 52.
- Число четверной точности (128 бит): S: 1, E: 15, M: 112.

В качестве величины сдвига принята $K = 2^{n-1} - 1$, где n — количество бит порядка.

Порядок байтов

В том случае, если число не может быть представлено одним байтом, имеет значение в каком порядке байты записываются в памяти компьютера или передаются по линиям связи.

Порядок от старшего к младшему

Big-endian: запись начинается со старшего и заканчивается младшим байтом: $A_n, \ldots, A_0$.

Порядок от младшего к старшему

Little-endian: запись начинается с младшего и заканчивается младшим: $A_0, \ldots, A_n$.

Интерпретация бинарных данных зависит от порядка байтов. Так, последовательность байтов 00FF является записью чистла 255_{10}, если порядок big-endian, и 65280_{10}, если порядок little-endian.

Ссылки

1. David Goldberg, What Every Computer Scientist. Should Know About Floating-Point Arithmetic
2. Что нужно знать про арифметику с плавающей запятой / Хабрахабр
3. А.В. Столяров. Программирование на языке ассемблера NASM для ОС Unix