**Лекция 2. Позиционные системы счисления. Представление чисел в микропроцессорах.**

Как представлять и хранить данные.

Как ими оперировать?

В микропроцессорной технике со времен фон Неймана в основном используется двоичная система счисления. При этом для человека привычнее десятичная система счисления по числу пальцев на руках. Поэтому представляют интерес правила перевода из двоичной системы счисления в десятичную и обратно и реализация операций с двоичным кодом. Помимо двоичной системы счисления в микропроцессорной технике часто применяется шестнадцатеричная, ее роль также будет рассмотрена в данной лекции.

**Позиционные системы счисления**

В этой системе счисления число есть строка цифр, каждой цифре приписан вес. Значение числа – взвешенная сумма его разрядов:

1734 = 1∙103 + 7∙102 + 3∙101 + 4∙100

Для представления дробных чисел можно использовать отрицательные степени числа 10:

187.25 = 1∙102 + 8∙101 + 7∙100 + 2∙10-1 + 5∙10-2

Величина r = 10 называется основанием (base, radix) системы счисления. В общем случае:

|  |  |
| --- | --- |
|  | (1) |

где *di* – разряды числа D, 0 ≤ *di* < *r* или *di є* [0, *r* - 1],

*dp-1* – старший разряд числа, most significant digit (MSD),

*d-n* – старший разряд числа, least significant digit (LSD),

*p* – количество разрядов целой части числа,

*n* – количество разрядов дробной части числа.

Если n = 0, то число *D* – целое. Если n > 0, p > 0 и фиксированы, то такой формат представления называется числом с фиксированной точкой. Если n и p могут изменяться, то такой формат чисел называется числами с плавающей точкой. Формат чисел с плавающей точкой может быть разным. В процессорах x86 и AVR используется стандарт IEEE-754.

**Роль двоичной системы счисления в микропроцессорной технике**

Поговорим более подробно про двоичную систему счисления, т.е. рассмотрим случай r = 2.

**Пример:**

110110(2) = ***1***∙25 + ***1***∙24 +***0***∙23 +***1***∙22 +***1***∙21 +***0***∙20 = 32 + 16 + 4 + 2 + 1 = 55(10)

Значение двоичной системы в цифровой технике настолько велико, что разряды двоичного числа получили свое собственное название – бит (bit – BInary digiT).

Очевидно, что электронный регистр является естественным хранилищем двоичных чисел. **Таким образом, с помощью регистров можно хранить целые и вещественные числа (в виде чисел с фиксированной и плавающей точкой).**

**Но есть особенности.**

**Пример** представление числа 0.1(10)

Если представить 0.1(10) как двоичную дробь, то она окажется периодической: 0.0(0011) (2). Подробно рассмотреть пример дальше, когда будет про перевод.

Получается надо бесконечное число разрядов, а их нет. Очевидно, что при хранении данных в регистре величины p и n конечны. Сумма n + p называется разрядностью числа. Исторически в микропроцессорной технике возникло понятие байта. Байт – единица хранения информации, соответствующая 8 двоичным разрядам, т.е. 8 битам. Кроме того, существует понятие полубайт или тетрада – 4 бита.

Слово – характерный размер данных, измеряемый в битах, для которого аппаратно реализована операция сложения (которое автоматически дает вычитании при использовании *дополнительного кода*). Помимо операции сложения, слово характеризует и другие важные характеристика процессора, о которых речь пойдет позже. Слово зависит от архитектуры процессора и является важной его характеристикой. Так, процессоры Intel Pentium I-IV 32-разрядные, Intel Core 2 – 64-разрядные. Микроконтроллер AVR, изучаемый в данном курсе – 8-разрядный или, что то же самое, 8-битный.

Разрядность задает диапазон возможных значений. Для целых 8-разрядных чисел возможны коды:

0000 0000, 0000 0001, …, 1111 1111

Сколько здесь кодовых комбинаций?

0, 1, … 255

Получается максимальный код плюс 1, с учетом комбинации 0000 0000. В двоичном коде:

1111 1111 + 1 = 1 0000 0000 = 28

В общем случае при n-разрядном представлении двоичного числа количество кодовых комбинаций N равно:

|  |  |
| --- | --- |
| N = 2n | (2) |

**Задание.**

Сколько комбинаций в слове у 32-битных процессоров?

**Шестнадцатиразрядная система счисления**

Разряд шестнадцатеричной системы счисления лежит в диапазоне от 0 до 15:

d ∈[0, 16) = [0, 15]

Записи разрядов для диапазона 0-9 соответствуют десятичной системе, а цифры 10-15 задаются латинскими буквами A-F.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Шестнадцатеричный разряд** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |
| **Десятичное значение разряда** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |

**Пример**

AF3(16) = A∙162 + F∙161 + 3∙160 = ?

*Связь двоичной и шестнадцатиричной систем счисления*

*b8*∙28 + *b*7∙27 + *b*6∙26 + *b*5∙25 + *b*4∙24 + *b*3∙23 + *b*2∙22 + *b*1∙21 + *b*0∙20 =

= (*b8*∙20) ∙28 + (*b*7∙23 + *b*6∙22 + *b*5∙21 + *b*4∙20) ∙24 + (*b*3∙23 + *b*2∙22 + *b*1∙21 + *b*0∙20) ∙20 =

= ( *…* ) ∙ 162 + ( *…* ) ∙ 161 + (*…*) ∙ 160

Можно показать, что выражения в скобках, например (*b*3∙23 + *b*2∙22 + *b*1∙21 + *b*0∙20) лежат в диапазоне [0, 15], т.е. удовлетворяет требованию к разрядам 16-разрядного числа (см. формулу (1))

**Пример**

1101 0101(2) = 13(10) 5(10) = D(16) 5(16) = D5(16)

Очевидно, легко сделать и обратное преобразование.

**Пример**

AF3(16) = 1010 1111 0011(2)

Из примеров ясно, что ценность 16-разрядной системы счисления – в ее близости в двоичной. Фактически, она в основном используется программистами (не микропроцессорами!), когда надо задать двоичное число большой разрядности в компактном виде.

В общем случае для перевода ***n*** – разрядного двоичного число надо  шестнадцатеричных разрядов. Общая формула перевода имеет вид:



Тетрады двоичного числа , записанные в 16-ричном виде, образуют запись 16-ричного представления числа.

**Общий подход к преобразованию чисел между системами счисления.**

Мы рассмотрели 3 системы счисления с основаниями 2, 10, 16. В работе надо уметь переводить из каждой системы в любую другую, т.е. каждой клетке таблицы 1 должно соответствовать некоторое правило перевода (кроме диагональных клеток). Некоторые клетки мы уже можем заполнить.

1. **Bin -> Dec**, уже рассмотрен выше, просто расписать по формуле (1)

2. **Bin -> Hex** – есть простой способ, рассмотренный выше

3. Попробуем **Dec -> Hex** расписать по формуле (1)

135(10) = 1∙A2 + 3∙A1 + 5∙A0

Получается сложно, потому что для перевода придется вести вычисления в шестнадцатеричной системе счисления. Другой способ – подгонка. Нам необходимо подобрать разряды d2, d1, d0 в разложении:

135(10) = d2∙162 + d1∙161 + d0∙160

Сколько раз по 162=256 содержится в 135? Ни одного, поэтому разряд d2 = 0. Разряд d1 указывает сколько раз по 161 содержится в 135. Очевидно, 8. Дальше легко определить d0.

(Почему надо идти сверху и брать максимальное значение?)

Табл. 1. Перевод между системами счисления

|  |  |  |  |
| --- | --- | --- | --- |
| Из | В | | |
| Bin(ary), 2 | Dec(imal), 10 | Hex(ademical), 16 |
| Bin(ary), 2 | x | формула (1) | Связь 2<->16 |
| Dec(imal), 10 | ? | x | подбор |
| Hex(ademical), 16 | Связь 2<->16 | формула (1) | x |

Остался перевод dec->bin. Рассмотрим для

**Общее правило перевода**

Пусть задано число в некоторой системе счисления. Требуется его перевести в систему счисления с основанием r. Рассмотрим отдельно перевод его целой и дробной части. В соответствии с (1) *p* разрядов задают целую часть числа, а *n* разрядов – дробную.

Целая часть числа в нужной нам системе с основанием *r* в соответствии с (1) имеет вид:

|  |  |
| --- | --- |
|  | (3) |

Задача перевода сводится к отысканию величин *di*. Разделим D на r и определим чему равны частное и остаток.



Поскольку d0 < r, то остаток от деления равен d0 (именно d0, а не дробь ), а частное равно Q. Следовательно, при делении числа в «старой» системе счисления на основание «новой» системы счисления, остаток от деления равен младшему разряду числа в «новой» системе счисления. Аналогично *d1* будет остатком от деления Q на r. Повторяя процедуру, пока Q ≠ 0, получим все разряды числа в системе счисления с основанием r в обратном порядке.

Рассмотрим перевод дробной части числа:

|  |  |
| --- | --- |
|  | (4) |

Умножим D на r:

|  |  |
| --- | --- |
|  | (5) |

Можно показать, что сумма всех слагаемых выражения, кроме d-1 строго меньше 1. Следовательно, целая часть числа P равна d-1. Обнулим целую часть P и снова умножим на r, получим d-2 и так далее.

Отметим, что мы не сделали никаких предположений относительно основания системы r, поэтому описанный способ подходит не только для dec -> bin, но и, например, для dec -> hex.

**Примеры:**

**13(10) = ?(2)**

**0.1(10) = ?(2)** – убедиться, что это действительно периодический код

**Лекция 3. Машинная реализация операций с числами. Дополнительный код.**

Постановочная часть. Как представить и работать со знаковыми числами.



Определение дополнительного кода для целого n-разрядного числа x в r-ичной системе счисления:

r – основание системы счисления

n – разрядность числа

Продемонстрируем основное свойство дополнительного кода, выполнив сложение на аппаратном сумматоре:

В случае, если , имеем:

Если интуитивно понятнее представить выражение следующим образом:

Поскольку , то

Таким образом,

То есть результатом является дополнительный код модуля отрицательного результата.

Вывод: Для осуществления операции вычитания на двоичном сумматоре, необходимо знать дополнительный код вычитаемого.

Однако, само вычисление дополнительного кода по данному определению трудоемко, т.к. само требует вычитания. Проблема решается за счет того, что существует быстрый способ расчета двоичного дополнительного кода:

Очевидно, что число представляет собой число из *n* единиц.

При вычитании получаем число, биты которого инверсны по отношению к битам x. Действительно выражение равно 1, если и равно 0, если . Кроме того, поскольку , то нигде не встречаются «заемы» от старшего бита. В итоге имеем:

Учитывая полученный результат ясно, что двоичный вычислитель превращается в вычитатель, если проинвертировать одно из входных чисел и задать «1» на вход переноса на младшем сумматоре.



Обоснование вывода формулы.

Формула для определения значения знакового числа по его битовому набору:

Рассмотрим простой и часто встречающийся код:

int a = -43;

long b = a;

Хочется, чтобы после выполнения кода

Однако, если просто скопировать 16-разрядный битовый набор переменной a в 16 младших разрядов переменной b, а остальные (старшие) разряды b оставить положить равными нулю (что вроде бы «логично»), то нетрудно сообразить, что результат достигнут не будет, хотя бы потому, что число с нулевым старшим разрядом (а у b аж 16 старших разрядов равны нулю!) не может быть отрицательным.

Подобробнее раскрыть, что старший разряд отвечает за знак

Рассмотрим сначала более простой случай – пусть число b – не 32-разрядное, а 17-и разрядное соответственно. Снова потребуем

Тогда:

Выделии старшее слагаемое из суммы в правой части тождества:

Отсюда

Повторяя рассуждения для 17 и 18 разрядных чисел и так далее до получения 32-разрядного числа, становится ясен смысл алгоритма распространения знака, который состоит в том, что при приведении целой знаковой переменной меньшей разрядности к знаковой переменной большей разрядности свободные старшие разряды не заполняются нулями (что «логично»), а заполняются значением старшего разряда исходного числа (что правильно).

До сих пор мы рассмотрели

**Машинное представление вещественных чисел. Формат IEE-754.**

Прежде чем глубоко погрузится в формат чисел с плавающей точкой, бросим въедливый взгляд на табл. , а именно на типы long и float. Оба типа имеют одинаковую разрядность – 32 бита или 4 байта. Многих это вводит в заблуждение, автор этих строк однажды даже видел следующее рассуждение на интернет-форуме: «поскольку оба типа 4-байтные, то их максимальное значение должно быть одинаковым, следовательно, значение 3.4·1038 для float ошибочное». Однако, ошибки никакой нет. Фокус состоит в том, что отведенные 32 бита можно использовать по-разному. У *целых беззнаковых* чисел биты (или битовый набор) интерпретируются по формуле (1), знаковые – по формуле (2). Правила интерпретации битового набора для вещественных чисел совершенно другие, и благодаря им получается теми же 32 битами задать числа с большим диапазоном. Рассмотрим эти правила.

Представление вещественных чисел основано на нормализованной экспоненциальной записи числа:

где *M* – мантисса, *e* – порядок. Для десятичной системы счисления , для двоичной системы счисления .

**Пример 1.** Представим десятичное число в нормализованной экспоненциальной записи**:**

Можно это представить себе так: множитель «двигает» точку в мантиссе *M* вправо, в результате получается исходное число.

**Пример 2.** Переведем то же самое десятичное число в двоичный вид и затем в двоичную нормализованную форму.

Учтем, что роль множителя аналогична десятичной системе, тогда Нормализованная экспоненциальная запись этого двоичного числа:

Если взять отрицательный порядок, например *e* = -3, то сдвиг запятой происходит влево:

Обратим внимание, что мантисса представляет собой вещественное число с фиксированной точкой:

Поскольку , то для любого значения мантиссы старший разряд равен единице, поэтому его можно отбросить, сэкономив один бит памяти.

Сделав предварительные замечания, рассмотрим правила интерпретации битового набора вещественных чисел по формуле, заданной стандартом IEEE-754:

|  |  |
| --- | --- |
|  | (6) |

– вещественная часть мантиссы;

– бит знака (*s* = 1 для x < 0);

– порядок числа, величина .

Как видно, формула учитывает равенство единице старшего бита мантиссы в экспоненциальной записи числа.

Порядок *e* может быть положительным и отрицательным. Для определения знака экспоненты, не вводя при этом еще одного бита знака, добавляют смещение к порядку в половину кодового диапазона байта.

Формат IEEE-754 определяет формат машинного представления вещественных чисел с одинарной точностью (32 бита, тип float в языке C) и двойной точностью (64 бита, тип double в язык C). Числа с разной точностью отличаются количеством разрядов мантиссы и порядка (см. Рис).

|  |
| --- |
|  |
| Рисунок. Формат IEEE-754 с двойной и одинарной точностью |

**Пример 3.** Представим знакомое нам число в формате с одинарной точностью.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0 |  |  |

Все-таки, как же хранить в 32-битной переменной значения, большие max long? Для этого рассчитаем максимальное число, которое можно представить с помощью чисел IEEE-754 с одинарной точностью. Максимальное число получится, если задать максимальную мантиссу и максимальный порядок .

Максимальное число типа unsigned long, если задать все его биты равными единице, равно:

Как видно, формат IEEE-754 обеспечивает , поэтому .

В отличие от целых чисел, для вещественных чисел имеется нетривиальное понятие минимального по модулю числа, также называемого **машинным нулем**. Зададим , , тогда

Получается неожиданный вывод, что используя (4) нельзя задать число, строго равное нулю. Как же задать строгий ноль? Только отойдя от правила (4) в особых случаях. В Таблица 1 приведены эти особые комбинации. В ней помимо понятных вариантов с нулем и бесконечностью, имеется специальное значение NaN (not a number, не число). NaN нужен, например, для обозначения некорректного результата при попытке посчитать квадратный корень из -1.

Таблица Специальные значения

|  |  |  |  |
| --- | --- | --- | --- |
| **Значение числа** | **Битовая комбинация числа** | | |
|  |  |  |
| +0 | 0 |  |  |
| -0 | 1 |  |  |
|  | 0 |  |  |
|  | 1 |  |  |
| NaN |  |  | (за исключением случая «все нули», соответствующего ) |

Понятие точности представления чисел с плавающей точкой также значительно сложнее по сравнению с целыми числами. Если записать вещественное число в целочисленную переменную (int, long), то его дробная часть просто отбросится. Если то же число записать в вещественную переменную (float), то дробная часть целиком не отбросится, но и без потерь не обойдется – т.к. разрядность ограничена.

Зададимся вопросом, какова будет разница между истинным числом округления при представлении произвольного числа (например, числа ). Оценим эту разницу:

- число, которое необходимо представить, x – округленное до конечной разрядности , величины – мантиссы чисел . Абсолютная ошибка зависит от порядка *e* (который у чисел один и тот же). Перейдем к относительной ошибке, которая от порядка *e* не зависит:

|  |  |  |  |
| --- | --- | --- | --- |
|  | (7) |  | (8) |

Мантисса округленного числа содержит конечное число разрядов и равна:

Мантисса истинного числа содержит бесконечное число разрядов и равна:

Тогда

Отсюда

Как известно, за счет округления до ближайшего можно снизить погрешность в два раза, поэтому

Максимальная величина относительной погрешности будет достигнута при максимальном (равном ) и минимальном (по определению равном единице). Тогда

Величина называется машинным эпсилоном. Машинный эпсилон – числовое значение, меньше которого невозможно задавать точность любого алгоритма, возвращающего вещественное число.

**Пример.** Рассмотрим, насколько может отличаться число от своего представления в формате IEEE-754 с одинарной точностью:

Типы данных для архитектуры ATMega.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Тип*** | ***Целый, вещественный*** | ***Знак*** | ***Разрядность*** | ***Диапазон значений*** |
| char = unsigned char | целый | беззнаковый | 8 | 0 … +255 |
| signed char | целый | знаковый | -128 ... +127 |
| **int** = signed int | целый | знаковый | 16 | -32768 … 32767 |
| unsigned int | целый | беззнаковый | 0 … 65535 |
| short int =  signed short int | целый | знаковый | -32768 … 32767 |
| unsigned short int | целый | беззнаковый | 0 … 65535 |
| long = long int = signed long int | целый | знаковый | 32 | -2147483648 … +2147483647 |
| unsigned long int | целый | беззнаковый | 0 … 4294967295 |
| float | вещественный | знаковый | 32 | ±1.7·10-38 - ±3.4·1038 |
| double | вещественный | знаковый | 32 | ±1.7·10-38 - ±3.4·1038 |
| long double | вещественный | знаковый | 32 | ±1.7·10-38 - ±3.4·1038 |

Самарский А.А., Гулин А.В. Численные методы: Учеб. Пособие для вузов. – М.: Наука. Гл. ред. физ-мат. лит., 1989. – 432 с.

IEEE-754