1.4.4 Байт, полубайт и «весь этот джаз»

Группа из восьми битов называется *байт (byte)*. Байт представляет $2^8 = 256$ цифровых комбинаций. Размер модулей, сохраненных в памяти компьютера, обычно измеряется именно в байтах, а не битах.

Группа из четырех битов (половина байта) называется *полубайт (nibble)*. Полубайт представляет $2^4 = 16$ цифровых комбинаций. Одна шестнадцатеричная цифра занимает один полубайт, а две шестнадцатеричные цифры – один байт. В настоящее время полубайты уже не находят широкого применения, однако этот термин все же стоит знать, да и звучит он забавно (в английском языке *nibble* означает откусывать что-либо маленькими кусочками).

Микропроцессор обрабатывает данные не целиком, а небольшими блоками, называемыми словами. Размер *слова (word)* не является величиной, установленной раз и навсегда, а определяется архитектурой каждого конкретного микропроцессора. На момент написания этой главы (в 2012 году) абсолютное большинство компьютеров использовало 64-битные процессоры. Такие процессоры обрабатывают информацию блоками (словами) длиной 64 бита. А еще не так давно верхом совершенства считались компьютеры, обрабатывающие информацию словами длиной 32 бита. Интересно, что и сегодня наиболее простые микропроцессоры и особенно те, что управляют работой таких бытовых

устройств, как, например, тостеры или микроволновые печи, используют слова длиной 16 бит или даже 8 бит.

В рамках одной группы битов конечный бит, находящийся на одном конце этой группы (обычно правом), называется *наименее значимым битом (least significant bit, lsb*), или просто *младшим битом*, а бит на другом конце группы называется *наиболее значимым битом (most significant bit, msb)*, или *старшим битом*. **Рис. 1.7 (a)** демонстрирует наименее и наиболее значимые биты в случае 6-битного двоичного числа. Аналогичным образом, внутри одного слова можно выделить наименее значимый байт (least significant byte, LSB), или младший байт, и наиболее значимый байт (most significant byte, MSB), или старший байт. **Рис. 1.7 (b)** показывает, как это делается в случае 4-байтного числа, записанного восемью шестнадцатеричными цифрами.

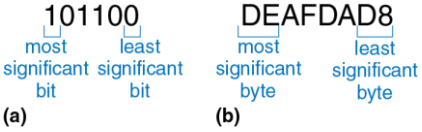

Рис. 1.7 **Наименее и наиболее значимые биты, и байты**

В силу удачного совпадения $2^{10} = 1024 \approx 10^3$. Этот факт позволяет нам использовать приставку *кило* (греческое название тысячи) для сокращенного обозначения 2^{10}. Например, 2^{10} байт – это один килобайт (1 КБ). Подобным же образом, *мега* (греческое название миллиона) обозначает $2^{20} \approx 10^6$, а *гига* (греческое название миллиарда) указывает на $2^{30} \approx 10^9$. Зная, что $2^{10} \approx 1$ тысяча, $2^{20} \approx 1$ миллион, $2^{30} \approx 1$ миллиард и помня значения степеней двойки до 2^9 включительно, будет легко приблизительно рассчитать в уме любую другую степень двух.

Если подходить абсолютно строго к терминологии, то микропроцессором называется такой процессор, все элементы которого размещается на одной микросхеме. До 70-х годов XX века полупроводниковая технология не позволяла разместить процессор целиком на одной микросхеме, поэтому процессоры мощных компьютеров представляли собой набор плат с довольно большим количеством различных микросхем на них. Компания Intel в 1971 году представила первый 4-битный микропроцессор, получивший в качестве названия номер 4004. В наши дни даже самые передовые суперкомпьютеры построены на микропроцессорах, поэтому в этой книге мы будем считать «микропроцессор» и «процессор» тождественными понятиями и использовать оба эти термина как синонимы.

Пример 1.6 ОЦЕНКА СТЕПЕНЕЙ ДВОЙКИ

Найдите приблизительное значение 2^{24} без использования калькулятора.

Решение. Представьте экспоненту как число кратное десяти и остаток.

$2^{24} = 2^{20} \times 2^4$, $2^{20} \approx 1$ миллион. $2^4 = 16$. Итак, $2^{24} \approx 16$ миллионов. На самом деле $2^{24} = 16\,777\,216$, но 16 миллионов – достаточно хорошее приближение для маркетинговых целей.

Так же как 1024 байта называют *килобайтом* (КБ), 1024 бита называют *килобитом* (Кб или кбит). Аналогичным образом, МБ, Мб, ГБ и Гб используются для сокращенного обозначения миллиона и миллиарда байтов и битов. Размеры элементов памяти обычно измеряются в байтах. А вот скорость передачи данных измеряется в битах в секунду. Максимальная скорость передачи данных телефонным модемом, например, составляет 56 килобит в секунду.

1.4.5 Сложение двоичных чисел

Сложение двоичных чисел производится так же, как и сложение десятичных, с той лишь разницей, что двоичное сложение выполнить гораздо проще (см. **Рис. 1.8**). Как и при сложении десятичных чисел, если сумма двух чисел превышает значение, помещающееся в один разряд, мы переносим 1 в следующий разряд. На **Рис. 1.8** для сравнения показано сложение десятичных и двоичных чисел. В крайней

правой колонке на **Рис. 1.8 (a)** складываются числа 7 и 9. Сумма 7 + 9 = 16, что превышает 9, а значит, больше того, что может вместить один десятичный разряд. Поэтому мы записываем в первый разряд 6 (первая колонка), и переносим 10 в следующий разряд (вторая колонка) как 1. Аналогичным же образом при сложении двоичных чисел, если сумма двух чисел превышает 1, мы переносим 2 в следующий разряд как 1. В правой колонке на **Рис. 1.8 (b)**, например, сумма $1 + 1 = 2_{10} = 10_2$, что не может уместиться в одном двоичном разряде. Поэтому мы записываем 0 в первом разряде (первая колонка) и 1 в следующем разряде (вторая колонка). Во второй колонке опять складываются 1 и 1 и еще добавляется 1, перенесенная сюда после сложения чисел в первой колонке. Сумма $1 + 1 + 1 = 3_{10} = 11_2$. Мы записываем 1 в первый разряд (вторая колонка) и снова добавляем 1 в следующий разряд (третья колонка). По очевидной причине бит, добавленный в соседний разряд (колонку), называется битом переноса *(carry bit)*.

```
   11     ← carries →      11
  4277                    1011
+ 5499                  + 0011
  9776                    1110
(a)                     (b)
```

Рис. 1.8 **Примеры сложения с переносом: (a) десятичное (b) двоичное**

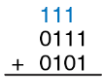

Рис. 1.9 Пример двоичного сложения

Пример 1.7 ДВОИЧНОЕ СЛОЖЕНИЕ

Вычислить $0111_2 + 0101_2$

Решение. На **Рис. 1.9** показано, что сумма равна 1100_2. Переносы выделены синим цветом. Мы можем проверить нашу работу, повторив вычисления в десятичной системе счисления. $0111_2 = 7_{10}$, $0101_2 = 5_{10}$. Сумма равна $12_{10} = 1100_2$

Цифровые системы обычно оперируют числами с заранее определенным и фиксированным количеством разрядов. Ситуацию, когда результат сложения превышает выделенное для него количество разрядов, называют *переполнением (overflow)*. Четырехбитная ячейка памяти, например, может сохранять значения в диапазоне [0, 15]. Такая ячейка переполняется, если результат сложения превышает число 15. В этом случае дополнительный пятый бит отбрасывается, а результат, оставшийся в четырех битах, будет ошибочным. Переполнение можно обнаружить, если следить за переносом бита из наиболее значимого разряда двоичного числа (см. на **Рис. 1.8**), из наиболее левой колонки.

Пример 1.8 СЛОЖЕНИЕ С ПЕРЕПОЛНЕНИЕМ

Вычислить $1101_2 + 0101_2$. Будет ли переполнение?

Решение. На **Рис. 1.10** показано, что сумма равна 10010_2. Результат выходит за границы четырехбитового двоичного числа. Если его нужно запомнить в 4-х битах, наиболее значимый бит пропадет, оставив некорректный результат 0010_2. Если вычисления производятся с числами с пятью или более битами, результат 10010_2 будет корректным.

```
  11 1
  1101
+ 0101
 10010
```

Рис. 1.10 Пример двоичного сложения с переполнением

1.4.6 Знак двоичных чисел

До сих пор мы рассматривали двоичные числа без знака *(unsigned)* – то есть только положительные числа. Часто, однако, для вычислений требуются как положительные, так и отрицательные числа, а это значит, что для знака двоичного числа нам потребуется дополнительный разряд. Существует несколько способов представления двоичных чисел со знаком *(signed)*. Наиболее широко применяются два: *Прямой Код (Sign/Magnitude)* и *Дополнительный Код (Two's Complement)*.

Прямой код

Представление отрицательных двоичных с использованием прямого кода интуитивно покажется вам наиболее привлекательным, поскольку совпадает с привычным способом записи отрицательных чисел, когда сначала идет знак минус, а затем абсолютное значение числа. Двоичное число, состоящее из N битов и записанное в прямом коде, использует наиболее значимый бит для знака, а остальные $N-1$ бита для записи абсолютного значения этого числа. Если наиболее значимый бит 0, то число положительное. Если наиболее значимый бит 1, то число отрицательное.

Пример 1.9 ПРЕДСТАВЛЕНИЕ ЧИСЕЛ В ПРЯМОМ КОДЕ

Запишите числа 5 и –5 как четырехбитовые числа в прямом коде

Решение: Оба числа имеют абсолютную величину $5_{10} = 101_2$. Таким образом, $5_{10} = 0101_2$ и $-5_{10} = 1101_2$.

К сожалению, стандартный способ сложения не работает в случае двоичных чисел со знаком, записанных в прямом коде. Например, складывая $-5_{10} + 5_{10}$ привычным способом, получаем $1101_2 + 0101_2 = 10010_2$. Что, естественно, есть полный абсурд.

Двоичная переменная длиной N битов в прямом коде может представлять число в диапазоне $[-2^{N-1} + 1, 2^{N-1} - 1]$.

Другой несколько странной особенностью прямого кода является наличие +0 и −0, причем оба этих числа соответствуют одному нулю. Нетрудно предположить, что представление одной и той же величины двумя различными способами чревато ошибками.

Дополнительный код

Двоичные числа, записанные с использованием дополнительного кода, и двоичные числа без знака идентичны, за исключением того, что в случае дополнительного кода вес наиболее значимого бита -2^{N-1} вместо 2^{N-1}, как в случае двоичного числа без знака. Дополнительный код гарантирует однозначное представление нуля, допускает сложение чисел по привычной схеме, а значит, избавлен от недостатков прямого кода.

В случае дополнительного кода нулевое значение представлено нулями во всех разрядах двоичного числа: $00\ldots000_2$. Максимальное положительное значение представлено нулем в наиболее значимом разряде и единицами во всех других разрядах двоичного числа: $01\ldots111_2 = 2^{N-1} - 1$. Максимальное отрицательное значение имеет единицу в наиболее значимом разряде и нули во всех остальных разрядах: $10\ldots000_2 = -2^{N-1}$. Отрицательная единица представлена единицами во всех разрядах двоичного числа: $11\ldots111_2$.

Ракета Ариан-5 ценой 7 миллиардов долларов, запущенная 4 июня 1996 года, отклонилась от курса и разрушилась через 40 секунд после запуска. Отказ был вызван тем, что в бортовом компьютере произошло переполнение 16-разрядных регистров, после которого компьютер вышел из строя.

Программное обеспечение Ариан-5 было тщательно протестировано, но на ракете Ариан-4. Однако новая ракета имела двигатели с более высокими скоростными параметрами, которые, будучи переданными бортовому компьютеру, и вызвали переполнение регистров.

(Фото: ESA/CNES/ARIANESPACE- Service Optique CS6)

Обратите внимание на то, что наиболее значимый разряд у всех положительных чисел – это «0», в то время как у отрицательных чисел – это «1», то есть наиболее значимый бит дополнительного кода можно рассматривать как аналог знакового бита прямого кода. Однако на этом сходство кончается, поскольку остальные биты дополнительного кода интерпретируются не так, как биты прямого кода.

В случае дополнительного кода, знак отрицательного двоичного числа изменяется на противоположный путем выполнения специальной операции, называемой *дополнением до двух (taking the two's complement)*. Суть этой операции заключается в том, что инвертируются все биты этого числа, а затем к значению наименее значимого бита прибавляется 1. Подобная операция позволяет найти двоичное представление отрицательного числа или определить его абсолютное значение.

Пример 1.10 ПРЕДСТАВЛЕНИЕ ОТРИЦАТЕЛЬНЫХ ЧИСЕЛ В ДОПОЛНИТЕЛЬНОМ КОДЕ

Найти представление -2_{10} как 4-битового числа в дополнительном коде.

Решение: начните с $+2_{10} = 0010_2$. Для получения -2_{10} инвертируйте биты и добавьте единицу. Инвертируя 0010_2, получим 1101_2. $1101_2 + 1 = 1110_2$. Итак, -2_{10} равно 1110_2.

Пример 1.11 ЗНАЧЕНИЕ ОТРИЦАТЕЛЬНЫХ ЧИСЕЛ В ДОПОЛНИТЕЛЬНОМ КОДЕ

Найти десятичное значение числа 1001_2 в дополнительном коде.

Решение: Число 1001_2 имеет старшую 1, поэтому оно должно быть отрицательным. Чтобы найти его модуль, инвертируем все биты и добавляем 1. Инвертируя 1001_2, получим 0110_2. $0110_2 + 1 = 0111_2 = 7_{10}$. Отсюда, $1001_2 = -7_{10}$.

Неоспоримым преимуществом дополнительного кода является то, что привычный способ сложения работает как в случае положительных, так и отрицательных чисел. Напомним, однако, что при сложении *N*-битных чисел *N*-ный бит (т.е. *N* + 1-й бит результата) не переносится.

Пример 1.12 СЛОЖЕНИЕ ЧИСЕЛ, ПРЕДСТАВЛЕННЫХ В ДОПОЛНИТЕЛЬНОМ КОДЕ

Вычислить (a) $-2_{10} + 1_{10}$ и (b) $-7_{10} + 7_{10}$ с помощью чисел в дополнительном коде

Решение: (a) $-2_{10} + 1_{10} = 1110_2 + 0001_2 = 1111_2 = -1_{10}$. (b) $-7_{10} + 7_{10} = 1001_2 + 0111_2 = 10000_2$. Пятый бит отбрасывается, оставляя правильный 4-битовый результат 0000_2.

Вычитание одного двоичного числа из другого осуществляется путем преобразования вычитаемого в дополнительный код и последующего его сложения с уменьшаемым.

Пример 1.13 ВЫЧИТАНИЕ ЧИСЕЛ В ДОПОЛНИТЕЛЬНОМ КОДЕ

Вычислить (a) $5_{10} - 3_{10}$ и (b) $3_{10} - 5_{10}$, используя 4-разрядные числа в дополнительном коде.

Решение:

(a) $3_{10} = 0011_2$. Вычисляя его дополнительный код, получим $-3_{10} = 1101_2$. Теперь сложим $5_{10} + (-3_{10}) = 0101_2 + 1101_2 = 0010_2 = 2_{10}$. Отметим, что перенос из наиболее значимой позиции сбрасывается, поскольку результат записывается в четырех битах.

b) Вычисляя дополнительный код от 5_{10}, получим $-5_{10} = 1011_2$.

Теперь сложим $3_{10} + (-5_{10}) = 0011_2 + 1011_2 = 1110_2 = -2_{10}$.

Дополнение нуля до двух также производится путем инвертирования всех битов (это дает $11\ldots111_2$) и последующим прибавлением 1, что делает значения всех битов равным 0. При этом перенос наиболее значимого бита игнорируется. В результате, нулевое значение всегда представлено набором только нулевых битов. В отличие от прямого кода дополнительный код не имеет отрицательного нуля. Нуль всегда считается положительным числом, так как его знаковый бит всегда 0.

Так же, как и двоичное число без знака, произвольное *N*-битное число, записанное в дополнительном коде, может принимать одно из 2^N возможных значений. Однако, весь этот диапазон разделен между

положительным и отрицательным числами. Например, 4-битное двоичное число без знака может принимать 16 значений от 0 до 15. В случае дополнительного кода, 4-битное число также принимает 16 значений, но уже от −8 до 7. В общем случае, диапазон *N*-битного числа, записанного в дополнительном коде, охватывает $[-2^{N-1}, 2^{N-1} - 1]$. Легко понять, почему в отрицательном диапазоне оказалось на одно значение больше, чем в положительном – в дополнительном коде отсутствует отрицательный нуль. Максимальное отрицательное число, которое можно записать, используя дополнительный код $10\ldots000_2 = -2^{N-1}$, иногда называют *странным числом (weird number)*. Чтобы дополнить это число до двух, инвертируем все его биты (это даст нам $01\ldots111_2$), прибавим 1 и получим в результате $10\ldots000_2$ – опять это же самое «странное» число. То есть, это единственное отрицательное число, которое не имеет положительной пары.

В случае дополнительного кода сложение двух положительных или отрицательных *N*-битовых чисел может привести к переполнению, если результат будет больше, чем $2^{N-1} - 1$, или меньше, чем -2^{N-1}. Сложение положительного и отрицательного числа, напротив, никогда не приводит к переполнению. В отличие от двоичного числа без знака перенос наиболее значимого бита не является признаком переполнения. Вместо этого индикатором переполнения является

ситуация, когда после сложения двух чисел с одинаковым знаком знаковый бит суммы не совпадает со знаковыми битами слагаемых.

Пример 1.14 СЛОЖЕНИЕ ЧИСЕЛ В ДОПОЛНИТЕЛЬНОМ КОДЕ С ПЕРЕПОЛНЕНИЕМ

Вычислить $4_{10} + 5_{10}$, используя четырехбитные числа в дополнительном коде. Произойдет ли переполнение?

Решение: $4_{10} + 5_{10} = 0100_2 + 0101_2 = 1001_2 = -7_{10}$. Результат не помещается в диапазон положительных четырехбитных чисел в дополнительном коде, оказываясь отрицательным. Если бы вычисление выполнялось с пятью или более битами, результат был бы такой $01001_2 = 9_{10}$, что правильно.

В случае необходимости увеличения количества битов произвольного числа, записанного в дополнительном коде, значение знакового бита должно быть скопировано в наиболее значимые разряды модифицированного числа. Эта операция называется знаковым расширением *(sign extension)*. Например, числа 3 и −3 записываются в 4-битном дополнительном коде как 0011 и 1101 соответственно. Если мы увеличиваем число разрядов до семи битов, мы должны скопировать знаковый бит в три наиболее значимых бита модифицированного числа, что дает 0000011 и 1111101.

Сравнение способов представления двоичных чисел

Три наиболее часто использующиеся на практике способа представления двоичных чисел – это двоичные числа без знака, прямой код и дополнительный код. **Табл. 1.3** сравнивает диапазон *N*-битных чисел для каждого из этих трех способов. Преимущества дополнительного кода заключаются в том, что его можно использовать для представления как положительных, так и отрицательных целых чисел, а привычный способ сложения работает для всех чисел, представленных в дополнительном коде. Вычитание осуществляется путем преобразования вычитаемого в отрицательное число (т.е. путем дополнения этого числа до двух) и последующего сложения с уменьшаемым. В дальнейшем в этой книге, если не указано иное, предполагается, что все двоичные числа представлены в дополнительном коде.

Табл. 1.3 Диапазон *N*-битных чисел

Система	Диапазон
Двоичные числа без знака	$[0, 2^N - 1]$
Прямой код	$[-2^{N-1} + 1, 2^{N-1} - 1]$
Дополнительный код	$[-2^{N-1}, 2^{N-1} - 1]$