Алгоритмические трюки для программистов

ИСПРАВЛЕННОЕ ИЗДАНИЕ

Alg_title.indd 1 10.07.2007 17:08:49

Hacker's Delight

Henry S. Warren, Jr.

ADDISON-WESLEY

Boston ♦ San Francisco ♦ New York ♦ Toronto ♦ Montreal
London ♦ Munich ♦ Paris ♦ Madrid
Capetown ♦ Sydney ♦ Tokyo ♦ Singapore ♦ Mexico City

Alg_title.indd 2 10.07.2007 17:08:49

Алгоритмические трюки для программистов

ИСПРАВЛЕННОЕ ИЗДАНИЕ

Генри Уоррен, мл.

Издательский дом "Вильямс"
Москва ♦ Санкт-Петербург ♦ Киев
2007

Alg_title.indd 3 10.07.2007 17:08:50

ББК 32.973.26-018.2.75
У64
УДК 681.3.07

Издательский дом "Вильямс"

Зав. редакцией *С.Н. Тригуб*

Перевод с английского канд. техн наук *И.В. Красикова*, *Г.И. Сингаевской*

Под редакцией канд. техн наук *И.В. Красикова*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com
115419, Москва, а/я 783; 03150, Киев, а/я 152

Уоррен, Генри, С.
У64 Алгоритмические трюки для программистов. : Пер. с англ. — М. : Издательский дом "Вильямс", 2007. — 288с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-0572-7 (рус.)

В этой книге слову "хакер" возвращено его первозданное значение — человека увлеченного, талантливого программиста, способного к созданию чрезвычайно эффективного и элегантного кода. В книге воплощен сорокалетний стаж ее автора в области разработки компиляторов и архитектуры компьютеров. Здесь вы найдете множество приемов для работы с отдельными битами, байтами, вычисления различных целочисленных функций; большей части материала сопутствует строгое математическое обоснование. Каким бы не был ваш профессионализм — вы обязательно найдете в этой книге новое для себя; кроме того, книга заставит вас посмотреть на уже знакомые вещи с новой стороны. Не в меньшей степени эта книга пригодится и начинающему программисту, который может просто воспользоваться готовыми советами из книги, применяя их в своей повседневной практике.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Addison-Wesley Publishing Company, Inc.

Authorized translation from the English language edition published by Addison-Wesley Publishing Company, Inc, Copyright © 2002

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2007

ISBN 978-5-8459-0572-7 (рус.)
ISBN 0-201-91465-4 (англ.)

© Издательский дом "Вильямс", 2007
© Addison-Wesley Publishing Company, Inc, 2002

ОГЛАВЛЕНИЕ

СОДЕРЖАНИЕ

Джозефу У. Гауду (Joseph W. Gaud),
моему школьному учителю алгебры,
который зажег во мне пламя
восхищения математикой

ПРЕДИСЛОВИЕ

Около 30 лет назад, во время первой летней практики, мне посчастливилось участвовать в проекте MAC в Массачусетском технологическом институте. Тогда я впервые смог поработать на компьютере DEC PDP-10, который в то время предоставлял более широкие возможности при программировании на ассемблере, чем любой другой компьютер. Этот процессор имел большой набор команд для выполнения операций с отдельными битами, их тестирования, маскирования, для работы с полями и целыми числами. Несмотря на то что PDP-10 давно снят с производства, до сих пор есть энтузиасты, работающие на этих машинах либо на их программных эмуляторах. Некоторые из них даже пишут новые приложения; по крайней мере в настоящее время есть, как минимум, один Web-узел, поддерживаемый эмулированным PDP-10. (Не смейтесь — поддержка такого узла ни в чем не уступает поддержке антикварного автомобиля "на ходу".)

Тем же летом 1972 года я увлекся чтением серии отчетов по новым исследованиям, публиковавшихся в институте под общим названием HAKMEM — довольно странный и эклектичный сборник разнообразных технических мелочей[1]. Тематика статей могла быть любой: от электронных схем до теории чисел; но больше всего меня заинтриговал небольшой раздел, в котором публиковались программистские хитрости. Практически в каждой статье этого раздела содержалось описание некоторой (зачастую необычной) операции над целыми числами или битовыми строками (например, подсчет единичных битов в слове), которую можно было легко реализовать в виде длинной последовательности машинных команд либо цикла, а затем обсуждалось, как можно сделать это же, используя только четыре, три или две тщательно подобранные команды, взаимодействие между которыми становилось очевидным только после соответствующих объяснений или самостоятельных исследований. Я изучал эти маленькие программные самородки с тем же наслаждением, с которым другие пьют пиво или едят сладости. Я просто не мог остановиться. Каждая такая программа оказывалась для меня находкой, в каждой из них была интеллектуальная глубина, элегантность и даже своеобразная поэзия.

"Наверняка, — думал я тогда, — таких программ должно быть намного больше. Должна же где-то быть книга, посвященная таким штукам".

Книга, которую вы держите в руках, потрясла меня. Автор систематически, на протяжении многих лет собирал такие программные перлы, а теперь свел их воедино, тематически организовал и снабдил каждый четким и подробным описанием. Многие из них можно записать в машинных командах, но книга полезна не только для тех, кто пишет программы на ассемблере. Главная тема книги — рассмотреть базовые структурные отношения среди целых чисел и битовых строк и эффективные приемы реализации операций над ними. Рассмотренные в книге методы столь же полезны и практичны при программировании на языках высокого уровня, например C или Java, как и на языке ассемблера.

Во многих книгах по алгоритмам и структурам данных описаны сложные методы сортировки и поиска, поддержания хэш-таблиц и двоичных деревьев, работы с записями и указателями. Здесь же рассматривается, что можно сделать с очень крошечной частью

[1] Почему HAKMEM? Сокращение от hacks memo; одно 36-битовое PDP-10 слово может содержать шесть 6-битовых символов, поэтому многие имена PDP-10, с которыми тогда работали программисты, были ограничены шестью символами. Аббревиатуры из шести символов широко использовались для сокращенных названий, иногда ими пользовались просто для удобства. Поэтому название сборника HAKMEM имело ясный смысл для специалистов, по крайней мере в то время.

данных — битами и массивами битов. Поразительно, сколько всего можно сделать, используя только операции двоичного сложения и вычитания вместе с некоторыми поразрядными операциями. Операции с переносом позволяют одному биту воздействовать на все биты, находящиеся слева от него, что делает сложение особенно мощной операцией при работе с данными способами, не получившими широкого распространения.

Сейчас у вас в руках книга именно о таких методах. Если вы работаете над оптимизирующим компилятором или просто создаете высокопроизводительный код, вам просто необходимо прочитать эту книгу. Возможно, в своей повседневной работе вы не часто будете использовать изложенные здесь приемы, но, если вам потребуется организовать цикл по всем битам слова, ускорить работу с отдельными битами во внутреннем цикле или вы просто столкнетесь с ситуацией, когда код получается слишком сложным, загляните в эту книгу — я уверен, вы найдете в ней помощь в решении стоящей перед вами проблемы. В любом случае чтение этой книги доставит вам огромное удовольствие.

Гай Л. Стил, мл. (Guy L. Steele, Jr.)
Берлингтон, Массачусетс
Апрель 2002

ВСТУПЛЕНИЕ

Caveat emptor[2]*: стоимость сопровождения программного обеспечения пропорциональна квадрату творческих способностей программиста.*

(Первый закон творческого программирования,
Роберт Д. Блис (Robert D. Bliss), 1992)

Перед вами сборник программных приемов, которые я собирал много лет. Большинство из них работают только на компьютерах, в которых целые числа представлены в дополнительном коде. Хотя в данной книге речь идет о 32-битовых машинах с соответствующей длиной регистра, бо́льшую часть представленных здесь алгоритмов легко перенести на машины с другими размерами регистров.

В этой книге не рассматриваются сложные вопросы наподобие методов сортировки или оптимизации компилируемого кода. Основное внимание уделено приемам работы с отдельными машинными словами или командами, например подсчету количества единичных битов в заданном слове. В подобных приемах часто используется смесь арифметических и логических команд.

Предполагается, что прерывания, связанные с переполнением целых чисел, замаскированы и произойти не могут. Программы на C, Fortran и даже Java работают в таком окружении, но программистам на Pascal и ADA следует быть осторожными!

Представление материала в книге неформальное. Доказательства приводятся только в том случае, если алгоритм неочевиден, а иногда не приводятся вообще. Все методы используют компьютерную арифметику, базовые функции, комбинации арифметических и логических операций и др., а доказательства теорем в этой предметной области часто сложны и громоздки.

Чтобы свести к минимуму количество типографских ошибок и опечаток, многие алгоритмы реализованы на языке программирования высокого уровня, в качестве которого используется C. Это обусловлено его распространенностью и тем, что он позволяет непосредственно комбинировать операции с целыми числами и битовыми строками; кроме того, компилятор языка C генерирует объектный код высокого качества.

Ряд алгоритмов написан на машинном языке. В книге применяется трехадресный формат команд, главным образом для повышения удобочитаемости. Использован язык ассемблера для некой абстрактной машины, которая является представителем современных RISC-компьютеров.

Отсутствие ветвлений в программе всячески приветствуется. Это связано с тем, что на многих машинах наличие ветвлений замедляет выборку команд и блокирует их параллельное выполнение. Кроме того, наличие ветвлений может препятствовать выполнению оптимизации компилятором. Оптимизирующий компилятор более эффективно работает с несколькими большими блоками кода, чем с множеством небольших.

Крайне желательно использовать малые величины при непосредственном задании операнда, сравнение с 0 (а не с другими числами) и параллелизм на уровне команд. Хотя программу

[2] Caveat emptor (лат.) – да будет осмотрителен покупатель.

часто можно значительно сократить за счет использования поиска в таблице, этот метод не слишком распространен. Дело в том, что по сравнению с выполнением арифметических команд загрузка данных из памяти занимает намного больше времени, а методы поиска в таблице зачастую не представляют большого интереса (хотя и весьма практичны).

Напоследок мне хотелось бы напомнить исходное значение слова хакер[3]. Хакер — это страстный любитель компьютеров, он создает что-то новое, переделывает или совершенствует то, что уже есть. Хакер очень хорошо разбирается в том, что делает, хотя часто не является профессиональным программистом или разработчиком. Обычно хакер пишет программы не ради выгоды, а ради собственного удовольствия. Такая программа может оказаться полезной, а может остаться всего лишь игрой интеллекта. Например, хакер может написать программу, которая во время выполнения выводит точную копию себя самой[4]. Таких людей называют хакерами, и эта книга написана именно для них, а не для тех, кто хочет получить совет о том, как взломать что-либо в компьютере.

Благодарности

Прежде всего я хочу поблагодарить Брюса Шрайвера (Bruce Shriver) и Денниса Аллисона (Dennis Allison), оказавших мне помощь в опубликовании книги. Я признателен коллегам из IBM, многие из которых упомянуты в библиографии. Особой благодарности достоин Мартин Хопкинс (Martin E. Hopkins) из IBM, которого по праву можно назвать "мистер Компьютер". Этот человек неудержим в своем стремлении подсчитать в программе каждый такт, и многие его идеи нашли отражение в этой книге. Благодарю также обозревателей из Addison-Wesley, которые значительно улучшили мою книгу. Со многими из них я не знаком, но не могу не упомянуть выдающийся 50-страничный обзор одного из них, Гая Л. Стила, мл. (Guy L. Steele, Jr.), где, в частности, затронуты вопросы перемешивания битов, обобщенного упорядочения и многие другие, которые обязательно войдут во второе издание книги (☺). Ряд предложенных им алгоритмов использован в данной книге. Помогла мне и его пунктуальность. Например, я ошибочно написал, что шестнадцатеричное число 0xAAAAAAAA можно разложить на множители 2·3·17·257·65537; Гай указал, что 3 необходимо заменить на 5. Он не упустил ни одной из подобных мелочей и намного улучшил стиль изложения материала. Кроме того, весь материал, связанный с методом "параллельного префикса", появился в книге исключительно благодаря Гаю.

Г.С. Уоррен мл. (H.S. Warren, Jr.)
Йорктаун, Нью-Йорк
Февраль 2002

[3] В последнее время "хакерами" часто называют тех, кто получает несанкционированный доступ к банковским системам, взламывает Web-узлы и ведет прочую разрушительную деятельность либо ради получения денег, либо для демонстрации всем своей "крутости". К настоящим хакерам таковые не имеют никакого отношения. — *Прим. перев.*

[4] Самая короткая такая программа на C, известная автору, написана Владом Таировым и Рашидом Фахреевым и содержит всего 64 символа:

```
main(a){printf(a,34,a="main(a){printf(a,34,a=%c%s%c,34);}",34);}
```

ГЛАВА 1

ВВЕДЕНИЕ

1.1. Система обозначений

Повсюду в книге при описании машинных команд используются выражения, отличающиеся от обычных арифметических выражений. В "компьютерной арифметике" операнды представляют собой битовые строки (или векторы) некоторой фиксированной длины. Выражения компьютерной арифметики похожи на выражения обычной арифметики, но в этом случае переменные обозначают содержимое регистров компьютера. Значение выражения компьютерной арифметики представляет собой строку битов без какой-либо специальной интерпретации; операнды же могут интерпретироваться поразному. Например, в команде сравнения операнды интерпретируются либо как двоичные целые числа со знаком, либо как беззнаковые целые числа. Поэтому, в зависимости от типа операндов, для обозначения оператора сравнения используются разные символы.

Основное отличие между обычными арифметическими действиями и компьютерными сложением, вычитанием и умножением состоит в том, что результат действий компьютерной арифметики приводится по модулю 2^n, где n — размер машинного слова. Еще одно отличие состоит в том, что компьютерная арифметика содержит большее количество операций. Кроме основных четырех арифметических действий, машина способна выполнить множество других команд: логические *и* (*and*), *исключающее или*, *сравнение*, *сдвиг влево* и др.

Если не оговорено иное, везде в книге предполагается, что длина слова равна 32 битам, а целые числа со знаком представляются в дополнительном к 2 коде. Если длина слова иная, такие случаи оговариваются особо.

Все выражения компьютерной арифметики записываются аналогично обычным арифметическим выражениям, с тем отличием, что переменные, обозначающие содержимое регистров компьютера, выделяются полужирным шрифтом (это обычное соглашение, принятое в векторной алгебре). Машинное слово интерпретируется как вектор, состоящий из отдельных битов. Константы также выделяются полужирным шрифтом, если обозначают содержимое регистра (в векторной алгебре аналогичного обозначения нет, так как там для записи констант используется только один способ — указание компонентов вектора). Если константа означает часть команды (например, непосредственно значение в команде сдвига), то она не выделяется.

Если оператор, например "+", складывает выделенные полужирным шрифтом операнды, то он подразумевает машинную операцию сложения ("векторное сложение"). Если операнды не выделены, то подразумевается обычное арифметическое сложение скалярных величин (скалярное сложение). Переменная x обозначает арифметическое значение выделенной полужирным шрифтом переменной $\boldsymbol{x}$ (интерпретируемое как знаковое или беззнаковое, что должно быть понятно из контекста). Таким образом, если $\boldsymbol{x} = \mathbf{0x80000000}$, а $\boldsymbol{y} = \mathbf{0x80000000}$, то при знаковой интерпретации $x = y = -2^{31}$, $x + y = -2^{32}$ и $\boldsymbol{x} + \boldsymbol{y} = \mathbf{0}$. (Здесь **0x80000000** — шестнадцатеричная запись строки битов, состоящей из одного единичного бита и следующих за ним 31 нулевого бита.)

Биты нумеруются справа налево, причем крайний справа (младший) бит имеет номер 0. Длины бита, полубайта, байта, полуслова, слова и двойного слова равны соответственно 1, 4, 8, 16, 32 и 64 бит.

Небольшие и простые фрагменты кода представлены в виде алгебраических выражений с оператором присваивания (стрелка влево) и при необходимости с оператором `if`. Для таких задач использование математических выражений предпочтительнее, чем составление программы на языке ассемблера.

Более сложные и громоздкие алгоритмы представлены в виде программ на языке C++, причем свойства C++ как объектно-ориентированного языка программирования нигде в книге не используются. По сути, программы написаны на ISO (ANSI) C, но с комментариями в стиле C++. Там, где различия между C и C++ не существенны, подразумевается просто язык C.

Полное описание языка C выходит за рамки данной книги. Однако для тех читателей, которые не знакомы с C, в табл. 1.1 приведено краткое описание основных элементов этого языка [32]. Это описание особенно полезно для читателей, которые не знакомы с C, но знают какой-либо иной процедурный язык программирования. В таблице перечислены также операторы, которые используются в алгебраических и арифметических выражениях этой книги. Все операторы языка C упорядочены по приоритетам: в начале таблицы расположены операторы, имеющие наиболее высокий приоритет, в конце — операторы с самым низким приоритетом. Буква L в столбце "Приоритет" означает, что данный оператор левоассоциативен, а именно:

$$a \bullet b \bullet c = (a \bullet b) \bullet c.$$

Буква R означает, что оператор правоассоциативен. Приведенные в таблице и используемые в книге алгебраические операторы следуют правилам ассоциативности и приоритетам операторов языка C.

В дополнение к перечисленному в табл. 1.1 используется ряд обозначений из булевой алгебры и стандартной математики (соответствующие пояснения приводятся по мере необходимости).

Таблица 1.1. Элементы языка C и алгебраические операторы

Приоритет	C	Алгебра	Описание
	`0x...`	**0x**…, **0b**…	Шестнадцатеричные, двоичные константы
16	`a[k]`		Выбор k-го компонента
16		$\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots$	Различные переменные или выборка битов (зависит от контекста)
16	`f(x,...)`	$f(\boldsymbol{x},\ldots)$	Вычисление функции
16		$\mathrm{abs}(\boldsymbol{x})$	Абсолютное значение (однако $\mathrm{abs}\left(-2^{31}\right) = -2^{31}$)
16		$\mathrm{nabs}(\boldsymbol{x})$	Отрицательное абсолютное значение
15	`x++, x--`		Прибавление, вычитание единицы (постфиксная форма, постинкремент, постдекремент)
14	`++x, --x`		Прибавление, вычитание единицы (префиксная форма, преинкремент, предекремент)
14	(*type name*) `x`		Приведение типа
14 R		$\boldsymbol{x}^k$	*x* в степени *k*

Окончание табл. 1.1

Приоритет	C	Алгебра	Описание
14	`~x`	$\neg x, \overline{x}$	Побитовое *не* (побитовое дополнение до единицы)
14	`!x`		Логическое *отрицание* (`if` $x = 0$ `then` 1 `else` 0)
14	`-x`	$-x$	Арифметическое отрицание
13 L	`x*y`	$x * y$	Умножение по модулю размера слова
13 L	`x / y`	$x \div y$	Знаковое целочисленное деление
13 L	`x / y`	$x \overset{u}{\div} y$	Беззнаковое целочисленное деление
13 L	`x % y`	$\mathrm{rem}(x, y)$	Остаток (может быть отрицательным) от деления ($x \div y$) знаковых аргументов
13 L	`x % y`	$\mathrm{remu}(x, y)$	Остаток от деления ($x \overset{u}{\div} y$) беззнаковых аргументов
		$\mathrm{mod}(x, y)$	Приведение x по модулю y к интервалу $[0,\ \mathrm{abs}(y)-1]$; знаковые аргументы
12 L	`x + y, x - y`	$x + y,\ x - y$	Сложение, вычитание
11 L	`x << y, x >> y`	$x \ll y, x \overset{u}{\gg} y$	Сдвиг влево, вправо с заполнением нулями (“логический” сдвиг)
11 L	`x >> y`	$x \overset{s}{\gg} y$	Сдвиг вправо со знаковым заполнением (“арифметический” или “алгебраический” сдвиг)
11 L		$x \overset{rot}{\ll} y,\ x \overset{rot}{\gg} y$	Циклический сдвиг влево, вправо
10 L	`x < y, x <= y` `x > y, x >= y`	$x < y,\ x \le y$ $x > y,\ x \ge y$	Сравнение знаковых аргументов
10 L	`x < y, x <= y` `x > y, x >= y`	$x \overset{u}{<} y,\ x \overset{u}{\le} y$ $x \overset{u}{>} y,\ x \overset{u}{\ge} y$	Сравнение беззнаковых аргументов
9 L	`x == y, x != y`	$x = y,\ x \ne y$	Равно, не равно
8 L	`x & y`	$x \,\&\, y$	Побитовое *и*
7 L	`x ^ y`	$x \oplus y$	Побитовое *исключающее или*
7 L		$x \equiv y$	Побитовая *эквивалентность* ($\neg(x \oplus y)$)
6 L	`x | y`	$x \mid y$	Побитовое *или*
5 L	`x && y`	$x \overline{\&} y$	Логическое *и* (`if` $x = 0$ `then` 0 `else if` $y = 0$ `then` 0 `else` 1)
4 L	`x || y`	$x \vec{\vee} y$	Логическое *или* (`if` $x \ne 0$ `then` 1 `else if` $y \ne 0$ `then` 1 `else` 0)
3 L		$x \parallel y$	Конкатенация
2 R	`x = y`	$x \leftarrow y$	Присваивание

Кроме функций “abs”, “rem” и прочих, в книге используется множество других функций, которые будут определены позже.

При вычислении выражения `x < y < z` в языке C сначала вычисляется выражение `x < y`, (результат равен 1, если выражение истинно, и 0, если выражение ложно), затем полученный результат сравнивается с `z`. Выражение $x<y<z$ с операторами сравнения “<” вычисляется как $(x<y)\&(y<z)$.

В языке C есть три оператора цикла: `while`, `do` и `for`. Цикл `while` имеет вид:

`while(`*выражение*`)` *оператор*

Перед выполнением цикла вычисляется *выражение*. Если *выражение* истинно (не нуль), выполняется *оператор*. Затем снова вычисляется *выражение*. Цикл `while` завершается, когда *выражение* становится ложным (равным нулю).

Цикл `do` аналогичен циклу `while`, однако проверочное условие стоит после оператора цикла, а именно:

`do` *оператор* `while(`*выражение*`)`

В этом цикле сначала выполняется *оператор*, затем вычисляется *выражение*. Если выражение истинно, операторы цикла выполняются еще раз, если выражение ложно, цикл завершается.

Оператор `for` имеет вид

`for(`e_1`;`e_2`;`e_3`)` *оператор*

Сначала вычисляется выражение e_1 — как правило, это оператор присваивания (аналог выражения-инициализации). Затем вычисляется e_2 — оператор сравнения (или условное выражение). Если значение условного выражения равно нулю (условие ложно), цикл завершается, если не равно нулю (условие истинно), выполняется *оператор*. Затем вычисляется e_3 (тоже, как правило, оператор присваивания), и вновь вычисляется условное выражение e_2. Таким образом, знакомый всем цикл “do i=1 to n” запишется как `for(i=1; i<=n; i++)` (это один из контекстов использования оператора постинкремента).

1.2. Система команд и модель оценки времени выполнения команд

Чтобы можно было хотя бы грубо сравнивать алгоритмы, представим, что они кодируются для работы на машине с набором команд, подобных современным RISC-компьютерам общего назначения (типа Compaq Alpha, SGI MIPS и IBM RS/6000). Это трехадресная машина, имеющая достаточно большое количество регистров общего назначения — не менее 16. Если не оговорено иное, все регистры 32-разрядные. Регистр общего назначения с номером 0 всегда содержит нули, все другие регистры равноправны и могут использоваться для любых целей.

Для простоты будем считать, что в компьютере нет регистров “специального назначения”, в частности, слова состояния процессора или регистра с битами состояний, например “переполнение”. Не рассматриваются также команды для работы с числами с плавающей точкой, как выходящие за рамки тематики данной книги.

В книге описаны два типа RISC: “базовый RISC”, команды которого перечислены в табл. 1.2, и “RISC с полным набором команд”, в который кроме основных RISC-команд входят дополнительные команды, перечисленные в табл. 1.3.

Таблица 1.2. Базовый набор RISC-команд

Мнемокод команды	Операнды	Описание команды
`add, sub, mul, div, divu, rem, remu`	`RT, RA, RB`	`RT←RA op RB` Здесь `op` — сложение (*add*), вычитание (*sub*), умножение (*mul*), знаковое деление (*div*), беззнаковое деление (*divu*), остаток от знакового деления (*rem*) или остаток от беззнакового деления(*remu*)
`addi, muli`	`RT, RA, I`	`RT←RA op I` Здесь `op` — сложение (*addi*) или умножение (*muli*), `I` — непосредственно заданное 16-битовое знаковое значение
`addis`	`RT, RA, I`	`RT←RA+(I || 0x0000)`
`and, or, xor`	`RT, RA, RB`	`RT←RA op RB` Здесь `op` — побитовое *и* (*and*), *или* (*or*) или *исключающее или* (*xor*)
`andi, ori, xori`	`RT, RA, Iu`	То же самое, но последний операнд является непосредственно заданным 16-битовым беззнаковым значением
`beq, bne, blt, ble, bgt, bge`	`RT, target`	Переход к метке `target`, если выполнено некоторое условие, т.е. если `RT=0`, `RT≠0`, `RT<0`, `RT≤0`, `RT>0` или `RT≥0` соответственно (`RT` — целое знаковое число)
`bt, bf`	`RT, target`	Переход к метке `target`, если выполнено некоторое условие (`true/false`); аналогичны командам `bne/beq` соответственно
`cmpeq, cmpne, cmplt, cmple, cmpgt, cmpge, cmpltu, cmpleu, cmpgtu, cmpgeu`	`RT, RA, RB`	`RT` содержит результат сравнения `RA` и `RB`; `RT` равен 0, если условие ложно, и 1, если условие истинно. Мнемокоды означают: сравнить на равенство, неравенство, меньше, не больше и т.д, как и в командах условного перехода. Суффикс “`u`” в названии обозначает сравнение беззнаковых величин
`cmpieq, cmpine, cmpilt, cmpile, cmpigt, cmpige`	`RT, RA, I`	Аналогичны командам серии `cmpeq`, но второй операнд представляет собой непосредственно заданное 16-битовое знаковое значение
`cmpiequ, cmpineu, cmpiltu, cmpileu, cmpigtu, cmpigeu`	`RT, RA, Iu`	Аналогичны командам серии cmpltu, но второй операнд представляет собой непосредственно заданное 16-битовое беззнаковое значение
`ldbu, ldh, ldhu, ldw`	`RT, d(RA)`	Загрузка беззнакового байта, знакового полуслова, беззнакового полуслова или слова в `RT` из ячейки памяти по адресу `RA + d`, где `d` — непосредственно заданное знаковое 16-битовое значение

Окончание табл. 1.2

Мнемокод команды	Операнды	Описание команды
mulhs, mulhu	RT,RA,RB	В RT помещаются старшие 32 бита результата умножения RA и RB (знакового и беззнакового)
not	RT,RA	В RT помещается побитовое дополнение RA до единицы
shl, shr, shrs	RT, RA, RB	В RT помещается значение RA, сдвинутое влево или вправо; величина сдвига задается шестью младшими битами второго операнда (RB). При выполнении команды shrs освободившиеся биты заполняются содержимым знакового разряда, в остальных командах освободившиеся биты заполняются нулями. (Значение величины сдвига вычисляется по модулю 64)
shli,shri,shrsi	RT, RA, Iu	В RT помещается значение RA, сдвинутое влево или вправо на величину, задаваемую пятью младшими битами непосредственно заданного значения I
stb,sth,stw	RS, d(RA)	Сохранение байта, полуслова или слова из регистра RS в ячейку памяти по адресу RA + d, где d — непосредственно заданное 16-битовое знаковое значение

Везде в описаниях команд исходные операнды RA и RB представляют собой содержимое регистров.

В реальной машине обязательно должны быть команды ветвления и обращения к подпрограммам, команды перехода по адресу, содержащемуся в регистре (для возврата из подпрограмм и обработки оператора выбора альтернативы типа switch), а также, возможно, ряд команд для работы с регистрами специального назначения. Конечно же, должны быть привилегированные команды и команды вызова служб супервизора. Кроме того, вероятно наличие команд для работы с числами с плавающей точкой.

Краткое описание некоторых дополнительных команд, которые может иметь RISC-компьютер, приведено в табл. 1.3.

Таблица 1.3. Дополнительные команды полного набора RISC-команд

Мнемокод команды	Операнды	Описание команды
abs, nabs	RT, RA	RT получает абсолютное значение (или отрицательное абсолютное значение) RA
andc, eqv, nand, nor, orc	RT, RA, RB	Побитовое *и с дополнением*, *эквивалентность*, *отрицание и*, *отрицание или* и *или с дополнением*
extr	RT, RA, I, L	Извлечение битов от I до I+L-1 из RA и размещение их в младших битах RT; остальные разряды заполняются нулями
extrs	RT, RA, I, L	Аналогична extr, но свободные биты заполняются содержимым бита знака