С. Л. Сергеев

Архитектуры вычислительных систем

bhv®

С. Л. Сергеев

Архитектуры вычислительных систем

Допущено УМО по классическому университетскому образованию
в качестве учебника для студентов высших учебных заведений,
обучающихся по направлению ВПО 010400
«Информационные технологии»

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.06
ББК 32.973.26-018.2
С32

Сергеев С. Л.

С32 Архитектуры вычислительных систем: учебник. — СПб.: БХВ-Петербург, 2010. — 240 с.: ил. — (Учебная литература для вузов)

ISBN 978-5-9775-0575-8

В учебнике рассмотрена архитектура компьютера на уровне системы команд и адресов. Изложение опирается на минимальное понимание работы "железа" и операционных систем, от читателя требуется лишь знание четырех действий арифметики. Описаны представление данных, диапазон и точность, системы счисления, коды чисел, разновидности команд передачи управления, структура циклов, методы организации переменных адресов. Подробно рассмотрены структура подпрограмм, организация вызова и возврата, методы передачи параметров и сохранения регистров и соответствующие им команды. Описаны конвейер команд и связанные с ним проблемы. Представлены современные направления развития архитектур: RISC- и CISC-процессоры, архитектуры со словом сверхбольшой длины.

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Фото	*Кирилла Сергеева*
Зав. производством	*Николай Тверских*

Рецензенты:

Е. И. Веремей, д. ф.-м. н., профессор, заведующий кафедрой компьютерных технологий и систем факультета прикладной математики — процессов управления Санкт-Петербургского государственного университета

В. А. Кузнецов, д. т. н., профессор кафедры прикладной математики и кибернетики Петрозаводского государственного университета

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.06.10.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 19,35.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0575-8

Оглавление

Введение

Архитектура ЭВМ — понятие обширное и расплывчатое. В различных разделах компьютерных наук (архитектуры ЭВМ, базы данных, операционные системы) принято описывать объект или технологию на двух уровнях: физическом и логическом. На физическом уровне объект описывается так, как он видится разработчику, на логическом — так, как он видится пользователю. Современный компьютер — это электронное ядро ("железо"), окруженное большим числом различных оболочек. Система команд, транслятор, операционная система — некоторые из них. И у каждой оболочки есть свой разработчик и свой пользователь. Разработчикам этих оболочек и их пользователям компьютер видится по-разному. Таким образом, вместо двух уровней рассмотрения появляется несколько. Иногда, например, предлагают разделить рассмотрение компьютера на шесть уровней (не считая "железа"): цифровой логический, микроархитектурный, архитектуры команд, операционной системы, языка ассемблера, языка высокого уровня.

В данной книге рассматривается уровень системы команд и адресов. Разумеется, деление на уровни условно. Кроме того, автор стремился к тому, чтобы читателям были понятны причины, породившие ту или иную деталь архитектуры системы команд и адресов. Это потребовало местами спускаться на более низкий уровень, разделяя исполнение команд на отдельные этапы, а местами подниматься на более высокий уровень — уровень операционной системы, рассматривая команды ввода/вывода и виртуальную память. И все же, изложение сосредоточено в области системы команд и опирается лишь на минимальное понимание работы "железа" и операционных систем, которое можно получить целиком из материала книги. От читателя не требуется никаких предварительных знаний, кроме четырех действий арифметики.

Еще раз подчеркнем: главная задача книги — не просто рассказать об устройстве компьютера, но и объяснить, почему он так устроен.

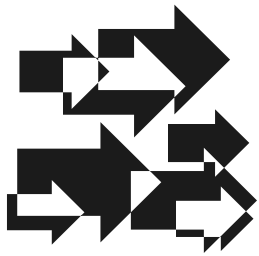
Книга состоит из семи глав.

- *Глава 1 "Представление данных в компьютере"*. В ней рассматриваются различные системы счисления, главным образом, двоичная, восьмеричная и шестнадцатеричная. Даются и обосновываются правила перевода чисел из одной системы счисления в другую. Описываются различные коды чисел, используемые в компьютерах. Показываются преимущества дополнительного и смещенного кодов. Исследуются проблемы диапазона и точности представления чисел. Показывается их взаимозависимость. Рассматриваются способы кодирования символов.
- *Глава 2 "Компьютерные вычисления"*. В ней рассматриваются всевозможные компьютерные операции с кодами чисел и символов (главным образом, операции математической логики, сдвиги и арифметические операции). Цель этой главы — описать все встречающиеся в компьютерах операции арифметико-логического типа (хотя все одновременно они вряд ли встречаются в каком-либо компьютере). Арифметические операции описываются как последовательность отдельных шагов. Это важно для понимания работы конвейера команд.
- *Глава 3 "Команды арифметико-логического типа и адресация"*. Цель этой главы — описать всевозможные модификации команд арифметико-логического типа, имеющиеся в машинах с разной адресностью и с разными способами формирования адресов. Приводится принципиальная схема компьютера в целом и принципиальная схема процессора, описываются основные этапы выполнения команды арифметико-логического типа, подробно рассматривается выполнение команд в машинах разной адресности. Анализируются достоинства машин с регистрами общего назначения и возникающие в них проблемы. Описываются прямой, косвенный, непосредственный и относительный адреса и особенности распределения памяти, связанные с использованием двух последних. Рассматривается также адресация устройств внешней памяти.
- *Глава 4 "Команды передачи управления и циклы"*. В начале главы приводятся разновидности простых команд передачи управления, выполняющих единственное действие — безусловную или условную передачу управления. Далее исследуются циклы — разновидности циклов и их структура. Подробно рассматривается важная разновидность циклов — с переменными командами и со счетчиками. Описывается структура таких циклов и методы организации переменных адресов. Во многих машинах имеются сложные команды управления циклом, совмещающие организацию передачи управления с продвижением индекса и/или со счетчиком. В заключение главы рассматриваются такие команды.
- *Глава 5 "Подпрограммы и ввод/вывод"*. Подпрограммы, наряду с циклами, являются главной структурной единицей программ на нижнем уровне.

В главе подробно рассматриваются проблемы взаимодействия подпрограмм с главной программой: организация вызова и возврата, различные методы передачи параметров и сохранения регистров и соответствующие им команды. Обсуждается проблема настройки подпрограмм по месту и главный прием такой настройки — базирование. Операции ввода/вывода обычно производят обмен сразу большим количеством слов или байтов и потому выполняются как цикл. Обычно они выполняются с помощью подпрограмм, исполняемых процессором или контроллером. Это делает логичным включение операций ввода/вывода в данную главу. В конце главы рассматриваются варианты организации ввода/вывода — программно управляемый, по прерываниям и прямой доступ к памяти.

- *Глава 6 "Параллельность работы и иерархия памяти".* В предыдущих главах показано, что адрес памяти зачастую определяется сложно — комбинацией базирования, индексирования и смещения. В этой главе описываются еще более сложные способы организации оперативной памяти — сегментная и страничная. Обсуждаются достоинства иерархической памяти, особенно ярко проявляющиеся в многозадачном режиме или в однозадачном, при высоких требованиях к объему памяти. Описывается также развитие идеи иерархии памяти — кэш-память. Рассматриваются разновидности кэша, методы (политики) чтения и записи.
- *Глава 7 "Организация процессора".* Описывается конвейер команд. Обсуждаются его достоинства и проблемы, с которыми он сталкивается: задержки работы устройств, конфликты по ресурсам и по данным, проблемы разгона конвейера и предсказания переходов. В конце главы обсуждаются варианты аппаратного решения проблем конвейера — RISC- и CISC-процессоры, архитектуры со словом сверхбольшой длины.

ГЛАВА 1



Представление данных в компьютере

1.1. Системы счисления

1.1.1. P-ичная система счисления

Система счисления — это совокупность правил, позволяющих считать и кратко записывать числа. Подразумевается, что форма записи чисел должна быть удобной для выполнения арифметических операций. В повседневной жизни мы используем десятичную систему счисления (можно также сказать: систему счисления с основанием десять). По аналогии с привычной для нас десятичной системой можно определить и другие системы счисления.

Опишем, например, восьмеричную. Ее базу составляют 8 символов (цифр) — 0, 1, 2, 3, 4, 5, 6, 7. (В десятичной системе базу составляют 10 символов — от 0 до 9.) Любое число в восьмеричной системе изображается только с помощью символов базы. (И точки, разделяющей целую и дробную части числа.)

$$b_n b_{n-1} \ldots b_1 . b_0 b_{-1} \ldots b_{-m}. \qquad (1.1)$$

Выражение (1.1), составленное из символов восьмеричной базы, обозначает число

$$b_n 8^n + b_{n-1} 8^{n-1} + \ldots + b_1 8^1 + b_0 8^0 + b_{-1} 8^{-1} + \ldots + b_{-m} 8^{-m}. \qquad (1.2)$$

Аналогично строятся и другие системы. Например, четверичная система счисления имеет базу из четырех символов: 0, 1, 2, 3. Любое число в ней изображается с помощью только этих символов, имеет вид (1.1) и обозначает

$$b_n 4^n + b_{n-1} 4^{n-1} + \ldots + b_1 4^1 + b_0 4^0 + b_{-1} 4^{-1} + \ldots + b_{-m} 4^{-m}.$$

Рассмотрим, к примеру, выражение $x = 2301.21$. Его можно интерпретировать и как десятичное, и как восьмеричное, и как четверичное число.

Как десятичное:

$$x = 2\times10^3 + 3\times10^2 + 0\times10^1 + 1\times10^0 + 2\times10^{-1} + 1\times10^{-2}.$$

Как восьмеричное:

$$x = 2\times8^3 + 3\times8^2 + 0\times8^1 + 1\times8^0 + 2\times8^{-1} + 1\times8^{-2}.$$

Как четверичное:

$$x = 2\times4^3 + 3\times4^2 + 0\times4^1 + 1\times4^0 + 2\times4^{-1} + 1\times4^{-2}.$$

Очевидно, это совершенно разные числа, хотя и выглядят одинаково. Во избежание путаницы там, где из контекста не ясно, в какой системе счисления записано число, оно снабжается индексом, например,

$$x = 2\times8^3 + 3\times8^2 + 0\times8^1 + 1\times8^0 + 2\times8^{-1} + 1\times8^{-2} = 2301.21_8.$$

Обобщая, можно сказать, что система счисления с основанием p ($2 \le p \le 10$) имеет базу 0, 1, 2, ..., $p-1$. Любое число в этой системе записывается в виде $b_n b_{n-1}\ldots b_1 b_0 . b_{-1}\ldots b_{-m}$, где $0 \le b_i \le p-1$, и интерпретируется как

$$b_n p^n + b_{n-1}p^{n-1} + \ldots + b_1 p^1 + b_0 p^0 + b_{-1}p^{-1} + \ldots + b_{-m}p^{-m}.$$

Надо заметить, что в качестве символов базы не обязательно использовать цифры. Можно использовать любые другие значки. Но все равно эти значки должны быть заменителями слов "ноль", "один" и т. д. Понимание этого позволяет достичь еще большего обобщения.

Пусть система с основанием $p > 0$ (теперь p может быть и больше 10) имеет базу a_0, a_1, ..., a_{p-1} — p различных символов. Тогда любое число, записанное в этой системе $b_n b_{n-1}\ldots b_1 b_0 . b_{-1}\ldots b_{-m}$, где все b_i есть символы из базы, интерпретируется как

$$b_n p^n + b_{n-1}p^{n-1} + \ldots + b_1 p^1 + b_0 p^0 + b_{-1}p^{-1} + \ldots + b_{-m}p^{-m}.$$

Символы базы могут быть произвольны, но обычно используют цифры от 0 до $p-1$, если $p \le 10$, или от 0 до 9 плюс дополнительные символы (если $p > 10$). В частности для шестнадцатеричной системы используют символы: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Правила счета в p-ичной системе такие же, как в десятичной. Надо только помнить, что после старшего символа базы идет число 10. Например, в восьмеричной: 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, ..., 17, 20, ..., 27, 30, ..., 77, 100, 101, ...

В шестнадцатеричной: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, ..., 19, 1A, 1B, ..., 1F, 20, ..., 99, 9A, 9B, ..., 9F, A0, A1, ..., FF, 100, 101, ...

В двоичной: 0, 1, 10, 11, 100, 101, 110, 111, 1000, ...

Арифметические операции с числами в p-ичных системах выполняются по тем же правилам, что и в десятичной, только используются свои таблицы сложения и умножения. Пример для $p = 2$ представлен на рис. 1.1.

x \ y	0	1
0	0	1
1	1	10

x \ y	0	1
0	0	0
1	0	1

Рис. 1.1. Таблицы сложения и умножения для $p = 2$

Для $p = 4$ — рис. 1.2.

x \ y	0	1	2	3
0	0	1	2	3
1	1	2	3	10
2	2	3	10	11
3	3	10	11	12

x \ y	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	10	12
3	0	10	12	21

Рис. 1.2. Таблицы сложения и умножения для $p = 4$

1.1.2. Правило перевода из одной системы счисления в другую

Целые

Перевести целое число x в p-ичную систему, значит представить его в виде

$$b_n b_{n-1} \ldots b_1 b_0 ,$$

где b_i — символы из базы p-ичной системы. Для этого надо найти многочлен

$$x = b_n p^n + b_{n-1} p^{n-1} + \ldots + b_1 p + b_0 .$$

Очевидно, можно записать этот многочлен в виде

$$x = x_1 p + b_0,$$

где

$$x_1 = b_n p^{n-1} + b_{n-1} p^{n-2} + \ldots + b_1.$$

Следовательно, b_0 — младшая цифра p-ичного представления числа x и может быть получена как остаток от деления x на p.

Частное — x_1 представим, в свою очередь, как сумму

$$x_1 = x_2 p + b_1.$$

Следовательно, следующая, вторая справа цифра p-ичного представления числа x есть остаток от деления x_1 на p. Продолжая делить каждое новое частное на p, будем получать все новые остатки, являющиеся последовательными (справа налево) цифрами числа x в p-ичной системе. Последнее частное, меньшее p, будет старшей цифрой x.

Пример 1. Перевод числа 1995_{10} в восьмеричную систему (рис. 1.3).

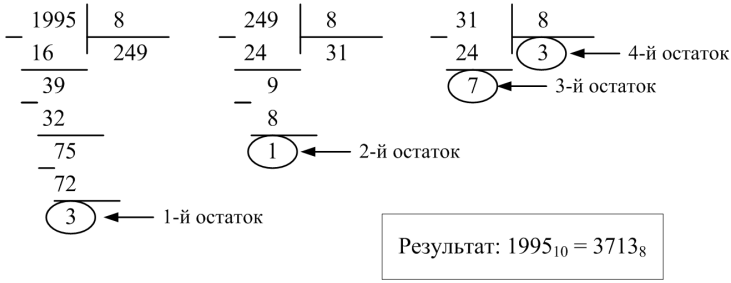


Рис. 1.3

Пример 2. Перевод того же числа в шестнадцатеричную систему (рис. 1.4).

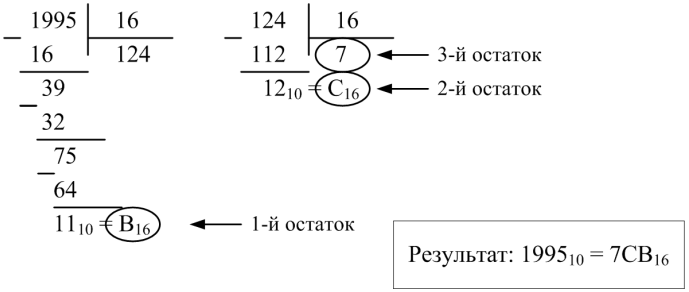


Рис. 1.4

Числа, меньшие единицы

Переведем теперь число y, меньшее 1, из десятичной системы в p-ичную. Будем искать y в виде многочлена

$$y = b_{-1}p^{-1} + b_{-2}p^{-2} + b_{-3}p^{-3} + \ldots + b_{-m}p^{-m},$$

который определяет p-ичную форму числа y: $y_p = 0.b_{-1}b_{-2}b_{-3}\ldots b_{-m}$.

Преобразуем многочлен к виду

$$y = p^{-1}\left(b_{-1} + y_1\right), \text{ где } y_1 = b_{-2}p^{-1} + b_{-3}p^{-2} + \ldots + b_{-m}p^{-m+1}.$$

Из того, что $y_1 < 1$ заключаем, что произведение $y \times p$ имеет целой частью b_{-1} — первую слева цифру p-ичного представления числа y. Умножая y_1 на p, получим число, целая часть которого — вторая цифра y_p, и т. д.

Заметим, что мы еще только ищем представление y в виде многочлена. Заранее нам не известны не только его коэффициенты, но и степень m. Иногда на каком-то шаге оказывается, что $y_i = 0$. Тогда процесс перевода заканчивается $(i = m)$. Чаще этого не происходит слишком долго, поэтому мы должны решить, сколько коэффициентов будем искать. (Иными словами, с какой точностью требуется выполнить перевод в p-ичную систему.)

Пример 3. Перевод десятичного числа 0.1995 в восьмеричную систему (на рис. 1.5, слева) и в шестнадцатеричную систему (рис. 1.5, справа).

0.	1995	0.	1995
	8		16
1.	5960	3.	1920
	8		16
4.	7680	3.	0720
	8		16
6.	1440	1.	1520

Рис. 1.5

Итак, $0.1995_{10} \approx 0.146111_8 \approx 0.33126\ E_{16}$.

Как следует из выведенного нами правила, число, появившееся перед точкой, в последующем умножении не участвует (отделено вертикальной чертой). Ответ получается при прочтении сверху вниз того, что расположено левее вертикальной черты.

Равенства, разумеется, приближенные, т. к. процессы перевода не закончены и младшие цифры восьмеричного и шестнадцатеричного чисел отброшены. Точные переводы получаются редко.

Пример 4. Перевод числа 0.1025 в восьмеричную (рис. 1.6, слева) и в шестнадцатеричную (рис. 1.6, справа) системы счисления.

0.	1025	0.	1025
	8		16
0.	8200	1.	6400
	8		16
6.	5600	10.	2400
	8		16
4.	4800	3.	8400

Рис. 1.6

Итак, $0.1025_{10} \approx 0.064_{8} \approx 0.1A3_{16}$.

Общее правило перевода

Теперь дадим общее правило перевода чисел из десятичной системы в p-ичную.

Число разбивается на две части — целую и дробную (левее точки и правее). Каждая часть переводится в p-ичную систему по своему правилу — для целых и для дробных. Полученные числа являются целой и дробной частями результата. Их объединяют в одно число (слева от точки и справа).

Пример 5. Перевод числа общего вида — 96.96_{10} в восьмеричную систему (рис. 1.7).

Целая часть

96	8	
8	12	8
16	8	1
16	4	
0		

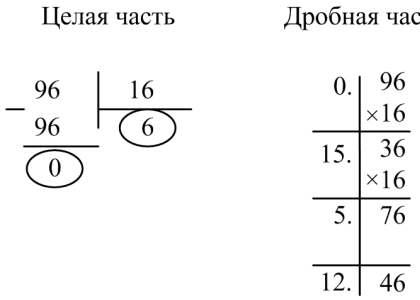
Дробная часть

0.	96
	×8
7.	68
	×8
5.	44

$95_{10} = 140_{8}$ $0.95_{10} \approx 0.75_{8}$ $96.96_{10} \approx 140.75_{8}$

Рис. 1.7

Пример 6. Перевод числа 96.96_{10} в шестнадцатеричную систему (рис. 1.8).



$96_{10} = 60_{16}$ $0.96_{10} \approx 0.F5C_{16}$ $96.96_{10} \approx 60.F5C_{16}$

Рис. 1.8

Для перевода из системы счисления с основанием q в систему с основанием p можно воспользоваться правилом перевода из десятичной системы в p-ичную с единственным уточнением — деление и умножение должны выполняться в q-ичной системе счисления.

В **примере 7** дается перевод числа 140_8 в десятичную систему. $q = 8$, $p = 10$, следовательно, вычисления надо производить в восьмеричной системе (рис. 1.9).

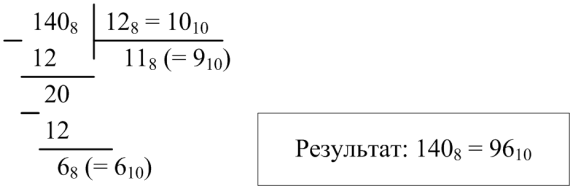


Рис. 1.9

Поскольку такие вычисления очень сложны своей непривычностью, перевод из произвольной системы в десятичную иногда выполняют по более простому правилу. При этом объем вычислений больше, но ведутся они в десятичной системе.

Вот это правило.

От числа $b_n b_{n-1} \ldots b_1 b_0 . b_{-1} b_{-2} \ldots b_{-m}$ переходят к многочлену

$$b_n p^n + b_{n-1} p^{n-1} + \ldots + b_1 p^1 + b_0 p^0 + b_{-1} p^{-1} + b_{-2} p^{-2} + \ldots + b_{-m} p^{-m},$$

значение которого вычисляется в десятичной системе. Например:

$$140.75_8 = 1\times 8^2 + 4\times 8^1 + 0\times 8^0 + 7\times 8^{-1} + 5\times 8^{-2} = 64 + 32 + \frac{7}{8} + \frac{5}{64} \approx 96.959.$$

1.1.3. Двоичная и вспомогательные системы

Для хранения p-ичного n-разрядного числа в памяти компьютера служит ячейка, состоящая из n одинаковых элементов. Каждый элемент — это устройство, способное находиться в одном из p устойчивых состояний. Каждому символу базы ставится в соответствие одно из состояний устройства. Записать символ в элемент ячейки, значит привести этот элемент в соответствующее состояние. Элементы ячейки упорядочены, как и разряды в числе, так что в каждый элемент записывается свой, вполне определенный разряд числа. Элементы ячейки называют также *разрядами*.

Наиболее простые и надежные устройства хранения и переработки информации работают с двоичными числами. Разряды этих устройств — двоичные. Двоичные разряды называют также битами, т. к. в них записывается один бит — минимальная единица информации. Базу двоичной системы составляют символы 0 и 1. В табл. 1.1 приведены некоторые числа в двоичной системе.

Таблица 1.1. Примеры двоичных чисел

Десятичные	0	1	2	3	4	5	6	7
Двоичные	000	001	010	011	100	101	110	111
Десятичные	8	9	10	11	12	13	14	15
Двоичные	1000	1001	1010	1011	1100	1101	1110	1111
Десятичные	16		32		64		1024	
Двоичные	10000		100000		1000000		10000000000	

Из таблицы видно, что двоичные числа значительно длиннее десятичных, но по ним невозможно понять, во сколько раз. Оценим эту величину. Пусть число x состоит из n разрядов в десятичной системе и из m — в двоичной. Это значит, что выполняются неравенства

$$10^{n-1} \le x < 10^n \text{ и } 2^{m-1} \le x < 2^m.$$

Логарифмируя их по основанию 2, получим:

$$(n-1)\times \log_2 10 \le \log_2 x < n\times \log_2 10 \text{ и } m-1 \le \log_2 x < m.$$

$\log_2 x$ удовлетворяет двум неравенствам одновременно, поэтому

$$\max\left[(n-1)\times\log_2 10\ ,\ m-1\right]\le\log_2 x<\min\left[n\times\log_2 10\ ,m\right].$$

Следовательно,

$$(n-1)\times\log_2 10<m\ \text{ и }\ m-1<n\times\log_2 10\ .$$

Отсюда

$$\left(1-\frac{1}{n}\right)\times\log_2 10<\frac{m}{n}<\log_2 10+\frac{1}{n}.$$

При больших n обе границы стремятся к $\log_2 10\approx 3.3$. Таким образом, при больших n двоичные числа длиннее десятичных примерно в 3.3 раза.

Использование двоичной системы счисления добавляет к обычной для недесятичных систем трудности — непривычности, еще одну — очень большую длину чисел. Работа с ними утомительна и увеличивает вероятность ошибок. В то же время преимущества использования двоичной системы в компьютерах столь убедительны, что не позволяют от нее отказаться.

Компромисс найден в использовании одной из вспомогательных систем — восьмеричной или шестнадцатеричной (для каждого типа компьютера выбирается одна из них). Дело в том, что переводы из восьмеричной системы в двоичную и обратно, так же как и из шестнадцатеричной в двоичную и обратно, настолько просты, что могут быть выполнены человеком очень быстро даже в уме. Поэтому используется такая схема: компьютер работает с двоичными числами, а человек читает и записывает их как восьмеричные (или шестнадцатеричные), мысленно осуществляя перевод из одной системы в другую. В этой схеме сохраняется непривычность манипулирования с недесятичными числами, однако длина этих чисел примерно такая же, как и десятичных.

Возьмем произвольное целое двоичное число: $b_n b_{n-1}\ldots b_1 b_0$.

Запишем его в виде многочлена

$$b_n 2^n+b_{n-1}2^{n-1}+\ldots+b_5 2^5+b_4 2^4+b_3 2^3+b_2 2^2+b_1 2^1+b_0 2^0.$$

Рассмотрим сумму трех правых слагаемых: $b_2 2^2+b_1 2^1+b_0 2^0$. Очевидно, что эта сумма лежит в диапазоне от 0 до 7 и может быть заменена одной восьмеричной цифрой. Обозначим ее через c_0.

Возьмем теперь следующие три слагаемых. Вынося за скобки 2^3, замечаем, что сумму в скобках можно заменить одной восьмеричной цифрой. Пусть это будет c_1.

$$b_5 2^5+b_4 2^4+b_3 2^3=\left(b_5 2^2+b_4 2^1+b_3 2^0\right)\times 2^3=c_1\times 8.$$

Следующие три слагаемых будут иметь общим множителем 2^6, а выражение в скобках снова заменим восьмеричным — c_2.

Аналогично будем поступать с каждой следующей тройкой слагаемых. Если число слагаемых в многочлене не кратно трем, добавим слагаемое вида $0\times 2^{n+1}$ или два слагаемых: $0\times 2^{n+2}+0\times 2^{n+1}$ с тем, чтобы последняя группа слагаемых также была тройкой. В результате придем к новому многочлену

$$c_k 8^k + c_{k-1}8^{k-1} + \ldots + c_1 8^1 + c_0 8^0,$$

где k равно $\frac{n}{3}$, $\frac{n+1}{3}$ или $\frac{n+2}{3}$ (выбирается целое).

Но это соответствует восьмеричному числу $c_k c_{k+1}\ldots c_1 c_0$.

Теперь возьмем двоичное число, меньшее единицы $0.b_{-1}b_{-2}b_{-3}\ldots b_{-m}$. Построим соответствующий многочлен:

$$b_{-1}2^{-1} + b_{-2}2^{-2} + b_{-3}2^{-3} + \ldots + b_{-m}2^{-m}.$$

Рассмотрим сумму из трех левых слагаемых. Если в ней вынести за скобки 2^{-3}, то в скобках получится сумма, которую можно заменить одной восьмеричной цифрой. Обозначим ее c_{-1}.

$$b_{-1}2^{-1} + b_{-2}2^{-2} + b_{-3}2^{-3} = \left(b_{-1}2^2 + b_{-2}2^1 + b_{-3}2^0\right)\times 2^{-3} = c_{-1}\times 2^{-3}.$$

В следующих трех слагаемых за скобку вынесем 2^{-6}. Сумму в скобках заменим восьмеричной цифрой c_{-2}.

Аналогично будем действовать с каждой следующей тройкой слагаемых. Если m не кратно трем, добавим слагаемое $0\times 2^{-m-1}$ или два слагаемых: $0\times 2^{-m-1}+0\times 2^{-m-2}$ так, чтобы последняя группа слагаемых также оказалась тройкой. В результате получим новый многочлен: $c_{-1}8^{-1} + c_{-2}8^{-2} + \ldots + c_{-r}8^{-r}$, соответствующий восьмеричному числу $0.c_{-1}c_{-2}\ldots c_{-r}$, где $r = \frac{m}{3}$, $\frac{m+1}{3}$ или $\frac{m+2}{3}$.

Общее правило перевода из двоичной системы в восьмеричную таково:

1. Начиная от десятичной точки, влево и вправо объединяем цифры двоичного числа в тройки (триады), дополняя, при необходимости, число нулями слева и справа.

2. Каждую триаду заменяем соответствующей восьмеричной цифрой по табл. 1.2.

Таблица 1.2. Триады

Двоичная триада	000	001	010	011	100	101	110	111
Восьмеричная цифра	0	1	2	3	4	5	6	7

Правило перевода из восьмеричной системы в двоичную:

1. Каждую восьмеричную цифру заменяем соответствующей триадой (именно триадой, т. е. 0 заменяем на 000, 1 — на 001 и т. д.).
2. Отбрасываем слева и справа лишние нули.

Если в этих двух правилах слова "восьмеричная" заменить на "шестнадцатеричная", "тройки" на "четверки", а "триады" на "тетрады", то получим правила перевода из двоичной системы в шестнадцатеричную и обратно. В табл. 1.3 приведены соответствующие тетрады.

Таблица 1.3. Тетрады

Двоичная тетрада	0000	0001	0010	0011	0100	0101	0110	0111
16-ричная цифра	0	1	2	3	4	5	6	7
Двоичная тетрада	1000	1001	1010	1011	1100	1101	1110	1111
16-ричная цифра	8	9	A	B	C	D	E	F

Примеры перевода из двоичной системы

Рассмотрим перевод числа $x = 1011011101.100111110_2$ в восьмеричную систему.

1. Разбиваем на триады (от точки вправо и влево) — рис. 1.10.
2. Дополняем нулями (рис. 1.11).
3. Заменяем триады восьмеричными цифрами: $x = 1335.474_8$.

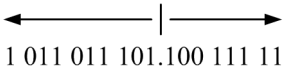


Рис. 1.10

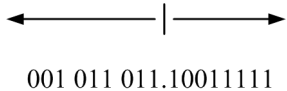
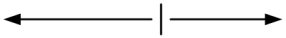


Рис. 1.11

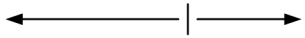
Рассмотрим перевод числа $x = 1011011101.100111110_2$ в шестнадцатеричную систему:

1. Разбиваем на тетрады (от точки вправо и влево) — рис. 1.12.



10 1101 1101.1001 1111

Рис. 1.12



0010 1101 1101.1001 1111

Рис. 1.13

2. Дополняем нулями (рис. 1.13).
3. Заменяем тетрады шестнадцатеричными цифрами: $x = 2DD.9E_{16}$.

Примеры перевода в двоичную систему

Переведем в двоичную систему число $x = 124.124_8$.

1. Заменяем цифры триадами: 001 010 100.001 010 100.
2. Отбрасываем лишние нули: $x = 1010100.\ 0010101_2$.

Переведем в двоичную систему число $x = 124.124_{16}$.

1. Заменяем цифры тетрадами: 0001 0010 0100.0001 0010 0100.
2. Отбрасываем лишние нули: $x = 100100100.\ 000100100 1_2$.

Обычно проблемы перевода из одной системы счисления в другую скрыты от человека, работающего с компьютером: человек передает компьютеру информацию в десятичной системе, компьютер переводит ее в двоичную, обрабатывает в двоичной системе, а перед выдачей результатов человеку переводит их в десятичную систему.

Редко, но все же иногда приходится общаться с компьютером на его языке. При этом человек формулирует входную информацию и получает выходную в восьмеричной или шестнадцатеричной системе. Перевод в двоичную и обратно осуществляется на уровне устройств ввода/вывода: мы нажимаем клавишу, на которой изображена, например, шестнадцатеричная цифра, а клавиатура передает в процессор соответствующую тетраду. И наоборот, из процессора передается двоичный код, а в принтере или на экране монитора каждой тетраде кода соответствует шестнадцатеричная цифра, которая и печатается.

1.1.4. Другие системы счисления

Рассмотренные нами системы счисления называют *системами с неотрицательной базой*, т. к. существуют системы, в которых часть элементов базы отрицательна.

Например, "троичная с симметричной базой". (Имеется в виду симметрия относительно нуля.) В ней элементы базы — 0, 1 и –1 (последнюю цифру обозначают с чертой сверху вместо минуса: $\bar{1}$).

В табл. 1.4 приводятся некоторые числа в этой системе.

Таблица 1.4. Троичная система с симметричной базой

Десятичные	0	1	2	3	4	5	6	7	8	9	10
Троичные	0	1	$1\bar{1}$	10	11	$1\bar{1}\bar{1}$	$1\bar{1}0$	$1\bar{1}1$	$10\bar{1}$	100	101
Десятичные	27		81		243		729				
Троичные	1000		10000		100000		1000000				

Здесь число по-прежнему интерпретируется как многочлен, но коэффициенты могут быть и отрицательными. К примеру, числу $1\bar{1}\bar{1}$ соответствует многочлен $1\times3^2-1\times3^1-1\times3^0=5$, а числу $10\bar{1}$ — многочлен $1\times3^2+0\times3^1-1\times3^0=8$. Важным достоинством троичной системы является единообразие изображения положительных и отрицательных чисел. Здесь не требуется знаков "–" и "+" (табл. 1.5).

Таблица 1.5. Отрицательные числа в троичной системе с симметричной базой

Десятичные	–1	–2	–3	–4	–5	–6	–7
Троичные	$\bar{1}$	$\bar{1}1$	$\bar{1}0$	$\bar{1}\bar{1}$	$\bar{1}11$	$\bar{1}10$	$\bar{1}1\bar{1}$

Просто изменять знак числа: все 1 надо поменять на $\bar{1}$, а $\bar{1}$ на 1. Троичная система даже предлагалась как конкурент двоичной для использования в компьютерах. Двоичная система победила лишь благодаря значительно большей надежности хранения информации в двоичных устройствах.

P-ичной системе счисления не обязательно иметь неотрицательную или симметричную базу. В табл. 1.6 приведены разновидности пятеричных систем счисления (в первом столбце — система с неотрицательной базой, в третьем — с симметричной). Заметим, что только системе с неотрицатель-

ной базой требуется специальный знак для изображения отрицательных чисел.

Таблица 1.6. Различные пятеричные системы

Десятичные числа	База системы			
	0, 1, 2, 3, 4	$\overline{1}$, 0, 1, 2, 3	$\overline{2}$, $\overline{1}$, 0, 1, 2	$\overline{3}$, $\overline{2}$, $\overline{1}$, 0, 1
0	0	0	0	0
1	1	1	1	1
2	2	2	2	$1\overline{3}$
3	3	3	$1\overline{2}$	$1\overline{2}$
4	4	$1\overline{1}$	$1\overline{1}$	$1\overline{1}$
5	10	10	10	10
6	11	11	11	11
7	12	12	12	$1\overline{3}\overline{3}$

Следует подчеркнуть, что основание системы счисления — это вовсе не число символов базы. Система счисления может иметь даже отрицательное основание. Например, система с основанием –2 и базой, состоящей из цифр 0, 1. Числа в этой системе интерпретируются, как и обычно, полиномом:

$$a_n(-2)^n + a_{n-1}(-2)^{n-1} + \ldots + a_1(-2)^1 + a_0(-2)^0.$$

В табл. 1.7 представлены некоторые числа в этой системе счисления.

Таблица 1.7. Система с основанием –2

$p = 10$	0	1	2	3	4	5	6
$p = -2$	0	1	110	111	100	101	11010
$p = 10$	7	8	9	10	11	12	13
$p = -2$	11011	11000	11001	11110	11111	11100	11101

Могут быть системы с иррациональным или даже комплексным основанием.

Все перечисленные системы являются позиционными. В них значение цифры зависит от занимаемой ею позиции в числе. В числе 1991 первая единица

означает одну тысячу, а последняя — одну единицу, первая девятка — девять сотен, а вторая — девять десятков.

Римская система — непозиционная. В ней нет определенной базы и значения цифр не зависят от занимаемого места. Правда, они иногда прибавляются, а иногда вычитаются, так что можно сказать, что в ней присутствует элемент позиционности.

XXXVIII = 10 + 10 + 10 + 5 + 1 + 1 + 1 = 38,

MCMXCI = 1000 – 100 + 1000 – 10 + 100 + 1 = 1991.

Непозиционной является также самая простая из систем счисления — единичная. В ней используется только одна цифра — 1, а значение числа равно количеству единиц: 111 = 3, 111111 = 6.

Есть также целая группа непозиционных систем счисления, которые исследовались на возможность использования в компьютерах. Их называют *системами в остаточных классах* (СОК). В теории чисел доказано, что если m_1, m_2, ..., m_n — попарно взаимно простые числа, то каждому числу x, при $0 \le x < m_1 \times m_2 \times \ldots \times m_n$ взаимно-однозначно соответствует набор остатков от деления x на m_1, m_2, ..., m_n. Пусть, например, $m_1 = 7$, $m_2 = 11$, $m_3 = 13$. $m_1 \times m_2 \times m_3 = 1001$. Любому неотрицательному числу соответствуют три остатка от деления на 7, 11, 13. Причем, если оно не превышает 1000, ни одно другое число из этого диапазона не имеет того же набора остатков. Число 100 в такой системе запишется как 2, 1, 9; число 200 — как 4, 2, 5; число 1000 — как 6, 10, 12.

Интерес к СОК связан с тем, что в этих системах арифметические операции могут выполняться быстрее, чем в позиционных системах. Если, например, в системе с базой m_1, m_2, m_3 два числа представлены остатками x — a_1, a_2, a_3 и y — b_1, b_2, b_3, то числу $z = x + y$ соответствуют остатки c_1, c_2, c_3, где $c_i = a_i + b_i$ или, если $a_i + b_i \ge m_i$, $c_i = a_i + b_i - m_i$. Отсюда видно, что все остатки могут вычисляться независимо, а следовательно, одновременно, что повысит скорость вычислений. Аналогично обстоит дело с другими операциями.

Однако СОК имеют существенные недостатки, среди которых — сложность вычислений с дробными числами, что препятствует их широкому внедрению.

1.2. Представление двоичных чисел

1.2.1. Целые

Будем рассматривать представление двоичных чисел в ячейке памяти, состоящей из n двоичных разрядов. В компьютерах разных типов длина ячеек различна, да и в одном компьютере обычно используются ячейки с разными n. Ячейки с $n = 8$ называются байтами. В примерах будем использовать ячейки с $n = 8, 16, 32$. (Они, по-видимому, наиболее распространены.)

Беззнаковые числа

Ячейки с n двоичными разрядами как раз достаточно для записи n-разрядного целого двоичного числа x (без знака). Так как двоичные числа состоят только из нулей и единиц, наибольшее n-разрядное двоичное число $x_{\max}$ состоит из n единиц (рис. 1.14), а наименьшее — из n нулей.

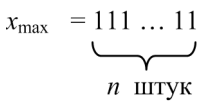


Рис. 1.14

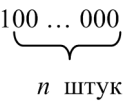


Рис. 1.15

Для быстрого перевода числа, состоящего из n единиц, в десятичную систему достаточно сообразить, что оно на 1 меньше числа, изображенного на рис. 1.15.

Но это число равно 2^n. Таким образом, $x_{\max} = 2^n - 1$. Следовательно, в байте, например, можно записать любое целое без знака от 0 до $2^8 - 1 = 255$, а в двухбайтовой ячейке — от 0 до 65 535.

В ячейках целые числа выравниваются по правому краю. Это выражение означает, что правые границы числа и ячейки совпадают, т. е. разряд единиц числа записывается всегда в крайний правый разряд ячейки. Если число имеет меньше разрядов, чем ячейка, то в левые, свободные разряды ячейки, записываются нули. Например, число 1 записывается в байте как 00000001, число 17_{10} — как 00010001.

Прямой код

При записи в ячейку целых со знаком один разряд приходится отводить для запоминания знака. Обычно знаковым объявляется самый левый разряд ячей-

ки. В знаковый разряд записывается 1, если число отрицательно, 0 — если положительно. Для хранения самого числа остается $n-1$ разряд. Поэтому наибольшее число со знаком имеет вид как на рис. 1.16, что соответствует $2^{n-1}-1$.

$$x_{\max} = \underbrace{011\ \ldots\ 11}_{n-1\ \text{штука}}$$

Рис. 1.16

Для байта это 127, для ячейки из двух байтов — 32 767.

Рассмотренная система кодирования целых со знаком, в которой левый бит отводится под знак, а остальные — под абсолютное значение числа, называется *прямым кодом*. Кроме прямого, в компьютерах используются обратный, дополнительный и смещенный коды.

Обратный код

Положительные числа в *обратном коде* записываются так же, как в прямом. Изменения касаются только отрицательных чисел. Для получения обратного кода отрицательного числа все биты прямого кода (кроме знакового) заменяются на противоположные.

Обратные коды некоторых двухбайтовых чисел приведены в табл. 1.8.

Таблица 1.8. Прямой и обратный коды некоторых чисел

Число	**Прямой код**	**Обратный код**
32 767	0111111111111111	0111111111111111
+5	0000000000000101	0000000000000101
+1	0000000000000001	0000000000000001
–1	1000000000000001	1111111111111110
–5	1000000000000101	1111111111111010
–32767	1111111111111111	1000000000000000

Между прямым и обратным кодами отрицательных чисел имеется соотношение: $|x_{\text{пр}}| + |x_{\text{обр}}| = 2^{n-1} - 1$, где n — длина ячейки в битах.

Если число двухбайтовое, то $|x_{\text{пр}}| + |x_{\text{обр}}| = 111111111111111 = 2^{15} - 1$.

Очевидно, что для получения прямого кода отрицательного числа следует применять ту же процедуру, т. е. обратный код $x_{\text{обр}}$ есть $x_{\text{пр}}$: $(x_{\text{обр}})_{\text{обр}} = x_{\text{пр}}$.

Дополнительный код

Дополнительный код положительных чисел совпадает с их прямым кодом. Для получения дополнительного кода отрицательного числа его сначала переводят из прямого кода в обратный, а затем прибавляют 1. То есть $|x_{\text{доп}}| = |x_{\text{обр}}| + 1$.

Или получают непосредственно из прямого: $|x_{\text{доп}}| = 2^{n-1} - |x_{\text{пр}}|$.

Примеры приведены в табл. 1.9.

Таблица 1.9. Прямой и дополнительный коды

Число	Прямой код	Дополнительный код
32 767	0111111111111111	0111111111111111
+5	0000000000000101	0000000000000101
+1	0000000000000001	0000000000000001
–1	1000000000000001	1111111111111111
–5	1000000000000101	1111111111111011
–32 767	1111111111111111	1000000000000001

Здесь также имеет место формула $(x_{\text{доп}})_{\text{доп}} = x_{\text{пр}}$. С прямым, обратным и дополнительным кодами связана проблема двух нулей. Вот ее суть.

Во многих алгоритмах содержится операция сравнения некоторого результата с нулем. Например, чтобы определить, разделился ли x на y без остатка, надо сравнить остаток с нулем. Так как, по определению, остаток имеет знак делимого, при делении может получиться как остаток +0, так и –0. И хотя +0 = –0, их прямой и обратный коды различны (табл. 1.10).

Таблица 1.10. Прямой и обратный коды нуля

Число	Прямой код	Обратный код
+0	0000000000000000	0000000000000000
–0	1000000000000000	1111111111111111

Поэтому, действуй мы в прямом или в обратном коде, нам придется сравнивать остаток как с одним, так и с другим кодом нуля.

При попытке получить дополнительный код числа –0 из его обратного кода возникнет перенос единицы за пределы ячейки. (Получится число 10000000000000000.) Однако такой результат в дополнительном коде не является переполнением. Выходящая за пределы разрядной сетки единица должна быть просто отброшена. (Это станет ясно после изучения операции сложения в дополнительном коде в *главе 2*.)

Таким образом, дополнительные коды чисел +0 и –0 совпадают:

$$+0 = -0 = 0000000000000000.$$

За счет этого диапазон представимых чисел в дополнительном коде на единицу больше, чем в прямом и обратном (табл. 1.11).

Таблица 1.11. Коды числа –32 768

Число	Прямой код	Обратный код	Дополнительный код
–32 768	Не представимо	Не представимо	1000000000000000

Смещенный код

Недостатком дополнительного кода (как и обратного) является наличие различных правил их формирования для положительных и отрицательных чисел. *Смещенный код* свободен от этого недостатка. Идея смещенного кода заключается в том, чтобы вовсе не иметь знакового бита. Вся ячейка, включая знаковый бит, используется для записи беззнакового кода. Коды положительных и отрицательных чисел образуются по единой формуле:

$$x_{\text{смещ}} = 2^{n-1} + x_{\text{пр}}.$$

В частности для двухбайтовой ячейки $x_{\text{смещ}} = 2^{15} + x_{\text{пр}}$.

В табл. 1.12 приведены примеры чисел в смещенном коде.

Таблица 1.12. Примеры чисел в смещенном коде

Число	Смещенный код
32 767	1111111111111111
+5	1000000000000101
+1	1000000000000001
0	1000000000000000