Ю. С. Каневский

Компьютерная Арифметика

Конспект лекций

Киев 1994

Каневский Ю.С.
"Компьютерная арифметика", Конспект лекций.

Ответственный за выпуск Хижняк И.В.
Оригинал-макет подготовлен фирмой "ДиаСофт"
НИПФ ''ДиаСофт', 252055, г.Киев-55, а/я 100 Фирма "ВИПОЛ", эак.№ 4-4423 т. 500 экз., 1994г

Исправлено и добавлен список литературы в 2015 г.

Содержание.

Глава 1 ВВОДНЫЕ ЗАМЕЧАНИЯ

Уровень внедрения вычислительной техники в экономику нашей страны становится в настоящее время одним из решающих факторов ускорения научно-технического прогресса и экономического развития всех отраслей народного хозяйства.

Сегодня трудно себе представить деятельность человека без электронных вычислительных машин (ЭВМ). Появившись около 40 лет назад, ЭВМ открыли новую страницу в истории человеческих знаний и возможностей, высвободили тысячи вычислений, значительно облегчили труд ученых, дали возможность изучать сложнейшие процессы. Сейчас нет ни одной отрасли народного хозяйства, где нельзя было бы применить ЭВМ; более того, целые разделы науки и техники не смогут существовать без них.

Появление ЭВМ было подготовлено историческим развитием средств вычислений. Древнейшим счетным инструментом, который дала сама природа человеку, была его собственная рука. От пальцевого счета берут свое начало пятиричная (одна рука) и десятичная (две руки) системы счисления. Издревле употреблялся еще один вид инструментального счета – деревянные палочки с зарубками (бирки) и веревки с узелками. Но с ростом и расширением торговли они не смогли удовлетворить потребности в средствах вычислений. И вскоре появился специальный счетный прибор, известный в древности под названием «абак», представляющий собой доску с вертикальными желобками, в которых передвигались камешки. Русский абак – счеты – появились на рубеже, XVI – XVII вв. Главное их отличие – десятичный принцип счисления. Форма счетов, установленная более 250 лет назад, в настоящее время почти не изменилась. В XVII в. появились и первые логарифмические линейки.

С развитием общества росли потребности в различных вычислениях, которые становились все более трудоемкими. Это явилось причиной появления механических счетных устройств. Первым среди них стало суммирующее устройство (1623) Б. Паскаля. В нем часовой механизм был превращен в счетный и вместо стрелок двигался диск с нанесенными на нем числами. Впоследствии Б. Паскаль сделал несколько вариантов суммирующего устройства, но ни одно из них не получило практического применения: устройства были ненадежны в работе и без специальной подготовки пользоваться ими было невозможно. Тем не менее, значение суммирующих устройств Паскаля в истории развития вычислительной

техники огромно: они послужили переходным этапом от простых приборов к машинам с механическими счетчиками доказали возможность выполнения меаническим устройством определенной части умственной деятельности человека. Создание этих устройств явилось мощным толчком к разработке новых счетных устройств. Работа над счетными устройствами велась в двух направлениях: создание чисто суммирующих устройств и устройств, выполняющих четыре действия арифметики (арифмометр).

В России первое суммирующее устройство было изобретено и изготовлено в 1770 г. Е. Якобсоном – часовым мастером и механиком в г. Несвиже. Суммирующие устройства, создаваемые в то время, не имели применения, они скорее выставлялись напоказ; чем использовались по назначению. Это объяснялось их ненадежностью, неудобством в эксплуатации, серьезными конструктивными недостатками из-за отсутствия необходимой материально-технической и технологической базы. Процессы ввода чисел и выполнения операций в этих устройствах были медленными.

Коренной перелом в создании счетных устройств произошел в середине XIX в., когда появилась необходимая технологическая база, обеспечивающая требуемую для них точность изготовления деталей. Кроме того, общественно-экономическая обстановка (бурный рост промышленности, развитие банков и железных дорог) требовала создания надежных и быстродействующих счетных устройств. Для этого необходимо было, в первую очередь, изменить «медленную установку» чисел. Приближенно эту задачу решило изобретение клавишного ввода. Принципиально решена проблема была лишь с появлением радиоэлектроники. Тем не менее благодаря клавишному механическому вводу в середине 80-х годов XIX в, удалось организовать промышленный выпуск суммирующих устройств, получивших широкое распространение в первой половине нашего столетия. Начиная с 50-х годов в клавишных устройствах стали использовать электропривод, а затем и электронику.

Параллельно с развитием счетных суммирующих устройств создавались арифмометры. Первым в мире арифмометром стала «арифметическая машина» Г. Лейбница, появившаяся в конце XVII в. Сначала Лейбниц пытался лишь улучшить машину Паскаля, но выяснилось, что для выполнений, операций умножения и деления необходим совершенно новый принцип. Лейбниц блестяще разрешил эту задачу, предложив использовать цилиндр, на боковой поверхности которого, параллельно образующей, было расположено девять ступенек, различной длины. Этот цилиндр впоследствии назвали ступенчатым валиком. Машина не

получила широкого распространения, но основная идея Лейбница – идея ступенчатого валика – осталась действенной и плодотворной даже в XX в. На принципе ступенчатого валика был построен и арифмометр Томаса — первое в мире счетное устройство, изготовляемое промышленностью. Создавались арифмометры и другой конструкции. Основным их элементом было зубчатое колесо Однера с переменным числом зубьев.

В 1878 г. в России П. Л. Чебышевым был создан арифмометр, преимуществом которого являлось то, что перенос десятков из младшего разряда в старшие происходил постепенно в процессе накопления единиц и распространялся на последующие разряды. Идеи, заложенные в арифмометре Чебышева, лежат в основе многих видов современных вычислительных устройств.

Существенным недостатком суммирующих устройств и арифмометров считается невозможность значительного увеличения скорости вычислений. Производительность устройств определяется быстротой рук человека. Поэтому ввод информации и управление операциями необходимо передать в ведение машины. Впервые автоматизировал вычислительный процесс в XIX в. английский ученый Ч. Бэббедж, создав проект арифметической машины – прообраз современных компьютеров. Машина состояла из «склада» для хранения чисел (памяти); «мельницы» – для производства арифметических операций (арифметического устройства); устройства, управляющего в определенной последовательности операциями машины (устройства управления); устройства ввода и вывода данных. Для ввода данных предполагалось использовать перфорированную карту. Время на производство арифметических операций оценивалось Ч.Бэббеджем так: сложение и вычитание – 1 с; умножение и деление – 1 мин. Идеи Ч. Бэббеджа не были поддержаны современниками. К ним обратились только в 40-х годах XX столетия при создании автоматической универсальной вычислительной машины «Марк-2».

Первая действующая счетно-аналитическая машина была создана . Г.Холлеритом для автоматизации длительной, однообразной и утомительной работы по обработке данных переписи населения в США в 1880 г. Как и в машине Ч. Бэббеджа, в качестве носителей информации использовались перфокарты, но все остальное оборудование: простой пробойник (перфоратор), сложный пробойник, сортировальная машина и табулятор – было оригинально.

Конец XIX в. и начало,XX в. характеризуются бурным развитием., электроники, телефонии, радиотехники, а позднее – «электроники.

Большой материал, накопленный в, этой области, позволил создать вычислительную машину немеханического типа.

В 1947 г. была закончена работа над релейной вычислительной машиной «Марк-2», в котрой впервые использовалась двоичная система исчисления, а для запоминания чисел, выполнения арифметических раций и операций управления - электромеханические реле (13 тыс.), обладающие двумя устойчивыми состояниями. В машине операции сложения и вычитания занимали примерно. 0,125 с, умножения – 0,25 с.

Одной из удачных конструкций релейных вычислительных машин была машина РВМ-1, сконструированная и построенная под руководством советского инженера Н.И.Бессонова в 1956 г. Главный недостаток релейных вычислительных машин – отсутствие хранимой в памяти программы (малая оперативная память), а также невысокая скорость работы и малая надежность.

В 1943 г. в Гарвардском университете под руководством американских ученых Д. Моучли и Д. Эккерта приступили к созданию электронной вычислительной машины (ЭВМ). К этому времени уже были известны и построены диод (1904), триод (1905), триггер (1918). Машина создавалась по заказу артиллерийского управления и предназначалась для расчета баллистических таблиц. Завершенная в конце 1945 г. машина, получившая название ЭНИАК, имела громадные размеры: содержала 18 тыс. электронных ламп и 1,5 тыс. реле, потребляла около 150 кВт электроэнергии – мощность, достаточная для работы небольшого завода. Использование электронных ламп позволило резко повысить скорость выполнения машинных операций: сложение – 0,0002 с, умножение – 0,0028 с. Управление счетом осуществлялось с помощью программ, набираемых вручную на многочисленных коммутационных досках и переключателях. Несоответствие между временем решения задачи и временем ее подготовки вручную было настолько большим, что выигрыш от скорости вычисления почти полностью покрывался проигрышем во времени на подготовительных операциях.

Создание вычислительной машины ЭНИАК положило начало бурпому развитию ЭВМ нового поколения. В СССР первая малая электронная счетная машина (МЭСМ) – прототип современных ЭВМ – была создана в 1951 г. под руководством С. А. Лебедева. Для МЭСМ характерно наличие универсального арифметического устройства, выполнявшего 50 арифметических или логических операций в секунду. Связанное с универсальным арифметическим устройством оперативное запоминающее устройство, в свою очередь, было соединено с долговременным запоминающим устройством, на котором осуществлялся ввод и хранение

команд. В случае математической ошибки или переполнения разрядной сетки машина останавливалась. Потребляемая ею мощность .составляла 25 кВт. По сравнению со специализированной машиной ЭНИК, созданная С.А. Лебедевым машина имела принципиально новое решение.

Одной из первых в мире ЭВМ с параллельной обработкой кодов была МЭСМ. Эта машина стала базовым прототипом для мирового цифрового математического машиностроения и обусловила переход к новому периоду развития искусства программирования. Появление МЭСМ послужило мощным толчком для разработки широкого круга вопроса вычислительной математики: на машине было решено большое количество задач ядерной физики, осуществлен расчет линии электропередачи Куйбышев – Москва, решены задачи ракетной баллистики и др., решение которых вручную надолго задержало бы развитие некоторых важных направлений отечественной науки и техники. Разработка МЭСМ носила экспериментальный характер и явилась необходимым этапом создания первой быстродействующей электронной счетной машины.

В процессе создания МЭСМ разрабатывались, монтировались и опробовались быстродействующие устройства и узлы большой электронной счетной машины (БЭСМ), монтаж и отладка которой были завершены в 1953 г.

В течение нескольких последующих лет БЭСМ с быстродействием 8 тыс. опер./с была самой быстродействующей машиной в Европе. На ней были решены многие задачи, считавшиеся ранее неразрешимыми из-за большого объема вычислений. Весьма примечательным было то, что ряд технических решений, воплощенных в БЭСМ, предвосхитил идеи ЭВМ второго поколения. Так в состав машины входило специальное устройство контроля, а независимое подключение к памяти арифметического устройства и устройств ввода и вывода соответствовало структуре мультипрограммных машин. Особо важное значение имел схемный метод обращения к подпрограмме и возможность модификации команд с помощью систем местного и центрального управления командами, что открыло новые возможности в развитии искусства программирования, Структура и основные схемы БЭСМ стали классическими; они были положены в основу быстродействующих машин БЭСМ-2, М-2 и др.

В 1953 г. под руководством Ю. А. Базилевского была создана цифровая вычислительная машина (ЦВМ) «Стрела», в 1954 г. под руководством Б. И. Рамеева – ЭВМ «Урал». Почти одновременно с этими машинами появились такие ЭВМ, как М-3, «Минск-2» и др., которые составили семейство отечественных ЭВМ первого поколения.

Характерными чертами ЭВМ первого поколения можно считать не только использование электронных ламп в основных и вспомогательных схемах, но и наличие параллельного арифметического устройства, разделение памяти на быстродействующую оперативную ограниченного объема (выполненную на электронно-лучевой трубке или на ферритовых сердечниках) и медленную внешнюю большого объема (использовавшую накопители на магнитных барабанах и лентах), применение полупроводниковых диодов и магнитных сердечников в логических элементах машины, перфолент и перфокарт как носителей информации при вводе и выводе данных. Среднее быстродействие ЭВМ первого поколения достигало десятка тысяч арифметических операций в секунду.

Поиск структур, обеспечивающих максимальную загрузку всех устройств за счет совмещения их работы во времени, привел к появлению ЭВМ второго поколения, в которых на смену ламповым схемам пришли транзисторные. Основу технической базы ЭВМ второго поколения составили полупроводниковые диоды и транзисторы.

По сравнению с ЭВМ первого поколения ЭВМ второго поколения отличались более высокой надежностью, меньшим потреблением энергии, более высоким быстродействием. Их быстродействие достигалось за счет повышения скорости переключения счетных и запоминающих элементов и изменений в структуре машины.

Наиболее мощная отечественная ЭВМ второго поколения – БЭСМ-6, созданная под руководством С. А. Лебедева. Трудно переоценить то значение и влияние на развитие вычислительной техники и других областей науки, которое оказало создание этой высокопроизводительной оригинальной по архитектуре и структуре отечественной вычислительной машины.

В нашей стране на основе БЭСМ-6 были созданы центры коллективного пользования, управления, координационно-вычислительные и др. До настоящего времени БЭСМ-6 широко используется в системе проектирования для разработки математического обеспечения новых ЭВМ, моделирования сложных физических процессов и процессов управления.

Архитектуру и структуру семейства ЭВМ БЭСМ-1, БЭСМ-2, М-20, БЭСМ-3М, БЭСМ-4, характеризуют целостность концепции и изящные инженерные решения. Наиболее полно это проявилось в БЭСМ-6; несмотря на то, что машина – сложная система, механизмы функционирования ее устройств, их функциональные связи легко понимаются, четко интерпретируются и, следовательно, машина БЭСМ-6 эксплуатируемся легко. Элементная база БЭСМ-6 совершенно новая. Все связи в машине записаны формулами булевой алгебры. Машины семейства БЭСМ ни по

системе команд, ни по внутренней структурной организаций не являются копией какой-либо отечественной или зарубежной установки. В создании и развитии этих машин принимал активное участие акад. В. А. Мельников.

Для ЭВМ второго поколения характерен параллелизм в работе отдельных блоков, начиная от «перекрытия» времени выполнения отдельных команд и кончая параллельным выполнением двух команд или более из одной или из разных программ, что позволило достичь быстродействия до миллиона операций в секунду. Дальнейшее увеличение быстродействия ЭВМ тормозилось конструктивным выполнением электронных схем, собираемых из отдельных элементов – резисторов, конденсаторов, диодов, транзисторов.

Миниатюризация конструктивных, элементов затруднялась необходимостью работы с каждым элементом в отдельности. Выходом из этих затруднений явилась интегральная технология. Малые интегральные схемы (МИС) стали базой машин третьего поколения.

В интегральных схемах роль электронных приборов и элементов выполняют небольшие группы молекул. Основой для таких схем служат полупроводниковые материалы, чаще всего кремний. Специально выращенные большие кристаллы кремния, имеющие очень высокую степень химической чистоты, разрезаются на отдельные пластины, на поверхности которых или внутри специальным способом формируются участки, обладающие свойствами конденсаторов, сопротивлений, диодов, транзисторов и т.д. Достаточно тончайшим металлическими выводами или просто «каналами связи» внутри кристалла соединить одни его участки с другими, выполняющими ту или иную функцию, и интегральная схема готова. Одна интегральная схема заменяет большое число различных деталей и позволяет избавиться от многих недостатков полупроводниковых схем. Переход на интегральные схемы способствовал повышению качества ЭВМ, уменьшению их габаритов и потребляемой ими энергии.

Интеграция различных элементов устранила многие причины, вызывающие возникновение неисправностей. Во-первых, у интегральных схем, состоящих из десятков элементов, небольшое количество вводов и выводов, а до перехода на интегральные схемы их было гораздо больше. Во-вторых, миниатюризация уменьшила нежелательные связи между элементами. Это положительно сказалось на увеличении быстродействия машины. Достоинств у интегральных схем немало. И все же такие характеристики ЭВМ, выполненных на интегральных схемах, как быстродействие и надежность, не являются главными, определяющими и

основополагающими. Более существенное новшество – изменившиеся методы производства этих машин и организации их работы. .

В чем же основное отличие ЭВМ третьего поколения от ее предшественников?

1. ЭВМ третьего поколения оперируют произвольной буквенно-цифровой информацией. В них фактически соединились два направления предыдущих поколений машин: ЭВМ для делового, коммерческого применения; с обработкой алфавитной информации и ЭВМ, предназначенные для научных учреждений и обработки цифровой информации,

2: Изменился порядок работы ЭВМ третьего поколения; эти машины Построены по принципу независимой параллельной работы различных их устройств; процессоров, средств внешней памяти. Независимую работу устройств обеспечивают каналы, управляемые специальным устройством, куда поступает информация от пользователей ЭВМ. Это устройство и осуществляет первичную переработку информации, освобождая основное устройство от непроизводительной работы. Благодаря параллельной работе отдельных устройств ЭВМ может выполнять серию операций: переписывать информацию для очередной задачи с магнитной ленты или магнитного диска, выводить информацию для соответствующего устройства, вводить информацию и т.д.

Типичные представители ЭВМ третьего поколения – машины единой системы (ЕС ЭВМ). Это семейство машин, предназначенных .для решения иаучно-технических, экономических и управленческих задач, применения в различного рода АСУ и системах обработки данных. Они созданы совместными усилиями коллективов ученых, инженеров и рабочих НРБ, ПНР, ГДР, ПНР, СССР и ЧССР. Промышленный выпуск первых моделей ЕС ЭСМ был начат в 1972 г. Высокое быстродействие (миллионы операций в секунду) и широкие возможности являются базой для эффективного использования моделей ЕС ЭВМ.

В машинах третьего поколения в одной интегральной схеме совмещается несколько элементов. Это большое достижение миниатюризации, но еще не предел. ЭВМ четвертою поколения создаются на больших интегральных схемах (БИС). В одной такой схеме объемом всего лишь в доли кубического сантиметра размещается блок, занимавший в ЭВМ первого поколения целый шкаф. В результате достигнуто существенное повышение производительности ЭВМ, Если в ЭВМ третьего поколения быстродействие достигает 20-30 млн. операций в секунду, то в машинах четвертого поколения производительность достигает сотен миллионов операций в секунду. Соответственно возрастает и объем памяти. Наряду с усовершенствованием традиционных устройств памяти на магнитных

дисках и лентах создается память без движущихся частей. Общий объем внешней памяти в крупных машинах четвертого поколения превосходит 10^{12} символов, что эквивалентно библиотеке, состоящей из нескольких миллионов объемистых томов.

По мере развития ЭВМ стабильно проявляется тенденция к увеличению их быстродействия. Это объясняется следующими обстоятельствами. Так как основной узел ЭВМ – сумматор, производящий только операцию сложения, то, с одной стороны, решение любой задачи должно быть сведено к выполнению какого-то количества простых действий. Например, при умножении числа 241 на число 358 фактически необходимо число 241 сложить 16 раз. Аналогичным образом выполняются и другие арифметические операции. С другой стороны, предположим, что требуется решить задачу, объем вычислений в которой составляет 10^8 операций сложения.

Если вычислительная машина работает со скоростью 10 опер/с, то решение задачи займет 10^7 с, что составит примерно 3000 ч. Работать бесперебойно такой длительный срок ЭВМ не всегда могут. Поэтому естественно стремление пользователя решать даже самые сложные задачи за короткий промежуток времени, чтобы неисправности в работе ЭВМ не влияли на результат решения.

Производительность традиционых вычислительных систем повышалась двумя путями: развитием элементной базы и развитием архитектуры самих систем.

Если в первом направлении достигнут практически уже предел (достигнута максимально возможная скорость переключения, определяемая скоростью света), то по второму направлению имеются большие резервы, которые открываются в связи с использованием методов параллельной обработки информации. Сюда можно отнести конвейерные методы, или же системы (машины) потока данных.

Системы пятого поколения в структурном аспекте будут отличаться именно применением таких параллельных структур. Второй отличительной чертой вычислительных систем пятого поколения будет способность производить не только числовые расчеты (как это делают современные машины.), но и обработку смысловой информации с выполнением операции анализа и вывода. Необходимость в подобных системах определяется расширением областей применения ЭВМ в таких нетрадиционных сферах, как образование, услуги, населению и т.п. Здесь речь идет о возможностях создания искусственного интеллекта. Третьей особенностью систем пятого поколения станет элементная база. Это

будет не только сверхбольшие интегральные схемы (СБИС), но и созданные на их основе ЭВМ, имеющие некоторое сходство по структуре с искусственным интеллектом, а также оптоэлектроника с использованием когерентного излучения. А так как скорость света значительно выше скорости электронов, то повысится как быстродействие машины, так и пропускная способность линий связи, по которым информация должна поступать в ЭВМ. Для решения этой задачи сделано уже немало. Созданы световоды с малыми потерями – на расстоянии в 1 км интенсивность света в них уменьшается всего в 2 раза.

Перспективно параллельное преобразование информации, представляемой в виде голограмм и соответствующих вычислительных сред. Если соединить в одно целое быстродействие запоминающих устройств и возможности голографии, то машины будущего смогут вместить в своей памяти и выдавать по первому же приказу все информационное богатство, накопленное человеком за многовековой путь развития.

Поиск новых принципов построения ЭВМ, совершенствование уже известных алгоритмов выполнения арифметических и логических операций – главнейшие задачи для специалиста в области проектирования и создания ЭВМ. Весь дальнейший, материал конспекта посвящен принципам построения алгоритмов функционирования машин и методам формального описания их работы, используемых при проектировании цифровых вычислительных машин.

Глава 2 СИСТЕМЫ СЧИСЛЕНИЯ И ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В ЭВМ

Системы счисления были созданы в процессе хозяйственной деятельности человека, когда у него появилась потребность в счете, а по мере развития научной и технической деятельности возникла также необходимость записывать числа и производить над ними вычисления.

Системой счисления называется совокупность символов и приемов, позволяющих изображать числа. В общем случае, это специальный язык, алфавитом которого являются символы, называемые цифрами, а синтаксисом – правила, позволяющие однозначно сформировать запись чисел. Запись числа в некоторой системе счисления называют кодом числа. Кратко число записывается следующим образом:

$$A = a_n a_{n-1} \ldots a_2 a_1 a_0 .$$

Отдельную позицию в изображении числа принято называть разрядом, а номер позиции – номером разряда. Число разрядов в записи числа называется разрядностью и совпадает с его длиной. В техническом плане длина числа интерпретируется как длина разрядной сетки. Если алфавит системы счисления имеет p различных значений, то разряд a_i в числе рассматривается как p-ичная цифра, которой может быть присвоено каждое из p значений.

Каждой цифре a_i числа A однозначно соответствует ее количественный эквивалент – $K(a_i)$. Количественный эквивалент числа A, заданного в определенной системе счисления, является некоторой функцией количественных эквивалентов всех его цифр, т.е. $K(A)_p = f[K(a_n),..., K(a_1)]$.

Очевидно, что при конечной разрядной сетке количественный эквивалент числа A будет принимать в зависимости от количественных эквивалентов отдельных разрядов значения от $K(A)_{min}$ до $K(A)_{max}$.

Тогда диапазон (D) представления чисел в данной системе счисления, т.е. интервал числовой оси, заключенный между максимальным и минимальным числами, представленными заданной разрядностью (длиной разрядной сетки), составит

$$D = K(A)_{(p)max} - K(A)_{(p)min}.$$

Существует бесчисленное множество способов записи чисел цифровыми знаками. Однако любая система счисления, предназначенная для практического использования, должна обеспечивать:

1) возможность представления любого числа в заданном диапазоне чисел;

2) однозначность представления чисел;

3) краткость и простоту их записи;

4) легкость овладения системой, а также простоту и удобство оперирования ею.

В настоящее время в зависимости от целей применения используются различные системы. Например, человеком для счета и выполнения действий над числами применяется десятичная система счисления, для исчисления времени – система счисления времени, для нумерации – римская система счисления, в вычислительной технике обычно используется двоичная система счисления и т.д. В зависимости от способа записи чисел и способа вычисления их количественного эквивалента системы счисления можно классифицировать следующим образом (рис.2.1).

В подавляющем большинстве случаев системы счисления строятся по следующему принципу:

$$A_{(p)} = a_n p_n + a_{n-1} p_{n-1} + \ldots + a_1 p_1,$$

где $A_{(p)}$ – запись числа в системе с базисом p;
a_i – база или набор цифр системы счисления с p-ичным алфавитом;
p_j – базис системы счисления или совокупность весов отдельных разрядов системы, $j = \overline{1,n}$.

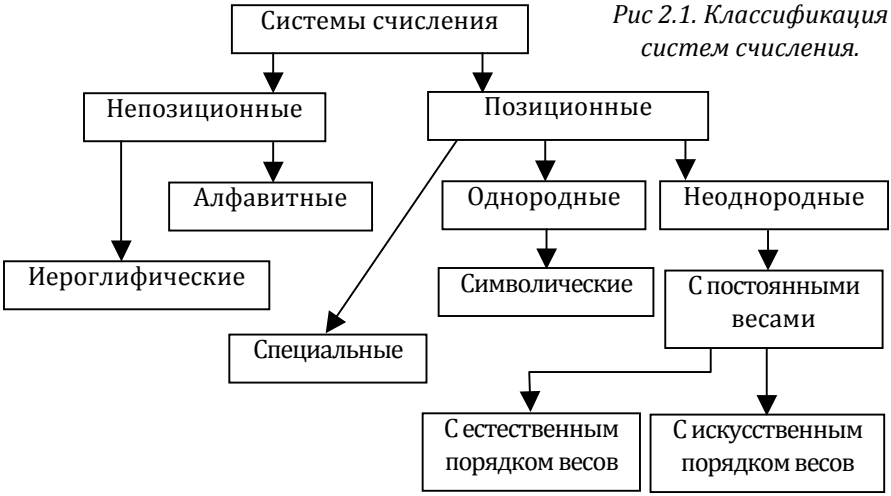


Рис 2.1. Классификация систем счисления.

База системы счисления может быть положительной, и тогда в ней в качестве значений цифр используются символы 0,1,..., p–1. Она может быть также смешанной, и тогда в ней наряду с положительными цифрами имеются и отрицательные. Например, для симметричной базы с нулем число положительных значений цифр равно числу отрицательных. Значения цифр алфавита в этом случае при p = 2t + 1 (т.е. при нечетном основании) составляют следующий рад: –t, –t+1,..., –1, 0,1,.... t–1, t.

Базис системы счисления, как уже отмечалось, – это совокупность весов отдельных ее разрядов: Например, базис десятичной системы представляет собой набор 10^0, 10^1, 10^2,...,10^n. Вес разряда R_i числа любой системе счисления – это отношение $R_i = p_i/p_0$.

2.1. Непозиционные системы счисления

Алфавит непозиционных систем счисления содержит неограниченное количество символов (цифр), а количественный эквивалент любой цифры постоянен и зависит только от ее начертания, но не от позиции в числе. Такие системы строятся по принципу аддитивности, т.е. количественный эквивалент числа определяется как сумма количественных эквивалентов рядом стоящих цифр:

$$K(A)_Q = Q_1 + Q_2 + \ldots + Q_k = \sum_{i=1}^{k} Q_i ,$$

где Q_i – символы, образующие базис системы

$$Q = \{Q_1, Q_2, \ldots, Q_k\}.$$

Наиболее известными представителями непозиционных систем счисления являются иероглифические и алфавитные. Иероглифические – это такие системы счисления, у которых каждая цифра представлена своим символом, значком или иероглифом. Наиболее известной из них является римская система счисления.

Значение записанного числа в римской системе определяется как сумма записанных подряд цифр, причем, если слева от цифры стоит меньшая, то значение последней принимается со знаком "минус", например: IX = $9_{(10)}$; XI = $11_{(10)}$, т.е. здесь существует отклонение от правила независимости значения цифры от положения в числе. В настоящее время римская система используется, в основном, для нумерации.

Запись чисел в алфавитных системах строится по такому же принципу.

К основным недостаткам непозиционных систем счисления можно отнести:

1) отсутствие нуля;

2) теоретически такие системы должны содержать бесконечное количество символов;

3) арифметические действия с числами в таких системах сложны.

2.2. Позиционные системы счисления

Это такие системы счисления, у которых алфавит содержит ограниченное количество символов, а значение каждой цифры в числе определяется не только ее начертанием, но и находится в строгой зависимости от позиции в числе. Например, в десятичной системе счисления число 777 содержит три одинаковые цифры, но значение каждой из них определяется ее позицией.

Позиционные системы имеют ряд достоинств по сравнению с непозиционными, основным из которых является удобство выполнения арифметических операций.

В общем виде число A в позиционной системе счисления может быть представлено следующим образом:

$$A = a_{n-1}p_{n-1}...p_0 + a_{n-2}p_{n-2}...p_0 + ...+ a_1p_1...p_0 \qquad (2.1)$$

где p_j – основания системы счисления, $j=\overline{0,n}$;

a_i – цифра i-го разряда числа, причем $a_i = \overline{0,p_j-1}$ есть база системы счисления;

$P_i = \prod_{j=0}^{i} p_j$ – вес i-го разряда числа.

Как видно, такие системы строятся не только по принципу аддитивности, но и по принципу мультипликативности, т.е. количественный эквивалент числа определяется как сумма рядом стоящих цифр со своими весами.

Позиционные системы счисления разделяются на ряд подклассов.

2.2.1. Неоднородные позиционные системы счисления

В неоднородных позиционных системах счисления p_j-тые значения не зависят друг от друга и могут принимать любые значения, эти системы еще называют системами со смешанным основанием.

В неоднородных системах счисления в каждом i-м разряде количество допустимых символов может быть различно, при этом $0 \le a_i < p_i$, где p_i –

основание системы счисления в i-м разряде. Запись целого числа в таких системах производится в соответствие с (2.1).

Примером неоднородной позиционной системы счисления может служить система счисления времени, для которой: $P_0 = 1$ сек, $P_1 = 60$ сек, $P_2 = 60$ мин, $P_3 = 24$ часа, $P_4 = 365$ суток.

Например, время, в 2 года, 25 суток, 14 часов, 35 минут, 48 секунд, выраженное в единицах младшего разряда определится по (2.1);

$$A = 2\times365\times24\times60\times1 + 25\times24\times60\times1 + 14\times60\times1 + 35\times60\times1 + 48\times1.$$

Специально для применения в ЭВМ была создана неоднородная двоично-пятеричная система счисления, в которой в нечетных разрядах основание $P_1 = 5(a_i = \overline{0,4})$, а в четных разрядах основание $P_2 = 2(a_i = \overline{0,1})$. Так как произведение двух соседних (четного и нечетного) разрядов равно 10, то двумя двоично-пятеричными разрядами можно кодировать одну десятичную цифру (табл.2.1).

Таблица 2.1

a_{10}	$a_{2\text{-}5}$
0	00
1	01
2	02
3	03
4	04
5	10
6	11
7	12
8	13
9	14

Пример: Записать число 398_{10} в двоично-пятеричной системе счисления: $A = 398_{10} = 03\times14\times13$. Здесь n = 6, основания $P_1 = 5$, $P_2 = 2$, $P_3 = 5$, $P_4 = 2$, $P_5 = 5$, $P_6 = 2$, а цифры $a_1 = 3$, $a_2 = 1$, $a_3 = 4$, $a_4 = 1$, $a_5 = 3$, $a_6 = 0$. Для вычисления количественного эквивалента числа $A_{2\text{-}5}$ подставим эти значения в (2.1):

$$A = 0\times5\times2\times5\times2\times5 + 3\times2\times5\times2\times5 + 1\times5\times2\times5 + 4\times2\times5 + 1\times5 + 3 =$$
$$= 0 + 300 + 50 + 40 + 5 + 3 = 389_{10}.$$

2.2.2. Однородные позиционные системы

Однородные позиционные системы счисления являются частным случаем позиционных систем при pj = pj для всех i и j, т.е. в них веса отдельных разрядов представляют собой ряд членов, геометрической профессия со знаменателем p. Поэтому число в однородных системах может быть представлено в общем случае полиномом вида:

$$A = a_n p^n + a_{n-1}p^{n-1} + \ldots + a_1 p^1 + a_0 p^0 + a_{-1}p^{-1} + \ldots + a_{-k}p^{-k} \qquad (2.2)$$

или

$$A = \sum_{i=-k}^{n} a_i ,$$

причем $a_j = \overline{0,(p-1)}$, а знаменатель геометрической прогрессии p носит название основания системы счисления. Очевидно, что основанием однородной позиционной системы может быть любое натуральное число со своим знаком, так как в определении позиционных систем счисления не

наложено никаких ограничений на величину основания. Поэтому возможно бесчисленное множество позиционных систем счисления. Например, десятичное число $A = 777_{10}$ в троичной системе счисления с символами 0, 1, 2 будет иметь следующий вид:

$$A = 1001210_3.$$

Обычно число в однородной позиционной системе записывается в сокращенном виде:

$$a_n\, a_{n-1} \ldots a_1\, a_0\, a_{-1} \ldots a_{-k},$$

а название системы определяет ее основание: десятеричная, двоичная, восьмеричная и т.д.

Для любой однородной позиционной системы счисления справедливо, что ее основание изображается символами 10 в своей системе, т.е. любое число в своей системе можно записать символами этой системы в виде:

Таблица 2.2

Основание системы					
16	10	8	4	3	2
0	0	0	0	000	0000
1	1	1	1	001	0001
2	2	2	2	002	0010
3	3	3	3	010	0011
4	4	4	10	011	0100
5	5	5	11	012	0101
6	6	6	12	020	0110
7	7	7	13	021	0111
8	8	10	20	022	1000
9	9	11	21	100	1001
a	10	12	22	101	1010
b	11	13	23	102	1011
c	12	14	30	110	1100
d	13	15	31	111	1101
e	14	16	32	112	1110
f	15	17	33	120	1111
10	16	20	40	121	10000

$$A = a_n 10^n + a_{n-1}10^{n-1} + \ldots + a_1 10^1 + a_0 10^0 + a_{-1}10^{-1} + \ldots + a_{-k}10^{-k}.$$

Правильная дробь в р-ичной системе счисления будет иметь вид:

$$A = a_{-1}p^{-1} + a_{-2}p^{-2} + \ldots + a_{-k}p^{-k}.$$

Это выражение можно представить в следующем виде:

$$A = \left(a_{-1}p^{k-1} + a_{-2}p^{k-2} + \ldots + a_{-k}\right) p^{-k} = \frac{a_{-1}p^{k-1} + a_{-2}p^{k-2} + \ldots + a_{-k}}{p^k} \qquad (2.3)$$

Этим выражением мы пользуемся при чтении десятичных дробей. Этим же приемом целесообразно пользоваться при чтении дробей с любым другим основанием р.

Некоторые числа» представленные в однородных системах с различными основаниями, приведены в таблице 2.2.

Из таблицы видно, что вес R_i разряда (i – номер разряда) числа в позиционной системе счисления представляет собой отношение

$$R_i = \frac{p^i}{p^0}.$$

Если разряд имеет вес $R_i = 10^m$, то следующий старший разряд имеет вес $R_{i+1} = 10^{m+1}$, а младший – $R_{i-1} = 10^{m-1}$. Такая взаимосвязь разрядов требует передачи информации между ними (переносов при сложении и займов при вычитании).

2.3. Кодированные позиционные системы счисления

Это такие системы, в которых цифры одной системы счисления кодируются при помощи цифр другой системы, а число в общем виде записывается следующим образом:

$$A = (a_{n,k}p^n + a_{n-1,k}p^{n-1} + ... + a_{1,k}p^1 + a_{0,k}p^0)P^k + (a_{n,k-1}p^n + ... + a_{0,k-1}p^0)P^{k-1} + ... + (a_{n,0}p^n + ... + a_{0,0}p^0)P^0,$$

где p – основание системы счисления, символами которой кодируются числа;

P – основание исходной системы счисления.

При построении кодированных позиционных систем в качестве весов разрядов кодирующей системы могут быть выбраны как члены геометрической прогрессии (т.е. веса однородной позиционной системы счисления), так и произвольные числа. В первом случае системы называются кодированными системами счисления с естественными весами разрядов, во втором – с искусственными весами разрядов.

Примером системы счисления с естественными весами разрядов может служить двоично-десятичная система с весами 8-4-2-1. В ней каждая цифра десятичного числа кодируется двоичной тетрадой. Например, десятичное число 1593 в этой двоично-десятичной системе имеет вид 0001 0101 1001 0011.

Примером системы счисления с искусственными весами разрядов могут служить двоично-десятичная система счисления с весами 2-4-2-1. Десятичное число 1593 в этой системе примет соответственно следующий вид:

0001 1011 1111 0011.

Как нетрудно заметить, этот код является самодополняющимся, т.е. в этой системе десятичная цифра 9 кодируется как 1111, что позволяет рационально строить счетчики и арифметические схемы. Код называется самодополняющимся, если двоичные коды любых двух десятичных цифр,

дополняют друг друга до 9 (т.е. если их десятичная сумма равна 9: $a'_{10} + a''_{10}$ 9), дополняют друг друга до $15_{(10)} = 1111_{(2)}$.

Таблица 2.3

десятичная цифра	код 8421	код 2421	код 8421+3
0	0000	0000	0011
1	0001	0001	0100
2	0010	0010	0101
3	0011	0011	0110
4	0100	0100	0111
5	0101	0101	1000
6	0110	1100	1001
7	0111	1101	1010
8	1000	1110	1011
9	1001	1111	1100

Еще одним самодополняющимся кодом является "код с избытком три" – 8421+3. Он получается из естестенного кода 8421 добавлением к нему числа $3_{10} = 0011_2$. В результате такого добавления получается система с непостоянными весами разрядов. В самом деле, например, цифра 1_{10} кодируется как 0100_2, т.е. в третьем двоичном разряде 1 имеет вес единицы. Тогда при кодировании цифры $9_{10} = 1100_2$ старшая двоичная цифра должна иметь вес 8, но при кодировании цифры $5_{10} = 1000_2$ она имеет вес 5. Отметим, что все приведенные коды являются системами однородными, так как в каждом двоичном разряде может быть только две цифры: 0 или 1. В таблице 2.3 приведены десятичные цифры, представленные в некоторых двоично-десятичных кодах.

Следует также отметить, что двоично-десятичные коды обладают определенной избыточностью, так как для кодирования десятичных цифр применяется только 10 комбинаций из 16-ти, что можно использовать для обнаружения некоторых ошибок.

2.4. Системы счисления специального назначения

Достоинством этих позиционных систем, специально созданных для упрощения или ускорения вычислений в ЭВМ, является простота алгоритмов выполнения некоторых арифметических операций, недостатком – необходимость перевода из классических позиционных систем в специальные. Их применяют для реализации некоторых вычислительных процессов, в которых не требуется изменять систему счисления при вводе и выводе данных, либо это изменение достигается простыми средствами.

Например, позиционные системы счисления с отрицательным основанием позволяют представить без знака любое вещественное число, положительное или отрицательное. Одной из самых интересных специальных систем является уравновешенная троичная система счисления,

т.е. система по основанию 3 с цифрами +1, 0, –1 (или $\bar{1}$). При записи чисел в этой системе:

$$10\bar{1} = 8_{(10)}$$

$$1\bar{1}\bar{1}0{,}1\bar{1} = 15\frac{2}{9}_{(10)}$$

$$\bar{1}110{,}\bar{1}1 = -15\frac{2}{9}_{(10)}$$

Достоинства уравновешенной троичной системы:

1. Знак числа задается его старшей ненулевой цифрой.
2. Переход к числу с противоположным знаком производится заменой всех 1 на $\bar{1}$ и наоборот.
3. Операции округления до ближайшего целого сводятся к отбрасыванию дробной части.

Складывать в этой системе очень просто, если учесть, что $\bar{1} + \bar{1} = \bar{1}1$, $1 + \bar{1} = 0$, $1 + 1 = 1\bar{1}$. Вычитание сводится к переходу к числу, противоположному по знаку, и последующему сложению. Правила умножения на +1 обычны, а при умножении на $\bar{1}$ знак частного произведения меняется на противоположный.

Пример:

$A = 1\bar{1}01 = 19_{10}$

$B = 11\bar{1}0 = 33_{10}.$

0000

$\bar{1}101$

$1\bar{1}01$

$1\bar{1}01$

$10\bar{1}\bar{1}1\bar{1}0 = 627_{10}$

2.5. Позиционные системы счисления с непостоянными весами разрядов

Система счисления с непостоянными весами разрядов уже упоминалась – двоично-десятичная система 8421+3. Теперь познакомимся с некоторыми позиционными системами счисления с непостоянными весами разрядов, которые на практике в качестве помехоустойчивых кодов, например, с кодом Грея.

В двоичном коде при переходе от изображения одного числа к изображению соседнего может происходить одновременное изменение цифр в нескольких разрядах, что может служить источником значительных ошибок (например, при переходе от 7 к 8). Это обусловлено тем, что в

однородной позиционной системе счисления любой код является разрешенным, а числа находятся на кодовом расстоянии от n до 1. Кодовым расстоянием между двумя кодовыми комбинациями называется число одноименных разрядов в этих комбинациях, которые имеют различные символы. В коде Грея соседние числа находятся на кодовом расстоянии, равном единице, т.е. различаются цифрой только в одном разряде (таблица 2.4).

Таблица 2.4

десятичная система счисления	двоичная система счисления	код Грея
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Двоичные разряды в коде Грея не имеют постоянного веса. Например в числе 3_{10}, представленном в коде Грея, единица второго разряда имеет вес, равный трем, а в числе 2 – вес 1.

Коду Грея характерен ряд особенностей:

1) В нем нет одновременного изменения цифр в нескольких разрядах при счете.

2) Смена значений каждого разряда при последовательном переходе от комбинации к комбинации происходит вдвое реже, чем в обычном двоичном коде (в младшем разряде двоичного кода происходит чередование элементов 0-1-0-1-..., во втором разряде – 00-11-00-11-..., в четвертом – 0000-1111-0000-..., а в коде Грея для младшего разряда – 00-11-00- 11-..., для второго - 0000-1111-0000-....

2) В коде Грея можно выделить оси симметрии, относительно которых наблюдается идентичность состояния некоторых разрядов. Главная ось симметрии расположена между кодами $(2^{n-1} - 1)$ и 2^{n-1} (отсюда название "отраженный" или рефлексный код).

Недостатки кода Грея и других позиционных систем счисления с постоянными весами разрядов обусловлены тем, что в них вес символа (1) не определяется номером разряда (или меняет знак). Из-за этого их трудно применить для обработки информации в ЭВМ. Поэтому перед вводом в ЭВМ или декодированием данных, представленных в позиционных системах счисления с непостоянными весами разрядов, их преобразуют в более простой и удобный двоичный код.

2.6. Символические системы счисления

Позиционные системы счисления имеют четкие межразрядные связи. Это свойство обеспечивает простоту выполнения арифметических операций, и вместе с тем наличие межразрядных переносов приводит к ограничению скорости выполнения этих операций. Поэтому разработка непозиционных систем счисления, в которых отсутствуют межразрядные связи и вместе с тем просто осуществляются арифметические операции, позволяет повысить скорость вычислений.

В символических системах, в отличие от позиционных, цифры являются символами, каждый из которых в отдельности никак не характеризует какое-либо число. Определенным комбинациям цифр условно поставлены в соответствие определенные числа Примером символической системы счисления является система остаточных классов (СОК).

Говорят, что целые числа A и B сравнимы по mod S, если им соответствует один и тот же остаток от деления на третье число S, что выражается записью $A = B(\mathrm{mod}\ S)$. Число в СОК изображается в виде остатков от деления заданного числа на ряд взаимно простых чисел S_1, S_2,..., S_n. При этом образуется число с весами разрядов, соответственно равными S_1, S_2,..., S_n, т.е.

$$A_{СОК} = a_1, a_2, \ldots, a_n, \qquad (2.4)$$

где $a_i = A_{10} - K_i \times S_i$ и $K_j = [A_{10}/S]$, где [x] – целая часть x. Следовательно, $A = a_i(\mathrm{mod}\ S_i)$. Остаток a_i называют также вычетом числа A по модулю S_i. В таблице 2.5 на основании (2.4) приведены трехразрядные числа для первых пятнадцати десятичных чисел, представленных в СОК с весами разрядов, соответственно равными $S_1 = 2$, $S_2 = 3$, $S_3 = 5$.

Таблица 2.5

десятичная система	СОК		
	S_1	S_2	S_3
0	0	0	0
1	1	1	1
2	0	2	2
3	1	0	3
4	0	1	4
5	1	2	0
6	0	0	1
7	1	1	2
8	0	2	3
9	1	0	4
10	0	1	0
11	1	2	1
12	0	0	2
13	1	1	3
14	0	2	4
15	1	1	0

В СОК поразрядными являются операция счета, а также операции сложения, вычитания и умножения, что является несомненным достоинством СОК.

СОК применяется в специализированных ЭВМ, в которых диапазон исходных чисел и промежуточных результатов строго фиксирован и операция деления практически отсутствует.

2.7. Перевод чисел из одной системы счисления в другую

Существует два основных метода перевода чисел из одной системы счисления в другую: табличный и расчетный.

Табличный метод прямого перевода основан на сопоставлении таблиц соответствия чисел различных систем счисления. Этот метод очень громоздок и требует большого объема памяти для хранения таблиц, но применим для любых, систем счисления (не только для позиционных). Суть другого вида табличного метода состоит в том, что имеются таблицы эквивалентов в каждой системе только для цифр, т.е. баз, этих систем и степеней основания (положительных и отрицательных), т.е.,базисов систем. Задача перевода сводится к тому, что в выражения полиномов (2.2) и (2.3) для исходной системы счисления подставляют эквиваленты из новой системы для всех цифр и их весов разрядов и производят действия (умножения и сложения) по правилам арифметики с новым основанием Р. Полученный при этом результат будет изображать число в новой системе счисления.

Пример: число $A_{10} = 113$ перевести в систему с Р = 2.

Таблица эквивалентов:

десятичное число	двоичное число
10^0	0001
10^1	1010
10^2	1100100

Тогда: $A_{10} = 113 = 1\times10^2 + 1\times10^1 + 3\times10^\circ = 0001\times1100100 + + 0001\times1010 + 0011x0001 = 1110001_2$. Этот метод применим только к позиционным системам счисления,

Рассматриваемый ниже расчетный метод применим только к однородным позиционным системам счисления, однако более удобен.

2.7.1. Перевод целых чисел из одной позиционной системы счисления в другую

Пусть задано число А в произвольной позиционной системе счисления по основанию 1 и его необходимо перевести в новую систему с основанием Р, т.е. преобразовать к виду

$$A = a_n p^n + a_{n-1} p^{n-1} + ... + a_1 p^1 + a_0 p^0, \qquad (2.5)$$

где: $a_i = 0 : p-1$ – база новой системы счисления.

Выражение (2.5) можно записать в виде:

$$A = A_1 \times p + a_0,$$

где: $A_i = (a_n p^{n-1} + a_{n-1} p^{n-1} + ... + a_2 p^1 + a_1)$ – целая часть частного; a_0 - остаток от деления А на Р, который является цифрой младшего разряда искомого числа, записанной в символах старой системы счисления.

При делении числа A_i на Р получим остаток a_i и т.д. Другими словами выражение (2.5) представляется в соответствии со схемой Горнера:

$$A = (...((a_n p + a_{n-1})p + a_{n-2})p + ...+ a_1)p + a_0,$$

после чего его правая часть последовательно делится на основание новой системы Р.

При этом деление продолжается до тех пор, пока не окажутся выполненными соотношения: $A_n < p$, $A_{n+1} = 0$.

Правило перевода целых чисел из системы в систему формулируется следующим образом: чтобы перевести целое число из одной позиционной системы счисления в другую необходимо исходное число последовательно делить на основание новой системы счисления, записанное в исходной системе, до получения частного, равного нулю. Число в новой системе счисления записывается в виде остатков от деления, начиная с последнего.

Пример: перевести десятичное число 138 в двоичную и восьмеричную систему и произвести обратный перевод:

а) при переводе из десятичной системы последовательно делим ис ходное число на основание 2 и на основание 8:

138,	69,	34,	17,	8,	4,	2,	1,	0	– частное
0	1	0	1	0	0	0	0	1	– остаток

138,	17,	2,	0 – частное
2	1	2	– остаток

т.е. получаем: $138_{10} = 10001010_2 = 212_8$,

б) при переводе из двоичной системы счисления в десятичную ис ходное число необходимо делить на основание новой системы, т.е. 1010_2.

Деление выполнять в двоичной системе трудно. Поэтому на практике при необходимости перевода чисел из системы с малым основанием в систему с большим основанием удобно пользоваться общей записью чисел в виде полинома (2.5).

В общем случае можно вычислить многочлен $A = a_m l^m + ...+ a_1 l + a_0$ непосредственно или в виде $A = (...(a_m l + a_{m-1})l + ...+ a_1)l + a_0$, представив в системе по основанию Р a_i и 1 и выполнив все действия по правилам арифметики Р. При переводе двоичных чисел в десятичную систему счисления обычно подсчитывают сумму степеней основания 2, при

которых коэффициенты ai равны единице. Расчеты ведутся при этом в десятичной системе счисления.

Пример: Перевести число 10001010_2 в десятичную систему:

$$A = 1\times2^7 + 1\times2^3 + 1\times2^1 = 128 + 8 + 2 = 138_{10}.$$

2.7.2. Перевод правильных дробей

Пусть правильную дробь А, заданную в произвольной позиционной системе счисления по основанию 10, необходимо перевести в новую систему с основанием Р, т.е. преобразовать ее к виду:

$$A = a_{-1}p^{-1} + a_{-2}p^{-2} +...+ a_{-k+1}p^{-k+1} + a_{-k}p^{-k}. \qquad (2.6)$$

Если, аналогично переводу целых чисел разделить обе части выражения (2,6) на p^{-1}, т.е. умножить на р, то получим:

$$A\times p = a_{-1} + A_1,$$

где $A_1 = (a_{-2}p^{-1} + a_{-3}p^{-2} +...+ a_{-k}p^{-k+1})$ – дробная часть произведения;

a_{-1} – целая часть результата.

Полученная при этом цифра целой части результата и будет первой цифрой искомого числа.

Умножив теперь дробную часть результата A_1 снова на Р, получим:

$$A_1\times p = a_{-2} + A_2,$$

где: $A_2 = (a_{-3}p^{-1} + a_{-4}p^{-2} +...+ a_{-k}p^{-k+2})$ – дробная часть произведения (нового); a_2 - следующая цифра искомого числа.

Следовательно, при переводе выражение (2.6) вычисляется в соответствии со схемой Горнера

$$A_{(p)} = p^{-1}(a_{-1}p^{-1}(a_{-2}+...+ p^{-1}(a_{-k+1} + p^{-1}a_{-k})...));$$

Пример: Преобразовать число $0{,}138_{10}$ к восьмеричному виду.

$0,138 \times 8$
$1,104 \quad \rightarrow 1$
$0,104 \times 8$
$0,832 \quad \rightarrow 0$
$0,832 \times 8$
$6,656 \quad \rightarrow 6$

Результат $0{,}138_{10} = 0{,}106_8$.

Таким образом, для перехода от десятичного представления дробного числа к восьмеричному нужно производить умножение в десятичной системе на восемь.

2.8. Выбор системы счисления для применения в ЭВМ

Очевидно, что непозиционные системы непригодны для использования в ЭВМ в силу своей громоздкости и трудности выполнения арифметических операций.

Из позиционных систем наиболее удобны однородные системы счисления, так как одинаковое основание, т.е. одинаковое количество символов во всех разрядах приводит к наиболее рациональному использованию оборудования и к наиболее простым алгоритмам выполнения арифметических операций. Поэтому проанализируем однородные позиционные системы счисления на предмет их применения в ЭВМ. При этом будем учитывать следующие факторы:

1. Наличие физических элементов, способных изобразить символы системы.
2. Экономичность системы, т.е. количество элементов, необходимое для представления многоразрядных чисел.
3. Трудоемкость выполнения арифметических операций в ЭВМ.
4. Быстродействие вычислительных систем.
5. Наличие формального математического аппарата для анализа и синтеза вычислительных устройств.
6. Удобство работы человека-с машиной.
7. Помехоустойчивость кодирования цифр на носителях информации.

Таким образом, задача выбора системы счисления для применения в ЭВМ сводится по сути к задаче рациональной величины основания системы Р. Поэтому рассмотрим с указанных выше позиций системы счисления с разными основаниями.

2.8.1. Наличие физических элементов

Любой из символов, применяемых для записи чисел, должен в ЭВМ изображаться в виде одного или нескольких состояний какого-то физического элемента. Очевидно, что элемент будет тем проще, чем меньше состояний ему требуется иметь, т.е. чем меньше основание системы счисления. Например, для реализации двоичной системы счисления можно применить реле, магнитные, полупроводниковые элементы и т.п.

Троичную систему счисления можно еще естественно реализовать при помощи конденсаторов и магнитных элементов. Р-позиционные элементы (при $P > 3$) носят искусственный характер и с увеличением Р их реализация усложняется. Так, для реализации десятичного элемента используется триггер из более чем 10 транзисторов.

Таким образом, по этому критерию наиболее пригодной для использования в ЭВМ является двоичная система счисления.

2.8.2. Экономичность системы счисления

Найти оптимальное значение основания системы счисления по этому критерию довольно сложно, т.к. чем больше основание, тем меньшее количество разрядов и, значит, элементов требуется для изображения числа, однако, тем большее количество символов должен отображать каждый элемент, т.е. иметь большее количество устойчивых состояний, что приводит к его усложнению.

Числа при больших основаниях имеют ряд недостатков. Во-первых, нужно иметь названия и обозначения для P цифр, чего обычно нет для больших значений P.

Вторым и гораздо большим недостатком является трудность, возникающая при попытках выполнения вычислений с помощью обычных методов. Например, таблица умножения становится чрезмерно громоздкой, чтобы ее можно было помнить наизусть или ею оперировать.

С этой точки зрения, использование чисел с небольшим основанием дает ряд преимуществ. Например, когда основанием является число P = 3, то в таблице умножения существует только единственное нетривиальное умножение, а именно: $2\times2 = 4 = 11_3$. Однако запись числа при этом удлиняется.

Для оценки экономичности системы счисления в качестве критерия выберем количество цифроразрядов *D*\, необходимое для изображения числа в i-той системе счисления, исходя из условия, что D| пропорционально объему оборудования, т.е.

$$D_i = P_i \times n_i, \qquad (2.8)$$

где P_i – основание i-той системы счисления; n_i – количество разрядов представляемого числа. Правомерность последнего утверждения обусловлена тем, что например, при умножении время умножения на один разряд пропорционально основанию p, а на все разряды - пропорционально $P \times n$.

Количество чисел, которые можно представить в i-той системе счисления, определяется следующим образом:

$$N_i = P_i^{n_i} . \qquad (2.9)$$

Откуда

$$n_i = \log_{p_i} N_i. \qquad (2.10)$$

Подставив выражение (2.10) в (2.8) получим:

$$D_i = P_i \log_{p_i} N_i = \frac{P_i}{\log_{N_i} P_i}.$$

Оптимальное значение величины D_i можно определить, если допустить, что величина P_i изменяется не дискретно, а непрерывно и количество чисел N принять равным для всех i систем. Тогда и D_i будет непрерывной величиной, находящейся в логарифмической зависимости с P_i:

$$D_{(P)} = \frac{P}{\log_N P}. \qquad (2.12)$$

Взяв производную от функции (2.12):

$$\frac{dD}{dP} = \frac{1 \times \log_N P - p \times 1/p \log_N e}{(\log_N e)^2}. \qquad (2.13)$$

и исследовав ее на экстремум, получим:

$$P_{опт} = e = 2{,}73\ldots .$$

Теперь, чтобы оценить экономичность систем с целочисленными основаниями, определим относительное значение $D_{i(отн)}$

$$D_{i(отн)} = \frac{D_i}{D_{опт}} \qquad (2.14)$$

где $D_{опт} = e \cdot \ln N$.

Подставив выражения (2.11) и (2.15) в (2.14), получим:

$$D_{i(отн)} = \frac{p_i \times \log_{P_i} N}{e \times \ln N} = \frac{p_i \times \ln N \times \log_{P_i} e}{e \times \ln N} = \frac{p_i}{e \times \ln p_i},$$

решив которое для разных P_i, определим характеристику экономичности систем с различными основаниями (табл. 2.6).

Таблица 2.6.

2	3	4	5	6	7	8	9	10
1.062	1.004	1.062	1.143	1.232	1.30	1.42	1.507	1.597

Следовательно, по критерию экономичности системы наиболее приемлемой является система счисления с основанием P = 3. Затем следуют системы с P = 2 и P = 4, которые уступают ей на 5,8%. Однако, ввиду того; что троичных элементов фактически нет, приходится хранить троичный разряд в двух двоичных. С учетом этого наиболее экономичной оказывается снова двоичная система счисления.

2.8.3. Трудоемкость выполнения арифметических операций

По этому критерию наиболее эффективной является двоичная система, т.к. чем меньше цифр участвует в арифметических операциях, тем проще их выполнение.

2.8.4. Быстродействие вычислительных устройств

Этот критерий находится в прямой зависимости от простоты выполнения арифметических операций. Очевидно также, что с увеличением количества цифр в системе счисления быстродействие ЭВМ при прочих равных условиях будет падать.

Наиболее часто встречающейся операцией является операция алгебраического сложения. Она выполняется, как правило, за один такт, т.е. протекает с высокой скоростью. Быстродействие ЭВМ в значительной степени зависит от скорости выполнения операции умножения, которая, с одной стороны, встречается сравнительно часто, а с другой – является достаточно длительной. Поэтому, если посчитать скорость выполнения умножения достаточной характеристикой общего быстродействия ЭВМ, то получим следующие результаты. С учетом того, что на каждом этапе умножения максимальное количество сложений не может превзойти величину $(p_i - 1)$, а также учитывая, что количество этапов определяется разрядностью чисел *щ,* получим общее количество необходимых сложений:

$$K_i = (P_i - 1)n_i = (P_i - 1)\log_{P_i} N. \qquad (2.16)$$

Полученное выражение не имеет экстремумов. Для выявления $P_{опт>}$ необходимо сделать нормированные относительно $P_i = 2$ подсчеты по формуле (2.16) для ряда оснований:

$$K_{i(отн)} = \frac{(p_i - 1)\log_{P_i} N}{e \times \log_2 N} = \frac{p_i - 1}{e \times \log_2 P_i}.$$

Получим таблицу 2.7 результатов:

Таблица 2.7.

2	3	4	5	6	7	8	9	10
1.00	1.262	1.50	1.725	1.913	2.138	2.333	2.524	2.709

Таким образом, ЭВМ, работающая в двоичной системе счисления, характеризуется более высоким быстродействием относительно троичной системы на 26,2% и относительно десятичной – в 2,7 раза.

2.8.5. Наличие формального математического аппарата для анализа и синтеза вычислительных устройств

Таким аппаратом, позволяющим относительно просто и экономично строить узлы и блоки ЭВМ, является алгебра логики. Наибольшее развитие и законченность изучения, вследствие своей простоты и широкого практического применения, получила двоичная логика.

С ее помощью все арифметические и управляющие цепи можно строить на основе одного и того же набора двоичных элементов, применяя для их анализа и синтеза один и тот же математический аппарат, что значительно облегчает проектирование схем ЭВМ. Это обстоятельство также позволяет сделать вывод в пользу двоичной системы счисления по сравнению с другими системами.

2.8.6. Удобство работы человека с машиной

Ввиду того, что в своей практической деятельности человек привык пользоваться десятичными числами, то наиболее удобной по этому критерию *является* десятичная система счисления. Но решить какая система находится на втором месте сложнее, так как все они требуют перевода чисел. Очевидно, наиболее удобной для человека будет система, в которой проще всего выполняются арифметические действия, запоминаются таблицы сложения, вычитания, умножения, деления, т.е. двоичная.

2.8.7. Наибольшая помехоустойчивость кодирования цифр

Если считать одинаковым диапазон изменения носителя информации *для* всех систем, то очевидно преимущество систем с малыми основаниями. Это значит, что при наложении некоторой помехи на основной сигнал, изображающий цифру, наибольшая ошибка возможна в устройстве, использующем систему счисления с самым большим основанием. Следовательно, с позицией наибольшей помехоустойчивости предпочтение следует отдать двоичной системе счисления.

Таким образом, исходя из перечисленных критериев, наиболее приемлемой для применения в ЭВМ является однородная позиционная система счисления с основанием, равным двум. Однако, в некоторых случаях при синтезе вычислительного устройства какому-либо критерию придается большее значение, чем остальным. Тогда для применения выбирается система счисления, оптимальная по выбранному критерию.

Например, в некоторых случаях предпочтение отдают десятичному счислению, руководствуясь при этом не соображениями экономичности выбираемого счисления, а удобством общения человека с машиной.

В современных универсальных ЭВМ применяются как двоичная, так и десятичная системы счисления. Причем цифры последней кодируются двоичными символами, т.е. речь идет в действительности не о десятичной, а о двоично-десятичной системе счисления. Каждая из отмеченных систем имеет свои достоинства и недостатки, а также свои области применения.

Достоинствами двоичной системы счисления относительно двоично-десятичной являются:

1) экономия порядка 20 % оборудования;

2) примерно в 1,5 раза более высокое быстродействие (не путать с десятичной системой, у которой быстродействие в 2,7 раза ниже, чем у двоичной!);

3) упрощение логического построения устройства управления и вспомогательных цепей.

Достоинствами двоично-десятичной системы являются:

1) отсутствие надобности в переводе исходных данных и результатов расчетов из одной системы в другую;

2) удобство контроля промежуточных результатов путем вывода их на индикацию для визуального наблюдения;

3) более широкие возможности для автоматического контроля вычислений из-за наличия в двоично-десятичном коде избыточных комбинаций.

Двоичную систему счисления применяют в больших и средних ЭВМ, предназначенных для решения научно-технических задач, для которых характерен большой объем вычислений и сравнительно малый объем исходных данных и результатов вычислений. Ее также целесообразно применять в ЭВМ, предназначенных для управления технологическими процессами.

Двоично-десятичную систему счисления применяют для решения экономических задач, которые характеризуются большим объемом исходных данных, сравнительной простотой, малым объемом выполняемых над ними преобразований и большим количеством результатов вычислений. Эту систему целесообразно также применять в калькуляторах и в ЭВМ, предназначенных для инженерных расчетов.

2.9. Двоичная система счисления.

Под двоичной системой счисления понимается такая система, в которой для изображения чисел используются два символа, а веса разрядов меняются по закону $2^{\pm k}$ (где k – произвольное целое число). Классической двоичной системой является система с символами 0, 1. Ее двоичные цифры часто называют битами.

Чтобы овладеть любой системой счисления, надо уметь складывать и умножать в ней две любые цифры. Арифметические операции в двоичной системе счисления выполняются так же, как и в десятичной в соответствии с таблицами поразрядных вычислений.

Таблица 2.8 умножения двоичных чисел полностью определяется двумя правилами:

1) умножение любого числа на 0 дает 0;
2) умножение любого числа на 1 оставляет его без изменения.

Таблица 2.8.

x	0	1
0	0	0
1	0	1

Для сложения имеется только правило, согласно которому прибавление 0 к любому числу не меняет этого числа. Тогда таблица 2.9 сложения примет вид:

Таблица 2.9.

1)

+	0	1
0	0	1
1	1	0

2)

+	0	1
0	0	1
1	1	1

3)

+	0	1
0	0	1
1	1	1

Заполнение пустого места соответственно 0, 1 или сочетанием 10 приведет к таблицам сложения в трех различных системах счисления. Первая система счисления называется арифметикой вычетов по mod 2.
Вторая таблица представляет собой пример операции булевой алгебры. Третья таблица представляет собой сложение в обычной двоичной системе счисления с символами 0, 1.

Как уже отмечалось, в общем виде все двоичные числа представляются в виде полинома:

$$A = \sum_{i=-k}^{n} a_i \times 2^i . \quad (2.17).$$

Таблица 2.10.

a_i	b_i	$П_{i-1}$	S_i	$П_i$
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Перевод в двоичную систему счисления из десятичной производится либо по общему правилу перевода чисел из одной позиционной системы счисления в другую, либо десятичные числа переводятся в восьмеричную систему, а затем восьмеричные числа переводятся в двоичные по правилу перевода чисел для систем с кратным основанием. Обратный перевод производится аналогично либо при помощи общего вида записи числа (2.17) в виде полинома.

Сложение в двоичной системе счисления производится по правилам сложения полиномов. Поэтому при сложении чисел A и B i-й разряд суммы S_i и перенос $П_i$ из данного разряда в 0+1)-й будет определяться в соответствии со следующим выражением:

$$a_i + b_i + П_{i-1} = S_i + 2П_i \quad . \quad (2.18)$$

где $П_{i-1}$, $П_i \in \{0, 1\}$ и $S_i \in \{0,1\}$.

Выражению (2.18) соответствует таблица 2.10 сложения одноразрядных чисел. В соответствии с таблицей 2.10 можно суммировать многоразрядные двоичные числа.

Пример: Задано $A = 11_{10} = 1011_2$ и $B = 3_{10} = 11_2$. Найти сумму C = A + B.

В результате поразрядного сложения чисел Аи В получаем:

$$\begin{array}{rl} & 1011 \\ + & 0011 \\ \hline C= & 1110_2 = 14_{10}. \end{array}$$

2.9.1. Двоичная система с цифрами 1, $\bar{1}$

Из определения двоичной системы счисления следует, что для изображения чисел могут быть использованы не только символы 0,1, но и символы 1,-1 или 0, -1.

Символ -1 обычно изображают как Т, а систему, в которой используются символы 1, –1, называют системой $(1, \bar{1})$. Число A = 37ю запишется, например, в этой системе как $A = 11\bar{1}1\bar{1}1$, если положить в (2.17), что a_i принимает значение 1 или $\bar{1}$.

Особенностью двоичной системы счисления с цифрами 1, $\bar{1}$ является представление единым кодом как положительных, так и отрицательных чисел. Однако в ней отсутствует нуль и нет возможности представить некоторые числа в виде конечного множества символов.

В системе (1, $\bar{1}$) некоторые целые и дробные числа, например четные (20 = $11\bar{1}\bar{1}1$,111...), представляют в виде бесконечных дробей, т.е. с определенной погрешностью.

Вместе с тем, существуют числа, которые не имеют единственного изображения, например число 1 может быть представлено в виде

$$1 = 1\underbrace{\bar{1}\bar{1}...\bar{1}}_{k} = 1\underbrace{00...0}_{k} - \underbrace{11...1}_{k} = 2^k - (2^k - 2^0), \qquad (2.19)$$

где k = 1, 2,....

Соотношение (2.19) выражает связь между обычной двоичной системой и системой (1, $\bar{1}$).

Пример: Перевести число A = 1001012 в систему (1, $\bar{1}$).

Используя соотношение (2.19) перевод сводится к замене комбинаций 001 и 01 комбинациями соответственно 1ТТ и IT, т.е. A= 11 ГИТ.

Для получения конечного представления как четных, так и нечетных чисел в системе (1, $\bar{1}$) польским ученым И. Баньковским была предложена следующая запись чисел [53, 65]:

$$A = \sum_{i=-h}^{k} a_i \times 2^i - 2^{-h}.$$

В этом случае из-за аддитивной добавки положительных чисел будет меньше на одно число, чем отрицательных. Всего при n = k + 1 + h разрядов можно изобразить 2^n чисел: 2^{n-1}отрицательных, $(2^{n-1} - 1)$ положительных и нуль, вместо $(2^n - 1)$ числа в двоичной системе с цифрами 0, 1, ибо в последней число нуль представляется неоднозначно (+0, –0).

Для перевода чисел в систему (1, $\bar{1}$) по методу Баньковского необходимо учитывать следующее: в случае нечетного числа перевод осуществляют по правилу (2.19), а затем в разряд i = –h записывают единицу; при переводе четного числа его сначала превращают в нечетное добавлением единицы в младший разряд и только после этого переводят в систему (1, $\bar{1}$) по правилу (2.19) как нечетное число. Затем в полученном результате в разряд i = –h записывают $\bar{1}$.

Пример: Перевести в систему (1, $\bar{1}$) двоичное число A = 11000.

По правилу Баньковского это число вначале превращаем в нечетное число 11001. После этого заменяем в изображении числа комбинацию 001 на комбинацию $1\bar{1}\bar{1}$, приписываем в разряд после запятой цифру $\bar{1}$ и получаем: $A_{(1,\bar{1})} = 11\bar{1}\bar{1}\bar{1},\bar{1}$.

Правило перевода из двоичной системы счисления с цифрами 0, 1 в систему с цифрами 1, $\bar{1}$ можно формализовать следующим образом.

Для перевода положительных чисел вначале к исходному числу приписывается справа еще один разряд, значение которого есть 1. Затем в исходном изображении выделяют конструкции, состоящие из последовательности нулей и единицы справа, т.е. конструкции вида 00...01. Эти конструкции на основании (2.19) преобразуются к виду $1\bar{1}...\bar{1}\bar{1}$, т.е. самый старший нулевой разряд заменяется на 1, а остальные разряды включая 1 в младшем разряде конструкции, заменяются на $\bar{1}$.

Пример: Число A = $10001,10101_2$.

Приписываем справа разряд со значением 1

$$A'=10001,101011.$$

Выделяем конструкции вида 000...01.

$$A'=1\underbrace{0001},1\underbrace{0}\underbrace{1}\underbrace{01}1 .$$

Преобразуем выделенные конструкции и получаем

$$A' = 11\bar{1}\bar{1}\bar{1},11\bar{1}1\bar{1}1_{(1,\bar{1})} .$$

В этом случае добавлением 1 справа к исходной записи реализуется учет аддитивной поправки. Значение этой поправки должно быть одно и то же для всех чисел, поэтому, если в исходной записи дробная часть числа имеет меньше, чем h–1 значащих цифр, то ее надо заполнить нулями до h–1 разрядов и уже в h-й разряд записать 1. Целая часть числа дополняется нулями слева до требуемого количества разрядов k.

Пример: Число A = $10,1_2$ при k = 1, h = 3.

Вначале подготовим запись числа к преобразованию: A = $0010,10_2$

Дополним A разрядом справа со значением $\bar{1}$ и выделим конструкции вида 00...01.

$$A'= \underbrace{0}\underbrace{0}1\underbrace{0},\underbrace{1}\underbrace{0}\underbrace{1} .$$

Преобразуем выделенные конструкции и получим

$$A' = 1\bar{1}\bar{1}1,\bar{1}1\bar{1}_{(1,\bar{1})} .$$

Для представления отрицательных чисел в системе с цифрами 1, $\bar{1}$ вначале преобразуется запись абсолютной величины числа А путем дописывания слова и справа необходимого числа нулей. Затем в этом изображении все 1 заменяются на $\bar{1}$, а потом к полученному изображению дописывается 1 справа (аддитивная добавка). Далее выделяются конструкциивида $00...0\bar{1}$ и 00...01, которые преобразуются соответственно к виду $\bar{1}1...11$ и $1\bar{1}...\bar{1}\bar{1}$.

Пример: $A = -10{,}1_2$

$A = -0010{,}10_2$

$A' = \underbrace{00\bar{1}0}_{}\underbrace{,}_{}\underbrace{101}_{}$

$A' = \bar{1}11\bar{1},11\bar{1}_{(1,\bar{1})}$.

Двоичную систему счисления с цифрами 1, $\bar{1}$ рационально использовать как промежуточную при выполнении операций умножения и деления. Прямая реализация арифметических действий в этой системе заметно сложнее, чем в системе с цифрами 0,1.

Существуют другие двоичные системы счисления, например, система с символами 1, 0, $\bar{1}$, которая является избыточной.

2.9.2. Избыточная двоичная система

Избыточной системой счисления с основанием Р называется система, в которой для записи чисел используется количество символов большее, чем Р. Избыточная двоичная система (симметричная) связана с обычной следующим соотношением:

$$\underbrace{11...1}_{k} = \sum_{i=0}^{k-1} 2^i = 2^k - 2^0 = \underbrace{100...0\bar{1}}_{k+1} . \qquad (2.21)$$

На (2.21) производится переход от двоичной системы с цифрами (0, 1) в систему (1, 0, $\bar{1}$) и наоборот, например, $A = 111100012 = 1000\bar{1}0001_{(1,0,\bar{1})}$.

В любой избыточной системе одни и те же числа можно представить несколькими способами. Например, $A = 0{,}01110011 = 0{,}100\bar{1}0011 = 0{,}100\bar{1}010\bar{1}$.

При этом в избыточной системе можно уменьшить количество единиц в изображении числа. Симметричные избыточные системы счисления позволяют в ряде случаев упростить выполнение арифметических действий. Например, избыточную двоичную систему счисления используют, как мы увидим далее в главе 4, в некоторых алгоритмах ускоренного умножения.

В системе 1, 0, $\bar{1}$, как и в системе 1, $\bar{1}$, для обозначения отрицательной величины к числу не требуется присоединить дополнительный знак и, следовательно, при выполнении арифметических действий над числами не требуется использовать еще и правила знаков. Кроме того, в этой системе при выполнении операции можно избежать распространения переноса далее, чем в два и даже один соседний разряд. Другими словами, в этой системе за счет вводимой избыточности можно реализовать методы сложения, в которых каждый разряд суммы является функцией только смежных справа разрядов слагаемых. Это значит, что время выполнения операции сложения не зависит от длины операндов и эквивалентно времени сложения трех или даже двух разрядов. Отсюда следует, что коды чисел при сложении могут поступать в порядке от старших разрядов к младшим, т.е. начиная со старших разрядов.

При умножении, в отличие от обычной двоичной системы, образование программирования может также начинаться со старших разрядов. В этом случае процесс выполнения операции умножения может быть остановлен, когда будет получено требуемое количество цифр произведения, т.е. достигнута требуемая точность.

2.9.3. Навыки в обращении с двоичными числами

Хотя все правила выполнения операций в двоичной системе счисления очень просты, но тем не менее при работе с двоичными числами из-за отсутствия навыков возникают, разного рода неудобства. Ниже приведены некоторые простые приемы, которые позволяют довольно свободно обращаться с двоичными числами.

1. Число $\underbrace{100\ldots0}_{k} =: 2^k$.

Необходимо знать на память десятичные значения этих чисел от k = 0 до k = 12. Они приведены в таблице 2.13.

Таблица 2.13.

k	0	2	3	4	5	6	7	8	9	10	11	12
2	1	2	4	8	16	32	64	128	256	1024	2048	4096

2. Число $\underbrace{11\ldots1}_{k} = 2^k - 1$.

3. Необходимо знать на память десятичные значения двоичных чисел от 0 до 31 включительно. Эти числа в дальнейшем будут называться "малыми числами".

4. Двоичное число

$$\underbrace{a_{n-1}a_{n-2}a_{n-3}a_{n-4}a_{n-5}}_{\text{"малое число" a}}\underbrace{00\ldots00}_{k}$$

равно $a \times 2^k$.

Пример: $11011000 = 11011 \times 2^3 = 27 \times 8 = 216$,
$101000000 = 101 \times 2^6 = 5 \times 64 = 320$.

Двоичное число

$$\underbrace{a_{n-1}a_{n-2}a_{n-3}a_{n-4}a_{n-5}}_{\text{"малое число" a}}\underbrace{00\ldots00}_{k}\ \underbrace{b_5b_4b_3b_2b_1}_{\text{"малое число" b}}$$

равно $a \times 2^k + b$.

Пример: $10110000101 = 1011 \times 2^7 + 101 = 11 \times 128 + 5 = 1413$,
$1010100001101 = 10101 \times 2^8 + 1101 = 21 \times 256 + 13 = 5389$.

5. Если в n-разрядном числе много единиц и мало нулей, то для определения его количественного эквивалента можно из n-разрядного числа! записанного одними единицами, вычесть "малое число", в котором разряды со значением 1 соответствуют разрядам исходного числа с нулевым значением и наоборот.

Пример: A= 11111101001 соответствует $1111111111 = 2^{11} - 1$

$$\begin{array}{r} - \quad 10110 = 22 \\ \hline 11111101001 = 2025 \end{array},$$

т.е. $11111101001 = 2^{11} - 1 - 10110 = 2047 - 22 = 2025$.

Некоторые числа можно читать двумя способами.

Например: $1110101 = 2^7 - 1 - 1010 = 127 - 10 = 117$ или
$1110101 = 111 \times 2^4 + 101 = 7 \times 16 + 5 = 117$.

7. Зная на память малые числа легко проверять правильность выполнения операций сложения и вычитания двоичных чисел, если их можно разбить на группы, не связанные друг с другом переносами или заемами.

Пример:

левая группа:		правая группа:
10111 | 01011	23	11'
+ 101 | 01110	+5	+ 14
11100 | 11001	28	25

7. Чтение двоичных дробей.

A= $0,\underbrace{00\ldots001}_{n-1} = 2^{-n}$. Дробь A = $0,\underbrace{11\ldots11}_{k} = 1 - 2^k$.

Двоичная дробь читается по тем же правилам, что и десятичная: разряды справа от запятой читаются как целое число, которое является

числителем; читается как целое число, являющееся k-й степенью двух, причем k – номер младшего разряда справа от запятой.

Пример: A = 0,11011 = $27/2^5$ = 27/32, читается "двадцать семь тридцать вторых"

B = 0,00110110 = $27\times2^1/2^8$ = 54/256, читается "пятьдесят четыре двести пятьдесят шестых".

8. Двоичные дроби бывают периодическими. Например, периодическими являются дроби вида $\frac{1}{2^k-1} = 0,(\underbrace{00\ldots01}_{k})$ и $\frac{1}{2^k+1} = 0,(\underbrace{0\ldots0}_{k}\underbrace{11\ldots11}_{k})$.

Примеры:

$\frac{1}{3} = \frac{1}{2^2-1} = 0,(01) = 0,010101\ldots$

$\frac{1}{5} = \frac{1}{2^2+1} = 0,(0011) = 0,001100110011\ldots$

$\frac{1}{7} = \frac{1}{2^3-1} = 0,(001) = 0,001001001\ldots$

$\frac{1}{9} = \frac{1}{2^3+1} = 0,(000111) = 0,000111000111\ldots$

$\frac{1}{15} = \frac{1}{2^4-1} = 0,(0001) = 0,000100010\ldots$

$\frac{1}{17} = \frac{1}{2^4+1} = 0,(00001111) = 0,0000111100001111\ldots$

2.10. Представление двоичных чисел в ЭВМ

Под машинным изображением числа понимают представление числа A в разрядной сетке цифрового автомата. Цифровым автоматом называют устройство, характеризующееся набором внутренних состояний $A = \{a_1(t), a_2(t),\ldots, a_n(t)\}$, в которые оно попадает под воздействие команд программы решения задачи.

Машинное изображение числа условно обозначают символом [A]. При этом

$$A = [A]\times K_A, \qquad (2.22)$$

где K_A – коэффициент, величина которого зависит от формы представления числа в автомате. Под формой представления чисел в автомате понимают свод правил, позволяющий установить взаимное соответствие между записью числа и его количественным эквивалентом.

Если произвольное вещественное число A' таково, что $A' = [A]\times K_A$, то говорят, что такое число точно представляется в машине при заданной

форме представления чисел. Если же $A' \neq [A] \times K_A$, то произвольное вещественное число A' может быть представлено в машине приближенно либо вообще не может быть представлено. При приближенном представлении вещественное число A' заменяется некоторым числом [A], принадлежащим множеству машинных чисел. Множеству машинных чисел принадлежат только числа, которые делятся на два, так как любые два попарно соседних машинных числа отличаются друг от друга на величину 2^{-n}, где n – количество разрядов. Число, отличное от нуля, но представленное в машине как нуль, т.е. число $|A| < A_{min}$ называют машинным нулем. Числа, большие A_{max} , в машине также не могут быть представлены. В этом случае говорят о переполнении разрядной сетки.

Существует три формы представления чисел: естественная, с фиксированной запятой и нормальная. Естественной формой записи числа называется запись числа в виде полинома, представленного в сокращенном виде:

$$A = a_n\, a_{n-1} \ldots a_1 a_0\, a_{-1}\, a_{-2} \ldots a_{-k}. \qquad (2.23)$$

При этом отсчет весов разрядов ведется от запятой. При естественной форме записи запятая ставится на строго определенном месте – между целой и дробной частью числа.

Числа, представленные в такой форме, называют числами с естественной запятой. При этой форме представления для каждого числа необходимо указывать положение его запятой в одном из разрядов кода, т.е. место положения запятой должно быть предусмотрено в каждом разряде, для чего необходимо дополнительное оборудование.

Недостатки представления чисел в форме с естественной запятой обусловлены также усложнением арифметических цепей и трудностью оперирования с очень большими или очень малыми по абсолютной величине числами. Поэтому эту форму применяют лишь в калькуляторах, несмотря на привычность представления чисел.

Необходимость в указании положения полностью отпадает, если место запятой в разрядной сетке машины заранее фиксировано. Такая форма представления чисел называется представлением с фиксированной запятой. Поэтому машины, использующие запись чисел в этой форме, называются машинными с фиксированной запятой (точкой).

Кроме записи (2.23) число можно представить в виде произведения целой степени основания системы и цифровой частоты, являющейся правильной дробью:

$$A = p^m \times \sum_{i=-k}^{n} a_i \times p^{i-k} \quad . \quad (2.24)$$

Такая форма записи называется нормальной, показатель степени при основании р называется порядком числа, а цифровая дробная часть мантиссой. Так как ограничений на величину мантиссы нет, то положение запятой в ней может меняться при соответствующем изменении порядка. В силу этого машины, использующие нормальную запись числа, называются машинами с плавающей запятой.

2.10.1. Представление чисел в машинах с фиксированной запятой

Чтобы упростить функционирование цифрового автомата, необходимо ограничить входную информацию какой-то одной областью чисел (целой или дробной). В машинах с фиксированной запятой числа часто представлены в виде правильных дробей, т.е. запятую фиксируют перед старшим разрядом числа, причем числа, большие единицы, приводятся к такому виду при помощи масштабного коэффициента $К_А$. Представление чисел в виде правильных дробей обусловлено необходимостью уменьшить возможность переполнения разрядной сетки машины, т.е. обезопасить вычисления от потерь значащих цифр старших разрядов при выполнении арифметических операций.

Здесь учитывается тот факт, что результат умножения никогда не выходит за пределы разрядной сетки, если запятая расположена перед старшим разрядом. Но в этом случае результаты слржения и деления могут выйти за пределы разрядной сетки (при операции сложения – не более чем на один разряд, а операция деления встречается в среднем в 6 раз реже, чем умножение).

Можно было бы оперировать только малыми числами, так как переполнение при их сложении отсутствует. Однако это приводит к снижению точности представления чисел и точности вычислений. Поэтому всегда стремятся использовать числа, величины которых близки к максимальному значению. При этом на. них накладываются следующие ограничения:

1) абсолютная величина суммы двух чисел должна быть меньше единицы;

2) делитель по абсолютной величине должен быть больше делимого.

Нередко запятую фиксируют после младшего разряда числа. Тогда все данные представляются р виде целых чисел. В этом случае также необходимо масштабирование исходных данных.

В ячейке машины с фиксированной перед старшим разрядом запятой число записывается в разрядную сетку в виде значащей части дроби

со своим знаком, т.е. для записи n-значной дроби разрядная сетка должна содержать n + 1 разряд. Отдельных разрядов для записи целой части числа (0) и запятой не выделяется, так как их положение обусловлено формой записи чисел. (При записи данных в виде целых чисел знак также является старшим разрядом).

Знак числа обычно кодируется следующим образом: знаку “+” соответствует “+” в знаковом разряде, знаку “–” – “1” .

Величины чисел, представляемых в машинах с фиксированной перед старшим разрядом запятой, лежат в пределах:

$$2^{-n} \le |A| \le 1 - 2^{-n}.$$

Причем числа, меньше $A_{min} = 2^{-n} - 1$ – и большие $A_{max} = 1 - 2^{-n}$ мАшиной не воспринимаются. Область чисел, лежащих в пределах $-2^n < A < 2^n$, называется областью нечувствительности и определяется погрешностью представления чисел.

Величины чисел, представленных при фиксации запятой после младшего разряда, лежат в пределах $1 \le |A| \le 2^n - 1$.

То есть в обоих случаях диапазон представления чисел в машине с фиксированной запятой равен:

$$D = \frac{A_{max}}{A_{min}} = \frac{1 - 2^{-n}}{2^{-n}} = \frac{2^{-n} - 1}{1} \approx 2^{-n}.$$

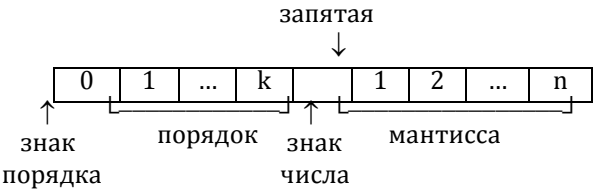
Например, при n = 40 $D \approx 2^{40} \approx 10^{13}$.

2.10.2. Представление чисел в машинах с плавающей запятой

В машинах с плавающей запятой числа представлены в нормальной форме

$$A = p^m \sum_{i=-k}^{n} a_i \, p^{i-m} = p^m \times a.$$

Для записи числа в ячейке машины отводится (k+1) разряд для фиксации порядка со своим знаком и n+1 разряд для мантиссы. То есть место, занимаемое в коде числа кодом мантиссы и кодом порядка заранее фиксировано.



Для повышения точности записи чисел, мантиссы в машинах с плавающей запятой представлены в так называемом нормализованном виде, при котором диапазон представления мантиссы лежит в пределах для p = 2: $0{,}5 \le |a| \le 1 - 2^{-n}$, т.е. старший разряд мантиссы есть всегда значащий.

Для $p = 2^s$ $(s > 0)$ имеем $1/p < |a| < 1$.

В свою очередь, диапазон представления порядка числа лежит в пределах: $-(2^к - 1) \le m \le 2^к - 1$.

Следовательно, минимальное число для p = 2, которое можно представить в ячейке машины, определится следующим образом:

$$A_{min} = 2^{-|m_{max}|} \times |a_{min}| = 2^{-(2^k-1)} \times 2^{-1} = 2^{-2^k}$$

и максимальное

$$A_{max} = 2^{|m_{max}|} \times |a_{max}| = 2^{2^k-1} \times (1 - 2^{-n}) = 2^{2^k-1},$$

(т.к. n обычно велико, то $1 - 2^{-n} \approx 1$) .

Очевидно, что диапазон представимых чисел в машинах с плавающей запятой значительно больше, чем в машинах с фиксированной запятой:

$$D = \frac{A_{max}}{A_{min}} = \frac{2^{2^k-1}}{2^{-2^k}} \approx 2^{2^{k+1}-1}. \qquad (2\text{-}27)$$

Например, если из 40 разрядов 5+1 займет порядок числа, а остальные 34 - мантисса, то диапазон представимых чисел составит:

$$D = 2^{63} \approx 10^{21}.$$

Сопоставляя между собой две основные формы представления чисел в ЭВМ, можно прийти к следующим выводам:

Диапазон представимых чисел в машинах с фиксированной запятой значительно меньше, чем в машинах с плавающей запятой, а точность вычислений зависит от величин исходных чисел. Программирование для машины с фиксированной запятой значительно сложнее, так как приходится вводить масштабные коэффициенты, чтобы избежать переполнения разрядной сетки при выполнении арифметических операций. При этом от правильности выбора масштабных коэффициентов зависит ошибка представления. Следовательно, их вычисление должно производиться таким образом, чтобы исключить возможность появления не

только переполнений разрядной сетки, но и машинного нуля, что бывает порою сложно осуществить даже опытному программисту.

Однако машины с плавающей запятой конструктивно более сложны, так как необходимо вводить дополнительное оборудование для выполнения операций над порядками чисел, а также предусмотреть операцию нормализации и выравнивания порядков чисел. Плавающая запятая менее удобна, чем фиксированная, в тех случаях, когда распределение разрядов в разрядной сетке не соответствует той схеме, которая заранее принята для чисел с плавающей запятой. Наиболее важный из этих случаев – это операции над командами. Кроме того, время выполнения операций над числами в машине с плавающей запятой больше, чем в машине с фиксированной запятой, что обусловлено необходимостью работы с порядками.

Как и при фиксированной запятой, здесь возможно переполнение разрядной сетки, которое выражается в том, что результат какой-либо операции имеет порядок больше допустимого. Это приводит к аварийной ситуации. При выполнении операций возможно получение чисел, имеющих порядок меньше допустимого и нормализованную мантиссу. Эти числа обычно рассматриваются как машинные нули, так же как и числа, имеющие нулевую мантиссу и допустимый порядок.

Иногда нормальную форму представления чисел называют полулогарифмической, так как порядок числа выражен в логарифмической форме.

2.11. Точность представления чисел в ЭВМ

При решении различных задач требуется различная точность получаемых результатов. Так, при решении инженерных задач достаточна точность до 3-4 десятичных знаков (10-13 двоичных), при решении научных задач – 5-6 десятичных или 16-20 двоичных знаков и при решении особо точных задач – 8-10 десятичных, т.е. 25-32 двоичных разрядов.

При ограниченной длине масштабных слов множество чисел, которые можно представить в машине является конечным. Поэтому представление чисел в ЭВМ, как правило, влечет за собой появление погрешностей и от длины разрядной сетки. Необходимо отметить, что запас точности в ЭВМ берется обычно в 1,5-2 раза больше, чем это необходимо, что обусловлено кроме всего прочего накоплением погрешностей в ходе вычислений.

Точность представления числа характеризуется абсолютной и относительной погрешностями. Поскольку чаще всего знак ошибки неизвестен, то целесообразно пользоваться абсолютной погрешностью приближенного числа.

Абсолютная погрешность – это модуль разности между истинным значением входной величины A и ее значением, полученным из машинного изображения [A], т.е.

$$\Delta[A] = |A - [A]|. \qquad (2.28)$$

Усредненная абсолютная погрешность представления чисел в машинах с фиксированной запятой определяется как среднее арифметическое между минимально представимым числом и его минимальной потерей, т.е.

$$\Delta = \frac{A_{min}}{2} = \frac{2^{-n}+0}{2} = 2^{-(n+1)} \qquad (2.29)$$

т.е. в машинах с фиксированной запятой усредненная абсолютная погрешность постоянна и равна половине младшего разряда.

Относительная погрешность представления определяется как отношение усредненной абсолютной погрешности к самому числу:

$$\varepsilon = |\Delta/A|.$$

Так как само число A меняется в пределах

$$A_{min} = 2^{-n} \le |A| \le 1 - 2^{-n} = A_{max},$$

то и относительная погрешность является величиной переменной, меняющейся соответственно в пределах:

$$\varepsilon_{min} \le e \le \varepsilon_{max}.$$

Для машин с фиксированной запятой она определяется следующим образом:

$$\varepsilon_{min} = \left|\frac{A}{A_{max}}\right| = \frac{2^{-(n-1)}}{1-2^{-n}} \approx 2^{-(n+1)}. \qquad (2.30)$$

Таким образом, относительная погрешность для машин с фиксированной запятой зависит от величины числа и при $A \to 2^{-n}$, относительная погрешность может достигать 100%.

В машинах с плавающей запятой абсолютная погрешность представления числа определяется следующим образом:

$$\Delta = \Delta a \times 2^{m}, \qquad (2.31)$$

где: Δa – погрешность представления мантиссы, которая определяется также как абсолютная погрешность представления чисел в машине с фиксированной запятой, т.е. $\Delta a = 2^{-(n+1)}$;

m – порядок числа, который изменяется в пределах:

$$-(2^k - 1) \le m \le (2^k - 1).$$

Следовательно, в отличие от машин с фиксированной запятой, в машинах с плавающей запятой абсолютная погрешность представления чисел зависит от порядка числа: она минимальна при наибольшем отрицательном m и максимальна при наибольшем положительном т. Ащ^ и Дтах определяется здесь следующим образом:

$$\Delta_{min} = 2^{-(n+1)} \times 2^{-(2^k-1)} = 2^{-(n+2^k)}. \qquad (2.32)$$
$$\Delta_{max} = 2^{-(n+1)} \times 2^{2^k-1} = 2^{2^k-n-2}.$$

Относительная погрешность представления чисел в машинах с плавающей запятой определяется по общему правилу:

$$\varepsilon = \frac{\Delta}{|A|} = \frac{\Delta a \times 2^m}{a \times 2^m} = \frac{\Delta a}{a},$$

т.е. ε не зависит от порядка числа и изменяется в пределах:

$$\varepsilon_{min} = \frac{\Delta a}{a_{max}} = \frac{2^{-(n-1)}}{1 - 2^{-n}} \approx 2^{-(n+1)} \qquad (2.33)$$
$$\varepsilon_{max} = \frac{\Delta a}{a_{min}} = \frac{2^{-(n+1)}}{2^{-1}} \approx 2^{-n}.$$

Следовательно, в машинах с плавающей запятой, в отличие от машин с фиксированной запятой, относительная погрешность изображения чисел во всем диапазоне представлений практически постоянна и для чисел с нормализованной мантиссой зависит только от количества разрядов мантиссы: чем их больше, тем меньше погрешность представления.

В некоторых практических применениях вычислительных средств информационной единицей являются не отдельные числа, а их блоки или массивы, т.е. последовательности или векторы, состоящие из сотен и тысяч чисел. В этих случаях нередко применяется промежуточная форма представления чисел в ЭВМ, так называемое представление с поблочно-плавающей или условной (векторной) запятой, при котором всему массиву чисел присваивается общий порядок и массив считается нормализованным, если хотя бы одно его слово является нормализованным.

Естественно, что относительная погрешность представления отдельных элементов массива будет при этом различной.

Как и в случае представления с фиксированной запятой, максимальный по абсолютной величине элемент будет представлен с минимальной, в то время как минимальный по абсолютной величине элемент массива – с максимальной относительной погрешностями. Однако, как правило, для указанных применений это обстоятельство не имеет существенного значения, так как основную информационную нагрузку в этих случаях несут максимальные элементы массивов. Вместе с тем, благодаря представлению чисел с поблочно-плавающей запятой, удается при приемлемой точности вычислений значительно сократить объем оборудования, а главное – время выполнения операций, так как действия над порядками в этом случае выполняются только один раз за время обработки всего массива чисел.

На основании вышеизложенного можно сделать вывод о том, что нельзя отдать предпочтение какой-либо одной форме представления чисел. Для научно-технических расчетов в ЭВМ общего назначения обычно применяют нормальную форму. Этим обеспечивается большой диапазон представления чисел, высокая точность вычислений, простота программирования. Усложнение аппаратуры этих ЭВМ имеет второстепенное значение.

В специализированных ЭВМ чаще применяют фиксированную или поблочно-плавающую (векторную) запятую, если информация обрабатывается отдельными массивами, так как эти формы обеспечивают простоту конструкции ЭВМ Программа для этих ЭВМ составляется только один раз, диапазон изменения величин известен заранее, масштабные коэффициенты подбираются один раз, требуемая точность вычислений также известна заранее и определяет длину разрядной сетки.

В современных ЭВМ используются обе формы представления чисел, т.е. фиксированная и плавающая запятая. При этом в большинстве случаев формат чисел с фиксированной запятой служит для представления целых двоичных и десятичных чисел и выполнения операций над ними, что, например, необходимо для операций над кодами адресов (операции индексной арифметики).

В режиме с плавающей запятой иногда используется система счисления с основанием 16. В этом случае число А считается нормализованным, если хотя бы один из четырех старших двоичных разрядов отличен от 0. Это несколько уменьшает точность представления чисел, но позволяет резко увеличить диапазон представляемых в машине чисел и ускорить выполнение некоторых операций, в частности нормализацию,

так как сдвиг мантиссы производится сразу на 4 двоичных разряда. При этом каждый сдвиг мантиссы на 4 разряда влево или вправо требует соответствующего изменения порядка шестнадцатиричного числа всего лишь на единицу.

Точность представления в этом случае повышают за счет использования формата двойной длины (представление с удвоенной точностью).

Для упрощения действий над порядками в таких ЭВМ их сводят обычно к операциям над целыми положительными числами, используя представление чисел со "смещенным порядком" (смещение на 2^k). Например, при восьми разрядах, отводимых на представление порядка (К = 7 плюс знаковый разряд), $2^к = 128$. Тогда порядки диапазона

$$-2^k \le m \le 2^k - 1, \text{ т.е. } -128 \le m \le 127$$

будут ограничиваться пределами:

$$0 \le m^* = (m + 2^k) \le 2^k - 1,$$

что соответствует $0 \le m^* \le 255$. Смещенные порядки называют характеристиками, чтобы не путать действительные и условные величины. Так как все характеристики являются положительными, то это упрощает выполнение некоторых действий над ними, например, сравнения. Для этого вычитание из одной характеристики другой можно производить без предварительного анализа их знаков.

Упражнения к главе 2

1. Записать число 267_{10} в двоично-пятеричной системе счисления.
2. Вычислить количественный эквивалент числа А= 131204, записанного в двоично-пятеричной системе счисления.
3. Записать в двоично-десятичных системах счисления с весами соответственно 8421, 2421 и 8421+3 следующие десятичные числа:
   A = 89,43; B = 58,36; C = 62,28; D - 27,56; E = 57,18.
4. Записать в коде Грея числа $A = 1110_2$ и $B = 1010_2$.
5. Определить количественный эквивалент числа А= 1001 и В = 1111, записанных кодом Грея.
6. Перевести числа А = 89,43; В = 58,36; С = 62,28; D = 27,56; Е = 57,18 в двоичную, восьмеричную и троичную системы счисления.
7. Перевести числа А = 1000011,0011102 и В= 1010011,1011102 в десятичную систему счисления.
8. Записать числа $A = 1000011,001110_2$ и $B = 1010011,101110_2$ в 12-и разрядную сетку машины с фиксированной запятой, предварительно вы-

рав $К_А$, и в 15-и разрядную сетку машины с плавающей запятой при n = 8 и m = 7.

9. Определить абсолютную и относительную погрешность представле ния чисел А и В для примера 10.
10. Привести двоичное число А = 0,011000100 в систему (1, $\bar{1}$).
11. Перевести число А = 1 $\bar{1}$ 1 1 $\bar{1}$, $\bar{1}$ из системы (1, $\bar{1}$) в двоичную систему счисления с цифрами (0,1).

Контрольные вопросы к главе 2

1. Что такое система счисления?
2. Чем отличаются позиционные системы счисления от непозиционных?
3. В чем общность и различие однородных и неоднородных позиционных систем счисления?
4. Что такое самодополняющийся двоично-десятичный код?
5. Назовите специальные системы счисления?
6. Что такое система остаточных классов?
7. Приведите пример позиционной системы счисления с непостоянными весами разрядов.
8. Сформулируйте правило перевода целых чисел из одной системы счисления в другую.
9. Как переводят дроби из одной системы счисления в другую?
10. Как переводят неправильные дроби из одной позиционной системы счисления в другую?
11. Назовите критерии выбора систем счисления для применения в ЭВМ.
12. Какая система счисления считается наиболее экономичной и почему?
13. Какая из систем обеспечивает наибольшее быстродействие вычислительного устройства и почему?
14. Назовите разновидности двоичных систем счисления.
15. Какое соотношение выражает связь между канонической двоичной системой и системой (1, $\bar{1}$)?
16. Как перевести число из канонической двоичной системы в систему (1, $\bar{1}$)?
17. Что такое избыточная двоичная система и как она связана с канонической?
18. В чем преимущества и недостатки избыточной двоичной системы счисления?
19. Какие приемы, упрощающие обращение с двоичными числами Вы знаете?
20. Что такое форма представления числа в машине?

21. Какие формы представления чисел Вы знаете?
22. Как представляются числа в машинах с фиксированной и плавающей точками?
23. Назовите преимущества и недостатки различных форм представления чисел в ЭВМ.
24. Как оценить точность представления чисел в машине с фиксированной точкой?
25. Оцените абсолютную и относительную погрешности представления чисел в ЭВМ с плавающей точкой.
26. Чем различаются операции сложения двоичных чисел в прямом, обратном и дополнительном кодах?
27. Что такое переполнение разрядной сетки и как его устранить?
28. Как складываются числа, представленные в формате с плавающей точкой?
29. Что такое денормализация числа влево и вправо?
30. Чем различаются способы умножения двоичных чисел в прямом коде?
31. Оцените время умножения по четырем способам?
32. В чем состоят особенности умножения чисел в дополнительном коде?
33. Какие методы и способы ускорения операции умножения Вы знаете?
34. Оцените эффективцость методов ускоренного умножения.
35. Какие способы деления чисел Вы знаете?
36. Сформулируйте правило деления чисел в дополнительном коде.
37. Назовите способы округления результатов и дайте их оценку.
38. Назовите особенности округления чисел, представленных в дополнительном коде.

Глава 3 ВЫПОЛНЕНИЕ ОПЕРАЦИЙ АЛГЕБРАИЧЕСКОГО СЛОЖЕНИЯ И СДВИГА В ЭВМ

3.1. Общие замечания

Важнейшей функцией большинства вычислительных устройств является выполнение арифметических операций. В связи с этим в ЭВМ выделяют специальный функциональный блок – арифметическое устройство (АУ) – предназначенный для выполнения операций над числовыми кодами. Числа, участвующие в арифметических операциях, выполняемых ЭВМ, называют операндами.

Для позиционных систем счисления с естественными весами все допустимые числа являются полиномами по степеням Р (основания системы счисления). Следовательно, все арифметические действия в этом случае выполняются по правилам алгебраического сложения, умножения иделения полиномов.

Основной операцией в ЭВМ являются операция сложения. По способу ее выполнения АУ могут быть параллельного, последовательного и параллельно-последовательного действия.

В АУ последовательного действия производится последовательное суммирование всех п разрядов a_i и b_i слагаемых А и В.

С учетом того, что $a_i \le p-1$ и $b_i \le p-1$, а также того, что алфавит цифр S_i результата точно такой же, как и у слагаемых, последовательное суммирование операндов должно выполняться на основании следующего равенства

$$a_i + b = \Pi_i + S_i. \qquad (3.1)$$

При этом перенос Π_i из разряда с номером i принимает следующие значения

$$\Pi_i = \begin{cases} 0, & \text{при } a_i + b_i < p-1, \\ 1, & \text{при } a_i + b_i \ge p-1. \end{cases}$$

Так как при сложении полиномов должны суммироваться все члены с одинаковыми степенями, то (3.1) перепишется в виде

$$ai + bi + \Pi_{i-1} = \Pi_i + S_i. \qquad (3.2)$$

Если для суммы установлена та же длина слова, что и для слагаемых, то правильное представление суммы будет существовать только при

$\Pi_n = 0$ и для ее определения потребуется n тактов машинного времени (n тактов суммирования). В случае $\Pi_n = 1$ потребуется n + 1 такт суммирования. Из (3.2) также следует, что значение цифры суммы зависит от значения всех предыдущих разрядов слагаемых.

Формирование одного разряда суммы S_i и переноса из значений цифр слагаемых и переноса с предыдущего разряда производится с помощью одноразрядного сумматора по основанию р, длительность такта суммирования которого равна τ (рис. 3.1).

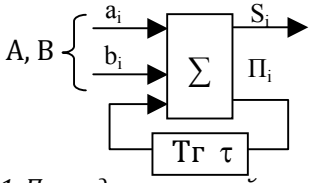


Рис. 3.1. Последовательный сумматор.

АУ параллельного действия содержит параллельный сумматор, в котором операция сложения одновременно выполняется над всеми n разрядами суммирования чисел А и В, следовательно, время выполнения операции сложения составляет один такт машинного времени (рис.3.2). Длительность этого такта с учетом того, что при формировании суммы может возникнуть перенос по всем разрядам результата, составит:

$$\tau_{max} = \tau_s + (n-1)\tau_n,$$

где τ_s и τ_n – время формирования одноразрядной суммы и переноса соответственно.

Параллельному способу выполнения операций соответствует минимальное время сложения при максимальном объеме оборудования (требуется n одноразрядных сумматоров).

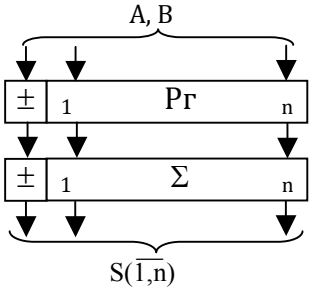


Рис 3.2. Параллельный сумматор.

Последовательному способу, наоборот, характерно максимальное время выполнения операции при минимальных затратах оборудования (один одноразрядный сумматор).

Параллельному способу выполнения операций соответствует минимальное время сложения при максимальном объеме оборудования (требуется n одноразрядных сумматоров). Последовательному способу, наоборот, характерно максимальное время выполнения операции при минимальных затратах оборудования (один одноразрядный сумматор).

Арифметические устройства параллельно-последовательного действия занимают промежуточное положение между двумя первыми типами АУ в отношении времени выполнения операции сложения и используемого оборудования. В таком АУ за один такт машинного времени находится сумма m разрядов слагаемых (часто 8-разрядных слов, которые называются байтами), т.е. искомая сумма определяется за |n/m| тактов машинного времени. В дальнейшем будет рассматриваться операция сложения только в АУ параллельного действия.

3.2. Операция алгебраического сложения в ЭВМ

При вычислении суммы двух чисел возможны два случая:

а) слагаемые имеют одинаковые знаки,

б) слагаемые имеют различные знаки.

В соответствии с этим алгоритмы получения суммы для каждого из вариантов значительно отличаются между собой. Так, алгоритм получения суммы двух чисел с одинаковыми знаками определяется следующим образом:

1. Сложить два числа.
2. Сумме приписать знак одного из слагаемых.

В то же время алгоритм получения алгебраической суммы записывается следующим образом:

1. Сравнить знаки слагаемых и, если они одинаковы, то выполнить сложение по первому алгоритму.
2. Сравнить слагаемые по абсолютной величине, если знаки слагаемых разные.
3. Если есть необходимость переставить числа местами (чтобы вычитать из большего меньшее).
4. Произвести вычитание двух чисел.
5. Результату присвоить знак большего по абсолютной величине слагаемого.

Как видно, первый алгоритм значительно проще второго. Следовательно, желательно преобразовать отрицательные числа таким образом» чтобы операцию вычитания заменить сложением, т.е. выполнять суммирование двух чисел следующим образом:

$$S = A + (-B).$$

Для этого необходимо изобразить положительные и отрицательные числа единым натуральным кодом, что возможно достичь естественным образом при использовании систем счисления с положительным основанием и симметричной базой либо с отрицательным основанием и неотрицательной базой. При использовании однородных позиционных систем счисления с положительным основанием и неотрицательной базой натуральным кодом можно представить только положительные-числа и нуль, т.е. в этом случае не существует единого натурального кода для положительных и отрицательных чисел. Это приводит к тому, что проблему представления чисел со знаком приходится решать на уровне слов при помощи специальных кодов: прямого, обратного и дополнительного. Способ построения этих кодов определяется функциями кодирования, которые должны обеспечить:

1. Запись алгебраического знака числа.
2. Представление отрицательных чисел при помощи вспомогательных положительных, которые отличаются по изображению от положительных исходных чисел, т.е. области изображений положительных и отрицательных чисел не должны пересекаться.
3. Полную идентичность алгоритмов выполнения операций над числами различных знаков и, следовательно, полную идентичность необходимого при этом оборудования.

3.2.1. Прямой код

Прямым кодом отрицательного числа называется его изображение в естественной форме записи, у которого в знаковом разряде проставляется единица. Прямой код положительного двоичного числа совпадает с его обычным изображением в естественной форме, так как знак кодируется нулем.

На основании данного определения, функция кодирования чисел в прямом коде для правильных дробей вида: $A = a_{зн}\, a_{-1}a_{-2}\ldots a_{-n}$ запишется следующим образом

$$[A]_{пр} = \begin{cases} A, & \text{при } A \geq 0 \\ 1 + |A|, & \text{при } A < 0 \end{cases} \qquad (3.3)$$

Величина числа А будет определяться в прямом коде следующим выражением:

$$A = (1 - 2a_{зн}) \sum_{i=-k}^{n} a_i P_i , \qquad (3.4)$$

при этом знаковому разряду не приписывается никакого веса. Очевидно,что диапазон изменения машинных изображений для прямого кода двоичной дроби лежит в пределах

$$-(1 - 2^{-n}) \le [A]_{пр} \le (1 - 2^{-n}). \qquad - \quad (3.5)$$

В геометрической интерпретации область положительных чисел будет совпадать с областью их изображений, а для отрицательных чисел эти области будут отличаться (рис. 3.3):

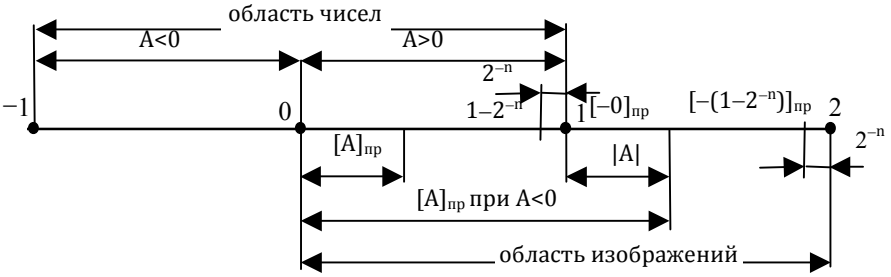


Рис. 3.3. Геометрическая интерпретация прямого кода.

Пример: записать числа А = +0,101011 и В = –0,0111010 в прямом коде.

$[A]_{пр}$ = 0,101011;
$[B]_{пр}$ = 1,0111010.

Отметим, что с ростом абсолютной величины числа А его код тоже возрастает во всей области изображения чисел.

В прямом коде нуль имеет два значения: положительное – 0,000... и отрицательное – 1,000... . Обычно в ЭВМ используется положительный нуль, но в процессе вычислений может возникнуть и его отрицательное изображение. Оба изображения полностью эквивалентны и применение любого из них не приводит к ошибке.

3.2.2. Сложение чисел в прямом коде

Правила сложения чисел, в прямом коде не отличаются от обычных правил сложения, т.е. если оба слагаемых имеют одинаковые знаки, то их цифровые части складываются, а сумме приписывается общий знак. Если

слагаемые с разными знаками, то из цифровой части большего по абсолютной величине числа вычитается меньшее, а сумме приписывается знак большего слагаемого. При этом числовые разряды кода обрабатываются отдельно от знаковых, так как последние не имеют веса.

Рассмотрим, 4 случая Получения суммы при $|A| > |B|$ и $|A| + |B| < 1$.

1. $A > 0; B > 0; C > 0$.

$$[A]_{пр} = A; [B]_{пр} = B; [C]_{пр} = A + B.$$

Пример: задано A = 0,10110; B = 0,00101, найти $[C]_{пр}$.

$$[C]_{пр} = A + B = 0,10110 + 0,00101 = 0,11011.$$

2. $A > 0; B < 0; C > 0$.

$$[A]_{пр} = A; [B]_{пр} = 1 + |B|; [C]_{пр} = A - |B|.$$

Пример: задано A = 0,10110; B = –0,00101, найти $[C]_{пр}$.

$$[A]_{пр}=0,10110; [B]_{пр}= 1,00101;$$
$$[C]_{пр} = A - B = 0,10110 - 0,00101 = 0,10001.$$

3. $A < 0; B > 0; C < 0$

$$[A]_{пр} - 1 + |A|; [B]_{пр} = B; [C]_{пр} = 1 + (|A| - |B|).$$

Пример: задано A = –0,10110; B = 0,00101, найти $[C]_{пр}$.

$$[A]_{пр}= 1,10110; [B]_{пр} = 0,00101;$$
$$[C]_{пр} = 1 + |A - B| = 1 + |0,10110 - 0,00101| = 1,10001.$$

4. $A < 0; B < 0; C < 0$

$$[A]_{пр} = 1 + |A|; [B]_{пр} = 1 + |B|; [C]_{пр} = 1 + |A| + |B|.$$

Пример: задано A = –0,10110; B = –0,00101, найти $[C]_{цр}$.

$$[A]_{пр}= 1,10110; [B]_{пр}= 1,00101;$$
$$[C]_{пр} = 1 + |0,10110 + 0,00101| = 1,11011.$$

Таким образом, в прямом коде знаковый разряд и цифровую часть числа нельзя рассматривать как единое целое, а выполнение операции сложения затруднено тем, что необходимо кроме сумматора иметь еще в составе машины и вычитатель кодов чисел. Эти недостатки настолько серьезны, что прямой код для выполнения операции алгебраического сложения не применяется, но зато удобен при выполнении операций умножения и деления.

3.2.3. Дополнительный код

Идея замены операции вычитания операцией сложения основана на применении некоторых вспомогательных положительных чисел, однозначно связанных с исходными отрицательными числами, в связи с чем эти вспомогательные числа можно считать исходными, но некоторым образом закодированными. Эти вспомогательные числа могут быть найдены, если знаковому разряду приписать вес $-p^{n+1}$. Величина числа A будет определяться в этом случае следующим образом:

$$A = -a_{зн}p^{n+1} + \sum_{i=-k}^{n} a_i P_i , \qquad (3.6)$$

при условии, что $a_{зн}$ принимает значение 0 для положительных чисел и 1 для отрицательных.

Если $a_{зн} = 0$, мы имеем множество положительных чисел и нуль. Если же $a_{зн} = 1$, то все числа отрицательны и их абсолютная величина есть

$$|A| = p^{n+1} - \sum_{i=-k}^{n} a_i P_i ,$$

т.е. она является дополнением количественного эквивалента, задаваемого основными разрядами числа a_n,..., a_{-k} до величины p^{n+1}. По этой причине такой код называют дополнительным. При этом наибольшая абсолютная величина числа A есть $|A| = p^{n+1}$, что соответствует случаю a_n,..., $a_{-k} = 0$.

Из вышеизложенного следует, что с учетом знаков функция кодирования правильных дробей в дополнительном коде записывается следующим образом:

$$[A]_д = \begin{cases} A, & \text{при } A \geq 0 \\ p + A, & \text{при } A < 0 \text{ или } p - |A| \end{cases} \qquad (3.6)$$

При этом для отрицательных чисел всегда имеет место

$$|A| + [A]_д = p \qquad (3.7)$$

или для двоичной системы $|A| + [A]_д = 2$.

Диапазон изменения машинных изображений двоичных чисел для формы представления о запятой, фиксированной перед старшим разрядом составляет

$$-1 \leq [A]_д \leq (1 - 2^{-n}). \qquad (3.8)$$

Функция кодирования целых k-разрядных чисел в дополнительном коде запишется таким образом

$$[A]_д = \begin{cases} A, & \text{при } A \geq 0, \\ p^{k+1} + A, & \text{при } A < 0. \end{cases}$$

Тогда диапазон изменения машинных изображений двоичных чисел для формы представления с запятой, фиксированной после младшего разряда составит:

$$-2^k \leq [A]_д \leq (2^k - 1). \qquad (3.9)$$

А соотношение (3.7) примет вид:

$$|A| + [A]_д = 2^{k+1}, \qquad (3.10)$$

где k – количество числовых разрядов.

Геометрическая интерпретация дополнительного кода правильной дроби при p = 2 представлено на рис. 3.4.

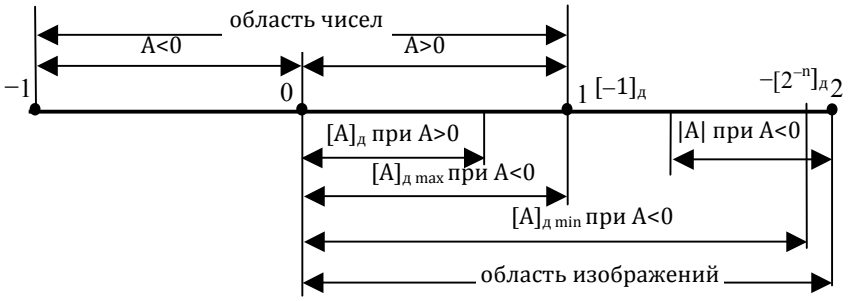


Рис 3.4. Геометрическая интерпретация дополнительного кода.

Как видно из рис. 3.4 с ростом абсолютной величины дополнительный код положительного числа возрастает, а отрицательного - убывает.

Ввиду того, что область чисел и область изображений равны по длине модулю p = 2, между числами и их изображениями имеет место однозначное соответствие. При этом область положительных чисел совпадает с областью изображений. Поэтому изображения положительных двоичных дробей не отличаются от их обычной двоичной записи, а для изображения правильной отрицательной дробив к ней нужно прибавить модуль 2, по которому сравнивается число с его изображением, т.е. получить дополнение до двух.

Примеры: Записать числа A = –0,101101 и B = –0,011010 в дополнительном двоичном коде:

$$\begin{array}{r} 10{,}000000 \\ A = \overline{\quad} \; 0{,}101101 \\ \hline [A]_д = 1{,}010011 \end{array} \qquad \begin{array}{r} 10{,}000000 \\ A = \overline{\quad} \; 0{,}011010 \\ \hline [A]_д = 1{,}100110 \end{array}$$

В случае отрицательных чисел нет необходимости суммировать А с основанием системы счисления, так как между отрицательными цифрами числовых разрядов прямого кода a_i и цифрами дополнительного кода a'_i имеются простые соотношения. Приравняем абсолютные величины одинаковых чисел, представленных прямым и дополнительным кодами; тогда

$$\sum_{i=-k}^{n} a_i p_i = p^{n+1} - \sum_{i=-k}^{n} a'_i p_i \; . \qquad (3.11)$$

Представим p^{n+1} в виде

$$p^{n+1} = \sum_{i=-k}^{n} (p-1)p^i + p^{-k} \qquad (3.12)$$

Подставляя (3.12) в (3.11) получим:

$$\sum_{i=-k}^{n} a_i p_i = \sum_{i=-k}^{n} (p - 1 - a'_i)p^i + p^{-k}$$

Приравнивая коэффициенты при одинаковых степенях р, имеем:

$$\begin{aligned} a_i + a'_i &= p - 1, \quad i = -k,..., n; \\ a_k + a'_k &= p \; . \end{aligned} \qquad (3.13)$$

Отсюда следует, что преобразование цифр прямого кода в дополнительный может быть произведено как поразрядная операция путем получения в каждом разряде дополнения до р–1 и прибавления к преобразованному числу единицы младшего разряда. В частности, для получения дополнительного кода отрицательной двоичной дроби необходимо в ее знаковом разряде записать 1, все цифры исходного числа заменить на инверсные (1 на 0 и 0 на 1) и прибавить единицу младшего разряда к преобразованному числу или, что то же самое, младшую единицу числа и все нули справа от нее оставить без изменения, а разряды слева от этой единицы заменить на инверсные.

Обратный переход от дополнительного кода числа к его естественному изображению производится аналогично.

Пример: привести $[A]_д = 1{,}010010$ и $[B]_д = 1{,}100110$ к естественному виду

$$A = -(0{,}101101 + 0{,}000001) = -0{,}101110;$$

$$B = -(0{,}011001 + 0{,}000001) = -0{,}011010.$$

Нуль в дополнительном коде имеет единственное значение: 0,000...00.

3.2.4. Алгебраическое сложение в дополнительном коде

При сложении в дополнительном коде знаковый разряд и цифровая часть числа рассматриваются как единое целое, поэтому машина оперирует с отрицательными числами как с неправильными дробями. Правильный знак суммы получается в процессе суммирования содержимого знаковых цифр операндов и единицы переноса из цифровой части, если она есть.

Для доказательства этих утверждений полагаем, что числа представлены в форме с запятой, фиксированной перед старшим разрядом. Считаем также для конкретности, что $|A| > |B|$. Рассмотрим следующие 4 возможных случая при $|A| + |B| < 1$:

1) $A > 0; B > 0; C > 0$.

Ввиду того, что изображение положительного числа в дополнительном коде не отличается от его изображения в прямом коде, сумма двух чисел определяется по общим правилам:

$$[C]_д = C = [A]_д + [B]_д = A + B.$$

Правильный знак при этом получается автоматически, так как знаковые цифры равны 0, а $|[A]_Д > [B]_д| < 1$ по условию, т.е. нет 1 переноса в знаковый разряд.

2) $A > 0; B < 0; C > 0$.

Сумма положительна, значит, результат должен быть получен в прямом коде.

$$[A]_д = A;\ [B]_д = 2 + B;\ [C]_д = C;$$
$$[C]_д = [A]_д + [B]_д = A + 2 + B.$$

Правильный знак суммы получается, так как, с одной стороны, один знаковый разряд равен 1, с другой, $-|[A]_Д + [B]_д| > 1$ в связи с тем, что $|[A]_д| = |A| > |B|$ (по условию) и $|B| + |[B]_д| = 1$, т.е. есть перенос в знаковый разряд суммы, что дает результирующий знак 0. В этом случае возникает перенос из знакового разряда суммы, который не должен учитываться, так как суммирование производится по модулю 2.

Другими словами, найденная сумма сравнима с результатом по модулю 2, т.е. требует коррекции (−2), которая производится автоматически, так как для изображения числа 2 в разрядной сетке ЭВМ нет места.

Пример: Заданы A = 0,10101, B = –0,01001. Найти $[C]_д$.

$$|A|_д = 0{,}10101$$
$$\underline{+[B]_д = 0{,}10111}$$
$$[C]_д = 0{,}01001$$

3) A < 0; B > 0; C < 0.

$$[A]_д = 2 + A;\ [B]_д = B;\ [C]_д = C_д = 2 + C;$$
$$[C]_д = [A]_д + [B]_д = 2 + A + B.$$

Так как сумма отрицательна, то она сразу получается в дополнительном коде. Правильный знак получается, так как один знаковый разряд равен 1 и нет 1 переноса в знаковый разряд суммы вследствие того, что

$$|[A]_д| + |[B]_д| < 1,$$

так как $[B]_д = |B| < |A|$, а

$$|A| + |[A]_д| = 1.$$

Пример: Заданы A = –0,10101, B = 0,01001. Найти $[C]_д$.

$$[A]_д = 1{,}01010$$
$$\underline{+[B]_д = 0{,}01001}$$
$$[C]_д = 1{,}10011$$

4) A<0; B<0; C < 0.

Так как сумма отрицательна, то результат получается в дополнительном коде,

$$[A]_д = 2 + A;\ [B]_д = 2 + B;\ [C]_д = 2 + C.$$

Правильный знак результата получается, так как оба знаковых разряда содержат 1 и есть 1 переноса из цифровых разрядов в знаковые вследствие того, что

$$|[A]_д| + |[B]_д| > 1,$$

так как |A| + |B| < 1 (по условию) и

$$|A| + |B| + |[A]_д| + |[B]_д| = 2.$$

В этом случае возникает перенос из знакового разряда суммы, который не должен учитываться, так как суммирование производится по модулю 2, т.е., как и во втором случае, сумма требует коррекции (-2), которая прор^водится автоматически, так как разрядная сетка ЭВМ не имеет разряда для изображения 2.

Пример: Заданы A = –0,10101; B = –0,01001. Найти $[C]_д$.

$$[A]_д = 1,01011$$
$$+[B]_д = 1,10111$$
$$[C]_д = 1,00010 \qquad [C]_{пр} = 1,11110$$

Таким образом, во всех случаях знак суммы формируется автоматически в результате применения общих правил поразрядного суммирования знаковых и цифровых разрядов операндов, при этом правила и результат суммирования не изменяется, если слагаемые поменять местами.

При сложении в дополнительном коде и неправильном выборе масштабных коэффициентов K_A слагаемых, в первом и в четвертом случае выполнения операции сложения возможно переполнение разрядной сетки. Признаком переполнения является отличие знака полученной суммы от знаков слагаемых.

Пример: Заданы A = 0,10101, B = 0,01110. Найти $[C]_д$. .

$$[A]_д = 0,10101$$
$$+[B]_д = 0,01110$$
$$[C]_д = 1,00011$$

Заданы A = –0,10101, B = –0,01110. Найти $[C]_д$.

$$[A]_д = 1,01011$$
$$+[B]_д = 1,10010$$
$$[C]_д = 0,00101$$

Из приведенных примеров видно, что при алгебраическом сложении двух двоичных чисел в дополнительном коде признаком переполнения является перенос в знаковый разряд суммы при отсутствии переноса из ее знакового разряда (положительное переполнение) или наличие переноса из знакового разряда суммы при отсутствии переноса в ее знаковый разряд (отрицательное переполнение). Если оба переноса есть или их нет, то переполнение отсутствует.

Для обнаружения переполнения разрядной сетки вводят вспомогательный разряд в знаковую часть изображения числа, который называют разрядом переполнения. Такое представление числа называется модифицированным, т.е. модифицированный дополнительный код отличается от обычного использованием двух знаковых разрядов. Тогда для модифицированного дополнительного двоичного кода функция кодирования запишется следующим образом

$$[A]^{м}_{д} = \begin{cases} A, & \text{при } A \geq 0, \\ 2^2 + A, & \text{при } A < 0. \end{cases}$$

Знак "+" числа запишется как 00, знак "–" как 11.

Геометрическая интерпретация дополнительной двоичного кода представлена на рис. 3.5.

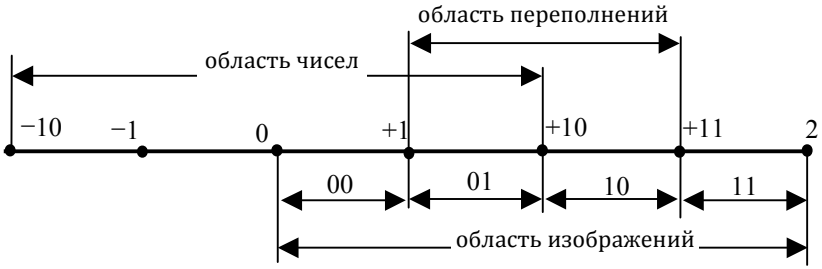


Рис. 3.5 Геометрическая интерпретация модифицированного дополнительного кода.

Область всех положительных чисел совпадает с областью изображений. Поэтому их изображения не отличаются от двоичной записи. Для перехода от отрицательного числа к его изображению нужно добавить к нему модуль 4.

Можно выделить 4 подобласти в области изображений:

1. Подобласть изображений положительных правильных дробей, признаком принадлежности к которой является сочетание 00 в знаковых (целых) разрядах кода.

2. Подобласть изображений положительных переполнений, признаком принадлежности к которой комбинация 01 в знаковых разрядах.

3. Подобласть изображений отрицательных правильных дробей, равных сумме двоичной дроби и модуля 4, признаком принадлежности к которой является комбинация 11 в знаковых разрядах.

4. Подобласть изображений отрицательных переполнений, равных сумме отрицательного переполнения с модулем 4, признаком принадлежности к которой является комбинация 10 в знаковых разрядах изображения.

Таким образом, признаком переполнения разрядной сетки при сложении в модифицированном коде будет наличие разных цифр в знаковых разрядах суммы. При этом левый знакбвый. разряд всегда сохраняет правильный код знака результата. Поэтому для устранения возникшего переполнения необходимо сдвинуть результат на один разряд вправо,

соответственно изменив масштабный коэффициент, а в правом знаковом разряде результата продублировать содержимое левого знакового разряда.

Пример: Задан ı а) A = 0,10101, B = 0,01110. Найти $[C]^{м}_{д}$.

$$[A]^{м}_{д} = 00,10101$$

$$+[B]^{м}_{д} = 00,01110$$

$$[C] = \overline{01,00011}$$

б) A = – ,10101, B =–0,01110.

$$[A]^{м}_{д} = 11,01011$$

$$+[B]^{м}_{д} = 11,10010$$

$$[C]^{м}_{д} = \overline{10,11101}$$

3.2.5. Обратный код

Если знаковый разряд числа имеет вес $-(p^{n+1} - p^{k})$, а не $-p^{n+1}$, как это было в дополнительном коде, то количественный эквивалент числа $A = a_{зн}a_n...a_0...a_{-k}$, заданного в позиционной системе счисления с основанием p, будет определяться как

$$A = -a_{зн}(p^{n+1} - p^{k}) + \sum_{i=-k}^{n} a_i P_i , \qquad (3.6)$$

Такие коды принято называть обратными, так как в них взаимное преобразование отрицательных чисел не требует привлечения арифметических операций. Положительные числа в обратном коде имеют такое же изображение, как и в прямом и дополнительном кодах. Наибольшая величина положительного числа вновь есть $p^{n+1} - p^{k}$, что для n-разрядной двоичной дроби составит $1–2^{-n}$. Отрицательных чисел теперь будет на одно меньше ввиду неоднозначности представления нуля. Действительно, код $a_{зн} = 1$ и $a_i = p - 1$, i = –k,...,n, имеет численный эквивалент 0, так же как и код $a_{зн} = a_i = 0$. Связь между значениями цифр прямого кода a_i и цифр обратного кода a_i'' выводится так же, как и для дополнительного кода и имеет вид

$$a_i + a_i'' = p - 1, \; i = -k,..., n.$$

Отсюда взаимное преобразование прямого и обратного кодов осуществляется как поразрядная операция получения дополнения до p–1 от заданных цифр. Следовательно, обратный код отрицательной двоичной

дроби равен ее инверсному значению и образуется по следующему правилу: в знаковом разряде числа проставляется 1, а все остальные цифры заменяются на взаимообратные. Переход от обратного кода к прямому аналогичен.

Функция кодирования в обратном коде запишется для двоичных дробей следующим образом:

$$[A]_o = \begin{cases} A, & \text{при } A \geq 0, \\ 2 - 2^{-n} + A, & \text{при } A < 0. \end{cases}$$

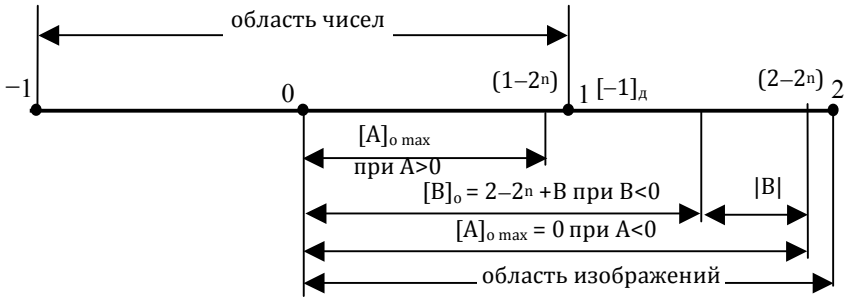


Рис. 3.6 Геометрическая интерпретация обратного кода

Пример: Заданы: A = –0,10101, B = –0,01110. Найти $[A]_o$ и $[B]_o$.

$[A]_o = 1,01010$; $[B]_o = 1,10001$.

Таким образом, преимуществом обратного кода перед дополнительным является более простая связь с прямым кодом, так как преобразование числа из прямого кода в обратный и наоборот является поразрядной операцией, что упрощает и ускоряет преобразование чисел.

Особенностью обратного кода является то, что при получении суммы, очевидно, необходимо предусмотреть коррекцию результата на 2^{-n}, так как инверсное значение чисел отличается от их дополнения на единицу младшего разряда.

3.2.6. Алгебраическое сложение в обратном коде

В обратном коде, как и в дополнительном, операция вычитания заменяется операцией сложения. При этом знаковый разряд и цифровая часть числа рассматриваются так же, как единое целое, вследствие чего машина оперирует с отрицательными числами, как с неправильными дробями. Правильный знак суммы получается автоматически в процессе суммирования цифр знаковых разрядов операндов и единицы переноса из цифро-

вой, части, если она есть. Характерной особенностью операции сложения в обратном коде является наличие циклического переноса (если он возникает) из знакового разряда в младший разряд цифровой части, благодаря которому осуществляется коррекция суммы на 2^{-n}. Это обусловлено тем, что сложение производится по модулю $2 - 2^{-n}$.

Рассмотрим 4 возможных случая сложения двух чисел при условии $|A| > |B|$ и $|A| + |B| < 1$.

1) $A > 0; B > 0; C > 0$.

$$[A]_o = a; [B]_o = B; [C]_o = [A]_o + [B]_o = A + B.$$

Так как обратный код положительных чисел не отличается от самих чисел, то вычисление их суммы совпадает с ее получением в прямом коде.

2) $A > 0; B < 0; C > 0$.

Сумма положительна, значит, результат должен быть получен в прямом коде.

$$[A]_o = A; [B]_o = 2 - 2^{-n} + B; \{C]_o = A + B + 2 - 2^{-n}.$$

Полученная сумма отличается от истинной на $(2 - 2^{-n})$, т.е. нужна коррекция на эту величину. Коррекция на 2 получается автоматически, так как для изображения 2 в разрядной сетке ЭВМ нет места. Коррекцию на 1 младшего разряда получают путем прибавления в младший разряд суммы единицы переполнения, которая возникает при сложении цифр знаковых разрядов, т.е. выполняется, так называемый, циклический перенос.

Пример: Заданы A = 0,10101, B = –0,01001 Найти $[C]_o$.

$$\begin{array}{r} [A]_o = 0{,}10101 \\ +[B]_o = 1{,}10110 \\ \hline 10{,}01011 \\ \quad\quad\quad\quad \longrightarrow +1 \\ \hline [C]_o = 0{,}01100 \end{array}$$

3) $A < 0; B > 0; C < 0$.

Сумма отрицательна, значит, результат должен получиться в обратном коде.

$$[A]_o = 2 - 2^{-n} + A; [B]_o = B; [C]_o = 2 - 2^{-n} + C = [A]_o + [B]_o = 2 - 2^{-n} + A+B,$$

т.е. результат коррекции не требует.

Пример: Заданы: A = –0,10101; B = +0,01001. Найти $[C]_o$.

$$[A]_o = 1{,}01010$$
$$\underline{+[B]_o = 0{,}01001}$$
$$[C]_o = 1{,}10011$$

Во втором и третьем случаях при A = B результат равен – 0, т.е. 1,1...11,

4) A < 0; B < 0; C < 0.

Сумма отрицательна, значит, результат должен получиться в обратном коде.

$$[A]_o = 2 - 2^{-n} + A;\ [B]_o = 2 - 2^{-n} + B;\ [C]_o = 2 - 2^{-n} + C =$$
$$= [A]_o + [B]_o = 2 - 2^{-n} + A + 2 - 2^{-n} + B.$$

Как и во втором случае необходима коррекция результата только на единицу младшего разряда, так как для изображения 2 в разрядной сетке ЭВМ нет места. Поправка вносится аналогичным второму случаю образом.

Пример: Заданы A = –0,10101; B = –0,01001 Найти $[C]_o$.

$$[A]_o = 1{,}01010$$
$$\underline{+[B]_o = 1{,}10110}$$
$$11{,}00000$$
$$\underline{\quad\quad\quad\quad\quad\quad \longrightarrow +1}$$
$$[C]_o = 1{,}00001$$

Очевидно, что во всех рассмотренных случаях сложения кодов слагаемые можно поменять местами и при этом код суммы не изменится. Требуемая для реализации коррекции суммы на величину 2^{-n} цепочка циклического переноса представляет собой цепь переноса из знакового разряда сумматора в младший, т.е. сумматор по переносам замыкается в кольцо. Эта цепочка может оставаться все время замкнутой, так как перенос из знакового разряда возникает только тогда, когда действительно необходимо реализовать коррекцию суммы. Время, необходимое для получения результата, возрастает незначительно, для сумматора дополнительного кода оно составляет $t_{сум} = (n - 1)e_n + e_s$, а для сумматора обратного кода – $t_{сум} = ne_n + e_s$.

В целом же и по аппаратным затратам, и по времени реализации, и по структуре алгоритма алгебраическое сложение обратных кодов фактически эквивалентно сложению дополнительных кодов.

Таким образом, обратный код достаточно удобен для выполнения операции алгебраического сложения, если учесть простоту перевода

отрицательных чисел из прямого кода в обратный и наоборот. Однако выполнять операции умножения и деления в обратном коде, как следует из главы 4, смысла нет.

В обратном коде переполнение разрядной сетки возможно при неправильном выборе масштабных коэффициентов в первом и в четвертом случаях. Признаком переполнения, как и в других кодах, является отличие знака результата от знаков слагаемых.

Для упрощения обнаружения переполнения разрядной сетки в ЭВМ используется модифицированный обратный код, функция кодирования которого записывается таким образом:

$$[A]^{м}_{о} = \begin{cases} A, & \text{при } A \geq 0, \\ 2^{2} - 2^{-n} + A, & \text{при } A < 0. \end{cases}$$

Признаком переполнения разрядной сетки являются разные цифры в знаковых разрядах суммы.

Пример: Заданы а) A = 0,10101, B = 0,01110. Найти $[C]^{м}_{о}$.

$$[A]^{м}_{с} = 00,10101$$

$$\underline{+[B]^{м}_{о} = 00,01110}$$

$$[C]^{м}_{о} = 01,00011$$ – положительное переполнение.

б) A = –0,10101, B = –0,01110.

$$[A]^{м}_{с} = 11,01011$$

$$\underline{+[B]^{м}_{о} = 11,10010}$$

$$[C]^{м}_{о} = 10,11011$$

$$\underline{\qquad\qquad \longrightarrow +1}$$

$$[C]^{м}_{о} = 10,11100$$ – отрицательное переполнение.

Таким образом, для замены операции вычитания, операцией сложения необходимо произвести специальное кодирование чисел.

В современных ЭВМ операции в режиме фиксированной запятой выполняются, как правило, над целыми числами. Отрицательные числа хранятся в оперативной памяти и участвуют в операциях в дополнительном коде. Операции выполняются над числами, выравненными по младшим разрядам, правее которых считается фиксированная запятая. Поэтому операнды – положительные целые числа, занимающие только часть разрядов слова, имеют во всех разрядах левее старшей значащей цифры

"0", а отрицательные – единицы. При необходимости удлинение операндов в сторону старших разрядов производится простым добавлением разрядов, содержимое которых равно содержимому знакового разряда (0 или 1).

Знаковые разряды участвуют в операциях сложения-вычитания наравне с цифровыми. Предварительный анализ знаковых разрядов операндов не производится, так как при выполнении команды "вычесть" вычитаемое автоматически преобразуется в инверсный код.

По окончании процесса суммирования производится анализ результата и выработка его признака: больше, меньше, равен "0", переполнение.

3.3. Операция сдвига

К операции сдвига приходится обращаться по меньшей мере при выполнении сложения в машине с плавающей запятой, а также при выполнении операций умножения и деления в ЭВМ обоих типов. Во всех этих операциях производится сдвиг мантисс, поэтому будут рассматриваться только числа с фиксированной запятой.

Сдвиг прямого кода числа на к разрядность вправо эквивалентен умножению этого числа на $2^{к}$. Ввиду того, что при сдвиге вправо младшие разряды сдвигаемого числа выходят за пределы разрядной сетки машины и теряются, пофешность представления сдвинутого кода числа имеет отрицательный знак для кодов положительных чисел и положительный знак для кодов отрицательных чисел. Для ее уменьшения необходимо предпринимать округление чисел. При сдвиге прямого кода отрицательной дроби сдвигается только ее мантисса, а знак остается без изменения.

Пример: Заданы A = 1,011010, k = 1

левый сдвиг	правый сдвиг
$2^{1}\times A = 1{,}110100$	$2^{-1}\times A = 1{,}001101.$

Сдвиг прямого кода числа влево на k разрядов эквивалентен умножению числа на 2 . Эта операция корректна до тех пор, пока старшие значащие цифры кода не начнут выходить за пределы разрядной сетки, т.е. пока число по абсолютной величине не станет больше 1. При сдвиге влево освобождающиеся справа разряды имеют значение нуль.

Сдвиг положительного числа влево или вправо в дополнительном или обратном коде ничем не отличается от сдвига положительного числа в прямом коде.

Под сдвигом отрицательного числа А, записанного инверсным, т.е. дополнительным или обратным кодом, понимается преобразование инверсного кода отрицательного числа А в инверсный код отрицательного числа $A\times2^{-k}$ в случае сдвига вправо и $A\times2^{-k}$ в случае сдвига влево.

Это означает, что сдвиг вправо на k разрядов отрицательного числа, записанного дополнительным кодом, должен преобразовать код

$$[A]_д = 2 + A \text{ в код } [A\times2^{-k}]_д = 2 + A\times2^{-k}. \qquad (3.16)$$

Если выполнять механическую операцию сдвига кода числа $[A]_д$ на k разрядов вправо, то получим:

$$[A]_д\times2^{-k} = (2 + A)\times2^{-k} = 2^{-(k-1)} + A\times2^{-k},$$

т.е. результат будет отличаться от требуемого на величину:

$$[A\times2^{-k}]_д - [A]_д\times2^{-k} = 2 - 2^{-(k-1)}.$$

Коррекция производится путем заполнения старших освободившихся разрядов кода знаковыми единицами, так как

$$2 - 2^{-(k-1)} = 1 + 2^{-1} + 2^{-2} + ... + 2^{-(k-1)}.$$

Пример: сдвинуть число $[A]_д$ = 1,011011000 на 3 разряда вправо

$$[A\times2^{-3}]_о = = 1,111011011.$$

В примере при последующих сдвигах вправо возникает ошибка, уменьшение которой потребует округления результата.

При сдвиге вправо отрицательных дробей, представленных обратным кодом, необходимо получить:

$$[A\times2^{-k}]_о = 2 - 2^{-n} + A\times2^{-k}. \qquad (3.17)$$

При механическом сдвиге получим код:

$$[A]_о\times2^{-k} = (2 - 2^{-n} + A)\times2^{-k} = (2 - 2^{-n})\times2^{-k} + A\times2^{-k},$$

т.е. необходима поправка на величину:

$$[A\times2^{-k}]_о = [A]_о\times2^{-k} = 2 - 2^{-n} - (2 - 2^{-n})\times2^{-k} = (1 + 2^{-1} + ... + 2^{-(n-1)} + 2^{-n}) -$$
$$- (2^{-k} + 2^{-k-1} + ... + 2^{-n} + 2^{-n-1} + ... + 2^{-n-k}) = 1 + 2^{-1} + ... + 2^{-k+1} - 2^{-n-1} - ... - 2^{-n-k}.$$

Коррекция производится путем заполнения старших освободившихся разрядов знаковыми единицами, при этом 1 в знаковом разряде восстанавливается после каждого сдвига числа на 1 разряд (т.е. она фактически остается неподвижной), а младшие отрицательные единицы

выходят за пределы разрядной сетки и не учитываются, т.е. отбрасываются.

Пример: Задано $[A]_o = 1{,}01011$, $k = 2$,

$[A \times 2^{-2}]_o = 1{,}11010$.

Сдвиг вправо кода двоичного числа с заполнением освободившихся старших разрядов знаковыми символами, носит название модифицированного сдвига.

Дополнительный код отрицательного числа, сдвинутый на k разрядов влево, должен быть равен:

$$[A \times 2^k]_д = 2 + A \times 2^k. \qquad (3.18)$$

Непосредственное умножение $[A]_д$ на 2 дает:

$$[A]_д \times 2^k = (2 + A) \times 2^k = 2^{k+1} + A \times 2^k.$$

Для получения истинного кода необходима поправка на величину:

$$[A \times 2^k]_д - [A]_д \times 2^k = 2 - 2^{k+1},$$

которая вносится автоматически, так как нет разрядов для записи целых чисел. При этом освобождающиеся справа разряды заполняются нулями. Сдвиг допусти́м до тех пор, пока в разряде справа от запятой не появится код "0".

Пример: заданы $[A]_д = 1{,}11011$ и $k = 2$, тогда $[A \times 2^k]_д = 1{,}01100$.

При сдвиге влево отрицательной дроби, заданной обратным кодом, должна получиться дробь:

$$[A \times 2^k]_o = 2 - 2^{-n} + A \times 2^k. \qquad (3.19)$$

Умножение на 2 обратного кода числа дает:

$$[A]_o \times 2^k = 2^{k+1} - 2^{k+n} + A \times 2^k.$$

отличающийся от истинного на величину:

$$[A \times 2^k]_o - [A]_o \times 2^k = (2 - 2^{k+1}) + (2^{k-n} - 2^{-n}).$$

Поправка вносится следующим образом: поправка на величину $(2 - 2^{k+1})$ вносится автоматически тем, что старшие разряды кода выходят за пределы разрядной сетки. Поправку на величину $(2^{k-n} - 2^{-n})$ можно внести, заполняя младшие освободившиеся разряды единицами. Это заполнение можно рассматривать как результат действия циклического переноса из знакового разряда в младший, т.е. при каждом сдвиге все

цифры изображения дроби циклически смещаются влево на 1 разряд, а знаковая единица по цепи циклического переноса смещается в разряд 2^n.

Пример: задано $[A]_o = 1{,}11010$ и $k = 2$, тогда $[A \times 2^k]_o = 1{,}01011$.

Таким образом, общим правилом для сдвига дробей вправо при представлении числа любым инверсным кодом является наличие передачи из знакового разряда в старший цифровой разряд и восстановление знака, т.е. знак не сдвигается. При сдвиге прямого кода отрицательного числа знак также не должен сдвигаться, однако передач из знакового в старший цифровой разряд нет. Так как количество разрядов ограничено, то при сдвиге вправо инверсного кода числа, начиная с какого-то момента, получается "0" при положительном исходном числе и $2 - 2^{-n}$ при отрицательном.

Количество сдвигов правильной дроби влево ограничено условием $|A \times 2^k| < 1$, т.е. сдвиг допустим лишь до тех пор, пока сохраняется знак результата. Перемена знака при сдвиге влево является признаком переполнения, который для отрицательных и положительных чисел совпадает с признаком переполнения, возникающим при сложении кодов двух чисел. При сдвиге влево обратного кода числа необходима также циклическая передача содержимого знакового разряда в младший разряд кода 2^{-n} и восстановление знака.

3.4. Сложение чисел в машинах с плавающей запятой

Результат сложения двух чисел $A = a \times p^{m_a}$ и $B = b \times p^{m_b}$, представленных в форме с плавающей запятой, должен быть тоже числом вида $C = c \times p^{m_c}$. При этом для слагаемых и результата должно выполняться равенство:

$$a \times p^{m_a} + b \times p^{m_b} = c \times p^{m_c}. \qquad (3.20)$$

Так как числа с разными порядками суммировать нельзя, то для того чтобы сложить два числа, представленных в нормальной форме, необходимо предварительно привести их к общему порядку, т.е. преобразовать одно из слагаемых, например B, следующим образом:

$$B = b \times p^{m_b} = b' \times p^{m_{общ}}.$$

После этого можно вынести степень основания системы за скобки и выполнить сложение мантисс:

$$A+B = a \times p^{m_{общ}} + b' \times p^{m_{общ}} = (a+b')p^{m_{общ}}. \qquad (3.21)$$

Преобразованная мантисса должна быть правильной дробью с тем, чтобы при сложении ничем не отличаться от обычных (непреобразованных) мантисс. Поэтому преобразованию подвергается всегда меньшее слагаемое, так как в противном случае происходит переполнение разрядной сетки мантиссы преобразуемого числа. При этом часть разрядов преобразуемой мантиссы может теряться. Поэтому арифметические действия над числами с плавающей запятой являются по своей сути приближенными, а не точными. В связи этим вопросам точности вычислений при представлении чисел в нормальной форме надо уделять особое внимание.

При сложении чисел в нормальной форме можно выделить 4 этапа:

На *первом этапе* уравниваются порядки слагаемых: меньший порядок увеличивается до большего, а мантисса преобразуемого числа сдвигается вправо на соответствующее количество разрядов. С этой целью производится вычитание порядков числа. Знак и модуль разности будет определять, соответственно, какое из слагаемых нужно преобразовать и на сколько разрядов следует сдвинуть его мантиссу. При этом младшие разряды мантиссы могут пропадать, вследствие чего в слагаемое, сдвигаемое вправо, вносится погрешность.

На *втором этапе* производится преобразование мантисс слагаемых в один из модифицированных кодов: дополнительный или обратный.

На *третьем этапе* выполняется сложение мантисс по правилам сложения чисел с фиксированной запятой.

На *четвертом этапе* производится нормализация результатов (в случае необходимости), затем результат переводится в прямой код, к нему приписывается общий порядок слагаемых и выполняется округление мантиссы результата.

В зависимости от абсолютных величин мантисс слагаемых сумма может получиться: нормализованной, денормализованной вправо, денормализованной влево (переполнение).

Следует отметить, что положительные нормализованные числа всегда имеют 1 в разряде p^{-1}, а отрицательные числа, записанные инверсным кодом, имеют 0 в разряде p^{-1}. Поэтому у нормализованных чисел значения цифр разрядов с весами p^0 и p^{-1} не совпадают ни для положительных чисел, ни для изображений отрицательных чисел. Вследствие этого несовпадение цифр в знаковых разрядах, говорит о денормализации влево (переполнение), а совпадение цифр знакового и старшего разряда мантиссы - о нарушении нормализации вправо.

При переполнении (денормализация результата влево) мантисса результата сдвигается на один разряд вправо, а порядок увеличивается на

единицу, так как сумма двух мантисс может быть денормализована влево не более чем на 1 разряд. Количество разрядов, на которое может нарушиться нормализация вправо, ограничено только длиной разрядной сетки. Поэтому при денормализации вправо мантисса сдвигается влево до появления в старшем разряде 1, при значении знака 0, или 0 при 1 в знаковом разряде, а из порядка вычитается число единиц, равное числу сдвигов мантиссы.

Обычно количество сдвигов влево ограничивают числом разрядов сумматора. Поэтому после выполнения предельного числа сдвигов мантиссу результата представляют машинным нулем. Мантиссу результата представляют также машинным нулем, если в процессе ее сдвига влево порядок числа окажется меньше допустимого, т.е. абсолютная величина результата будет меньше, чем минимально возможное машинное число. Однако приравнивание результата нулю при отрицательном переполнении порядка допустимо только тогда, когда результат должен складываться со значительно большей величиной. Если исчезновение порядка не отмечается каким-то образом, то могут возникнуть неожиданные ситуации, обусловленные ограниченностью разрядной сетки мантиссы и округлением результатов, например, может сказаться $(A\times B)\times C = 0$, а $(A\times C)\times B \neq 0$.

При сложении может произойти истинное переполнение разрядной сетки числа, т.е. переполнение разрядной сетки порядка. В этом случае минимум одно из слагаемых должно иметь максимальный порядок, а мантисса результата должна получиться денормализованной влево. При этом в ЭВМ формируется признак переполнения порядка.

Таким образом, после суммирования мантисс обоих слагаемых с выравненными порядками сначала следует проверить, не нарушена ли нормализация влево, и, если нарушена, то восстановить нормализацию. Если нормализация влево не нарушена, необходимо проверить нарушение нормализации вправо, и в случае необходимости, восстановить нормализацию.

Примеры:

1) $[A]_{пр}= 0\underbrace{011}_{m_a}\underbrace{1\,10101}_{a}$; $[B]_{пр}= 0\underbrace{101}_{m_b}\underbrace{0\,11001}_{b}$. Найти $[C]_{пр}$.

$[A]'_{пр}= 0\underbrace{101}_{m_a}\underbrace{1\,0010101}_{a}$;

$[a]^{м}_{д} = 11{,}1101011;$

$$\begin{array}{l} {}^{+}[b]^{м}_{д} = 00{,}1100100 \\ \hline [c]^{м}_{д} = 00{,}1001111 \end{array}$$ результат нормализован, поэтому производим только его усечение: $[C]_{пр} = 0\ 101\ 0\ 10011$.

2) $[A]_{пр} = 0\underbrace{101}_{m_a}\underbrace{1\ 10101}_{a}$; $[B]_{пр} = 0\underbrace{011}_{m_b}\underbrace{0\ 11001}_{b}$. Найти $[C]_{пр}$.

$[B]'_{пр} = 0\underbrace{101}_{m_b}\underbrace{0\ 00110011}_{b}$;

$[a]^{м}_{д} = 11{,}0101100$;

$$\begin{array}{l} {}^{+}[b]^{м}_{д} = 00{,}0011001 \\ \hline [c]^{м}_{д} = 11{,}1000101 \end{array}$$

Нормализация: $[c]_{д} = 1{,}000101$,. так как результат денормализован вправо на 1 разряд. Коррекция порядка: $m' = m_{общ} - 1 = 100$. Перевод в прямой код и усечение мантиссы результата:

$[c]_{пр} = 1{,}111011$. Окончательно имеем:

$[C]_{пр} = 0\ 100\ 1\ 11101$.

При выполнении сдвига влево необходимо следить, чтобы справа не было введено ошибок.

3) $[A]_{пр} = 0\underbrace{101}_{m_a}\underbrace{1\ 10101}_{a}$; $[B]_{пр} = 0\underbrace{100}_{m_b}\underbrace{1\ 11001}_{b}$. Найти $[C]_{пр}$.

$[B]'_{пр} = 0\underbrace{101}_{m_b}\underbrace{1\ 011001}_{b}$;

$[a]^{м}_{д} = 11{,}010110$;

$$\begin{array}{l} {}^{+}[b]^{м}_{д} = 11{,}100111 \\ \hline [c]^{м}_{д} = 10{,}111101 \end{array}$$

Результат денормализован влево, поэтому мантиссу сдвигаем вправо на один разряд:

$$[c]_{д} = 1{,}0111101.$$

Переводим в прямой код и производим усечение мантиссы: $[C]_{пр} = 0\ 110\ 1\ 1\ 0000$.

В системах команд современных ЭВМ имеется большой набор команд для выполнения сложения и вычитания как с длинными так и с короткими операндами с нормализацией и без нормализации результата. Выполнение операций вычитания отличается от сложения тем, что в начале операции знак второго операнда искусственно меняется на обратный.

Так как иногда основание системы счисления берется равным $2^4 = 16$, то при выравнивании порядков сдвиги производятся на количество разрядов, кратное 4. В основной памяти и в АУ порядки и мантиссы хранятся в прямом коде. Поэтому в АУ мантиссы с одинаковыми знаками всегда складываются как положительные числа, а результату присваивается знак первого операнда. Если же знаки не равны, то при алгебраическом сложении мантисса вычитаемого преобразуется в инверсный код. Если результат от сложения кодов мантисс получается в инверсном коде, то в конце выполнения операции он переводится в прямой код.

Переполнение, т.е. денормализация влево, устраняется сдвигом мантиссы вправо на четыре разряда и увеличением характеристики на единицу. Если возникает при этом переполнение порядка, то формируется требование прерывания. При нормализации влево может появиться характеристика меньше нуля. В этом случае также вырабатывается требование прерывания, а порядку и мантиссе результата присваиваются нулевые значения.

Если в результате алгебраического сложения получилась мантисса, равная 0, то реакция ЭВМ на этот результат зависит от того, разрешено или нет прерывание при потере значимости. Знак суммы при нулевой мантиссе результата всегда положителен.

3.5. Округление чисел в ЭВМ

Погрешность результата, в основном, зависит от численного метода вычислений, обусловливающего количество операций, выполняемых с округлением, топологию связей операндов, т.е. порядок следования операций. Погрешность результата зависит также от длины разрядной сетки, формы представления чисел, ошибок входных данных и способа округления результатов операций.

Поэтому системы счисления и длины разрядной сетки ЭВМ, а также формы представления числа в машине зависят, в значительной мере, от требуемой точности вычислений. Точность вычислений определяется погрешностью выполнения арифметических операций. Суммарная погрешность вычислений зависит от всех перечисленных факторов, и стремление повысить точность результата, воздействуя лишь на один из них, может не дать положительного эффекта.

Методическая погрешность вычислений возникает всегда вследствие того, что в ЭВМ обрабатывается не непосредственное математи-

ческое описание задачи, а ее модель, представленная в терминах численного анализа, которая является всегда приближенной.

Под вычислительной погрешностью ε_o понимается суммарная погрешность, состоящая из ошибок округления выполняемой операции и трансформированной (наследственной) погрешности от погрешностей операндов.

Под округлением понимается такое преобразование числовой информации, при котором некоторый m-значный числовой набор превращается в n-значный, $n < m$.

Арифметические операции сложения и вычитания в режиме с фиксированной запятой, при условии отсутствия переполнения, выполняются без округления, т.е. не генерируют ошибок. Однако результат выполнения этих операций будет получен с погрешностью, определяемой погрешностями задания операндов, т.е. эти операции в ходе вычислительного процесса изменяют лишь вычислительную погрешность по правилам алгебраического сложения погрешностей операндов.

При оптимальном округлении имеем для операций умножения и деления

$$-\frac{1}{2}\cdot 2^{-n} \le \varepsilon_o \le \frac{1}{2}\cdot 2^{-n}.$$

Таким образом, при выполнении арифметических операций в режиме с фиксированной запятой источниками ошибок округления являются только операции умножения и деления.

Источниками погрешностей при сложении в машине с плавающей запятой являются сдвиг вправо мантиссы одного из исходных чисел при выравнивании порядков, сдвиг вправо мантиссы при нормализации ре* зультата, а также искусственная установка нуля в качестве результата При отрицательном переполнении порядков. Поэтому при нормальной форме представления чисел сама операция алгебраического сложения также является источником погрешностей.

3.5.1. Округление чисел в прямом коде

Если предположить, что исходная информация не содержит никаких ошибок и все вычислительные процессы выполняются абсолютно точно, то всегда существует третий тип ошибок – ошибки округления, которые возникают при переводе чисел из одной системы счисления в другую и последующем представлении их в ограниченной разрядной сетке машины, а также при получении внутри машины чисел, разрядностью, большей чем это допустимо, например, при умножении. В этом случае число A округляют, т.е. заменяют его машинным числом [A]

заданной разрядности. Округление (обозначим его знаком Ŗ) называется оптимальным, если для любого машинного числа [A] справедливо ŖA= [A]. Пусть $[A]_1$ и $[A]_2$ – два последовательных машинных числа, тогда при округлении вещественное число A такое, что $[A]_1 < A < [A]_2$ заменяется либо числом $[A]_1$, либо числом $[A]_2$. Если ŖA < A, то говорят об округлении по недостатку, если ŖA > A, то говорят об округлении по избытку. Округление называют симметричным, когда ŖA = −Ŗ(−A). Различают три вида симметричного округления.

1. Округление в направлении к нулю – когда вещественное число округляется до ближайшего к нулю машинного числа.

2. Округление в направлении от нуля – когда округление производится до машинного числа, лежащего дальше от нуля, чем вещественное число A.

3. Округление по дополнению – когда округление производится до ближайшего машинного числа.

В качестве параметров, по которым будут сравниваться способы округления, целесообразно использовать максимальную величину модуля погрешности, т.е. Δ_{max}, где $\Delta = |A-[A]|$, и математическое ожидание погрешности округления $\bar{\Delta}$.

1. *Округление к нулю или усечение.* Для конкретности считаем, что числа в машине представлены в прямом коде с запятой, фиксированной перед старшим разрядом, т.е. $|[A]| = a_{-1}, ..a_{-n}$. Пусть в результате каких-либо действий над машинными числами внутри машины сформировалось число [A]', имеющее k = n+t разрядов. Очевидно, что самый простой способ округления состоит в отбрасывании "хвоста" числа $[A]^1$, который состоит из лишних разрядов, т.е. разрядов с номерами $a_{-n-1}, a_{-n-2}, ..., a_{-n-t}$. Тогда величина абсолютной погрешности приближения есть

$$\Delta = \left| \sum_{i=1}^{t} a_{-n-i} \cdot p^{-n-i} \right| \qquad (3.22)$$

Очевидно, что при усечении $\Delta_{max} = p^{-n}(1-p^{-t})$. Если считать, что появление чисел с абсолютной величиной A, но разных знаков равновероятно и равновероятны все значения "хвоста" чисел одного знака, то математическое ожидание погрешности в данном случае равно нулю, т.е., $\bar{\Delta} = 0$.

На практике обычно вероятность появления чисел равного знака при выполнении определенной программы не одинакова, и поэтому представляет интерес округление абсолютных величин, т.е. фактически чисел одного знака. При этом значение Δ_{max} останется прежним, но

математическое ожидание погрешности округления будет отлично от нуля и определится как

$$\overline{\Delta} = 2^{-1} \cdot p^{-n}(1 - p^{-t}) \qquad (3.23)$$

Это значит, что при действиях с числами одного знака погрешность усечения носит систематический характер, что приводит к накоплению погрешностей. Это обстоятельство заставляет исследовать другие способы округления, которые рассматриваются пока для прямых кодов.

2. *Округление от нуля*. Реализация данного способа требует анализа "хвоста" на нуль, затем отбрасывается "хвост" и, если отсекаемая часть не равна нулю, к абсолютной величине остающейся части добавляется единица в младший разряд. Тогда

$$R[A'] = \pm \left(\sum_{i=-n}^{-1} a_i \cdot p^i + p^{-n} \right) \qquad (3.22)$$

Это добавление может вызвать распространение переносов через все разряды числа. Помимо дополнительных временных затрат это может привести к переполнению в разрядной сетке. Следовательно, способ сложнее в реализации, хотя его основные характеристики точно такие же, как при усечении.

4. *Округление по недостатку*. Реализация данного способа базируется на анализе знака округленного числа. Если $[A]' > 0$, то округление заключается в отбрасывании "хвоста". Если же $[A]' < 0$, то «хвост» также отбрасывается, а к величине оставшейся части прибавляется единица в младший разряд, если "хвост" не равен нулю. Таким образом, реализация данного сособа более усложнена по сравнению со способом округления к нулю пза счет анализа знака числа $[A]'$, хотя величина Δ_{max} осталась при этом прежней. При $A' \geq 0$ данный способ совпадает с усечением, а при $A' < 0$ – с округлением от нуля. Отсюда ясно, что он не может конкурировать с усечением результатов.

4. *Округление по избытку*. Этот способ во всем подобен предыдущему с тем отличием, что добавление единицы в младший разряд сохраняемой части числа производится, когда больше нуля и «хвост» не равен нулю. При $A' < 0$ просто отбрасывается. Характеристики данного способа точно такие же, как у предыдущего, за исключением знака величины $\overline{\Delta}$, который меняется на противоположный.

5. *Округление по дополнению.* Данный способ представляет собой объединение способов округления от нуля и к нулю. Его реализация связана с коррекцией сохраняемой части числа A', которая производится по результатам анализа значения старшей цифры отсекаемой части a_{-n-1}, т.е.

цифры дополнительного (n+1)-го разряда (ДР). В этом случае округленное число Ŗ[A]' будет иметь значение

$$Ŗ[A]' = \pm\left(\sum_{i=-n}^{-1} a_i \cdot p^i + \alpha p^{-n}\right) \qquad (3.25)$$

где $\alpha = \begin{cases} 1, & \text{при } a_{-n-1} \geq p/2 \\ 0, & \text{при } a_{-n-1} < p/2 \end{cases}$.

Когда а = 0, происходит округление к нулю, в противном случае – от нуля. Значение максимума погрешности при этом составляет

$$\Delta_{max} = 2^{-1} \cdot p^{-n}.$$

Пример: Заданы А = 0,11011111; 8 = 0,10101101.

Округлить А и В до n = 6.

$$Ŗ A = 0{,}110111 \quad +1 \quad \underline{\qquad} \quad 0{,}111000$$

$$Ŗ B = 0{,}101011 \quad +0 \quad \underline{\qquad} \quad 0{,}101011$$

В случае равновероятного появления чисел разных знаков и равномерного распределения вероятностей появления различных значений «хвоста» числа математическое ожидание погрешности округления равно нулю. Однако при округлений чисел одного знака значение $\bar{\Delta}$ отлично от нуля. Поэтому при округлении чисел одного знака данный способ дает систематические ошибки округления, хотя и меньшие, чем при усечении.

Таким образом, по своим характеристикам способ округления по дополнению лучше, чем усечение. Систематические ошибки при округлении чисел одного знака обусловлены в данном случае тем, что округление особенно неточно производится, когда значение отсекаемой части близко половине единицы младшего сохраняемого разряда. Этот недостаток устраняется в следующем способе округления.

6. *Усовершенствованное округление по дополнению.* В этом случае решение о коррекции сохраняемой части числа А' принимаемся на основе анализа значения всех разрядов его отсекаемой части $A_o = a_{-n-1},\ldots,a_{-n-t}$, а не только старшего. Пусть для четного a_{-n} $\varphi = 0$, иначе $\varphi = 1$. Тогда параметр α в выражении (3.25) определится по следующим правилам:

$$\alpha = \begin{cases} 1, & \text{если } A_0 > 2^{-1} \cdot p^{-n}, \\ \varphi, & \text{если } A_0 = 2^{-1} \cdot p^{-n}, \\ 0, & \text{если } A_0 > 2^{-1} \cdot p^{-n}. \end{cases}$$

При этом величина $\Delta_{max} = 2^{-1} \cdot p^{-n}$, т.е. остается прежней. Однако систематические ошибки сводятся к нулю, так как при одинаковой вероятности появления четного и нечетного значения цифры младшего разряда сохраняемой части значения $\varphi = 0$ и $\varphi = 1$ равновероятны, т.е. $\bar{\Delta} = 0$. Таким образом, усовершенствованный способ дает хорошее округление в случае чисел одного знака, однако это достигается за счет усложнения его реализации.

7. *Упрощенное округление по дополнению.* При реализации способов округления по дополнению из-за возможного возникновения переносов при суммировании сохраняемой части числа с единицей округления необходимо выполнить операцию сложения, что требует дополнительного времени, т.е. снижает реальное быстродействие ЭВМ и кроме того, может повлечь за собой переполнение разрядной сетки. Распространения переносов не будет, если при $a_{-n} = p - 1$ отменить коррекцию. Для $p = 2$ этот способ округления состоит в том, что младший разряд сохраняемой части числа принудительно устанавливается в единицу, если старший разряд отбрасываемой части равен единице.

При этом, если в неокругленном результате разряд a_{-n} равен единице, то он не изменяется при округлении, а максимальная погрешность при отбрасывании "хвоста" для положительных и отрицательных чисел составляет $\Delta_{max} = \pm p^{-n}(1 - p^{-t})$. Если в неокругленном результате операции значение разряда a_{-n} есть нуль, то установка его в единицу вносит погрешность, максимальная величина которой может достигать, соответственно, $\Delta_{max} = \pm p^{-n}(1 - p^{-t})$. Таким образом, при равновероятном появлении нуля и единицы в младшем разряде сохраняемой части для знакопеременных чисел мы снова получили симметричное распределение погрешностей с $\bar{\Delta} = 0$. Однако в общем случае для чисел одного знака $\bar{\Delta} \neq 0$. Математическое ожидание будет в этом случае равно нулю только при $t = 1$, т.е. когда длина отсекаемой части составляет один двоичный разряд. Только в этом важном для практики случае округление происходит правильно, без накопления систематических ошибок.

8. *Вероятностное округление.* Для такого округления необходимо иметь датчик случайных величин (0 или 1), единица с выхода которого прибавляется к младшему разряду сохраняемой части числа. Погрешность округления при равновероятном распределении значений отбрасываемой части является случайной величиной с нулевым математическим ожиданием.

Таким образом, самым простым способом округления является усечение, при котором не требуется дополнительных затрат времени и оборудования. Однако на практике важнее всего точность вычислений, кото-

рая определяется величиной $\Delta_{max.}$ Только для трех способов округления по дополнению максимальная ошибка близка к половине единицы младшего разряда машинного числа, т.е. является наименьшей. Наиболее быстродействующим из них является упрощенный способ, а наиболее точным – усовершенствованный. Поэтому предпочтение тому или другому способу округления следует отдавать только после анализа требований, предъявляемых к быстродействию и погрешности вычислений конкретной машины. Мы будем в дальнейшем пользоваться первым способом округления по дополнению.

3.5.2. Особенности округления чисел, заданных инверсными кодами

Так как положительные числа в прямом, обратном и дополнительном кодах представляются одинаково, то и правила округления положительных чисел во всех трех кодах будут одними и теми же. Однако округление отрицательных чисел, записанных в инверсных кодах, имеет ряд особенностей.

1. *Обратный код.* При округлений по дополнению всегда округляется абсолютная величина, вследствие этого из дополнительного разряда (ДР) отрицательной дроби необходимо вычесть единицу, т.е. прибавить к округляемой дроби обратный код $-1_{окр.} = 1{,}11\ldots10$, где цифра "0" записана в ДР. Цепочка циклического переноса должна при этом охватывать и этот ДР.

Пример: Задано: A = –0,101101, n = 4, t = 2, $[A]_o = 1{,}010010$.

Округление абсолютной величины:

$$\begin{array}{r} |A| = 0{,}1011|01 \\ + \quad\quad\quad\quad |1\ \text{окр} \\ \hline 0{,}1011|11 \end{array}$$

$|A|_{окр} = 0{,}1011, \quad [A]_0 = 1{,}0100.$

Непосредственное округление обратного кода дает:

$$\begin{array}{rl} & \text{ДР} \\ A = 1{,}0100 & |10 \\ + \ 1{,}1111 & |0\ \text{окр} \\ \hline 11{,}0011 & |10 \\ + \ \text{└——→} & |1\ \text{ЦП} \\ \hline 1{,}01000 & |00 \end{array} \qquad \text{т.е. } [A_{окр}]_o = 1{,}0100.$$

Добавлять код 1,11...10 и перестраивать цепочку циклического переноса для охвата ДР, т.е. (n+1)-го разряда сумматора весьма неудобно.

Поэтому отрицательные дроби обычно округляются после их перевода Нз обратного в прямой код.

3. *Дополнительный код.* При округлении отрицательной дроби, заданной в дополнительном коде, различают два случая. Если справа от ДР находится хотя бы одна единица, то в ДР прибавляется единица округления, после реализации которой все разряды начиная с дополнительного отбрасываются. Если в ДР находится единица и эта единица является младшей в коде числа, то все разряды, начиная с дополнительного, просто отбрасываются. Прибавлять единицу в ДР в последнем случае нельзя, так как это исказит результат.

Пример: Заданы A = –0,1010001; B = –0,1010100; n = 4, t = 3.

$[A]_д$ = 1,0101111, $[B]_д$ = 1,0101100.

Округление абсолютных величин:

|A| = 0,1010|001
+ _____|1 окр
0,1010|101
$|A|_{окр}$ = 0,1011;
$[A_{окр}]_д$ = 1,0110.

|B| = 0,1010|100
+ _____|1 окр
0,1011|000
$|B|_{окр}$ = 0,1011;
$[B_{окр}]_д$ = 1,0101.

Непосредственное округление дополнительных кодов чисел A и B.

ДР
$[A]_д$ = 1,0101|111
+ _____|1 окр
1,0110|011
$[A_{окр}]_д$ = 1,0110.

ДР
$[B]_д$ = 1,0101|100
+ _____|0 окр
1,0101|100
$[B_{окр}]_д$ = 1,0101

Таким образом, округление отрицательных дробей, заданных в дополнительном коде, производится проще, чем в случае обратного кода.

3.5.3. Погрешности выполнения арифметических операций

Точность вычисления определяется числом цифр результата, заслуживающих доверия. Значащими цифрами целого числа называются все его цифры, кроме нулей, стоящих левее первой отличной от нуля цифры. Нули в конце целого числа – всегда значащие цифры. Цифра a_i приближенного числа [A] называется верной, если выполняется неравенство

$$|A - [A]| \le 0{,}5 \cdot p^{n-i+1}.$$

т.е. если абсолютная величина разности между точным числом A и его приближенным значением не превосходит половины единицы разряда, в котором стоит a_i. Или

$$[\Delta A] = |A - [A]| \le 0{,}5 \cdot p^{n-i+1}.$$

Погрешности выполнения арифметических операций могут быть оценены, если рассматривать их как результат выполнения элементарных операций над операндами A и B, заданными машинными изображениями [A] и [B] с абсолютными погрешностями соответственно [ΔA] и [ΔB].

Результат операции алгебраического сложения этих чисел будет иметь вид:

$$A+B = [A]+[B]+([\Delta A] + [\Delta B]),$$
$$\Delta(A+B) = [\Delta A] + [\Delta B].$$

Отсюда следует, что абсолютная погрешность алгебраической суммы в худшем случае не может быть меньше абсолютной погрешности наименее точного из слагаемых. Поэтому, чтобы не производить лишних вычислений, не следует сохранять лишние разряды и в более точных слагаемых.

При выполнении операций умножения получим:

$$A \times B = [A][B] + [A][\Delta B] + [B][\Delta A] + [\Delta A][\Delta B].$$

Так как произведение [ΔA]×[ΔB].– величина второго порядка малости, то ею можно пренебречь, т.е.

$$A \times B \approx [A] \times [B] + [A][\Delta B] + [B][\Delta A],$$

т.е. абсолютная погрешность произведения

$$\Delta AB = [A][\Delta B]+[B][\Delta A].$$

При выполнении операции деления получим:

$$\frac{A}{B} = \frac{[A] + [\Delta A]}{[B] + [\Delta B]} = \frac{[A] + [\Delta A]}{[B]} \times \frac{1}{1 + [\Delta B] / [B]}.$$

Второй сомножитель в правой части уравнения разложим в ряд. После преобразований получим:

$$\frac{A}{B} = \frac{[A]}{[B]} - \frac{[A][\Delta B]}{[B]^2} + \frac{[A][\Delta B]^2}{[B]^3} + \frac{[\Delta A]}{[B]} - \frac{[A][\Delta B]}{[B]^2} + \dots .$$

Пренебрегая членами второго порядка малости, можно упростить

$$\frac{A}{B} = \frac{[A]}{[B]} + \frac{[\Delta A]}{[B]} - \frac{[A][\Delta B]}{[B]^2}.$$

Отсюда абсолютная погрешность частного

$$\Delta\frac{A}{B} = \frac{[\Delta A]}{[B]} - \frac{[A][\Delta B]}{[B]^2}.$$

Аналогичным образом можно получить выражения для относительных погрешностей при алгебраическом сложении.

$$\delta_{A+B} = \frac{[A]}{[A] + [B]} \times \frac{[\Delta A]}{[A]} + \frac{[B]}{[A] + [B]} \times \frac{[\Delta B]}{[B]},$$

при умножении

$$\delta_{AB} = \frac{[\Delta A]}{[A]} + \frac{[\Delta B]}{[B]} = [\delta_A] + [\delta_B],$$

при делении

$$\delta_{A/B} = \frac{[\Delta A]}{[A]} - \frac{[\Delta B]}{[B]} = [\delta_A] - [\delta_B],$$

Следовательно, при умножении и делении основной вклад в относительную погрешность вносят наименее точные (имеющие наибольшую относительную погрешность) числа. Поэтому при умножении и делении чисел с различной относительной погрешностью не следует сохранять лишние знаки у чисел с меньшей относительной погрешностью.

При возведении числа A в степень имеем

$$B = [A]^m = [A]\times[A]...[A].$$

Тогда

$$[\delta B] = [\delta A] + [\delta A] + ... + [\delta A] = m[\delta A].$$

Таким образом, при возведении в степень m приближенного числа [A] его относительная погрешность увеличивается в m раз.

При вычислении $B = \sqrt[m]{[A]}$ получим

$$[\delta B] = \frac{1}{m}[\delta A].$$

При вычислениях, если не выполняется строгий подсчет погрешностей, рекомендуется пользоваться следующими правилами подсчета числа правильных цифр.

1. При алгебраическом сложении приближенных чисел в результате необходимо сохранять столько разрядов, сколько их в приближенном операнде с наименьшим числом разрядов.

2. При выполнении операций умножения и деления в результате следует сохранять столько значащих цифр, сколько их в приближенном данном с наименьшим числом верных цифр.

3. При возведении приближенного числа в квадрат или куб в результате необходимо сохранить столько значащих цифр, сколько их в основании степени.

4. При извлечении квадратного и кубического корней из приближенного числа в результате следует сохранить столько значащих цифр, сколько их в подкоренном числе.

5. При вычислении промежуточных результатов следует сохранить на один разряд больше, чем рекомендуют правила 1–4. В окончательном результате этот дополнительный разряд отбрасывается.

6. Если некоторые данные имеют больше разрядов (при сложении-вычитании) или больше значащих цифр (при других действиях), чем другие, то их целесообразно предварительно округлить, сохраняя лишь одну "запасную" цифру.

7. Если данные можно брать с произвольной точностью, то для получения результата с n верными цифрами исходные данные следует брать с таким числом цифр, которые согласно предыдущим правилам обеспечивают n+1 цифру в результате.

Необходимо, однако, помнить, что эти правила верны, если операнды содержат только верные цифры и число действий невелико. Следует также помнить, что оценка точности вычислений на машинах зависит не только от состава выполняемых операций, но и от порядка следования Их друг за другом.

3.6 Точность выполнения операций в машине с плавающей запятой

Оценка погрешности при вычислении на машине особенно затруднена при использовании чисел, представленных с плавающей запятой, так как в этом случае вычисления неточны по самой своей природе.

Здесь возможно перемещение ошибки из младших разрядов мантиссы в старшие, если преждевременно округлить младшие разряды числа. Такая ситуация возникает, например, при денормализации результата вправо, если порядки исходных данных отличаются на единицу. При этом, с одной стороны, мантисса может содержать некоторую погрешность (из-за сдвига вправо и округления одной из мантисс), с другой стороны, при нормализации результата может потребоваться большое количество сдвигов влево, в результате которых погрешность мантиссы результата может быть выведена чуть ли не в старшие раз-

ряды. В этом случае полностью исправляют ошибку с помощью цифры младшего разряда, которая была выдвинута в схему округления при выравнивании порядков. Если этого не делать, то очевидно, что наибольшую потерю точности следует ожидать при сложении и вычитании почти равных величин с соответствующими знаками, т.е. при операциях вида A – B и A + (–B).

Возможно также непредвиденное переполнение при округлении результата. Причем такая ситуация может возникнуть как при сложении, так и при умножении двух чисел.

Пример: Заданы $[A]_{пр}= \underset{m_a}{0\,010}\,\underset{a}{0\,1110}$; $[B]_{пр}= \underset{m_b}{0\,100}\,\underset{b}{0\,1001}$.

Вычислить C = A × B.

$m_c = m_a + m_b = 0\,010 + 0\,100 = 0\,110$,

$c = a \times b = 0{,}01111110$.

Мантисса результата денормализована вправо на один разряд. Следовательно, необходима ее нормализация и последующее округление до4-х разрядов.

После нормализации имеем:

$c' = 0{,}1111110$, $m'_c = 0\,101$.

После округления получим:

$$\begin{array}{l} c' = 0{,}1111|110 \\ +\ \underline{\qquad\quad|1\ \text{окр}} \\ \quad 1{,}0000|010 \end{array}$$,т.е. $c'_{окр} = 1{,}0000$.

Для исключения подобных ситуаций необходимо после округления проверять, не нарушена ли нормализация влево и устранять ее известным образом. Тогда в нашем примере получим окончательно

$$[C]_{пр}= \underset{m_b}{0\,110}\,\underset{b}{0\,1000}$$.

Поскольку степень преобразования мантиссы зависит от величины большего числа, то результат сложения нескольких чисел может зависеть от порядка их суммирования, т.е. в общем случае ассоциативный закон при сложении чисел с плавающей запятой не выполняется. Этот закон может не соблюдаться также, когда мантиссу результата принудительно обнуляют при отрицательном переполнении порядков. Строго говоря, приравнение мантиссы нулю при исчезновении порядка допустимо только, когда результат должен складываться со значительно большей величиной. Если исчезновение порядка не отмечается каким-то образом, то могут возникнуть неожиданные ситуации, когда

$(A \times B) \times C = 0$, а $(A \times C) \times B \neq 0$. Эти ситуации обусловлены ограниченностью разрядной сетки мантиссы и округлением результатов.

Пример: Заданы $A = 0{,}1110 \cdot 2^{-3}$; $B = 0{,}1001 \cdot 2^{-5}$; $C = 0{,}1000 \cdot 2^{+4}$.

$|m_{max}| = 7$; $n = 4$.

После умножения $A \times B$ и округления результата получим $A \times B = 0{,}0010 \cdot 2^{-7}$, т.е. мантисса этого результата представляется машинным нулем и последующая операция $(A \times B) \times C$ даст нулевой результат. Вместе с тем, $A \times C = 0{,}1110 \cdot 2^{0}$ и тогда $(A \times C) \times B = D$ после нормализации и округления составит: $D = 0{,}1000 \cdot 2^{-5}$.

Некоторым выходом из создавшегося положения могла бы послужить возможность задания входных данных задачи в ненормализованной форме, которая бы отражала степень принятой точности, и при этом в выходных данных имелась информация о том, какова точность ответа. Некоторые возможности в этом плане дает ненормализованная арифметика. Ее правила состоят в следующем: пусть l_A – количество нулей, стоящих вначале дробной части числа A. Тогда сложение и вычитание выполняются по обычным правилам за тем исключением, что все сдвиги опускаются. Умножение и деление выполняются как обычно за тем исключением, что результат сдвинут вправо или влево, так как дробная часть ответа будет начинаться в точности с $\max(l_A, l_B)$ нулей. В принципе, те же правила используются и для традиционных вычислений вручную. Когда итогом вычислений является нуль, то в качестве результата выдается ненормализованный нуль. Это означает, что в действительности результат может быть и не равен нулю, но мы, просто, не знаем ни одной его значащей цифры.

Следует отметить, что арифметика ненормализованных чисел не может в полной мере служить выходом из создавшегося положения. Имеются примеры, где указываемая такой арифметикой точность больше действительной (например, нахождение k-й степени числа для большого k), и существует еще больше примеров, когда в этой арифметике дается невысокая точность, в то время как в арифметике нормализованных чисел в действительности получались бы гораздо более точные результаты.

Другой подход к проблеме оценки погрешности связан с, так называемой, "интервальной" арифметикой [35,47], в которой вычисления выполняются с учетом верхней и нижней оценок для каждого числа. Так, например, если известно, что $A_0 \le A \le A_1$ и $B_0 \le B \le B_1$ то $A_0 + B_0 \le A + B \le A_1 + B_1$, $A_0 - B_1 \le A - B \le A_1 - B_0$ и при положительных A_0 и B_0 $A_0B_0 \le AB \le A_1B_1$, $A_0/B_1 \le A/B \le A_1/B_0$. Аналогичные правила можно

записать и для других случаев. Вычисления кодов A_0, A_1, B_0, B_1 проводятся с соответствующим округлением (к нулю для нижних границ и от нуля для верхних). Такие вычисления лишь вдвое больше по объему, чем те же вычисления в обычной арифметике и могут оказаться весьма ценными, так как обеспечивают точную оценку ошибки. Однако возникают трудности, когда $B_0 < B_1$ и мы пытаемся делить на B. Кроме того, ввиду зависимости промежуточных результатов друг от друга, окончательные оценки часто оказываются не слишком удовлетворительными.

Необходимо помнить, что любой метод анализа ошибок не надежен, так как операнды обычно не являются независимыми. Это означает, что ошибки имеют тенденцию компенсировать или усиливать друг друга непредсказуемым образом и поэтому невозможно гарантировать достаточную точность в промежуточных вычислениях.

Таким образом, ошибки при выполнении операций арифметики плавающей запятой обусловлены, в основном, округлением мантисс одного из операндов и результата операции. При этом, если мантисса представлена в прямом или дополнительном коде, то в этом коде она и округляется. Если она представлена в обратном коде, то перед округлением переводится в прямой код.

Правила округления мантисс достаточно просты. Округление производится добавлением единицы с весом $2^{-(n+1)}$ в ДР результата или установкой в единицу n-го его разряда, если значение ДР есть 1. Округление не производится, когда результат отрицателен и младшая значащая "1" находится в ДР.

Следует отметить, что для прямого кода соблюдается большее число стандартных математических законов, чем для дополнительного. Поэтому с этой точки зрения прямой код для представления чисел с плавающей запятой имеет некоторое теоретическое преимущество [35].

При выполнении операции сложения чисел с плавающей запятой естественно стремиться к такой ее реализации, которая могла бы быть осуществлена как можно быстрее и как можно точнее. Наибольшую точность при этом можно достичь, если складывать (2n+1) разрядные операнды, т.е. учитывать точное значение сдвигаемой мантиссы, и округлять результат до n-разрядного. Однако длина слагаемых и длительная нормализация влево (до 2n тактов) существенно увеличивают время выполнения операции и усложняют схему сумматора. Наименьшее время выполнения операции сложения будет достигнуто при совмещении округления сдвинутой мантиссы со сложением n-разрядных мантисс и выполнении последующего округления мантиссы результата упрощенным способом по дополнению.

3.7. Вычисления с двойной точностью

Если арифметика однократной точности, с которой шла речь до сих пор, дает слишком большую погрешность, то точность вычислений можно увеличить при помощи средств программистского характера, используя для представления каждого числа два или более слов памяти.

Для выполнения арифметических действий над числами с плавающей запятой двойная точность необходима почти всегда в отличие от случая фиксированной запятой. Вместе с тем, ввиду того что вычисления с двойной точностью проще для чисел с фиксированной запятой, мы будем рассматривать только операции над мантиссами.

Двойная точность часто желательна не только для снижения погрешности задания мантисс, но также для увеличения диапазона изменения порядка. Тогда двухсловное представление чисел с плавающей запятой будет иметь вид

либо

В последнем случае используются два байта для порядка и шесть байтов для мантиссы. Можно использовать также три машинных слова: одно для представления порядка и два для мантиссы. В этом случае говорят о тройной точности, хотя мантисса представлена только с двойной точностью.

Вычислениям с двойной точностью характерен ряд особенностей по сравнению с вычислениями однократной точности. Главная из них состоит в том, что округление не выполняется, если это оказывается неудобным, т.е. в этом случае допускается большая погрешность в младшем разряде мантиссы, имеющем существенно меньший вес. Главной заботой при этом является повышение скорости вычислений.

Сложение с двойной точностью отличается от случая однократной точности тем, что для выполнения операции используется раздельное сложение менее значимых и более значимых половин мантисс с учетом возможного переноса из менее значимой части числа.

Произведение $A = A_1 + A_2$ на $B = B_1 + B_2$ имеет четыре компоненты: C_1; C_2; C_3; C_4:

$$
\begin{array}{r}
B_2A_2 \\
B_2A_1 \\
A_2B_1 \\
+\ A_1B_1 \\
\hline
C_1C_2C_3C_4
\end{array}
$$

Так как нужна только старшая половина мантиссы результата, то можно даже не вычислять произведение младших половин A_2 и B_2.

Операция деления двойной точности точности чисел с плавающей запятой несколько сложнее. Запишем пару чисел, участвующих в делении, в виде

$$\frac{A_1 + \varepsilon A_2}{B_1 + \varepsilon B_2},$$

где ε – величина, обратная размеру машинного слова, а B_1 – нормализовано. Эту дробь можно разложить в ряд

$$\frac{A_1 + \varepsilon A_2}{B_1 + \varepsilon B_2} = \frac{A_1 + \varepsilon A_2}{B_1} \times \frac{1}{1 + \varepsilon(B_2/B_1)} = \frac{A_1 + \varepsilon A_2}{B_1} \times \left(1 - \varepsilon\left(\frac{B_2}{B_1}\right) + \varepsilon^2\left(\frac{B_2}{B_1}\right)^2 - \ldots\right).$$

Так как $0 \le |B_2| < 1$ и $1/p \le |B_1| < 1$, то можно пренебречь ошибкой, связанной с отбрасывание членов порядка ε^2. Таким образом, необходимо вначале вычислить $C = C_1 + \varepsilon C_2 = (A_1 + \varepsilon A_2)/B_1$, а затем вычесть из результата $\varepsilon \cdot C \cdot B_2/B_1$.

Среднее время выполнения операций с удвоенной точностью увеличивается немногим более в два раза.

Контрольные вопросы к главе 3

1. Что такое прямой код отрицательного числа?
2. Как складываются отрицательные числа в прямом коде?
3. Сформулируйте правило преобразования отрицательных чисел в дополнительный код и объясните, как оно получено.
4. Почему получается правильный знак результата при сложении чисел в дополнительном коде?
5. Перечислите способы обнаружения переполнений разрядной сетки.
6. Чем отличается обратный код отрицательного числа от его дополнительного кода?
7. Назовите особенности сложения чисел в обратном коде.
8. Как выполняется сдвиг инверсного кода числа влево и вправо? Какие при этом вводятся поправки и почему?
9. Перечислите этапы и особенности сложения чисел в машине с плавающей запятой.

9. Назовите способы округления чисел в ЭВМ.
10. Чем отличаются между собой способы округления по дополнению?
12. В чем состоят особенности округления инверсных кодов чисел?
13. Оцените погрешности выполнения арифметических операций?
14. Назовите источники погрешностей вычислений в машине с плавающей запятой.
15. Какие способы повышения точности вычислений в машине с плавающей запятой Вы знаете?
16. Перечислите особенности вычислений с удвоенной точностью.

Глава 4 ВЫПОЛНЕНИЕ ОПЕРАЦИЙ УМНОЖЕНИЯ И ДЕЛЕНИЯ В ЭВМ

4.1. Общие сведения об операции умножения

Операция умножения является наиболее частой после сложения. Одним из древних считается староегипетский способ умножения, основанный на использовании операции удвоения. По этому способу вычисляют все возможные значения 2^i (i = 0,1,2,... к) и $B \times 2^i$ до тех пор, пока не будет выполнено условие $2^{k+1} > A$. Значение А определяют в виде суммы соответствующих степеней двойки (представив его в виде полинома по основанию 2).

Произведение получают суммированием значений $B \times 2^i$ для тех i, которые входят в определение числа А. Хотя в этом способе используются элементы двоичного умножения, числа А и В представляются в системе счисления с произвольным основанием p > 2. Для p = 2 этот способ ничем не отличается от обычного двоичного умножения методом накопления сумм частных произведений.

Пример: Заданы B = 154, A = 23, $23_{10} = 10111_2$,

$2^0 = 1 \rightarrow 154$, $2^1 = 2 \rightarrow 308$, $2^2 = 4 \rightarrow 616$, $2^3 = 8 \rightarrow 1232$,

$2^4 = 16 \rightarrow 2464 \rightarrow$ Конец, т.к. следующая степень двойки больше 23.

Так как число 23 образуется из чисел 1, 2, 4 и 16, то произведение должно быть составлено из чисел 154, 308, 616 и 2464. Их сумма, равная 3542, является произведением 154 на 23.

Простаферетический (от греческого "простезис" – прибавление и "афай-резис" – отнятие) способ умножения основан на тождестве

$$AB = (A+B)^2/4 - (A-B)^2/4. \quad (4.1)$$

По значениям суммы и разности сомножителей из таблицы вида X^2 получаются величины $(A+B)^2/4 - (A-B)^2/4 = D$, а затем окончательное произведение формируется после выполнения операции вычисления (C – D). Модификацией умножения по этому способу является умножение, определяемое тождеством: $AB = 1/2[(A+B)^2 - A^2 - B^2]$.

В основу древнеиндийского способа умножения "крест на крест" положено то обстоятельство, что при умножении двух чисел А и В, представленных в позиционной однородной системе счисления по основанию p, соответственно правилам умножения полиномов цифра разряда (с весом) p^1 может быть получена как сумма по mod p всех одноразрядных произведений $(a_i b_0 + a_{i-1} b_1 + ... + a_1 b_{i-1} + a_0 b_i)$, где a_i и b_i –

цифры i-х разрядов, соответственно, чисел A и B, и переноса $П_{i-1}$ из разряда p^{i-1} произведения. Действительно, если представить числа A и B в виде полиномов по основанию p, то получим

$$A = a_n p^n + a_{n-1} p^{n-1} + ... + a_2 p^2 + a_1 p^1 + a_0 p^0,$$
$$B = b_k p^k + b_{k-1} p^{k-1} + ... + b_2 p^2 + b_1 p^1 + b_0 p^0, \qquad (4.2)$$

Тогда $$C = A \times B = (a_n p^n + a_{n-1} p^{n-1} + ... + a_1 p^1 + a_0 p^0) \times$$
$$\times (b_k p^k + b_{k-1} p^{k-1} + ... + b_1 p^1 + b_0 p^0).$$

Результат C также является полиномом по основанию p, поэтому он может быть представлен как:

$$C = c_m p^m + c_{m-1} p^{m-1} + ... + c_2 p^2 + c_1 p^1 + c_0 p^0, \qquad (4.3)$$

где m = n + k.

Тогда:

$$C_0 = a_0 b_0;$$
$$C_1 = a_1 b_0 + b_1 a_0 + П_0, \qquad (4.4)$$
$$C_2 = a_2 b_0 + a_1 b_1 + a_0 b_2 + П_1,$$
$$C_3 = a_3 b_0 + a_2 b_1 + a_1 b_2 + a_0 b_3 + П_2 \text{ и т.д.}$$

Например: A = 132; B = 23, т.е.

$A = 1 \cdot 10^2 + 3 \cdot 10^1 + 2 \cdot 10^0$;

$B = 2 \cdot 10^1 + 3 \cdot 10^0$;

$C = A \times B = 6 \cdot 10^0 + (9 \cdot 10^1 + 4 \cdot 10^1) + (3 \cdot 10^2 + 6 \cdot 10^1 \cdot 10^1 + 2 \cdot 10^3) =$
$= 3 \cdot 10^3 + 0 \cdot 10^2 + 3 \cdot 10^1 + 6 \cdot 10^0.$

Таким образом, для последовательного определения разрядов результата умножения двух чисел по способу "крест на крест" требуется только одноразрядный сумматор по основанию p. Параллельное формирование разрядов результата потребует применения значительно большего объема оборудования.

Эти способы умножения при p > 2 используются редко. Например, древнеегипетский способ применяется в некоторых ЭВМ ранних поколений, простаферетический способ используется в ЭВМ со сравнительно небольшой длиной слова, позволяющей применить постоянное ЗУ для хранения квадратов чисел. Способ "крест на крест" используется в некоторых специализированных ЭВМ, выполняющих большое количество умножений по основанию p > 2. Его модификация используется в двоичных ЭВМ при аппаратной реализации операции умножения.

4.2. Умножение в ЭВМ, выполняемое методом накопления частных произведений

Операция умножения в современных ЭВМ чаще всего выполняется суммированием сдвинутых на один или несколько разрядов частных произведений, каждое из которых является результатом умножения множимого на соответствующий разряд или разряды множителя.

При точном умножении двух чисел количество значащих цифр произведения может в пределе достигать двойного количества значащих цифр сомножителей. Еще сложнее возникает ситуация при умножении нескольких чисел. Поэтому в произведении только в отдельных случаях используют двойное количество разрядов, обычно же ограничиваются количеством разрядов, которое имели сомножители. Здесь учитывается то обстоятельство, что правила приближенных вычислений рекомендуют оставлять в произведении столько же значащих цифр, сколько их содержится в наименее точном из сомножителей. Младшие разряды результата при этом отбрасываются, а старшие, обычно округляются по известным правилам с тем, чтобы ошибка произведения стала знакопеременной и ее математическое ожидание было равно 0 с учетом равновероятности любых значений отброшенных младших разрядов.

Наиболее просто операция умножения в ЭВМ выполняется в прямом коде. При этом на первом этапе определяется знак произведения путем сложения знаковых цифр сомножителей по модулю 2.

Затем производится перемножение модулей сомножителей по правилам арифметики согласно двоичной таблицы умножения. Результату присваивается полученный знак.

Так как умножение производится в двоичной системе счисления, частные произведения либо равны 0 (при умножении на 0), либо самому сомножителю (при умножении на 1), сдвинутому на соответствующее количество разрядов.

Пример: Заданы A = 0,1101; B = 0,1011. Получить произведение A×B путем умножения с младших разрядов множителя B.

A = 0,1101
B = 0.1011
1101
1101
0000
+ 1101
0,10001111

Пример: Заданы A = 0,1101; B = 0,1011. Получить произведение A×B путем умножения, начиная со старших разрядов множителя.

```
A = 0,1101
B = 0,1011
      1101
       0000
        1101
  +      1101
  0,10001111
```

Процессом накопления суммы частных произведений можно управлять с помощью цифр множителя в соответствии с выражением:

$$C = A \times B = A \times \sum_{i=-n}^{-1} b_{-i} 2^{i} = A \cdot b_1 2^{-1} + A \cdot b_2 2^{-2} + \ldots + A \cdot b_{n-1} 2^{-(n-1)} + A \cdot b_n 2^{-n}, \quad (4.5)$$

где C – искомое произведение; A – множимое; B – множитель.

Управление процессом умножения может начинаться как с младших разрядов множителя, так и со старших. При этом полную сумму (произведение) можно получить двумя путями:

1) сдвигом множимого на требуемое количество разрядов и добавлением полученного очередного частного произведения к ранее накопленной сумме,

2) сдвигом суммы ранее подученных частных произведений на каждом шаге на один разряд и последующим добавлением к сдвинутой сумме неподвижного множимого либо 0.

Основываясь на вышеизложенном, можно создать четыре варианта схем машинного умножения:

1) умножение младшими разрядами множителя со сдвигом накапливаемой суммы вправо,

2) умножение младшими разрядами множителя со сдвигом множимого влево,

3) умножение старшими разрядами множителя со сдвигом суммы частных произведений влево,

4) умножение старшими разрядами множителя со сдвигом множимого вправо.

Рассмотрим более детально каждую из четырех схем умножения.

1) *Умножение младшими разрядами множителя со сдвигом суммы частных произведений вправо.*

Выражение (4.5) можно вычислять по схеме Горнера

$$C = (...(...(0 + A\cdot b_n)2^{-1} + A\cdot b_{n-1})2^{-1} + ... + A\cdot b_{n-i})2^{-1} + ... + A\cdot b_2)2^{-1} + A\cdot b_1)2^{-1}, \quad (4.6)$$

которая может быть сведена к n-кратному выполнению цикла вида:

$$C_{i+1} = (C_i + A\cdot b_{n-i})2^{-1}$$

при начальных значениях i = 0; $C_0 = 0$.

В каждом цикле множимое либо добавляется к сумме частных произведений (если $b_{n-i} = 1$), либо нет (если $b_{n-i} = 0$), после чего сумма частных произведений умножается на 2^{-1}, т.е. сдвигается на один разряд вправо. После окончания n-го цикла образуется искомое произведение, т.е.

$$C_n = C = A\times B.$$

Очередную цифру множителя, управляющую суммированием частных произведений, удобнее всего снимать с младшего разряда регистра множителя, в котором в каждом цикле производится сдвиг содержимого на один разряд вправо.

Тогда реализация данного способа потребует n-разрядного сдвигающего регистра множителя, n схем "И", пропускающих множимое на вход сумматора при $b_{n-i} = 1$ и запрещающих его передачу при $b_{n-i} = 0$, n-разрядного регистра множимого, n+1-разрядного сумматора и 2n+1-разрядного сдвигового регистра частных произведений, в котором накапливается сумма и имеются цепи для ее сдвига вправо на один разряд. Структура умножителя имеет вид (рис. 4.1).

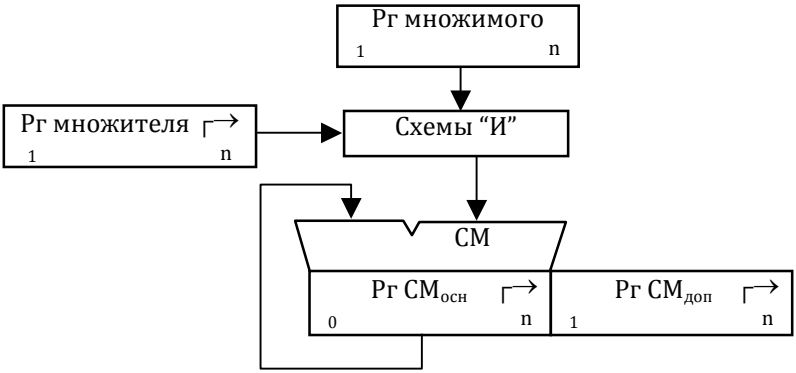


Рис.4.1. Первая схема умножения

Если дополнительные разряды регистра СМ исключить, то произведение будет вычисляться с тем же количеством разрядов, что и сомножители. Однако для округления результата желательно удлинить СМ на один дополнительный разряд с тем, чтобы после вычисления произведения прибавить к дополнительному разряду единицу округления и вывести результат без дополнительного разряда.

Пример: Заданы A = 0,101011; B = 0,10011. Получить C = A×B.

В примерах выполняется умножение только модулей чисел. Для наглядности разряд регистра множителя, управляющий передачей множимого в сумматор, отделен сплошной вертикальной линией. Исходное состояние сумматора - нуль.

Рг. множителя			1 0 1 0 1 1 Рг. Множимого
			0 0 0 0 0 0 0 0 0 0 0 0 0 Рг. СМ
	1 0 0 1 1 1	+ A	1 0 1 0 1 1
			0 1 0 1 0 1 1 0 0 0 0 0 0 Рг. СМ
		Сдвиг	0 0 1 0 1 0 1 1 0 0 0 0 0 →
Сдвиг Рг →	0 1 0 0 1 1	+ A	1 0 1 0 1 1
			1 0 0 0 0 0 0 1 0 0 0 0 0 Рг. СМ
		Сдвиг	0 1 0 0 0 0 0 0 1 0 0 0 0 →
Сдвиг Рг →	0 0 1 0 0 1	+ A	1 0 1 0 1 1
			1 0 0 1 0 1 1 0 1 0 0 0 0 Рг. СМ
		Сдвиг	0 1 0 0 1 0 1 1 0 1 0 0 0 →
Сдвиг Рг →	0 0 0 1 0 0	+ 0	0 0 0 0 0 0
		Сдвиг	0 0 1 0 0 1 0 1 1 0 1 0 0 →
Сдвиг Рг →	0 0 0 0 1 0	+ 0	0 0 0 0 0 0
		Сдвиг	0 0 0 1 0 0 1 0 1 1 0 1 0 →
Сдвиг Рг →	0 0 0 0 0 1	+ A	1 0 1 0 1 1
			0 1 1 0 1 0 0 0 1 1 0 1 0 Рг. СМ
		Сдвиг	0 0 1 1 0 1 0 0 0 1 1 0 1 →

A×B = 0,101011 × 0,100111 = 0,011010001101.

Здесь сумматор содержит вспомогательный разряд переполнений, отделенный от остальных разрядов сплошной линией, в качестве которого можно использовать знаковый разряд сумматора.

2) *Умножение младшими разрядами множителя со сдвигом множимого влево.*

Представим выражение (4.5) в виде:

$$C = 2^{n}[A\cdot b_n + 2A\cdot b_{n-1} + \ldots + 2^{n-2}A\cdot b_2 + 2^{n-1}A\cdot b_1], \qquad (4.7)$$

Вычисление выражения (4.7) сводится к n-кратному выполнению цикла:

$$C_{i+1} = C_i + A_i \times b_{n-i},$$

где $A_i = 2A_{i-1}$, при начальных значениях i = 0; $C_0 = 0$; $A_0 = A$.

В каждом цикле умножения множимое сдвигается на один разряд влево и либо передается в СМ (при $b_{n-i} = 1$), либо нет (при $b_{n-i} = 0$).

Для реализации данного способа умножения требуется n-разрядный сдвигающий регистр множителя, 2n-разрядный сдвигающий регистр множимого и 2n-разрядный сумматор (рис. 4.2).

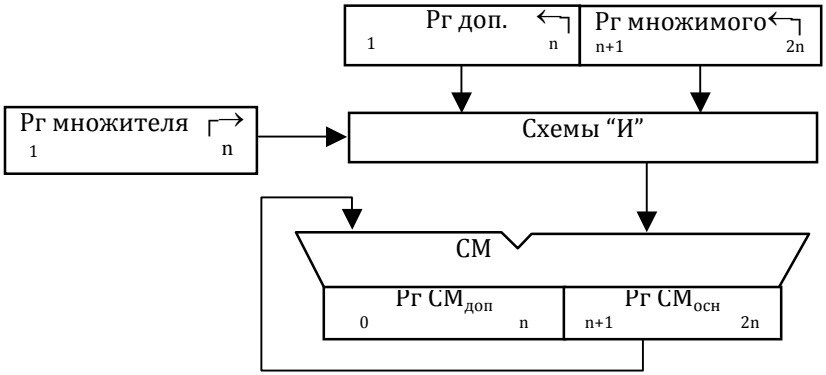


Рис. 4.2. Вторая схема умножения

Пример: A = 0.101011; B = 0.100111

Рг. мн-ля B	1 0 0 1 1 1	Рг. Множимого A	1 0 1 0 1 1	Рг. A
			0 0 0 0 0 0 0 0 0 0 0 0	Рг. СМ
	1 0 0 1 1 1	+ A	0 0 0 0 0 0 1 0 1 0 1 1	Рг. СМ
		Сдвиг A	1 0 1 0 1 1 0	Рг. A
Сдвиг Рг B →	0 1 0 0 1 1	+ A	0 0 0 0 1 0 0 0 0 0 0 1	Рг. СМ
		Сдвиг A	1 0 1 0 1 1 0 0	Рг. A
Сдвиг Рг B →	0 0 1 0 0 1	+ A	0 0 0 1 0 0 1 0 1 1 0 1	Рг. СМ
		Сдвиг A	1 0 1 0 1 1 0 0 0	Рг. A
Сдвиг Рг B →	0 0 0 1 0 0	+ 0	0 0 0 1 0 0 1 0 1 1 0 1	Рг. СМ
		Сдвиг A	1 0 1 0 1 1 0 0 0 0	Рг. A
Сдвиг Рг B →	0 0 0 0 1 0	+ 0	0 0 0 1 0 0 1 0 1 1 0 1	Рг. СМ
		Сдвиг A	1 0 1 0 1 1 0 0 0 0 0	Рг. A
Сдвиг Рг B →	0 0 0 0 0 1	+ A	0 1 1 0 1 0 0 0 1 1 0 1	Рг. СМ

C = A×B = 0,011010001101.

Знаковый разряд отделен сплошной вертикальной чертой. Для большей наглядности старшие разряды Рг.А, значение которых "0", опущены.

3) *Умножение старшими разрядами множителя со сдвигом суммы частных произведений влево.*

Если преобразовать выражение (4.5) к виду

$$C = 2^{-(n+1)}(\ldots(\ldots(0 + A\cdot b_1)2 + A\cdot b_2)2 + \ldots + A\cdot b_{n-i})2^{-1} + \ldots + A\cdot b_{n-1})2 + A\cdot b_n)2, \quad (4.8)$$

то умножение двух чисел А и В сводится к n-кратному повторению цикла

$$C_{i+1} = (C_i + A_i \times b_{i+1})\cdot 2,$$

с начальными значениями i = 0; $C_0 = 0$.

Тогда управление умножением будет производиться цифрами множителя, начиная со старших разрядов. Сумма частных произведений в каждом цикле будет сдвигаться на один разряд влево.

Таким образом, для реализации данной схемы необходимы: сдвигающий влево регистр множителя с n разрядами, n-разрядный регистр множимого и 2n-разрядный накапливающий сумматор со сдвигом влево (рис. 4.3).

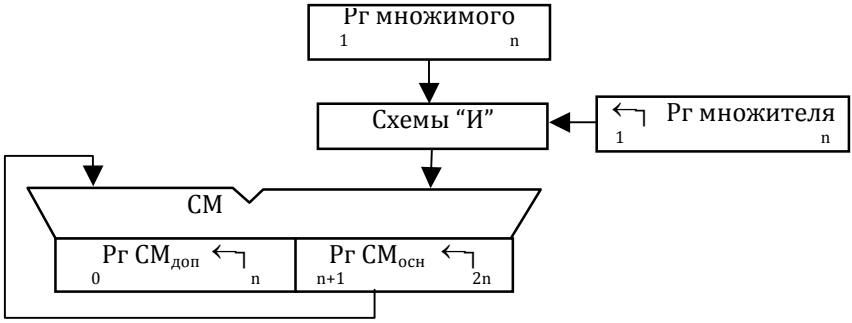


Рис. 4.3.Третья схема умножения

Старшие разряды накапливающего сумматора могут быть упрощены за счет того, что они суммируют сумму частных произведений из Рг $СМ_{доп}$ с переносом от сложения кода из Рг $СМ_{осн}$ с множимым.

Пример: Заданы А = 0.101011; В = 0.100111.

Выполнить умножение старшими разрядами множителя со сдвигом суммы частных произведений влево.

Рг. множит. B 1 0 0 1 1 1		Рг. множимого A 1 0 1 0 1 1	Рг. A
		0 0 0 0 0 0 0 0 0 0 0 0	Рг. СМ
	1 0 0 1 1 1	+ A	0 0 0 0 0 0 1 0 1 0 1 1	Рг. СМ
		Сдвиг СМ	0 0 0 0 0 1 0 1 0 1 1 0	←
Сдвиг Рг B ←	0 0 1 1 1 0	+ 0	0 0 0 0 0 1 0 1 0 1 1 1	Рг. СМ
		Сдвиг СМ	0 0 0 0 1 0 1 0 1 1 1 0 0	←
Сдвиг Рг B ←	0 1 1 1 0 0	+ 0	0 0 0 0 1 0 1 0 1 1 0 0	Рг. СМ
		Сдвиг СМ	0 0 0 1 0 1 0 1 1 0 0 0	←
Сдвиг Рг B ←	1 1 1 0 0 0	+ A	1 0 1 0 1 1	Рг. A
			0 0 0 1 1 0 0 0 0 0 1 1	Рг. СМ
		Сдвиг СМ	0 0 1 1 0 0 0 0 0 1 1 0	←
Сдвиг Рг B ←	1 1 0 0 0 0	+ A	1 0 1 0 1 1	Рг. A
			0 0 1 1 0 0 1 1 0 0 0 1	Рг. СМ
		Сдвиг СМ	0 1 1 0 0 1 1 0 0 0 1 0	←
Сдвиг Рг B ←	1 0 0 0 0 0	+ A	1 0 1 0 1 1	Рг. A
			0 1 1 0 1 0 0 0 1 1 0 1	Рг. СМ

C = A×B = 0,011010001101.

4) *Умножение старшими разрядами множителя со сдвигом множимого вправо.*

Если представить выражение (4.5) в виде:

$$C = A\cdot 2^{-1}\cdot b_1 + A\cdot 2^{-2}\cdot b_2 + \ldots + A\cdot 2^{-(n-1)}\cdot b_{n-1} + A\cdot 2^{-n}\cdot b_n, \qquad (4.9)$$

то вычисление произведения может быть сведено к n-кратному выполнению цикла:

$$A_{i+1} = A_i,$$
$$C_{i+1} = C_i + A_{i+1} \times b_{i+1},$$

при начальных условиях i = 0; $A_0 = A$, $C_0 = 0$.

Таким образом, в каждом цикле множимое сдвигается на один разряд вправо и в зависимости от значения управляющего (старшего) разряда множителя либо передается в накапливающий сумматор, либо нет.

С учетом этого для реализации данной схемы умножения необходим n-разрядный регистр множителя со сдвигом влево, 2n-разрядный регистр множимого со сдвигом вправо и 2n-разрядный сумматор (рис. 4.4).

Если потребовать, чтобы ошибки произведения не превышали единицы младшего разряда, то можно значительно сократить количество дополнительных разрядов регистра множимого и сумматора.

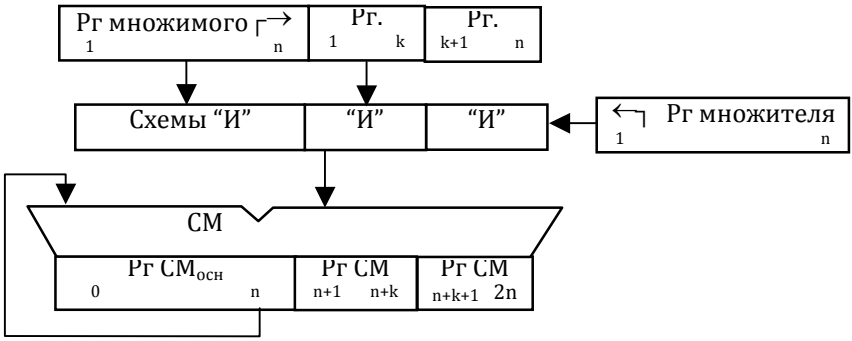


Рис. 4.4. Четвертая схема умножения

При k-дополнительных разрядах погрешность произведения обусловлена двумя причинами:

1) в каждом цикле передачи множимого в СМ после k-го будут пропадать его младшие разряды. Тогда при условии, что все (n–k) разрядов множимого равны 1, максимальная погрешность произведения составит:

$$\Delta_{1max} = -2^{-(n+k+1)} - (2^{-(n+k+1)} + 2^{-(n+k+2)}) - (2^{-(n+k+1)} + 2^{-(n+k+2)} + 2^{-(n+k+3)}) - \ldots$$
$$- 2^{-(n+k)}[(1 - 2^{-1}) + (1 - 2^{-2}) + \ldots + (1 - 2^{-n+k})] =$$
$$= -2^{-(n+k)}[(n - k - (1 - 2^{-(n-k)})] \approx -2^{-(n+k)}(n - k - 1). \quad (4.10)$$

2) После окончания умножения отбрасываются все k младших разрядов Рг.СМ. Если все они равны 1, погрешность от этого составит:

$$\Delta_{2max} = 2^{-n}(1 - 2^{k}) < 2^{-n}. \quad (4.11)$$

Если потребовать чтобы Δ_{2max} было $< 2^{-n}$, то из неравенства:

$$(n - k - 1)\cdot 2^{-(n+k)} < 2^{-n}$$

можно определить количество дополнительных разрядов k:

$$k > \log_2(n - k - 1). \quad (4.12)$$

Обе составляющие ошибки имеют отрицательный знак. Округление первой составляющей ошибки можно выполнять, если всякий раз прибавлять 1 к (n–k–1)-му разряду РгСМ, когда она выходит за пределы разрядной сетки регистра множимого при сдвиге его содержимого вправо. Округление второй составляющей можно выполнить путем добавления 1 к (n+1)-му разряду РгСМ после окончания умножения.

Пример: Заданы A = 0,101011; B = 0,100111.

Рг. множит. B		Рг.A		Рг.$A_{доп}$		
		РгСМ$_{осн}$	0 0 0 0 0 0	РгСМ$_{доп}$	0 0 0	СМ
		→ Рг.A	0 1 0 1 0 1		1 0 0	
	1 0 0 1 1 1	+ A	0 1 0 1 0 1		1 0 0	СМ
		→ Рг.A	0 0 1 0 1 0		1 1 0	
Сдвиг Рг B ← 0	0 1 1 1 0	+ 0	0 1 0 1 0 1		1 0 0	СМ
		→ Рг.A	0 0 0 1 0 1		0 1 1	
Сдвиг Рг B ← 0	1 1 1 0 0	+ 0	0 1 0 1 0 1		1 0 0	СМ
		→ Рг.A	0 0 0 0 1 0		1 0 1	
Сдвиг Рг B ← 1	1 1 0 0 0	+ A	0 1 1 0 0 0		0 0 1	СМ
		→ Рг.A	0 0 0 0 0 1		0 1 0	
Сдвиг Рг B ← 1	1 0 0 0 0	+ A	0 1 1 0 0 1		0 1 1	СМ
		→ Рг.A	0 0 0 0 0 0		1 0 1	
Сдвиг Рг B ← 1	0 0 0 0 0	+ A	0 1 1 0 1 0		0 0 0	СМ
		Округление			+ 1	
			0 1 1 0 1 0		1 0 0	СМ

C = A×B = 0,011010.

4.3. Сравнение схем умножения методом накоплений

Анализ приведенных схем умножения показывает, что:

1) длительность процесса умножения по любой схеме составляет n циклов: $T_y = n \cdot \tau_ц$. Однако длительность циклов в разных схемах неодинакова. Так, во втором и четвертом случаях $\tau_ц = \tau_S$ (учитывая, что $\tau_S > \tau_{сдв}$), и эти схемы позволяют ускорить процесс выполнения операции умножения за счет совмещения операции сложения частных произведений и сдвигов множимого.

2) Первая схема имеет меньшее количество оборудования, а третья схема – несколько больше. Однако с учетом того, что обычно требуется n-разрядный результат, схема 4 может быть значительно упрощена.

3) Схемы 3 и 4 легче приспособить для выполнения операции деления.

4) Схемы 2 и 3 позволяют не устанавливать РгСМ в "0" перед выполнением операции, что удобно для исключения промежуточных пересылок в случае, когда произведение A×B нужно прибавить к числу, ранее вычисленному на сумматоре или наоборот.

На основании вышеизложенного можно заключить, что наиболее удобными для применения в ЭВМ являются 1 и 4 схемы умножения, что подтверждается практикой разработки вычислительных устройств.

4.4. Методы ускорения операции умножения

Любое ускорение операции умножения, даже ценой усложнения арифметических и логических схем, позволяет существенно повысить производительность ЭВМ, т.к. примерно 70% своего времени они тратят на выполнение этой операции.

Известны способы ускорения умножения, направленные на сокращение общего количества и времени выполнения операций сложения, необходимых для образования произведения. Эти способы делятся на логические и аппаратные.

Под аппаратными понимают такие способы, которые требуют для своей реализации введения дополнительного оборудования в основные арифметические цепи, благодаря чему достигается совмещение во времени отдельных составных частей процесса умножения. Они различаются как способы 1-го и 2-го порядка. Для реализации способов первого порядка необходимо количество оборудования, пропорциональное числу разрядов машинного слова n. Для реализации способов второго порядка требуется объем оборудования, пропорциональный n^2.

Под логическими понимают такие способы ускорения, при реализации которых сохраняется основная структура арифметических цепей умножителя, а ускорение достигается только за счет усложнения схемы управления.

4.4.1. Пропуск тактов суммирования

Простейшим логическим способом является пропуск тактов суммирования в тех случаях, когда очередная цифра множителя равна 0. Если обозначить через g_1 количество единиц в некотором множителе, а через g_0 – количество нулей, то для четырех способов умножения методом накопления будут справедливы следующие соотношения:

$$T_{уск}(1, 3) = g_1(\tau_S + \tau_{сдв}) + g_0 \cdot \tau_{сдв};$$
$$T_{уск}(2, 4) = g_1\tau_S + g_0 \cdot \tau_{сдв}.$$

В среднем, количество сложений при этом сокращается вдвое.

4.4.2. Сложение и вычитание множимого

Можно сократить и среднее, и максимальное количество суммирований при использовании как прямых, так и инверсных передач множимого в сумматор. Здесь учитывается то обстоятельство, что время умножения значительно сокращается, если при наличии в разрядах множителя нескольких нулей подряд производить его сдвиг сразу на несколько разрядов. Для этого видоизменяют код множителя с целью

представления его с меньшим количеством разрядов, содержащих единицу. Например, последовательность единиц в множителе 011...110 можно преобразовать в группу цифр $100...0\bar{1}0$, т.е. перейти к системе с цифрами 1, 0, $\bar{1}$.

Таким образом, в основе способа лежит представление числа как совокупности следующих последовательностей: нулей, единиц, нулей с изолированными единицами, единиц с изолированными нулями. При этом два или более соседних нулей или соседних единиц рассматриваются как последовательность. Например, если умножение начинается с младших разрядов и множитель содержит последовательность единиц, то производится вычитание множимого с соответствующим (младшим) весом, а затем сдвиг через все эти единицы.

Сдвиг через последовательность единиц прекращается на первом нуле. Если сразу за этим нулем расположена единица, то множимое вычитается и выполняется сдвиг через последовательность единиц. Если за этим нулем непосредственно следует второй нуль, то множимое прибавляется, а затем выполняется сдвиг через последовательность нулей, который прекращается на первой единице. Если за этой единицей следует ноль, то множимое прибавляется и производится сдвиг через последовательность нулей. Если за этой единицей непосредственно следует вторая единица, то производится вычитание множимого с соответствующим весом данного разряда, а затем выполняется сдвиг через последовательность единиц. Если в старшем разряде множителя стоит 1, входящая в последовательность единиц, то сдвиг необходимо продолжать до первого нуля после старшего разряда множителя.

Следует отметить, что при умножении со старших разрядов принимаются несколько другие правила определения "оптимального" множителя. И в том, и в другом случае, в среднем, на каждую операцию сложения выполняется сдвиг на 2,9 разряда, если схема рассчитана на сдвиг не более, чем на 6 разрядов одновременно.

В пределе, среднее число сложений-вычитаний, приходящееся на один разряд множителя, равно ⅓. Это наилучший результат, которого можно достичь при использовании логических методов.

Таким образом, переход от одной разновидности двоичной системы счисления к другой при преобразовании множителя дает выигрыш во времени выполнения операции в целом. При этом возникают определенной длины последовательности 0 или 1, что в конечном счете приводит к необходимости одновременного анализа нескольких разрядов множителя и сдвига на произвольное число разрядов.

4.4.3. Одновременное умножение на два разряда

Количество циклов, необходимых для реализации в ЭВМ операции умножения, можно сократить, если в каждом цикле анализировать не один, а два или более разрядов множителя, выполняя после анализа одну передачу множимого в сумматор и сдвиг множителя на два или более разрядов. Количество типов передач в РгСМ, соответствующих различным комбинациям нулей и единиц в анализируемых разрядах, должно быть при этом увеличено. В этом случае необходимо максимально использовать те передачи, которые уже имеются в арифметическом устройстве, в частности, передачу с преобразованием в инверсный код.

Для ускоренного умножения множитель разбивается на группы по два разряда и преобразуется таким образом, чтобы каждая группа содержала не более одной единицы, понимая под последней 1 или $\bar{1}$. Тогда для младшей пары разрядов при умножении с младших разрядов возможны следующие комбинации единиц и нулей в разрядах: 00, 01, 10 и 11.

Для первой комбинации прибавляется нуль, для второй – суммирование множимого, для третьей – суммирование сдвинутого на 1 разряд влево множимого, т.е. умноженного на два, а для четвертой – выполняется одно вычитание множимого и одно сложение после сдвига множимого на 2 разряда, т.е. пара разрядов множителя преобразуется к виду 1, 0, $\bar{1}$. Поскольку сложение после сдвига приходится на умножение на следующую пару разрядов, то вместо того, чтобы его выполнять добавляют единицу в следующую за данной пару разрядов. С учетом этого действия при умножении выполняются в соответствии с таблицей 4.1.

Таблица 4.1.

Анализируемая пара разрядов	Перенос из предыд. пары разрядов	Преобразованная пара разрядов	Примечание
00	0	00	
01	0	01	
10	0	10	Сдвинутое множимое
11	0	$0\bar{1}$	Запоминается единица для следующей пары
00	1	01	
01	1	10	Сдвинутое множимое
10	1	$0\bar{1}$	Запоминается единица для следующей пары
11	1	00	

Описанная процедура повторяется для всех пар разрядов множителя, а также для одной пары разрядов левее запятой, т.к. может оказаться необходимым добавить к ней (к ее нулям) единицу.

Например, множителю 0,100111 соответствует преобразованный множитель $0,101001\bar{1}$.

Пример: A =0,101011; B = 0,100111.

			00 1 0 1 0 1 1	Рг. Множимого A
			11 0 1 0 1 0 1	– A
Рг. множителя			01 0 1 0 1 1 0	2·A
			00 0 0 0 0 0 0 0 0 0 0 0 0	Рг. СМ
	1 0 0 1	11 – A	11 0 1 0 1 0 1	
			11 0 1 0 1 0 1 0 0 0 0 0	Рг. СМ
		Сдвиг	11 1 1 0 1 0 1 0 1 0 0 0	→
Сдвиг Рг →	0 0 1 0	01 + 2·A	01 0 1 0 1 1 0	
			01 0 0 1 0 1 1 0 1 0 0 0	Рг. СМ
		Сдвиг	00 0 1 0 0 1 0 1 1 0 1 0 0	→
Сдвиг Рг →	0 0 0 0	10 + 2·A	01 0 1 0 1 1 0	
			01 1 0 1 0 0 0 1 1 0 1 0 0	Рг. СМ
		Сдвиг	00 0 1 1 0 1 0 0 0 1 1 0 1	→

C = A×B = 0,011010001101.

В примере вертикальной чертой отделены два знаковых разряда сумматора и два анализируемых разряда регистра множителя.

Умножение выполняется за 3 такта, вместо 6.

Таким образом, ускорение умножения достигается за счет того, что:

1) при комбинации 11 вместо двух сложений выполняется одно вычитание, которое эквивалентно сложению, благодаря использованию инверсных кодов;

2) добавление 1 в соседнюю старшую группу множителя производится одновременно с вычитанием множимого и, следовательно, не увеличивает время умножения.

Следует отметить, что, в общем случае, при умножении на 2 разряда множителя двух знаковых разрядов в сумматоре недостаточно. Здесь возможны случаи при A → 1,0, когда даже во втором знаковом разряде появляется единица переполнения, т.е. будет искажен знак частного произведения. Действительно, если в первом и во втором тактах выполнялась операция "+2A", в сумматоре получим содержимое: в первом такте S_1 = 2A, во втором такте после сдвига содержимого сумматора на 2 разряда вправо $S_2 = 2A + 2^{-1}A$. Если A → 1,0, то $S_2 > 2$.

Будем исходить из худшего случая, когда во всех тактах умножения выполняются операции "+2A". Тогда в сумматоре будет получена последняя сумма следующего вида:

$$S_{n/2} = A\cdot 2 + A\cdot 2^{-1} + A\cdot 2^{-3} + \ldots + A\cdot 2^{-(n/2-1)} = 2A + A(1 - 2^{-(n/2-1)}).$$

При любом A сумма $S_{n/2} < 3$. Следовательно, при данном способе умножения сумматор должен иметь три знаковых разряда.

При одновременном анализе двух разрядов, начиная со старших, правила преобразования множителя существенно изменяются. На характер действий влияет значение соседней справа цифры по отношению к анализируемой паре разрядов. При этом анализ разрядов всегда начинается с пустой пары. В зависимости от значения соседнего справа разряда, происходит изменение состояния преобразуемых разрядов и выполнение действий в соответствии с таблицей 4.2.

Таблица 4.2.

Анализируемая пара разрядов b_i, b_{i+1}	Перенос из предыдущей пары разрядов c_{i-1}	Преобразованная пара разрядов b_i, b_{i+1}	Примечание
00	0	00	
01	0	01	
10	0	10	Предварительный сдвиг множимого
11	0	$0\bar{1}$	
00	1	01	
01	1	10	Предварительный сдвиг множимого
10	1	$0\bar{1}$	Запоминается единица для следующей пары разрядов
11	1	00	

Следует отметить, что объем оборудования арифметического устройства при умножении на 2 разряда увеличивается незначительно по сравнению с устройством, работающим без ускорения.

Одновременное умножение на три и более разряда, для реализации которого применим удобный способ ускорения, используется реже, так

как при этом увеличивается количество требуемых типов передач. Это приводит к значительному усложнению схем, реализующих умножение.

Вместе с тем, в некоторых ЭВМ, например, старших моделей серии ЕС для ускорения операции умножения используется шестнадцатиричная система счисления, т.е. система с основанием $P = 2^4$.

Одновременный анализ четырех двоичных разрядов в соответствии с таблицей 4.3 позволяет осуществлять сдвиг сразу на четыре двоичных разряда и в 4 раза уменьшает количество тактов суммирования, необходимых для выполнения операции умножения. При этом для накопления частных произведений также требуется сумматор дополнительного или обратного кода, а анализ начинается с младшей тетрады множителя.

Таблица 4.3.

Анализируемая цифра	Анализируемая тетрада	Действия в сумматоре при значении предыдущей тетрады	
		> 8	< 8
0	0000	+1A	0
1	0001	+2A	+1A
2	0010	+3A	+2A
3	0011	+4A	+3A
4	0100	+5A	+4A
5	0101	+6A	+5A
6	0110	+7A	+6A
7	0111	+8A	+7A
8	1000	−7A	+8A
9	1001	−6A	−7A
A	1010	−5A	−6A
B	1011	−4A	−5A
C	1100	−3A	−4A
D	1101	−2A	−3A
E	1110	−1A	−2A
F	1111	0	−1A

Этот способ может быть реализован, например, путем предварительного формирования таблицы значений 3A, 5A и 7A, занесения этой таблицы в соответствующие регистры и подключения их выходов ко входу сумматора. Получить таблицу можно следующим образом. В сумматор передается +A, а затем во втором такте добавляется +2A. Резуль-

тат сложения +3А передается в первый регистр таблицы и остается в сумматоре. В третьем такте путем прибавления +2А в сумматоре формируется результат +5А и передается во второй регистр таблицы, в четвертом такте получается +7А. Значения множимого ±2А, ±4А и +8А формируются при передаче в сумматор множимого с регистра А прямым или инверсным кодом со сдвигом соответственно на 1, 2 и 3 разряда влево. Значения множимого +6А формируются при передаче значения 3А из первого регистра таблицы прямым или инверсным кодом со сдвигом на один разряд влево.

С целью исключения четырех подготовительных тактов, которые необходимы для получения таблицы значений 3А, 5А, 7А, требуемое значение множимого можно сформировать за один такт схемным путем. Однако объем оборудования при этом существенно возрастает.

Следует отметить, что при реализации данного способа сумматор должен содержать 5 знаковых разрядов, а регистры таблицы по 4 знаковых разряда.

4.4.4. Умножение с запоминанием промежуточных переносов

Время умножения можно сократить путем уменьшения длительности каждого такта передачи множимого в СМ за счет исключения из него времени, затрачиваемого на распространение переносов. Суть этого способа ускорения состоит в том, что весь процесс получения произведения выполняется суммированием без распространения переносов с одновременным их накоплением и однократным распространением переносов на заключительном этапе умножений. При этом в каждом цикле умножения суммируются поразрядно три числа: множимое, если текущий разряд множителя равен 1, частичное произведение, как результат умножения на предыдущие разряды, и переносы, образованные в предыдущем цикле умножения. При этом переносы, полученные в предыдущем цикле, должны запоминаться до начала следующего цикла.

4.4.5. Матричный метод умножения

Когда множимое и множитель расположены в регистрах машины, нетрудно образовать сразу все частные произведения. Следовательно, при наличии дополнительных сумматоров можно складывать сразу несколько частных произведений, а в предельном случае и все. В этом случае формирование произведения можно себе представить как спуск по дереву сумматоров от слагаемых к их общей сумме. Время спуска по дереву будет зависеть от его организации и типа применяемых сумм-

маторов. Рассмотрим схему умножения на примере двух шестиразрядных чисел

$$
\begin{array}{cccccccccccc}
 & A = & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & & & & \\
 & B = & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & & & & \\
\hline
 & & a_6b_1 & a_5b_1 & a_4b_1 & a_3b_1 & a_2b_1 & a_1b_1 & & & & \\
 & a_6b_2 & a_5b_2 & a_4b_2 & a_3b_2 & a_2b_2 & a_1b_2 & & & & & \\
 a_6b_3 & a_5b_3 & a_4b_3 & a_3b_3 & a_2b_3 & a_1b_3 & & & & & & \\
+ & & \ldots & & & & & & & & & \\
\hline
c_{12} & c_{11} & c_{10} & c_9 & c_8 & c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1
\end{array}
$$

Эту схему умножения можно представить в виде матрицы (таблица 4.4), каждый элемент которой равен 0 или 1 для p = 2. Для получения произведения двух чисел элементы матрицы надо суммировать, как уже указывалось выше, в силу порядка.

Таблица 4.4

b_j	a					
	a_6	a_5	a_4	a_3	a_2	a_1
b_1	a_6b_1	a_5b_1	a_4b_1	a_3b_1	a_2b_1	a_1b_1
b_2	a_6b_2	a_5b_2	a_4b_2	a_3b_2	a_2b_2	a_1b_2
b_3	a_6b_3	a_5b_3	a_4b_3	a_3b_3	a_2b_3	a_1b_3
b_4	a_6b_4	a_5b_4	a_4b_4	a_3b_4	a_2b_4	a_1b_4
b_5	a_6b_5	a_5b_5	a_4b_5	a_3b_5	a_2b_5	a_1b_5
b_6	a_6b_6	a_5b_6	a_4b_6	a_3b_6	a_2b_6	a_1b_6

$$
\begin{array}{cccc}
 & a_3b_1 & a_2b_1 & a_1b_1 \\
+ & a_2b_2 & a_1b_2 & \\
+ & a_1b_3 & & \\
\end{array}
$$

Элементы, составляющие каждый i-й столбец матрицы, просуммируем с помощью обычных трехвходовых двоичных сумматоров SM. Значения переносов с этих сумматоров должны быть слагаемыми для (i+1)-го столбца. Это приведет к тому, что в каждом (i+1)-м столбце появление слагаемых будет происходить с запаздыванием по времени относительно i-го столбца. Произведем распределение первичных и запаздывающих слагаемых для i, j = $\overline{1, 4}$ так, как это показано на рис. 4.5.

Схема устройства для реализации этого дерева спуска, которое представляет собой одну из разновидностей матричного алгоритма умножения, представлена на рис. 4.6 для i, j = $\overline{1, 6}$.

Для одноразрядных сумматоров время образования суммы больше, чем время образования переноса $\tau_п$, поэтому наибольшее время спуска по дереву данного вида есть

$$T_ц = (n - 1)(\tau_s + 2\tau_п), \qquad (4.13)$$

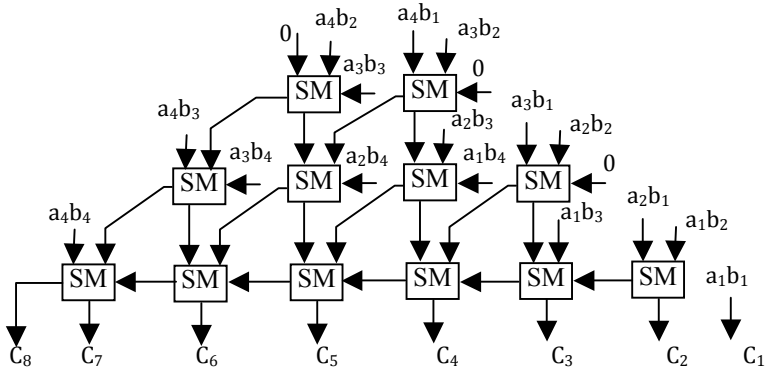


Рис. 4.5. Схема умножителя 4×4

если только все конъюнкции вида a_ib_j поступают на дерево спуска одновременно. Это время складывается из спуска по самой длинной вертикали, а затем последовательного распространения переноса через сумматоры нижнего ряда вплоть до самого левого.

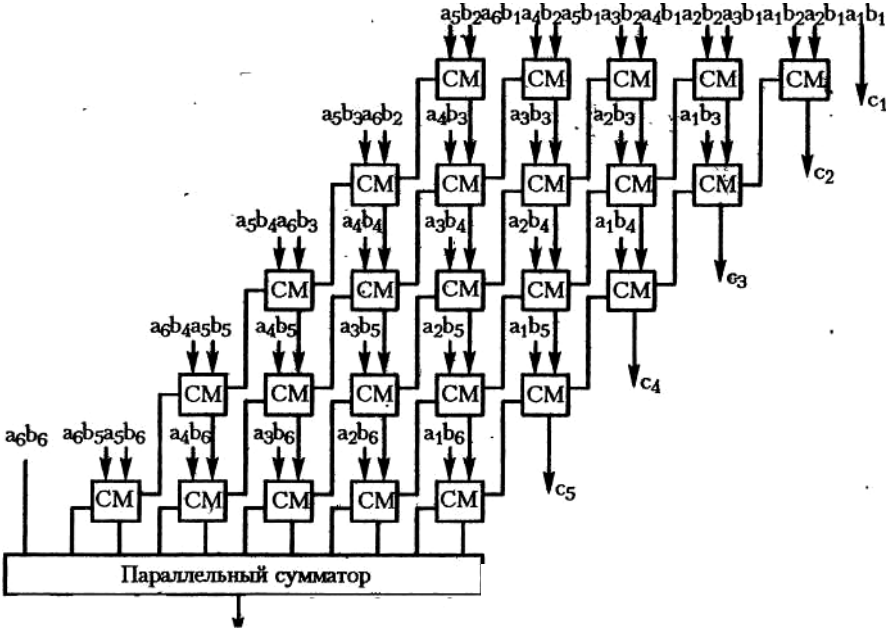


Рис. 4.6

Существенный вклад в величину $T_ц$ вносит время спуска по вертикали. Отдельная вертикаль в исследуемом дереве представляет собой последовательное соединение одноразрядных сумматоров. Дерево спуска по вертикали можно сделать таким, как показано на рис. 4.7. Если такое дерево имеет j ступеней, то справедливо неравенство

$$2^j \le m \le 2^{j+1} - 1,$$

где m – число слагаемых. Время спуска по этому дереву составит

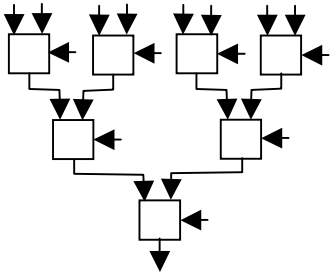


Рис. 4.7

$$\tau_{сп} = j\tau_s.$$

Указанная организация спуска по вертикали приводит к тому, что дерево сумматоров трансформируется (рис. 4.8). Новое дерево более "низкое", чем исходное, и естественно, что время спуска уменьшается до величины

$$T_ц = \log_2 n \cdot \tau_s + 2(n-2)\tau_п, \qquad (4.14)$$

если только справедливо $t = \tau_п/\tau_s < 1$, что обычно выполняется на практике.

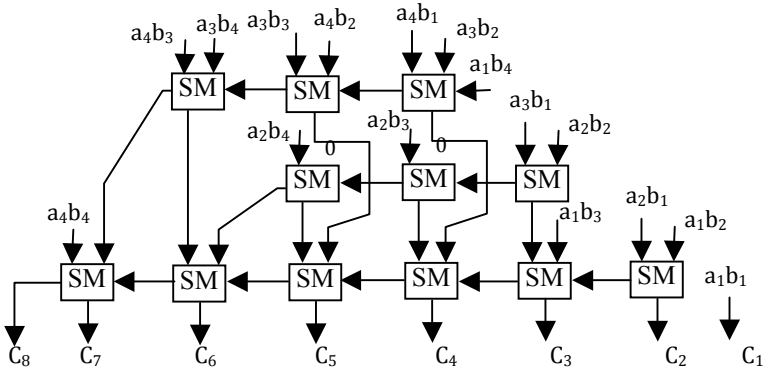


Рис. 4.8. Схема усовершенствованного умножителя 4×4

Таким образом, преобразование дерева спуска уменьшит время спуска по нему в g раз:

$$g = \frac{(n-1)(\tau_s + 2\tau_п)}{\log_2 n \cdot \tau_s + 2(n-2)\tau_п} = \frac{(n-1)(1+2t)}{\log_2 n + 2(n-2)t} \approx \frac{1+2t}{2t}.$$

Когда $\tau_п = \tau_s$, то оказывается, что g = 1,5. Если же t = ½, то g = 2, т.е. ускорение спуска равно почти 2 раза.

Как следует из соотношения (4.14), время формирования произведения деревом с измененной организацией оказывается приблизительно равным сложению чисел разрядности 2n в сумматоре с последовательным распространением переноса [50, 92].

Дальнейшее ускорение умножения может быть связано с применением в нижнем ряду дерева спуска сумматора не с последовательным распространением переноса, а какого-либо другого типа, который приводит к уменьшению времени сложения.

Задачей дерева спуска является сведение n частных произведений к их сумме. Рассмотренная структура дерева спуска не единственная. Будем строить дерево спуска из однотипных параллельных сумматоров, каждый из которых состоит из обычных двоичных одноразрядных сумматоров, не связанных друг с другом. Дерево построим следующим образом. Все n частных произведений будем одновременно суммировать на [n/3] сумматорах. Полученные m = 2[n/3] слагаемых вновь одновременно суммируем на [m/3] сумматорах и объединяем выходы с этих сумматоров в новый, массив и т.д. Через λ ступеней мы от n слагаемых приходим к двум слагаемым – векторам s и π (рис. 4.9).

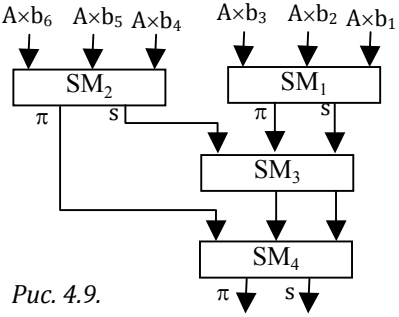


Рис. 4.9.

Наибольшее число слагаемых, которое может быть просуммировано таким деревом при λ ступенях, определится из рекуррентного соотношения

$$n_\lambda = [1{,}5\ n_{\lambda-1}] \qquad (4.15)$$

где $n_0 = 2$. Если число слагаемых n больше, чем значение n_λ, определенное из (4.15), то число ступеней в дереве спуска должно быть увеличено.

Время спуска по дереву приведенной структуры есть

$$\tau_{сп} = \lambda\tau_s.$$

Всего число параллельных сумматоров в дереве спуска есть n – 2 правда, они не все одинаковой разрядности. На рис. 4.10, представлена деталировка дерева спуска, изображенного на рис. 4.9 в двоичных одноразрядных сумматорах. Номера внутри этих сумматоров указывают их принадлежность к соответствующим параллельным сумматорам.

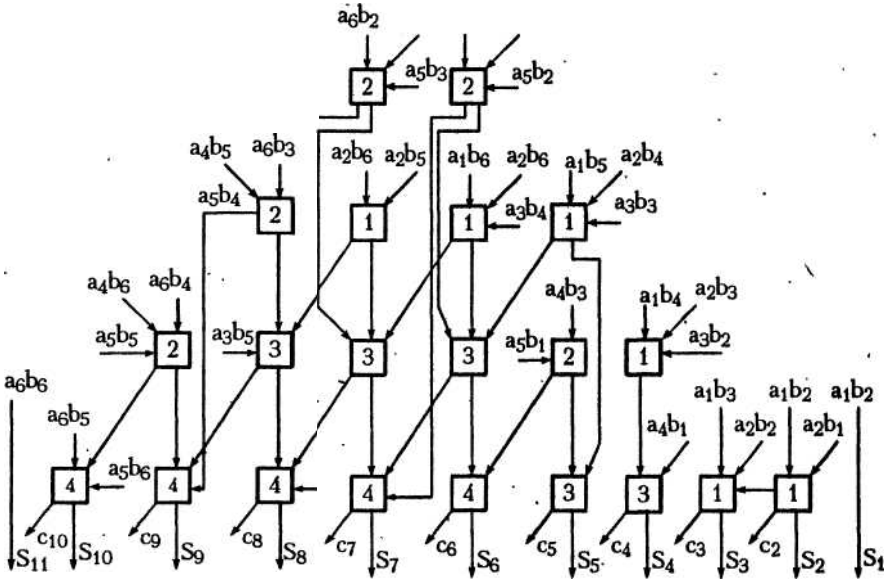


Рис.4.10.

При определенном компромиссе возможен одновременный учет лишь группы частных произведений. В этом случае полученная сумма наряду с новыми частными произведениями вновь подается в дерево спуска для последующих суммирований. Итеративный процесс заканчивается, когда все частные произведения будут учтены. Для уменьшения времени, отводимого для реализации одной итерации, полученную сумму следует направить не на вход дерева спуска, а на самую нижнюю допустимую ступень. Пусть n = uv, где n – общее число частных произведений, v – число итераций, u – число частных произведений, обрабатываемых на одной итерации; тогда полное время приведения n слагаемых к двум при использовании дерева спуска типа рис. 4.11 составит

$$\tau_{сп} = (\lambda(u) + 2v)\,\tau_s.$$

Если же использовать полное дерево, то время спуска $\tau^*_{сп} = \lambda(n)\tau_s$. Например, при n = 30, u = 6 и v = 5 получим $\tau_{сп} = 13\tau_s$ и $\tau^*_{сп} = 8\tau_s$. Заметим,

что требуемое оборудование существенно отличается для этих двух схем – при итерационном спуске число сумматоров есть u, а при прямом – n – 2. В нашем случае эти величины есть 6 и 28 соответственно.

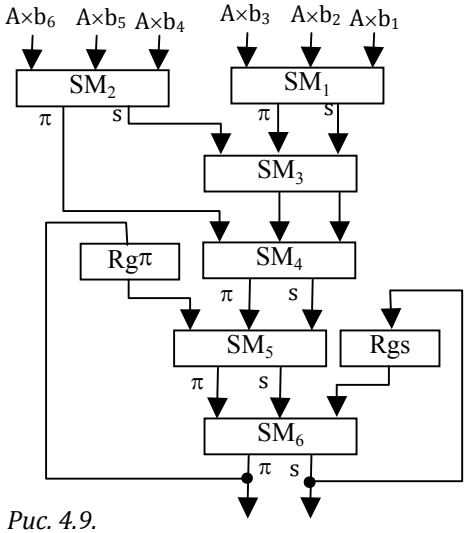


Рис. 4.9.

Матричные способы умножения дают выигрыш во времени умножения, но требуют большого объема оборудования по сравнению с умножением методом накопления частных произведений. Вместе с тем, в связи с быстрым прогрессом микроэлектроники в области создания больших интегральных схем (БИС) они все чаще применяются на практике.

Наконец, для ускорения операции умножения можно использовать таблицы. Двоичное умножение с помощью таблиц сводится либо к выбору из таблицы готового результата, либо к суммированию сдвинутых относительно друг друга частичных произведений, если в таблицах хранятся результаты умножения нескольких разрядов множимого на несколько разрядов множителя.

4.4.6. Быстрое умножение чисел большой разрядности

Пусть имеется умножитель, позволяющий перемножать m-разрядные числа в p-ичной системе счисления, а требуется перемножить n-разрядные числа и n = km, где k > 1 и целое. Нас будет интересовать вопрос о времени T(n) умножения n-разрядных чисел, выраженных в единицах времени умножения m-разрядных чисел.

Рассмотрим случай целых чисел и выберем вначале к = 2, тогда сомножители A и B можно представить в виде

$$A = p^m a' + a'',$$

где $a' = \sum_{i=0}^{m-1} a_{m+i} \cdot p^i$ и $a'' = \sum_{i=0}^{m-1} a_i \cdot p^i$. Произведение чисел A и B определится как $D = AB = p^{2m} a'b' + p^m \cdot (a'b''+a''b')+a''b''$. Для того, чтобы вычислить это произведение, надо произвести четыре умножения m-разрядных чисел, две операции сдвига и три операции сложения, следовательно,

$$T(2m) = 4T(m)+3\tau_s + 2\tau_{сдв}.$$

В то же время возможен и другой алгоритм вычисления произведения $A \times B$.

$$D = (p^{2m} + p^m)a'b' + p^m(a' - a'')(b'' - b') + (p^m + 1)a''b''.$$

Время вычисления произведения по этому алгоритму есть

$$T(2m) = 3T(m)+ 4\tau_s + 4\tau_{сдв}.$$

Обычно $T(m) >> 3\tau_s + 2\tau_{сдв}$, и второй алгоритм оказывается более быстрым.

Приведенный пример показывает, что при умножении k-кратных чисел количество умножений однократных чисел может быть сделано пропорциональным не k^2, как это кажется на первый взгляд, а меньшей величине, которая составляет:

$$T(km) \le (2k - 1)T(m)+Cm,$$

где C – константа, учитывающая необходимость выполнения помимо умножений еще и некоторого числа сложений и сдвигов.

Для n-разрядных чисел, представленных в двоичной системе счисления, существуют алгоритмы [35], которые позволяют вычислять 2n-разрядное произведение за время меньше, чем $Cn2^{\sqrt{(2\log 2n)}}\log_2 n$ шагов. Если при умножении двоичных чисел использовать быстрое преобразование Фурье, то

$$T(n) = Cn \cdot \log_2 n \log_2 \log_2 n.$$

Эти алгоритмы оказываются значительно более быстрыми; нежели очевидный алгоритм умножения, требующий для своей реализации времени, пропорционального величине n^2.

4.5. Умножение чисел, заданных в дополнительном коде

Операцию умножения проще всего выполнять в прямых кодах чисел. Вместе с тем, применение инверсных кодов позволяет существенно упростить операцию алгебраического сложения. Поэтому числа желательно хранить в ЗУ и умножать также в инверсном коде. В этом случае сомножители заданы в прямом коде, если они положительны, либо в инверсном коде, если они отрицательны. Необходимо получить произведение в прямом коде, если оно положительно, или в инверсном коде, если оно отрицательно. При этом, с целью устранения циклических переносов, рациональнее использовать дополнительный код. При умножении в дополнительном коде, также как и при алгебраическом сложении, требуется введение поправок в предварительный результат.

Если число А положительно, то его дополнительный код равен самому А, если же А отрицательно, то значащие цифры его дополнительного кода образуют величину $|[A]_д| = 1 - |A|$. Поэтому при умножении модулей кодов операндов в зависимости от сочетания знаков сомножителей возникают 4 случая.

1. $A > 0$, $B > 0$. Случай тривиальный, получаем сразу истинное значение положительного произведения:

$$[A]_д[B]_д = [A \times B]_д = A \times B.$$

2. $A > 0$, $B < 0$. В этом случае получается следующий результат (псевдопроизведение)

$$[A]_д |[B]_д| = A - |A| \times |B|.$$

Истинное произведение в дополнительном коде должно быть равно $|[A \times B]_д| = 1 - |A| \times |B|$. Поэтому требуется коррекция псевдорезультата – на величину $\Delta = 1 - |A|$, т.е. на величину дополнения модуля множимого до одного.

3. $A < 0$, $B > 0$. Получаем после непосредственного применения алгоритма умножения:

$$|[A]_д| \, [B]_д = B - |A| \times |B|$$

Истинный результат в дополнительном коде составит:

$$|[A \times B]_д| = 1 - |A| \times |B|.$$

Следовательно, требуется коррекция псевдорезультата на величину $\Delta = 1 - |B|$, т.е.на величину дополнения множителя до единицы.

4. $A < 0$, $B < 0$. После непосредственного применения алгоритма умножения получаем:

$$|[A]_д|\cdot|[B]_д| = (1 - |A|)(1 - |B|) = 1 - |A| - |B| + |A| \times |B|.$$

Псевдорезультат требует коррекции на величину: $\Delta = |A|+|B|$, так как лишняя 1 размещается в знаковом разряде и может быть просто заменена знаком произведения.

Как видим, сложность коррекции результата при умножении чисел в дополнительных кодах обусловлена тем, что исправлять нужно не только знак, но и цифровую часть произведения. Коррекцию можно производить или в процессе формирования результата, или сразу по окончании этого процесса. Первый способ применяют чаще, так как он не требует дополнительных тактов работы арифметического устройства, снижающих скорость счета и усложняющих схему управления умножением. Кроме того, при сдвигах, как правило, теряются один или оба сомножителя, а повторное обращение за ними в ЗУ нежелательно, так как снижает быстродействия ЭВМ.

Тогда коррекцию псевдорезультата для отрицательного множителя можно выполнить при умножении множимого А на единицу знака множителя В, рассматривая ее как отрицательную. То есть вместо суммирования А в этом цикле умножения, нужно выполнять вычитание А или же суммирование $[-A]_д$. Ввиду того, что в этом цикле прибавляется $2 - |A|$ вместо $1 - |A|$, то сразу же формируется и правильная знаковая цифра результата, так как в силу исходного сочетания знаков результат должен быть отрицательным и, значит, представлен в дополнительном коде. Таким образом, описанный прием обеспечивает коррекцию как величины результата, так и его знака.

Пример: Заданы: А = 0,1011; $[-A]^м_д$ =11,0101;

В = – 0,1101; $[B]^м_д$ = 11,0011. Найти $[A]_д \times [B]_д$.

00,1011	
×11,0011	
11,0101	$-A\cdot 2^0$
00,00000	$+A\cdot 0$
00,000000	$+A\cdot 0$
00,0001011	$+A\cdot 2^{-3}$
00,00001011	$+A\cdot 2^{-4}$
$\|[A]_д\|\cdot\|[B]_д\|$ = 11,01110001	

Получаем правильный результат со знаком.

При отрицательном множимом и положительном множителе требуется коррекция результата на величину -В, введение которой зависит от схемы умножения. Для любой схемы умножения справедливо

$$[A]_д \times B = [A]_д \times 0,b_1b_2...b_n = [A]_д \cdot b1 \cdot 2^{-1} + [A]_д \cdot b_2 \cdot 2^{-2} + ... + [A]_д \cdot b_n \cdot 2^{-n}. \quad (4.16)$$

Следовательно, при положительном множителе умножение производится как в прямых кодах, с тем отличием, что множимое передается в сумматор со знаком в модифицированном дополнительном коде. При этом в зависимости от схемы умножения выполняются модифицированные сдвиги либо множимого, либо содержимого сумматора.

Пример: Заданы: $A = -0,1011$; $[A]^м_д = -11,0101$;

$B = +0,1101$; $[B]^м_д = 00,1101$. Найти $[A]_д \times [B]_д$.

4-я схема умножения

11,0101	A
× 00,1101	B
00,0000	$-0 \cdot A \cdot 2^0$
11,10101	$+A \cdot 2^{-1}$
11,110101	$+A \cdot 2^{-2}$
00,0000000	+0
11,11110101	$+A \cdot 2^{-4}$
11,01110001	

1-я схема умножения

A	11,0101	
B	× 00,1101	
+A	11,0101	→
+0	11,10101	→
	11,110101	
+A	11,0101	
	11,001001	→
	11,1001001	
+A	11,0101	
	10,1110001	→
	11,01110001	

Для 1-й схемы выполняется модифицированный сдвиг суммы частных произведений, для 4-й – сдвиг множимого.

Если оба сомножителя отрицательны, то коррекция + |A|+|B| может быть получена объединением рассмотренных способов введения коррекций |A| и |B|. При этом получается правильный результат со своим знаком.

Пример: Заданы: $A = -0,1011$; $[A]^м_д = 11,0101$;

$B = -0,1101$; $[B]^м_д = 11,0011$. Найти $[A]_д \times [B]_д$.

11,0101	A
× 11,0011	B
00,1011	$-A \cdot 2^0$
00,01011	+ 0
00,001011	+ 0
11,1110101	$+A \cdot 2^{-3}$
11,11110101	$+A \cdot 2^{-4}$
00,10001111	

Если сомножители представлены обратным кодом, то методы коррекции псевдопроизведения значительно усложняются. Поэтому в этом случае проще избавиться от отрицательных сомножителей, благодаря использованию инверсных передач, а затем результат по необходимости преобразовать в обратный код.

Если сомножители представлены дополнительным кодом, можно применить также способ умножения, основанный на сопоставлении двух смежных цифр множителя.

В дополнительных кодах произведение имеет вид:

$$[C]_д = [A]_д \times [B]_д = - [A]_д \cdot b_0 2^0 + [A]_д \cdot b_1 2^{-1} + ... + [A]_д \cdot b_i 2^{-i} + ... + [A]_д \cdot b_n 2^{-n} \quad (4.17)$$

где b_0 – знаковый разряд множителя. Произведение можно представить следующим образом.

$$[C]_д = [A]_д \times [B]_д = [A]_д(b_1 - b_0)2^0 + [A]_д(b_2 - b_1)2^{-1} + ... + [A]_д(b_{i+1} - b_i)2^{-i} + ... + [A]_д(b_{n+1} - b_n). \quad (4.18)$$

При этом $b_{n+1} = 0$.

Тогда, с учетом того, что перед началом умножения в сумматоре находится код нуля, для четвертой схемы умножения на нулевом шаге будем иметь: $C_0 = 0 + (b_1 - b_0)[A]_д \cdot 2^0$.

На первом шаге: $C_1 = C_0 + (b_2 - b_1)[A]_д \cdot 2^{-1}$.

На i-м шаге: $C_i = C_{i-1} + (b_{i+1} - b_i)[A]_д \cdot 2^{-i}$.

Аналогичные результаты можно получить и для остальных схем умножения.

Таким образом, произведение формируется по следующим правилам:

1) Если данная цифра множителя равна 1, а цифра соседнего справа разряда множителя есть 0, то множимое со своим весом следует вычитать.

2) Если данная цифра множителя 0, а цифра соседнего справа разряда есть 1, то множимое следует прибавлять со своим весом.

3) Если данная цифра множителя такая же, как и цифра соседнего справа разряда, то данное частное произведение равно 0.

При этом множимое всегда передается в сумматор со своим знаком в модифицированном коде, а сдвиги множимого или суммы частных произведений (в зависимости от схемы умножения) должны быть модифицированными.

Ценность этого способа состоит в том, что при любом сочетании знаков сомножителей процесс умножения остается неизменным. Коррекция, необходимая при умножении сомножителей или их изображений, в

данном случае, вводится автоматически, так как знаковые разряды чисел обрабатываются точно так же, как и цифровые.

Примеры: 1) A = 0,1011; B = 0,1101;

	00,1011
×	00,1101
	00,1011
	00,00000
+	11,110101
	00,0001011
	11,11110101
	00,10001111

2) A =0,1011; B = – 0,1101; $[B]^{м}_{д}$ = 11,0011.

	00,1011
×	11,0011
	11,0101
	00,00000
+	00,001011
	00,0000000
	11,11110101
	11,01110001

Примеры: 3) A = – 0,1011; B = 0,1101; $[A]^{м}_{д}$ = 11,0101.

	11,0101
×	00,1101
	11,0101
	00,00000
+	00,001011
	11,1110101
	00,00001011
	11,01110001

4) A = –0,1011; B = –0,1101; $[A]^{м}_{д}$ = 11,0101; $[B]^{м}_{д}$ = 11,0011.

	11,0101
×	11,0011
	00,1011
	00,00000
+	11,110101
	00,0000000
	00,00001011
	00,10001111

Во всех случаях получены прежние результаты.

4.6. Умножение чисел в машинах с плавающей запятой

Если заданы $A = a \cdot 2^{m1}$ и $B = b \cdot 2^{m2}$ в нормальной форме, то их произведение составит

$$A \times B = ab2^{m1+m2}.$$

Следовательно, умножение прямых кодов нормализованных чисел состоит из следующих этапов:

1. Определение знака произведения путем сложения по mod 2 знаковых цифр мантисс сомножителей.

2. Алгебраическое сложение порядков сомножителей в инверсном коде с целью определения порядка произведения.

3. Умножение модулей мантисс сомножителей по правилам умножения чисел с фиксированной запятой.

4. Нормализация и округление мантиссы результата. Следует учесть, что сомножители являются нормализованными числами. Поэтому денормализация мантиссы произведения возможна только на один разряд вправо. Она устраняется путем сдвига мантиссы на один разряд влево и вычитания единицы из порядка результата.

5. Присвоение знака результату.

Первые три операции могут выполняться одновременно, так как они независимы. Операция определения знака произведения предполагает, что умножение мантисс выполняется в прямом коде. Однако можно умножать непосредственно изображения мантисс с последующей коррекцией результата, как и при числах с фиксированной запятой.

При выполнении операции умножения в машине с плавающей запятой может получиться переполнение отрицательного порядка, которое будет интерпретировано как машинный нуль, если программой пользователя игнорируется признак исчезновения порядка. Может также возникнуть и переполнение положительного порядка. В этом случае в первую очередь необходимо нормализовать мантиссу результата. Если и после этого переполнение порядка не устраняется, то машиной формируется признак переполнения порядка.

Округление мантиссы результата производится следующим образом. Если мантисса результата получилась без нарушения нормализации, то округление производится обычно добавлением единицы в (n+1)-й разряд сумматора. Если мантисса произведения денормализована вправо, то при 2n-разрядном результате вначале производится нормализация, а затем округление; при (n + 1)-разрядном результате вначале производится округление, а затем нормализация.

Во всех случаях единица округления добавляется в (n + 1)-й разряд сумматора только при наличии в нем кода "1".

4.7. Деление чисел с восстановлением остатков

Деление в ЭВМ, так же как и умножение, проще всего выполнять в прямом коде. Но в отличие от умножения дробных сомножителей, где не может возникнуть переполнение разрядной сетки, при делении правильных дробей такое переполнение возможно в случае, когда делимое больше делителя. Признаком переполнения является появление целой части в частном, что грубо искажает результат.

Знак частного при делении в прямом коде определяется как сумма по модулю 2 знаковых цифр делимого и делителя и присваивается частному в конце операции деления.

Частное определяется путем деления модулей исходных чисел. При этом во избежание переполнения разрядной сетки должно соблюдаться условие: |A| < |B|, где A – делимое, B – делитель.

Пусть требуется получить Y от деления A на B с точностью до i-го разряда. Тогда:

$$Y_i = A/B = 0, y_1, y_2, y_3,\ldots, y_i = y_1 \cdot 2^{-1} + y_2 \cdot 2^{-2} + y_3 \cdot 2^{-3} + \ldots + y_i \cdot 2^{-i}. \quad (4.19)$$

При любом значении i должно выполняться неравенство

$$0 \le A - B \cdot Y_i < B \cdot 2^{-i}, \quad (4.20)$$

т.е. остаток от деления (A-BYi) должен быть меньше делителя, умноженного на 2^{-i}. Преобразуем (4.20) к виду:

$$0 \le (A - B \cdot Y_i) 2^i < B. \quad (4.21)$$

Обозначив остаток $(A - B \cdot Y_i) 2^i = R_i$, получим

$$0 \le R_i < B, \quad (4.22)$$

Цифры частного определяются последовательно, начиная со старшего разряда. Допустим, что в результате выполнения i циклов получены старшие i разрядов частного, т.е. приближенное значение частного Y_i. В следующем (i + 1)-м цикле будет получено значение (i + 1)-го разряда частного. Исходными данными для этого цикла являются Y_i и R_i. Цифра Y_{i+1} может иметь одно из двух значении 1 или 0.

Если $y_{i+1} = 0$, то

$$Y_{i+1} = Y_i + y_{i+1} \cdot 2^{-(i+1)} = Y_i, \quad (4.23)$$

$$R_{i+1} = (A - B \cdot Y_{i+1}) 2^{i+1} = 2R_i \quad (4.24)$$

т.е. в частном записывается 0 при условии:

$$0 \le R_i = 2Ri < B. \quad (425)$$

Если $y_{i+1} = 1$, то

$$Y_{i+1} = Y_i + 2^{-(i+1)}, \quad (4.26)$$

$$R_{i+1} = (A - B \cdot Y_{i+1}) 2^{i+1} = (A - B \cdot Y_i - B \cdot 2^{-(i+1)}) 2^{i+1} = 2R_i - B, \quad (4.27)$$

т.е. цифра частного равна 1, если выполняется условие:

$$0 \le R_{i+1} = 2R_i - B < B \text{ или} \quad (4.28)$$

$$B \le 2R_i < 2B. \quad (4.29)$$

Так как всегда выполняется одно из условий (4.25) или (4.29), то для определения текущей цифры частного достаточно проверить одно

из них. Обычно проверяют условие (4.25). Левая часть этого неравенства выполняется заведомо, так как согласно (4.22) $0 \le R_i$, т.е. очередной остаток перед началом следующего шага деления всегда является неотрицательным числом.

Для проверки правой части неравенства сравним разность (2Rj * B) с нулем. Если эта разность окажется отрицательной, то в (i + 1) разряд частного запишем 0 и для подготовки исходных данных для (i + 2) цикла определим R_{i+1} следующим образом

$$R_{i+1} = (2R_i - B) + B = 2R_i, \qquad (4.30)$$

т.е. фактически восстановим прежний остаток.

Если разность $2R_i - B$ окажется положительной, то запишем в (i + 2) разряд частного 1, а в качестве исходного значения для следующего (i + 2) цикла используем вычисленную разность (см. 4.27):

$$R_{i+1} = 2R_i - B.$$

Исходными данными для 1-го цикла являются: $Y_0 = 0$,

$$R_0 = (A - B \cdot Y_0)2^0 = A < B,$$

т.е. по условию неравенство (4.22) выполняется и перед началом первого цикла. После завершения n-го цикла мы получим n-значное частное Y_n, вычисленное с недостатком $R_n = (A - B \cdot Y_n)2^n$, который равен остатку от деления A на B, сдвинутому влево на n разрядов.

Таким образом, правило деления с восстановлением остатков формируется так. Делитель вычитается из делимого и определяется, знак нулевого по номеру остатка. Если остаток положительный, т.е. $|A| > |B|$, то в псевдознаковом разряде частного проставляется 1, при появлении которой формируется признак переполнения разрядной сетки и операция прекращается. Если остаток отрицательный, то в псевдознаковом разряде частного записывается 0, а затем производится восстановление делимого путем добавления к остатку делителя. Далее выполняется сдвиг восстановленного делимого на один разряд влево и повторное вычитание делителя. Знак получаемого таким образом остатка определяет первую значащую цифру частного: если остаток положителен, то в первом разряде частного записывается 1, если отрицательный, то записывается 0. Далее, если остаток положителен, то он сдвигается влево на 1 разряд и из него вычитается делитель для определения следующей цифры частного. Если остаток отрицателен, то к нему прибавляется делитель для восстановления предыдущего остатка, затем восстановленный остаток сдвигается на 1 разряд влево и из него вычитается де-

литель для определения следующей цифры частного и т.д. до получения необходимого количества цифр частного с учетом одного разряда для округления, т.е. до обеспечения требуемой точности деления.

Пример: Заданы A = 0,101 B = 0,110, Найти Y = A/B.

$[B]^{м}_{д}$ = 11,010.

Определение знака частного и дополнения делителя до двух:

$S_{д}Y = 0 \oplus 0 = 0$

A	00	1 0 1	СМ				
$+[B]^{м}_{д}$	11	0 1 0	Рг В		Рг Y		
	11	1 1 1	СМ				0
+|B|	00	1 1 0					
	00	1 0 1	СМ				
	01	0 1 0	СМ	←			
$+[B]^{м}_{д}$	11	0 1 0					
	00	1 0 0	СМ			0	1
	01	0 0 0	СМ	←	01		
$+[B]^{м}_{д}$	11	0 1 0			011	0	
	11	1 1 0					
+|B|	00	1 1 0					
	00	1 0 0	СМ		0110		
	01	0 0 0	СМ	←			
$+[B]^{м}_{д}$	11	0 1 0					
	00	0 1 0	СМ		0110	1	
Округление:					+	1	
					0111	0	

Присвоение знака частному:

Y = 0,111.

Операция округления показана в регистре частного, хотя она должна выполняться в сумматоре. Вертикальными линиями отделены знаковые разряды остатков и разряд частного, в который заносится его текущая цифра.

Таким образом, цифры частного получаются как инверсное значение знаковых разрядов текущего остатка, которые принимают значение "00" или "11". Однако при сдвиге остатка влево в знаковых разрядах может возникнуть сочетание "01". В некоторых случаях для того, чтобы

цифры частного формировались как прямое значение знакового разряда текущего остатка, деление выполняют с инверсными знаками. При этом делимое передается в сумматор не прямым, а инверсным кодом, а на нулевом шаге выполняется операция "+B", вместо операции "–B".

4.8. Деление без восстановления остатков.

Рассмотренный способ деления с восстановлением остатков является аритмичным процессом с переменным числом шагов (3 шага при $2R_i < B$ и 2 шага при $2R_i > B$). Для ритмизации процесса на каждую цифру частного необходимо затратить по 3 шага, в результате чего увеличивается время выполнения операции. Вместе с тем, операцию можно упростить и получить каждую цифру частного за 2 шага.

Рассмотрим случай, когда $R_i < 0$. В предыдущем способе в этом случае выполнялись следующие операции:

1) Восстановление остатка

$$R' = R_i + |B| = 2R_{i-1} - |B| + |B| = 2R_{i-1}.$$

2) Сдвиг восстановленного остатка влево

$$R'_i = 2R' = 2R_{i-1} \cdot 2 = 4R_{i-1}.$$

3) Вычитание модуля делителя из восстановленного и сдвинутого влево остатка для определения следующего остатка

$$R_{i+1} = 4R_{i-1} - |B|$$

Если не восстанавливать остаток, а сразу сдвинуть отрицательный R| на один разряд влево, то получим:

$$R'_{i+1} = 2Ri = 2(2R_{i-1} - |B|) = 4R_{i-1} - 2|B|.$$

Результат, в данном случае, отличается от действительного на величину +|B|. Поэтому в качестве второго шага необходимо произвести коррекцию результата на ату величину:

$$R_{i+1} = 4R_{i-1} - 2|B| + |B| = 4R_{i-1} - |B|.$$

В результате, получаем требуемую величину следующего остатка R_{i+1} за 2 шага.

Таким образом, чтобы определить следующую цифру частного, необходимо сдвинуть текущий остаток влево на 1 разряд, а затем алгебраически прибавить к нему модуль делителя, которому приписывается знак, противоположный знаку текущего остатка. Знак полученного

таким образом следующего остатка и определяет следующую цифру частного: если остаток положительный, то в частном записывается 1, если отрицательный – то записывается 0. Операция сдвигов и алгебраических сложений повторяется до тех пор, пока в частном не получится требуемое количество цифр.

Пример: Заданы A = 0,101; B = 0,110. $[B]^{м}_{д}$ = 11,010. Определение знака частного:

$S_д Y = 0 \oplus 0 = 0.$

A	00	1 0 1	СМ		Рг Y	
$+[B]^{м}_{д}$	11	0 1 0	Рг B			
	11	1 1 1	СМ		0	
	11	1 1 0	СМ	← 0		Рг Y←
+|B|	00	1 1 0				
	00	1 0 0	СМ		01	
	01	0 0 0	СМ	← 01		Рг Y←
$+[B]^{м}_{д}$	11	0 1 0				
	00	0 1 0	СМ		011	
	00	1 0 0	СМ	← 011		Рг Y←
$+[B]^{м}_{д}$	11	0 1 0				
	11	1 1 0	СМ		0110	
	11	1 0 0	СМ	←0110		
+|B|	00	1 1 0				
	00	0 1 0	СМ		0110	1
	Округление:				+1	
					0111	0

4.9. Машинные схемы деления

В настоящее время во всех ЭВМ деление производится по способу без восстановления остатков. Это, во-первых, упрощает схему управления процессом деления и, во-вторых, увеличивает быстродействие ЭВМ.

При выполнении операции деления результат получится одинаковым, если сдвигать остатки от деления влево, либо делитель вправо. Следовательно, возможны 2 схемы выполнения деления:

1) деление без восстановления останков со сдвигом делителя вправо;

2) деление без восстановления остатков со сдвигом их влево.

Для реализации первого варианта необходимы: 2n-разрядный регистр делителя со сдвигом вправо, 2n-разрядный СМ, n-разрядный регистр частного со сдвигом влево (если округление результата выполняется в сумматоре, то частное пересылается в него по окончании операции) и схема управления (рис. 4.12).

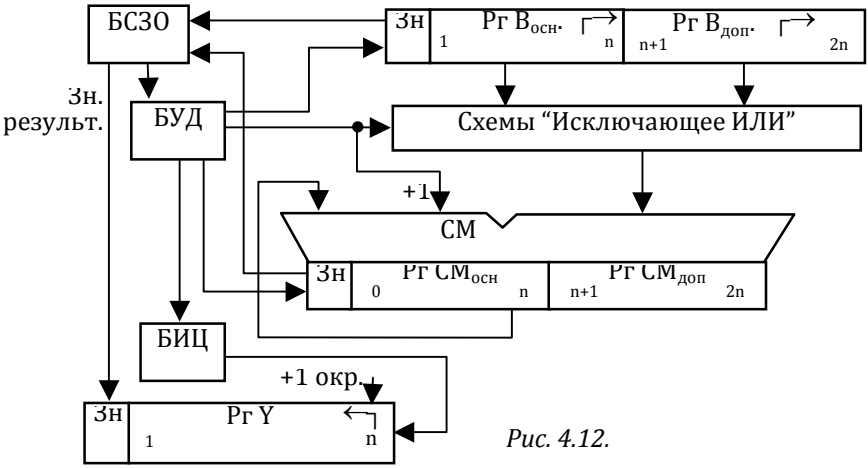


Рис. 4.12.

Передача в сумматор модуля делителя или его дополнения обеспечивается блоком управления делением (БУД), который анализирует знаковые цифры остатков, снимаемые с сумматора с помощью блока съема знаков остатков (БСЗО). Эти знаковые цифры остатков, проходя через блок инверсии цифр (БИЦ), инвертируются и подаются в младший разряд сдвигового регистра частного уже как цифры частного.

Для реализации второго варианта (рис. 4.13) необходим: n-разрядный регистр делителя, (n + 1)-разрядный регистр частного со сдвигом влево, n- или (n + 1)-разрядный сумматор со сдвигом влево и схема управления. Сумматор производит сдвиг текущего остатка влево и алгебраическое сложение его с делителем. Передача в СМ модуля делителя или его дополнения обеспечивается БУД, который анализирует выталкиваемые из СМ знаковые цифры остатков. Эти же цифры инвертируются в БИЦ и подаются в младший разряд Рг частного как цифры частного.

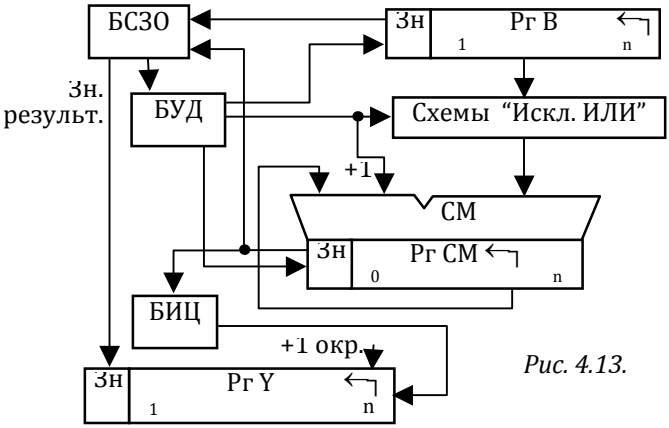


Рис. 4.13.

Анализ обеих схем показывает, что второй вариант примерно на 40% экономичнее по оборудованию по сравнению с первым.

Выбор типа делительного устройства при проектировании машины обычно не является самостоятельной задачей. Поэтому на практике, вначале по заданным техническим условиям выбирается схема множительного устройства вследствие того, что умножение является более частой операцией. После этого выбирается наиболее совместимая с устройством умножения схема делительного блока. Однако при проектировании специализированных ЭВМ, может быть принят другой порядок выбора структур отдельных устройств.

Если сравнивать приведенные схемы деления со схемами множительных устройств, то оказывается, что схема первого варианта деления во многом совпадает с четвертой схемой умножения, второй вариант схемы деления хорошо совместим с третьей схемой умножения.

4.10. Деление чисел в дополнительном коде

Так же, как и при умножении, здесь возможны два пути:

1) перевод операндов в прямой код, получение частного обычным способом и перевод его в дополнительный код перед записью в память;

2) деление операндов в дополнительном, коде с получением частного в требуемом виде.

Второй путь так же прост, как и первый. Необходимо только управлять логикой образования цифр частного, чему способствуют два обстоятельства:

1) делимое в процессе операции участвует всего лишь один раз, а дальше текущие остатки получаются автоматически;

2) нетрудно организовать получение цифр дополнительного кода отрицательного частного непосредственно в процессе деления путем передачи знаковых цифр остатков из СМ в регистр частного напрямую, минуя БИЦ.

Рассмотрим четыре возможных случая, определяемые комбинациями знаков дополнительных кодов операндов.

1-й случай. $A > 0; B > 0; Y = A/B > 0$.

Деление производится как обычно: знак частного определяется путем сложения знаковых цифр операндов по модулю 2 и одновременно формируется дополнение модуля делителя до двух. Цифры частного получаются, как уже известно, путем инвертирования знаковых цифр остатков от деления в БИЦ.

2-й случай. $A < 0; B > 0; Y = A/B < 0$.

Ход деления обычный с учетом двух следующих моментов:

1) Псевдознаковая цифра частного (сигнал о возможном переполнении разрядной сетки) должна получаться в этом случае равной знаковой цифре нулевого остатка, так как здесь производится вычитание модуля делимого из модуля делителя в силу исходного сочетания знаков операндов. Значит, знаковая цифра нулевого остатка должна поступать в регистр частного, минуя БИЦ, т.е. не инвертируясь.

2) Так как знак нулевого остатка имеет инверсное значение, то при продолжении деления обычным путем все другие остатки получаются также с инверсными знаками. Значит, в регистр частного также будут записываться инверсные цифры, и в конце деления в нем образуется обратный код отрицательного частного. Для округления результата из него надо вычесть единицу (n+1)-го разряда, а для перевода его в дополнительный код к нему надо прибавить 1 в n-й разряд. Тогда прибавление 1 в (n+1)-й разряд произведет суммарное действие округления и перевода обратного кода в дополнительный.

3-й случай. $A > 0; B < 0; Y \leq 0$.

Этот случай является зеркальным отражением предыдущего: псевдознаковая цифра частного получается, как обычно, путем инверсии знака первого остатка, а для того, чтобы получить инверсные цифры частного, нужно брать их равными знаковым цифрам соответствующих остатков без инвертирования их в БИЦ. В конце деления необходимо

прибавить к (n+1)-му разряду частного единицу для одновременного выполнения округления результата и перевода его в дополнительный код.

4-й случай. $A < 0; B < 0; Y > 0$.

В этом случае все цифры (включая псевдознаковую) должны быть равны знаковым цифрам соответствующих остатков. На первом этапе деления одновременно с определением истинного знака частного определяется величина модуля делителя, т.е. возникает ситуация, сходная с начальными условиями 2-го случая. Значит, псевдознаковая цифра частного определяется значением знаковой цифры нулевого остатка. Если далее производить деление обычным путем, то, как уже известно, из 2-го случая, будут получаться инверсные цифры модуля частного. Следовательно, знаковые цифры остатков надо посылать в регистр частного, минуя БИЦ, т.к. частное здесь должно быть положительным и его дополнительный код совпадает с прямым.

Таким образом, деление чисел, представленных в дополнительном коде, производится в 3 этапа.

На *первом этапе* определяется знак частного, путем сложения знаковых цифр операндов по модулю 2.

На *втором этапе* выполняется нулевой шаг деления для проверки частного на переполнение разрядной сетки. Делается это путем алгебраического сложения делимого с делителем, которому приписывается знак, противоположный знаку делимого. Если делимое положительное, псевдознаковая цифра модуля частного определяется как обратное значение знаковой цифры нулевого остатка. Если делимое отрицательное, псевдознаковая цифра берется тождественно равной знаку остатка.

На *третьем этапе* производятся все последующие (n+1) циклов деления по обычным правилам, но с той лишь разницей, что при отрицательном делителе цифры частного берутся тождественно равными знаковым цифрам соответствующих остатков. В конце деления в (n+1)-й разряд частного обязательно добавляется 1.

Пример: $A < 0; B > 0; Y \le 0$. Заданы $[A]_д = 1{,}011$; $[B]_д = 0{,}110$.

1. Определяется знак частного $1 \oplus 0 = 1$ и обращается делитель $2 - B = 1{,}010$.
2. Проверка на переполнение (нулевой шаг)

$$1{,}011 + 0{,}110 = 0{,}001.$$

Псевдознак модуля частного равен знаку нулевого остатка, так как делимое отрицательное, т.е. псевдознак частного равен 0.

	0,001	1-й остаток	0,0 0 1 0	$= [Y]_0$
1-й шаг	0,010	← 1-й сдвиг		
	+ 1,010	2-е сложение		
	1,100	2-й остаток		
2-й шаг	1,000	← 2-й сдвиг		
	+ 0,110	3-е сложение		
	1,110	3-й остаток		
3-й шаг	1,100	← 3-й сдвиг		
	+ 0,110	4-е сложение		
	0,010	4-й остаток		
4-й шаг	0,100	← 4-й сдвиг		
	+ 1,010	5-е сложение		
	1,110	5-й остаток		

3. Определяются значащие цифры частного обычным путем. Присваиваем знак, производим округление результата и переводим его из обратного в дополнительный код.

$[Y]_0 = 1,0010$

$+ \quad 1$

$[Y]_д = 1,0011 \qquad [Y]_{дОкр.} = 1,001.$

Пример: A > 0; B < 0; Y < 0. Заданы: $[A]_д = 0,101$; $[B]_д = 1,010$.

1. Знак частного $0 \oplus 1 = 1$.

   Модуль делителя $|B| = 0,110$.

2. Псевдознаковая цифра частного равна обратной величине знака нулевого остатка, так как делимое положительное

$$0,101+1,010=1,111.$$

Псевдознак частного равен нулю.

	1,111	1-й остаток	0,0 0 1 0	$= [Y]_0$
1-й шаг	1,110	← 1-й сдвиг		
	+ 0,110	2-е сложение		
	0,100	2-й остаток		
2-й шаг	1,000	← 2-й сдвиг		
	+ 1,010	3-е сложение		
	0,010	3-й остаток		
3-й шаг	0,100	← 3-й сдвиг		
	+ 1,010	4-е сложение		
	1,110	4-й остаток		

4-й шаг	1,100	←	4-й сдвиг
	+ 0,110		5-е сложение
	0,010		5-й остаток

3. Значащие цифры частного определяются знаками остатков, так как делитель отрицателен.

Производим округление частного и перевод его в дополнительный код:

$[Y]_0 = 1,0010$

$+ \quad 1$

$[Y]_д = 1,0011 \qquad [Y]_{дОкр.} = 1,001.$

Пример: A < 0; B < 0; Y > 0. Заданы: $[A]_д = 1,011$; $[B]_д = 1,010$.

1. Знак частного $1 \oplus 1 = 0$.

Модуль делителя |B| = 0,110.

2. Псевдознак модуля частного равен знаку нулевого остатка, так как делимое отрицательное.

1,011+0,110 = 0,001.

Псевдознак частного равен нулю.

	0,001		1-й остаток	0,1 1 0 1 = $[Y]_0$
1-й шаг	0,010	←	1-й сдвиг	
	+ 1,010		2-е сложение	
	1,100		2-й остаток	
2-й шаг	1,000	←	2-й сдвиг	
	+ 1,110		3-е сложение	
	1,110		3-й остаток	
3-й шаг	1,100	←	3-й сдвиг	
	+ 0,110		4-е сложение	
	0,010		4-й остаток	
4-й шаг	0,100	←	4-й сдвиг	
	+ 1,010		5-е сложение	
	1,110		5-й остаток	

Округление частного: $[У]_д = 0,111$.

Таким образом, алгоритм деления чисел в дополнительных кодах сводится к следующему. На каждом шаге деления производится алгебраическое сложение кода текущего остатка (на нулевом шаге делимого) и кода делителя, которому присваивается знак, противоположный знаку текущего остатка (делимого). При этом, если знаки вновь полученного остатка и делителя одинаковы, то в текущий разряд частного записывается единица, в противном случае – ноль. После этого

вновь полученный остаток и частное сдвигаются на один разряд влево. Деление прекращается после вычисления (n+1)-го разряда частного, в который прибавляется единица округления. При таком образе действий и условии, что |A| < |B|, знак частного получается автоматически, а код модуля частного соответствует его знаку.

4.11. Способы ускоренного деления

Необходимость ускорения деления следует из наличия весьма эффективных методов ускорения операций умножения и алгебраического сложения. Способы ускоренного деления делятся на две группы: для первой группы в каждом цикле формируется одна или несколько цифр частного и новый остаток; вторая группа способов предполагает выполнение деления через умножение или с использованием другой процедуры.

Суть одного из способов ускоренного деления первой группы состоит в том, что в частное можно сразу записать последовательность цифр (нулей или единиц), если в результате очередного шага деления получен остаток по абсолютной величине либо достаточно малый, либо близкий делителю. В первом случае, если остаток имеет k нулевых старших разрядов, то для определения очередных цифр частного нет нужды вычитать делитель k раз из остатка. Необходимо в k очередных разрядов частного сразу записать нули, сдвинуть остаток на (k+1) разряд влево, прибавить к нему алгебраически делитель и продолжать операцию деления.

Пример: Будем считать для простоты, что операнды заданы в виде целых чисел: A ~ 1010, B =1101.

```
 1010     | 1101
-1101     | 0,110001
 01110      ⌊_⌋
-1101       k
 00010000
-  ⌊_⌋
     k  1101
        0011
```

Если на каком-то шаге деления получен близкий делителю по абсолютной величине остаток R_i (оба имеют k одинаковых старших цифр), то можно ускорить деление следующим образом. Вместо k вычитаний и k сдвигов произвести вначале вычитание делителя B из остатка R_i затем вычисленную разрядность сдвинуть на k разрядов

влево, после чего к полученному числу $2^k(R_{2i} - B)$ прибавить делитель. При этом в частное записывается k единиц. Этот алгоритм позволяет уменьшить число алгебраических сложений на k–2.

Правомерность подобного изменения алгоритма деления основывается на том факте, что остаток после выполнения k шагов обычного алгоритма деления при совпадении k старших цифр остатка R_i и делителя B определится как

$$R_{i+k} = 2(...2(2R_i - B) - B...) - B = 2^kR_i - B(2^{k-1} + 2^{k-1} + ... + 2 + 1) = 2^k(R_i - B) + B,$$

т.е. значение остатка R_{i+k}, полученного по видоизмененному алгоритму, совпадает с величиной остатка R_{i+k}, который дает исходный алгоритм.

Пример: Заданы A = 1000, B = 1001, выполнить: а) обычное, б) ускоренное деление

```
а)  1000  | 1001
  - 1001  | 0,1110
    01110   i—j k
  -  1001
    01010
   -
      1001
      0001
```

```
    ⌐¬ k   | ⌐¬ k
    1000   | 1001
  - 1001   | 0,111000
    1111
    1000     Сдвиг на k=3 разр.
   +1001
    0001
```

Рассмотренный способ ускоряет операцию деления только в тех случаях, когда в остатках получаются последовательности нулей или остаток близок по величине делителю. В противном случае операция выполняется по обычному неускоренному алгоритму.

Развитием первого способа ускорения деления является способ, по которому всегда одновременно определяются две цифры частного.

Если A – делимое, B – делитель, то на основании 4.22 для остатка R_{i-1} справедливо:

$$0 \le R_{i-1} < B.$$

Если теперь R_{i-1} сдвинуть на два разряда влево, т.е. сформировать $R_{i-1} \cdot 2^2$, то следующую пару цифр частного и остаток R_i можно найти из условий, приведенных в таблице 4.5.

Таблица 4.5.

Условие для анализа	Пара цифр частного	R_i

$2^2 \cdot R_{i-1} < B$	00	$4R_{i-1}$
$B \le 2^2 \cdot R_{i-1} < 2B$	01	$4R_{i-1} - B$
$2B \le 2^2 \cdot R_{i-1} < 3B$	10	$4R_{i-1} - 2B$
$3B \le 2^2 \cdot R_{i-1}$	11	$4R_{i-1} - 3B$

Проверку всех четырех условий, в принципе, можно выполнять одновременно. Однако для этого требуется сравнительно большой объем оборудования: цепи сдвига для передачи B и 2B прямым и инверсным кодом, регистр для формирования и хранения устроенного делителя 3B, цепи для передачи 3B прямым и инверсным кодом и три сумматора.

Можно уменьшить объем оборудования, если учесть; что в большинстве случаев вывод о принадлежности $2^2 \cdot R_{i-1}$ к тому или иному из указанных в таблице диапазонов можно сделать путем анализа не всего делителя B и сдвинутого остатка $R_{i-1} \cdot 2^2$, а только нескольких их старших разрядов. Поэтому дальнейшим развитием изложенного алгоритма является определение каждой пары цифр частного исходя из анализа нескольких старших разрядов делителя B и сдвинутого остатка $R_{i-1} \cdot 2^2$. При этом, если старшие разряды остатка $R_{i-1} \cdot 2^2$ совпадают со старшими разрядами делителя, предполагается, что $2^2 \cdot R_{i-1}$ принадлежит к области с наибольшим значением пары цифр частного и исходя из этого, находится остаток R_i. Если R_i – отрицательный, это означает, что получена величина остатка, уменьшенная на B:

$$-B \le R'_i = (R_i - B) < 0.$$

Тогда коррекция остатка производится следующим образом. Если раньше проверялось условие

$$KB \le 2^2 \cdot R_{i-1} < B. < (K+1)B, \text{ где } K = 0,\ 1,\ 2,\ 3,$$

то теперь должно проверяться эквивалентное ему условие

$$(K-4)B \le 2^2 \cdot (R_{i-1} - B) < (K-3)B.$$

Окончательные правила проверки условий представлены в таблице 4.6.

Таблица 4.6

$R_{i-1} \ge 0$		$R'_{i-1} = (R_{i-1} - B) < 0$		Пара цифр частного	
Условие для анализа	R_i	Условие для анализа	R_i	$R_{i-1} \ge 0$	$R_{i-1} < 0$

$2^2 \cdot R_{i-1} < B$	$4R_{i-1}$	$2^2 \cdot R'_{i-1} < -3B$	$4\,R'_{i-1} + 4B$	00	$0\bar{1}$
$B \le 2^2 \cdot R_{i-1} < 2B$	$4R_{i-1} - B$	$-3B \le 2^2 \cdot R'_{i-1} < -2B$	$4\,R'_{i-1} + 3B$	01	00
$2B \le 2^2 \cdot R_{i-1} < 3B$	$4R_{i-1} - 2B$	$-2B \le 2^2 \cdot R_{i-1} < -B$	$4\,R'_{i-1} + 2B$	10	01
$3B \le 2^2 \cdot R_{i-1}$	$4R_{i-1} - 3B$	$-B \le 2^2 \cdot R'_{i-1} < 0$	$4\,R'_{i-1} + B$	11	10

В тех случаях, когда в ЭВМ обеспечивается большая скорость выполнения операции умножения, может оказаться целесообразным вместо операции деления применить один из итеративных способов определения частного, основанных на использовании операций только сложения и умножения. В отличие от способов деления, рассмотренных выше, методы использующие умножение, на каждой итерации уточняют приближение, а не определяют очередную цифру частного. Один из наиболее простых способов этой группы сводится к серии итераций, в каждой из которых делимое и делитель умножаются на один и тот же множитель, в результате чего частное не изменяется. Целью этих итераций является преобразование делителя, первоначально равного B, а в конце i-й итерации – B_i, в число, отличающееся от единицы не более, чем на единицу младшего разряда. Если для этого потребуется n итераций, то делимое A, а в конце i-й итерации – A_i по окончании n-й итерации будет с точностью до младшего разряда равно частному [50], т.е.

$$Y_n = A/B.$$

Множитель X для одновременного умножения на него делимого и делителя в каждой итерации можно выбрать, исходя из следующих соображений. Пусть известно, что после завершения i-й итерации выполняется, условие:

$$|B_i - 1| < 2^{-ki} \qquad (4.13)$$

т.е. $B_i = 1 + \varepsilon_i$, где

$$|\,\varepsilon_i\,| < 2^{-ki} \qquad (4.32)$$

Тогда делимое A_i, которое было получено к концу i-й итерации, с точностью до K_i-го разряда после запятой равно частному.

В качестве множителя X_{i+1} для (i+1)-й итерации выбирают

$$X_{i+1} = 2 - B_i,$$

который с учетом (4.31) равен $X_{i+1} = 2 - (1 + \varepsilon_i) = 1 - \varepsilon_i$. В результате умножения получим для делителя в конце (i+1)-й итерации новое значение

$$B_{i+1} = B_i \cdot X_{i+1} = 1 - \varepsilon_i^2 = 1 - \varepsilon_{i+1}. \qquad (4.33)$$

Из (4.32) и (4.33) следует, что $\varepsilon_{i+1} = -\varepsilon_i^2$, т.е.

$$\begin{cases} |\varepsilon_{i+1}| < 2^{-Ki+1}, \\ K_{i+1} = 2K_i. \end{cases} \qquad (4.34)$$

Из (4.34) следует, что после каждой итерации количество точных разрядов частного удваивается. Поэтому достаточно небольшого количества итераций, чтобы все разряды частного стали точными. Таким образом, исходной итерацией для каждой итерации являются преобразованные в предыдущих итерациях делимое A_i и делитель B_i. Содержанием каждой итерации является: 1) определение множителя X_{i+1} как дополнения делителя B_i до 2; 2) умножение делителя B_i на X_{i+1}; 3) умножение делимого A_i на X_{i+1}.

Необходимо отметить, что если в качестве исходного значения делителя выбрать его обычное нормализованное значение ($K_1 = 1$), то при первых итерациях количество верных разрядов растет медленно. Поэтому выбор множителя X_1, участвующего в первой итерации, необходимо осуществлять с помощью специальной таблицы. Входами этой таблицы являются старшие разряды B, а выходом – такое значение кода X_i, которое обеспечивает обращение к концу первой итерации нескольких старших разрядов произведения $B_1 = B \cdot X_1$ в единицы.

Ускорение операции деления может быть также достигнуто путем выполнения устройства для деления в виде итеративной сети, простейшая модификация показана на рис. 4.14 [32].

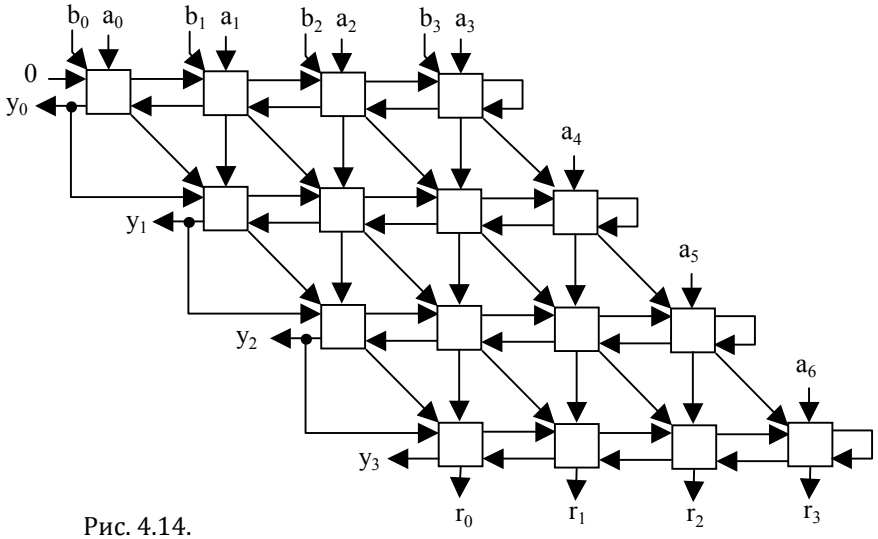


Рис. 4.14.

Каждый элемент сети (рис.4.14) содержит одноразрядный сумматор и элемент "исключающее ИЛИ", реализующий функцию $p\bar{q} \vee \bar{p}q$ и управляющий вводом делителя B в сумматор (рис.4.15).

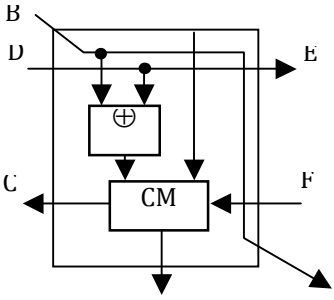


Рис.4.15

Шестиразрядное делимое A = $a_0a_1a_2a_3a_4a_5a_6$, трехразрядный делитель B = $b_0b_1b_2b_3b_4$ и каждый остаток R_i (последний остаток R4 = $r_0r_1r_2r_3$) представлены в дополнительном коде. Предполагается, что делимое и делитель являются положительными нормализованными дробями. Поэтому частное Y = $y_0y_1y_2y_3$ положительное число, лежащее в диапазоне ½ < Q < 2, т.е. y_0 – целая часть числа. Каждая цифра частного y является управляющим сигналом для следующей строки матрицы, определяющим, какую операцию (сложение или вычитание) нужно выполнять в этой строке. Вычитание, как видно из рис. 4.14 и рис. 4.15 выполняется путем формирования дополнительного кода делителя.

4.12 Деление чисел в машинах с плавающей запятой

Если числа A и B заданы в нормальной форме, to их частное будет равно:

$$Y = A/B = (a/b)\cdot 2^{m_1-m_2},$$

где a и b – мантиссы, а m_1 и m_2 – порядки, соответственно, чисел A и B. Отсюда следует, что операция деления в машинах с плавающей запятой выполняется в пять этапов.

1-й этап. Определение знака частного путем сложения по модулю 2 знаковых цифр мантисс операндов.

2-й этап. Деление модулей мантисс операндов по правилам деления чисел с фиксированной запятой.

3-й этап. Определение порядка частного путем вычитания порядка делителя из порядка делимого.

4-й этап. Нормализация результата и его округление.

5-й этап. Присвоение знака мантиссе результата.

Два первых этапа полностью совпадают с этапами деления чисел с фиксированной запятой. Третий этап представляет собой обычное сложение в инверсных кодах.

При делении нормализованных чисел денормализация результата возможна только влево и только на один разряд. Это обусловлено тем, что мантисса любого нормализованного числа лежит в пределах:

$$2^{-1} \le a_i < 1 - 2^{-n}.$$

Тогда наименьшая и наибольшая возможная величина мантиссы частного равна, соответственно

$$Y_{min} = \frac{2^{-i}}{1 - 2^{-n}}; \qquad Y_{max} = \frac{1 - 2^{-n}}{2^{-i}} = (2 - 2^{-(n-1)}) < 2,$$

т.е. мантисса частного, лежит в пределах .

$$2^{-1} < Y < 2.$$

Поэтому на четвертом этапе может возникнуть необходимость нормализации мантиссы частного путем ее сдвига вправо на один разряд и увеличения порядка частного на единицу. Если же перед делением сдвинуть долимое на один разряд вправо, то на 4-м этапе может потребоваться нормализация результата влево на 1 разряд.

Пример: Заданы $[A]_{пр} = 1\underbrace{110}_{m_a}\underbrace{1\,101}_{a}$; $[B]_{пр} = 1\underbrace{101}_{m_b}\underbrace{0\,110}_{b}$

1 этап. Определение знака частного: $1 \oplus 0 = 1$.

2-й этап. Деление мантисс: $|y| = |a|/|b| \approx 0{,}1101$.

3-й этап. Определение порядка частного: $[m_a]^м_д = 11{,}010$

$$\underline{+[m_b]^м_д = 00{,}101}$$

$$[m_y]^м_д = 11{,}111$$

Перевод порядка частного в прямой код:

$$[m_y]_{пр} = 1{,}001.$$

4-й этап. Частное получилось нормализованным, поэтому производится только округление мантиссы.

5-й этап. Результат $[y]_{пр}$ A/B = 1 001 1 111.

При выполнении третьего этапа порядок частного может оказаться больше допустимого, т.е. произойдет переполнение разрядной сетки, которое расценивается как аварийная ситуация. Может быть также получен порядок частного, который меньше допустимого. Если на

втором этапе вычислено частное |y| > 1, то на четвертом этапе при сдвиге частного вправо его порядок должен быть увеличен на единицу. При этом становится окончательно ясно, во-первых, возникло ли переполнение разрядной сетки порядка или нет, во-вторых, будет ли порядок частного меньше допустимого значения. Если имеет место исчезновение порядка, то результат деления по указанию программиста может быть заменен машинным нулем.

Контрольные вопросы к главе 4

1. Обоснуйте и подтвердите примерами правильность всех четырех способов умножения методом накопления.
2. Чем обусловлены и чему равны ошибки, возникающие при сокращенном умножении? Как округляются ошибки?
3. Какие логические методы ускорения Вы знаете? В чем они состоят?
4. Обоснуйте правила умножения чисел на 2 разряда множителя, начиная с младшего и со старшего разрядов.
5. Сколько знаковых разрядов должен иметь сумматор при умножении чисел на 2 разряда и почему?
6. Что такое матричный метод умножения?
7. В чем состоит суть быстрого умножения чисел большой разрядности?
8. Какие коррекции результатов требуются при умножении чисел разных знаков в дополнительных кодах и почему?
9. Обоснуйте корректность результатов умножения дополнительных кодов чисел с разными знаками, выполняемого путем анализа двух соседних разрядов множителя.
10. В чем состоят особенности умножения чисел в машине с плавающей запятой?
11. Обоснуйте и сформулируйте алгоритм деления чисел с восстановлением остатков.
12. Обоснуйте и сформулируйте алгоритм деления чисел без восстановления остатков.
13. Какая из машинных схем деления предпочтительнее и почему?
14. Обоснуйте и сформулируйте алгоритм деления чисел в дополнитель ном коде.
15. Какие логические методы ускорения Вы знаете и в чем они состоят?
16. Как можно выполнить деление чисел через умножение?
17. Какие аппаратные методы деления Вы знаете?
18. Как делятся числа в машине *с* плавающей запятой?

Глава 5 НЕОСНОВНЫЕ АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

Основными арифметическими операциями ЭВМ являются алгебраическое сложение, умножение и деление. Время выполнения этих операций, в основном, определяет производительность последовательных машин. Однако в ходе решения задачи очень часто необходимо выполнять другие, так называемые, неосновные операции, такие как извлечение квадратного корня, вычисление тригонометрических и других элементарных функций. Неосновные арифметические операции реализуются обычно с помощью стандартных программ, которые входят в состав математического обеспечения (МО) и вызываются простым обращением к соответствующей библиотеке подпрограмм. Однако для реализации этих подпрограмм требуется значительно больше времени, чем для выполнения основных арифметических операций.

Вместе с тем, в настоящее время существенно повысилась степень интеграции и снизилась стоимость электронных компонентов ЭВМ. В связи с этим с целью повышения производительности ЭВМ актуальным стал вопрос о передаче части функций МО машин аппаратным средствам. В первую очередь, это касается выполнения сложных арифметических операций, таких, как перевод из одной системы счисления в другую, вычисление численного значения многочлена, вычисление элементарных функций, выполнение операций комплексной арифметики. При этом интерес представляет исследование таких методов вычисления указанных операций, которые допускают аппаратную реализацию за время, соизмеримое со временем выполнения основных операций.

Обычно алгоритмы выполнения неосновных арифметических операций, ориентированные на аппаратную реализацию, существенно отличаются от алгоритмов программной реализации, которые строятся, как правило, на основе многократного выполнения операций сложения-вычитания и сдвигов. Рассмотрение алгоритмов неосновных арифметических операций начнем с операции извлечения корня квадратного.

5.1. Операция извлечения квадратного корня

Эта операция включается как команда в систему команд ЭВМ в том случае, когда она составляет не менее 2% от общего числа операций или является составной частью алгоритмов, которые необходимо выполнять в реальном времени, т.е. с высоким быстродействием.

Есть два пути решения й задачи. Первый путь связан с разработкой программы извлечения квадратного корня с использованием набора арифметических команд. При этом программа реализует итерационный метод извлечения квадратного корня. Например, в универсальных ЭВМ для вычисления квадратного корня применяется формула Ньютона:

$$B_{i+1} = 0{,}5\left(B_i + \frac{A}{B_i}\right),$$

где B_{i+1} есть (i+1)-е приближение $B = \sqrt{A}$, а i = 0, 1, 2....

Другой путь состоит в создании алгоритма извлечения квадратного корня, похожего по структуре на алгоритмы основных арифметических операций, например, деления.

Наиболее простой алгоритм сводится к подбору цифр в результате, начиная со старшего, имеющего вес 2^{-1}. При этом вычисление i-ой цифры b_i происходит следующим образом. После получения (i+1)-й цифры b_{i-1} в i-й разряд B для пробы помещается "1". Вычисляется разность $(A - B_i^2) = R_i$. Если $R_i > 0$, то b_i есть число, у которого цифры всех i разрядов совпадают с цифрами искомого результата B. Если $R_i < 0$, то в i-м разряде b_i нужно поставить "О" и переходить к вычислению (i + 1)-го разряда.

Так как вычисление этого разряда снова начнется с подстановки пробной "1", то в случае $(A - B_i^2) = R_i < O$ можно вместо "стирания" "1" в i-м разряде вычесть "1" из (i+1)-го разряда.

Подобным образом можно вычислять $\sqrt[3]{A}$, $\sqrt[4]{A}$ и т.д.

Однако определение остатка $R_i = (A - B_i^2)$ или хотя бы его знака является сложной процедурой, выполнение которой нежелательно. Поэтому рассмотрим возможность получения R_i из R_{i-1}. Для этого допустим, что найдены первые (i–1) цифры $B = \sqrt{A}$, т.е. получено $B_i = 0, b_1, b_2, \ldots b_i$. Очевидно, B_{i-1} – наибольшее число из (i–1)-разрядов, квадрат которого не превосходит A, т.е. остаток $R_{i-1} = (A - B_{i-1}^2) > 0$. Следующая i-я цифра b_i может быть равна 0 или 1. При этом, если $R_i \geq 0$, то $b_i = 1$ и $B_i = 0, b_1, b_2, \ldots b_{i-1}1$, если же $R_{i-1} = (A - B_{i-1}^2) < 0$, то $b_i = 1$ и $B_i = 0, b_1, b_2, \ldots b_{i-1}0$.

Остаток R_i при $b_i = 1$ можно получить следующим образом:

$$R_i = A - (0, b_1, b_2, \ldots, b_i)^2 = A - (B_{i-1} + 2^{-i})^2 =$$
$$(A - B_{i-1}^2) - (2 \cdot 2^{-i} \cdot B_{i-1} + 2^{-2i}) = R_{i-1} - 2^{-(i-1)}(B_{i-1} + 2^{-(i+1)}) =$$
$$= R_i - 2^{-(i+1)}(0, b_1, b_2, \ldots, b_{i-1}01).$$

Следующая цифра b_{i+1} результата определяется таким же образом, если $R_i \geq 0$. Однако, если $R_i < 0$, то прежде чем вычислять следующую цифру корня необходимо восстановить остаток, т.е. получить

$$R'_i = R_i + 2^{-(i+1)}(0, b_1,b_2,\ldots,b_{i-1}01) = R_{i-1}.$$

После этого в следующем такте необходимо вычислить новый остаток R_i

$$R_{i+1} = 2^{-i}(0, b_1,b_2,\ldots,b_{i-1}001) = R_{i-1} - 2^{-i}(B_{i-1} + 2^{-(i+1)}).$$

Новая (i+1)-я цифра результата равна инверсному значению знаковой цифры остатка R_{i+1}. Таким образом, чтобы получить остаток R_i нужно к (i+1)-му результату приписать справа пару цифр 01, сдвинуть его на (i–1) разрядов вправо и вычесть из предыдущего остатка R_{i-1}. При этом если $Ri \geq 0$, то цифра $b_i = 1$, если $R_i < 0$, то $b_i = 0$. В последнем случае необходимо восстановить предыдущий остаток и приступать затем к определению следующего остатка.

Такой способ получения остатка проще, чем вычисление разности $R_i = (A - B_i^2)$. Он лучше поддается автоматизации и может быть реализован в виде самостоятельной операции. Вычисление корня подобно вычислению частного. При этом роль делителя, постоянного в процессе деления, выполняет переменный делитель B_i. Переменный делитель сдвигается на один разряд вправо в каждом цикле. Как и при делении вместо сдвига делителя вправо можно сдвинуть остаток влево.

Пример: Задано A = 0,10111, найти B = $\sqrt{A}$ с точностью до 5-и знаков после запятой

0,10111		B_i
− 0,01		
0,01111		0,1
0,11110	← Сдвиг	
− 0,101		
0,01010		0,11
0,10100	← Сдвиг	
− 0,1101		
1,11010		
+ 0,1101	Восстановление	
0,10100		0,110
1,01000	← Сдвиг	
− 0,11001		
0,01111		0,1101
0,11110	← Сдвиг	
− 0,110101		
0,000111		B = $\sqrt{A}$ =0,11011

Как и при делении, можно отказаться от восстановления остатка. В этом случае если остаток R_i получился отрицательным, то в i-м разряде результата записывается $b_i = 0$, а в следующем цикле добавляется к B_i не 01, а 11 и вычитание заменяется сложением, т.е. выполняются следующие действия:

$$R_{i+1} = R_i + 2^{-i}(0, b_1,b_2,\ldots,b_{i-1}011)^2 = R_{i-1} - 2^{-(i-1)}\left(B_{i-1} + 2^{-(i+1)}\right) +$$
$$+ 2^{-i}\left(B_{i-1} + 2^{-(i+2)}\right) = R_{i-1} - 2^{-i}B_{i-1}(2-1) - 2^{-i}\left(2^{-(i+1)} - 2^{-(i+2)}\right) =$$
$$= R_{i-1} - 2^{-i}\left(B_{i-1} + 2^{-(i+2)}\right).$$

Таким образом, сразу получается правильный остаток R_{i+1}, операция вычисления корня квадратного становится регулярной и подобна операции деления без восстановления остатка.

Пример: Вычислить $B = \sqrt{A = 0{,}10111}$ с точностью до пяти знаков после запятой.

0,10111		B_i
− 0,01		
0,01111		0,1
0,11110	← Сдвиг	
− 0,101		
0,01010		0,11
0,10100	← Сдвиг	
− 0,1101		
1,11010		0,110
1,10100	← Сдвиг	
+ 0,11011		
0,01111		0,1101
0,11110	← Сдвиг	
− 0,110101		
0,000111		$B = \sqrt{A} = 0{,}11011$

Результат вычислений всегда получается с недостатком, поэтому желательно его округление. Для этого необходимо определить (n+1)-й разряд корня и прибавить к нему единицу.

Таким образом, процесс вычисления квадратного корня состоит из ряда однотипных циклов, выполняемых последовательно, и шага округления. В каждом цикле находится очередная цифра корня. Ее значение определяет знак результата алгебраического сложения остатка с переменным "делителем", которому приписывается знак, противоположный знаку предыдущего остатка. В первом цикле остаток – это подкоренное число.

Если новый остаток положительный, то в регистр результата операции заносится "1" и формируется норое значение переменного "делителя" путем приписывания к результату операции справа пары цифр "01" (начальный результат равен 0). Затем остаток сдвигается на один разряд влево и выполняется следующий цикл. Если же новый остаток отрицательный, то в регистр результата операции заносится "0", а новое значение "делителя" формируется путем приписывания к результату справа пары цифр "11". После чего текущий остаток сдвигается влево на один разряд и выполняется следующий шаг алгоритма.

Для извлечения корня квадратного из числа с плавающей запятой необходимо порядок числа разделить на два, а из мантиссы извлечь корень по правилам для чисел с фиксированной запятой. Действительно, число с плавающей запятой имеет вид:

$$A = a \cdot 2^m, \text{ откуда } B = \sqrt{A} = \sqrt{a} \cdot 2^{m/2}.$$

Для деления порядка на два его необходимо сдвинуть на один разряд вправо, если он четный. Вели же он нечетный, то необходимо прибавить к порядку единицу, затем сдвинуть порядок и мантиссу на один разряд вправо и вычислить корень. Следовательно, если $m = 2K - 1$, то

$$A = 2^{2K-1} \cdot a = 2^{2K}(a \cdot 2^{-1}) \quad \text{и} \quad B = \sqrt{A} = 2^K \sqrt{a \cdot 2^{-1}}.$$

Так как мантиссы всегда нормализованы и в первом цикле из нее вычитается число 0,01, то первый остаток будет всегда положительным, т.е. первая цифра результата всегда будет единица. Следовательно, при выполнении операции извлечения корня квадратного в машине с плавающей запятой результат всегда получается нормализованным.

Перед началом операции знак операнда и его величина анализируются на равенство нулю. При нулевой мантиссе операция не выполняется, а результату присваивается сразу значение нуль. Если знак операнда отрицательный, то ситуация рассматривается как аварийная, так как извлечение корня квадратного недопустимо из отрицательных чисел.

5.2. Вычисление сумм парных произведений

К получению сумм парных произведений приходится обращаться, например, при решении задач линейной алгебры. Оно сводится к вычислению выражения вида

$$\sum_{i=1}^{m} A_i B_i = A_1 B_1 + A_2 B_2 + \ldots + A_m B_m \qquad (5.1)$$

Трудоемкость данной операции характеризует величина m, которая в зависимости от вида решаемой задачи может меняться в широких пределах и принимать очень большие значения. В ЭВМ эта операция обычно выполняется программно путем вычисления отдельных произведений и последующего их накопления, что требует значительных затрат машинного времени. Длительность операции можно сократить с помощью дополнительных аппаратных средств. Для конкретности далее считаем, что числа A_i и B_i заданы в форме с запятой, фиксированной перед старшим разрядом.

В простейшем случае, для решения поставленной задачи можно применить m множительных устройств, т.е. столько, сколько необходимо вычислять произведений вида A_iB_i, а затем сложить результаты с помощью дерева, состоящего из (m−1) сумматоров. При этом иногда с целью повышения быстродействия для вычисления произведений A_iB_i применяют матричные множительные устройства.

Для уменьшения аппаратурных затрат при соответствующем росте времени выполнения операции поступают следующим образом. В выражении (5.1) одну из переменных, например Bi, записывают в виде полинома по степеням 2:

$$\sum_{i=1}^{m} A_iB_i = \sum_{i=1}^{m} A_i \times \sum_{i=1}^{m} 2^{-k} b_{i,k} . \qquad (5.2)$$

Изменяя порядок суммирования, получаем:

$$\sum_{i=1}^{m} A_iB_i = \sum_{i=1}^{m} 2^{-k} \times \sum_{i=1}^{m} A_ib_{i,k} . \qquad (5.3)$$

Ввиду того, что произведение $A_ib_{i,k}$ предполагает умножение на одну двоичную цифру, то арифметическое умножение в $A_ib_{i,k}$ может быть заменено логическим, тогда

$$\sum_{i=1}^{m} A_iB_i = \sum_{i=1}^{m} 2^{-k} \times \sum_{i=1}^{m} A_i\&b_{i,k} . \qquad (5.4)$$

Устройство, реализующее выражение (5.4); представляет собой дерево, состоящее на m сумматоров. Входные листья дерева содержат m/2 сумматоров, своими входами соединенных с m группами схем "И", которые выполняют логическое умножение. Корнем дерева является накопительный сумматор со сдвигом вправо, если умножение выполняется по 1-й схеме. Для получения суммы парных произведений потребу-

ется выполнить n циклов накоплений и сдвигов, где n – количество разрядов сомножителей.

Для небольших значений m можно уменьшить необходимые аппаратурные затраты, при определенном росте временных, путем предварительного формирования таблицы D(s) всех возможных сумм частных произведений. При этом учитывают, что для фиксированного k во вторую сумму выражения (5.4) будут входить только те члены $A_i \& b_{i,k}$, для которых $b_{i,k} = 1$. Следовательно, каждому номеру сочетаний из C_m^j, где j – количество единиц в k-х разрядах $b_{i,k}$ и фиксированному k будет соответствовать сумма [71]

$$S(N_k) = \sum_{i \to b_{i,k}} A_i = S(b_{i,k}).$$

Очевидно, что количество этих номеров N_k (т.е. количество возможных разрядных срезов) и, следовательно, количество сумм $S(N_k)$, которые составят некоторую область D(S), будет равно

$$D(S) = C_m^0 + C_m^1 + \ldots + C_m^j + \ldots + C_m^m = 2^m.$$

Окончательно получим

$$S(N_k) = S(b_{i,k}) = \sum_{i \to b_{i,k}} A_i,$$

$$\sum_{i=1}^{m} A_i B_i = \sum_{i=1}^{m} 2^{-k} \times \sum_{i \to b_{i,k}} A_i = \sum_{i=1}^{m} 2^{-k} \times S(b_{i,k}).$$

На основании двух последних выражений можно сформулировать следующий алгоритм вычисления выражения (5.1) [71]:

1. Предварительно вычисляется область всевозможных сумм D(S), количество которых равно числу всех возможных разрядных срезов множителей B_i, т.е. 2^m.

2. Область D(S) упорядочивается в соответствии с числами $b_{i,k}$ в виде таблицы так, чтобы число $b_{i,k}$ являлось номером (адресом) суммы $S(b_{i,k})$. Возможность указанной процедуры следует из (5.5), так как в данном случае это логическая операция.

3. Далее, приняв k = 1, определяется число (разрядный срез) $b_{i,k}$, по которому находится $S(b_{i,k})$.

4. $S(b_{i,k})$ умножается на 2^{-1} (т.е. сдвигается вправо на один разряд), если умножение лроизводится по первой схеме, а k изменяется на 1, т.е. k = k + 1. Это достигается тем, что все множители B_i сдвигаются на один разряд вправо (при первой схеме).

5. Соответственно, новым разрядам среза $b_{i,k}$ из таблицы D(S) находится новая сумма $S(b_{i+1,k})$, которая прибавляется к предыдущей сумме

$2^{-1}S(b_{i,k})$, а результат снова умножается на 2^{-1}. После этого пункт 5 повторяется до тех пор, пока k не станет равным n, т.е. пока не накопится сумма, равная $\sum_{i=1}^{m} A_i B_i$.

В том случае, когда числа представлены в дополнительных кодах, вычисление области D(S) не представляет трудностей, так как здесь по-прежнему сохраняются обычные правила сложения и вычитания. Например, если $B_i = b_{i,3н} B'_i$ представлены в дополнительных кодах, то выражение (5.1) с учетом выражения (4.17) можно записать следующим образом:

$$\sum_{i=1}^{m} A_i B_i = -\sum_{i=1}^{m} A_i \times 2^0 \, b_{i,3н} + \sum_{i=1}^{m} A_i B'_i = -\sum_{i=1}^{m} A_i \times 2^0 \, b_{i,3н} + \sum_{i=1}^{m} A_i \times \sum_{i=1}^{m} 2^{-k} b_{i,k} \ ,$$

то есть в этом случае имеет место коррекция результата вычислений, как и при обычном умножении, но только эта операция групповая.

Критерием оценки эффективности рассмотренного алгоритма является количество сложений, которое необходимо произвести, вычисляя сумму парных произведений обычным способом и по МВТ. Количество сложений, необходимых для вычисления этой зависимости обычным способом, равно

$$M_1 = m \cdot n + m - 1.$$

Количество сложений при МВТ составит

$$M_2 = 2^m - m - 1 + n,$$

так как из всех 2^m сочетаний m комбинаций – это множимые A_i и одна комбинация нулевая, которые вычислять не надо. Тогда выигрыш составит (при наличии быстрой памяти для таблицы)

$$K = M_1 / M_2 = \frac{m \cdot n + m - 1}{2^m - m - 1 + n}.$$

Выигрыш K является функцией количества разрядов n и количества произведений m. Принимая K = 1, получим зависимость, при которой оба алгоритма равноценны. Уравнение границы между выигрышем и проигрышем следующее:

$$1 = \frac{m \cdot n + m - 1}{2^m - m - 1 + n}.$$

Решая это уравнение относительно n, получим

$$n = \frac{2^m - 2m}{m - 1}. \qquad (5.6)$$

Из (5.6) явную зависимость m = F(n) получить невозможно. Поэтому на рис. 5.1 приведена графическая зависимость, построенная по (5.6), из которой следует, что область выигрыша в вычислениях невелика и находится между кривой m = F(n) и осью n, а также, что для n = 32 выигрыш возможен при величине m, лежащей в диапазоне $2 \le m \le 7$.

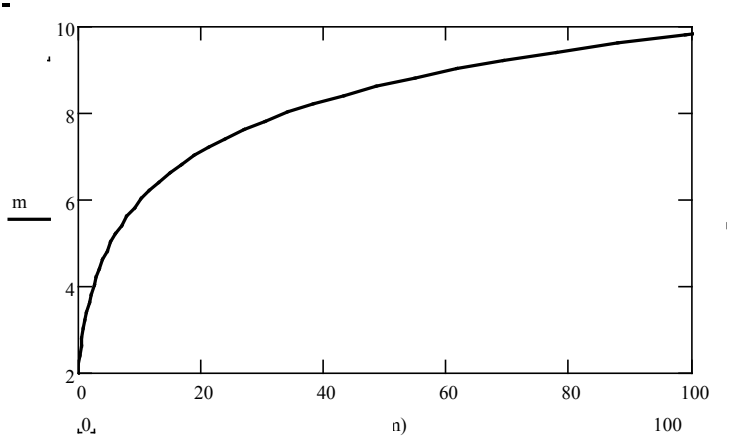


Рис. 5.1. Зависимость числа парных произведений от разрядности сомножителей.

Следует отметить, что если значения таблицы не вычислять заранее, а формировать в процессе накопления сумм парных произведений, то придем к способу вычислений, описанному выше, и тогда при любом m время вычислений составит n тактов. Однако объем оборудования при этом существенно возрастает, так как для реализации вычислений, как уже отмечалось, потребуется дополнительно дерево, состоящее из m сумматоров, и m групп схем "И", каждая из которых управляется одноименными разрядами соответствующего множителя B_i.

5.3. Арифметика комплексных чисел

В последнее время существенно возрос интерес к теории функций комплексного переменного, особенно в связи с выделением теории цифровой обработки сигналов в самостоятельную дисциплину. В рамках этой теории возможно не только выполнение задач спектрального

анализа, но и решение дифференциальных и интегральных уравнений при помощи преобразования Лапласа или Фурье, исследование систем автоматического управления, расчет электрических цепей и т.п. Алгоритмы решения задач перечисленных классов требуют выполнения действий над комплексными числами.

В ЭВМ комплексное число может быть представлено минимум парой вещественных чисел. Этому удовлетворяют три формы: алгебраическая, полярная и показательная. Предпочтение отдают обычно алгебраической форме, т.е. представлению комплексного числа A в виде

$$A = ReA+jImA,$$

где ReA и ImA – вещественные числа, а j – мнимая единица, удовлетворяющая условию $j^2 = -1$. В этой форме просто выполняется наиболее часто встречающаяся операция алгебраического сложения комплексных чисел A и B как две операции сложения в инверсных кодах соответствующих действительных и мнимых частей этих чисел:

$$A+B = (ReA+ReB) + j(ImA+ImB).$$

Как известно, произведение комплексных чисел вычисляется в соответствии с выражением

$$A\cdot B = (ReA\cdot ReB - ImA\cdot ImB) + j(ReA\cdot ImB + ReB\cdot ImA),$$

для реализации которого требуется выполнить 4 операции умножения и две операции алгебраического сложения вещественных чисел. Вместе с тем, это произведение можно записать, представив мантиссу одного из сомножителей, например B, по степеням основания 2

$$A\cdot B = ReA\cdot\sum_{i=1}^{m}2^{-i}Reb_i - ImA\cdot\sum_{i=1}^{m}2^{-i}Imb_i + j\ ReA\cdot\sum_{i=1}^{m}2^{-i}Imb_i + j\ ImA\cdot\sum_{i=1}^{m}2^{-i}Reb_i.$$

Изменив порядок суммирования, можно записать

$$A\cdot B = \sum_{i=1}^{m}2^{-i}(ReA\cdot Reb_i - ImA\cdot Imb_i) + j\sum_{i=1}^{m}2^{-i}(ReA\cdot Imb_i + ImA\cdot Reb_i). \qquad (5.3)$$

Из выражения (5.7) видно, что выражения в скобках, в зависимости от сочетания цифр Reb_i и Imb_i будут принимать следующие значения (табл. 5.1).

Таблица 5.1

Reb_i	Imb_i	$ReA \cdot Reb_i - ImA \cdot Imb_i$	$ReA \cdot Imb_i + ImA \cdot Reb_i$
0	0	0	0
0	1	–ImA	ReA
1	0	ReA	ImA
1	1	ReA – ImA	ReA + ImA

Вычисляют предварительно указанную таблицу. Затем анализируя последовательно друг за другом одноименные разряды действительной и мнимой частей комплексного числа B, по сочетанию цифр в этих разрядах из таблицы определяют соответствующие им значения. Далее эти значения складывают с соответственно сдвинутыми суммами частных произведений, в результате чего формируется полное произведение комплексных чисел.

При умножении комплексных чисел, представленных дополнительными кодами, выполняется коррекция результата, как и при умножении вещественных чисел. При этом в случае произвольных знаков чисел ReB и ImB выражение (5.7) записывается в виде

$$A \cdot B = \sum_{i=1}^{m} 2^{-i} (ReA \cdot Reb_i - ImA \cdot Imb_i) + j \sum_{i=1}^{m} 2^{-i} (ReA \cdot Imb_i + ImA \cdot Reb_i) - \\ - \left[(ReA \cdot Reb_0 - ImA \cdot Imb_0) + j(ReA \cdot Imb_0 + ImA \cdot Reb_0) \right].$$

В качестве критерия оценки эффективности описанного алгоритма примем количество сложений, которое необходимо для вычисления произведения комплексных чисел обычным способом M_1 и по МВТ – M_2

Экономия вычислений составляет [71]

$$Э = M_1 - M_2 = 2n,$$

а коэффициент выигрыша

$$K = M_1/M_2 = \frac{4n+2}{2n+2} \approx 2.$$

Обычно деление комплексных чисел записывается следующим образом:

$$\frac{A}{B} = \frac{ReA + jImA}{ReB + jImB} = \frac{(ReA + jImA)\,(ReB - jImB)}{(ReB + jImB)\,(ReB - jImB)} = \\ = \frac{ReA \cdot ReB + ImA \cdot ImB}{ReB^2 + ImB^2} + j\frac{ReA \cdot ImB + ImA \cdot ReB}{ReB^2 + ImB^2}.$$

Используя МВТ, а также учитывая тот факт, что число $B^* = ReB - jImB$ содержится как в знаменателе, так и в числителе, таблицу 5.2 удобно считать для B^*

Таблица 5.2.

Reb_i	Imb_i	$ReB \cdot Reb_i + lmB \cdot Imb_i$
0	0	0
0	1	ImB
1	0	ReB
1	1	ReB + ImB

Тогда получим

$$\frac{A}{B} = \frac{\sum_{i=1}^{m} 2^{-i} (Rea_i \cdot ReB + lma_i \cdot ImB)}{\sum_{i=1}^{m} 2^{-i} (ReB \cdot Reb_i + lmB \cdot Imb_i)} + j\frac{\sum_{i=1}^{m} 2^{-i} (-Rea_i \cdot ImB + lma_i \cdot ReB)}{\sum_{i=1}^{m} 2^{-i} (ReB \cdot Reb_i + lmB \cdot Imb_i)} \quad (5\text{-}8)$$

Отсюда следует, что для деления комплексных чисел при помощи МВТ необходимо вычислить всего одну таблицу, что позволяет свести к минимуму число подготовительных операций. В [71] изложен еще один более эффективный способ деления комплексных чисел.

Следует отметить, что в специализированных ЭВМ, предназначенных для выполнения операций комплексной арифметики может оказаться более целесообразным представление чисел по комплексному основанию [35, 53]. Например, система по основанию 2i позволяет любое комплексное число представить при помощи цифр 0, 1, 2 и 3, причем тех же цифр, взятых со знаком минус, не требуется. Система получила название мнимо-четверичной.

Пример: Перевести число A в десятичную систему

$$A = (31120,31)_{2i} = 3\cdot l6+1(-8i)+1(-4)+2\cdot(2i)+3(-1/2i)+1\cdot(-1/4) = 43\frac{3}{4} - 5\frac{1}{2}i,$$

то есть число $(a_{2n} \ldots a_1\ a_0\ a_{-1} \ldots a_{-2k})_{2i}$ равно

$$(a_{2n} \ldots a_2\ a_0\ a_{-2} \ldots a_{-2k})_{-4} + 2i(a_{2n-1} \ldots a_3\ a_1\ a_{-1} \ldots a_{-2k+1})_{-4}.$$

Поэтому перевод числа в мнимо-четверичную форму и обратно сводится к переводу в "минус-четверичную" форму и обратно. Интересное свойство этой системы состоит в том, что она допускает выполнение сложения, умножения и деления комплексных чисел целостным образом без раздельного рассмотрения вещественных и мнимых частей. Например, мы можем умножить два числа в этой системе так же, как и при любом другом основании, используя несколько иное правило переноса: в случае, если цифра становится больше 4, мы вычитаем 4 и переносим –1 на два столбца влево, а когда

получается отрицательная цифра, мы прибавляем к ней 4 и переносим +1 на два столбца влево.

Пример: Заданы A = $_{\times}12231 = (9-10i)_{2i}$. Получить произведение A×B.

B = $\underline{12231} = (9-10i)_{2i}$

12231

10320213

13022

13022

12231

021333121 $= (181-180i)_{2i}$

Можно построить аналогичные системы по основанию $\sqrt{2i}$, (i–1) и другим.

5.4. Методы вычисления элементарных функций

В цифровой вычислительной технике применяют следующие методы вычисления элементарных функций (ЭФ) [62, 92]: разделение в ряд Тейлора (степенные полиномы), аппроксимацию с помощью различных полиномов, табличные методы, рациональные приближения ЭФ, использование цепных дробей, итерационные (рекуррентные).

Степенные полиномы (отрезок ряда Тейлора, полином Чебышева и т.д.) вычисляются в ЭВМ чаще всего по схеме Горнера. При этом требуется выполнить m операций умножения и m операций сложения (m – степень полинома). Недостатки: ряд Тейлора очень медленно сходится для некоторых функций (натуральный логарифм, обратные тригонометрические и гиперболические функции), и поэтому время вычисления будет большим, а инструментальная погрешность увеличивается. Методическая погрешность этого метода монотонно растет с ростом аргумента, и поэтому приходится предварительно сводить аргумент в более узкую область с помощью соответствующих преобразований. Достоинством разложения в ряд Тейлора (в отличие от аппроксимации полиномом Чебышева) является то, что можно вычислять коэффициенты членов ряда непосредственно при вычислении функций и не хранить их в памяти ЭВМ. Однако при этом возрастает время вычисления ЭФ. Кроме того, единообразие вычисления всех ЭФ трудно обеспечить из-за плохой сходимости ряда для некоторых функций.

Метод полиномиальной аппроксимации используется в ЭВМ наиболее часто. Он характеризуется достаточно высоким единообразием вычисления всех ЭФ, однако при этом в памяти необходимо хранить большое количество коэффициентов всех полиномов. Для ускорения сходимости полинома аргумент предварительно сводится в более узкую

область. Методическая погрешность знакопеременна и равномерно распределена на интервале изменения аргумента. Для вычисления ЭФ с произвольной разрядностью в некоторых ЭВМ используется комбинированный таблично-полиномиальный алгоритм. Приближение любой ЭФ в приведенном интервале ведется с помощью подпрограмм не одним ортогональным полиномом, а их набором, каждый из которых применяется на подинтервалах с возрастанием степени аппроксимации от одного подинтервала к следующему.

Табличные методы основаны, главным образом, на кусочно-линейной и криволинейной аппроксимации. Для вычисления ЭФ этим методом требуется выполнить малое число арифметических операций, однако объем таблиц и время поиска в них может быть большим. Поэтому этот метод применяется в машинах с небольшой разрядностью слов.

При методе рационального приближения ЭФ функцию представляют в виде отношения двух полиномов, причем число членов в каждом полиноме намного меньше, чем при соответствующем разложении в ряд Тейлора. Однако коэффициенты полиномов должны обязательно храниться в памяти. Для вычисления ЭФ следует вычислить два полинома и выполнить операцию деления. Надо проверять устойчивость метода.

Метод цепных дробей характеризуется однообразием вычисления всех ЭФ, при этом количество констант мало. По сравнению с методом степенных полиномов в этом методе число шагов меньше, однако на каждом шаге выполняется операция деления, занимающая много времени. Хотя для цепных дробей область сходимости более широкая, чем для ряда Тейлора, но с ростом аргумента резко возрастает необходимое число звеньев дроби. Это заставляет приводить аргументы к интервалу, не более широкому, чем при разложении в ряд Тейлора. Оценка погрешности является сложной. Метод используется в малых машинах, где быстродействие не очень важно. Например, в машине "Проминь" метод цепных дробей реализован микропрограммно.

Итерационные методы предполагают вычисление последовательных приближений функции по итерационной формуле $Y_{j+i} = f(Y_i)$. Необходимость в ряде случаев вычислять операции деления и умножения на каждой итерации уменьшает быстродействие вычислений. Оценка погрешности удобна, алгоритмы вычисления некоторых функций (базового набора) достаточно единообразны. Загрузка памяти наименьшая, так как константы можно вычислить непосредственно перед счетом функций по той же схеме. Это снижает быстродействие, но является

решающим преимуществом при использовании в машинах с произвольной разрядностью [62].

Для большинства других методов характерно отсутствие единообразной методики вычисления вбех ЭФ. Это приводит к тому, что выбирается набор, так называемых, базовых функций, вычисляемых выбранным методом, а остальные ЭФ выражаются через базовые и вычисляются на их основе. Например, в машине МИР базовый набор ЭФ состоит из функций $\ln x$, a^x, $\cos x$, $\arcsin x$. Такая методика приводит к многоуровневой организации управления, усложнению структуры управляющего устройства и к увеличению времени вычисления.

В последнее время разработаны более эффективные итерационные алгоритмы вычисления ЭФ. Этот метод чаще всего называется методом "цифра за цифрой", так как после n итераций алгоритма получается значение функции с точностью до единицы n-го разряда. Однако следует заметить, что в отличие традиционных методов "цифра за цифрой" (например, деления), в данном методе на каждом шаге итерации получается полноразрядный приближенный результат, а приращение прибавляется к нему, так что на любом шаге из-за переносов может измениться любая цифра результата, вплоть до старшего разряда. Важным преимуществом этого метода является его эффективность для непосредственного вычисления почти всех ЭФ. Кроме того, с помощью этого метода можно также находить корни полиномов, выполнять преобразования координат и преобразования систем счисления чисел. Погрешность вычислений можно легко компенсировать с помощью дополнительных разрядов. Области сходимости достаточно широкие, а для некоторых функций вообще не требуется приведения аргумента. Так как алгоритмы метода построены на простых операциях сдвига и сложения, то их аппаратная реализация является простой и эффективной. Метод позволяет вычислять большинство ЭФ за время трех операций деления, которые можно выполнять параллельно. Недостатки метода – большое количество используемых констант и применение операционного устройства со специфической структурой.

Таким образом, метод "цифра за цифрой" обладает следующими преимуществами при структурной реализации: высокое быстродействие алгоритмов, основанных на операциях сдвига и сложения; единообразие вычисления почти всех ЭФ и оправданные аппаратурные затраты; простая организация вычислительного процесса, малое число уровней управления; удобство аппаратной компенсации пофешностей, возникающих при реализации алгоритмов на ЭВМ.

5.4.1. Метод "цифра за цифрой"

В основе данного метода лежит процедура преобразования прямоугольных координат точки в полярные, которую приспособили для вычисления большинства ЭФ. Метод "цифра за цифрой" в геометрическом смысле есть последовательность преобразований вектора в плоскости ХУ, т.е. последовательность поворотов радиуса вектора на стандартные углы вокруг начала координат с одновременным изменением длины вектора. В машине этот процесс представляется арифметическими преобразованиями координат вектора.

Например, для вычисления функций $\sin\varphi$, $\cos\varphi$ берется вектор с координатами (1, 0) и поворачивается последовательно на углы L_i (i = 0, 1,...,), так, чтобы алгебраическая сумма всех этих углов была равна φ. Тогда, очевидно, координаты итогового вектора равны искомым функциям $Y = \sin\varphi$, $X = \cos\varphi$. Согласно известным формулам, при каждом повороте вектора на угол L_i новые координаты вектора выражаются через предыдущие [3, 62]:

$$Y_{i+1} = \cos L_i(Y_i + X_i\, tgL_i); \qquad (5.9)$$

$$X_{i+1} = \cos L_i(X_i - Y_i\, tgL_i). \qquad (5.10)$$

Стандартные углы L_i(i = 0, 1,..,) заранее выбирают такими, чтобы tgL_i были равны целой степени основания системы счисления, используемой в ЭВМ. Тогда операции умножения на tgL_i вырождаются в операции сдвига. Для того, чтобы вектор в процессе таких поворотов приближался к требуемому положению, определяемому аргументом φ, необходимо поворачивать его на каждом шаге по (или против) часовой стрелке, в зависимости от того, является ли текущий угол наклона вектора к оси абсцисс больше или меньше аргумента φ, т.е. знак $\xi = \pm 1$ очередного угла поворота L_i выбирается из условия

$$\operatorname{sign}\xi_i = \operatorname{sign}\left(\varphi - \sum_{j=1}^{i-1}\xi_i L_j\right). \qquad (5.11)$$

Кроме того, абсолютные значения углов поворота должны уменьшаться по мере приближения вектора к требуемому положению. Доказано, что для двоичной системы счисления такой процесс сходится при $L_i = \operatorname{arctg}2^{-i}$ и $|\varphi| < \pi/2$, т.е. в результате поворотов исходного вектора последовательно на углы $\pm\operatorname{arctg}2^0$, $\pm\operatorname{arctg}2^{-1}$, $\pm\operatorname{arctg}2^{-2}$,... вектор приближается к требуемому (под углом φ к оси абсцисс) с методической погрешностью, которая не превышает последнего из примененных углов поворота. Поэтому для n-разрядных двоичных чисел достаточно выполнить n поворотов.

При дальнейших поворотах на углы $\mathrm{arctg}2^{-n}$, $\mathrm{arctg}2^{-n-1}$,... приращения координат при использовании чисел с фиксированной запятой будут равны машинному нулю.

Если в формулах (5.9) и (5.10) отбросить множитель $\cos L_i$, то они будут описывать поворот вектора с одновременным увеличением его длины в $1/\cos L_i$ раз. Тогда в результате n-кратного итерационного применения формул длина вектора увеличится в $C_T = \prod_{i=0}^{n-1} \frac{1}{\cos L_i}$ раз (Ст называют коэффициентом деформации тригонометрического вектора). Чтобы скомпенсировать это удлинение, в качестве начального вектора берут вектор длиной $1/C_T$.

Таким образом, с учетом изложенного процесс вычисления функций $\sin\varphi$, $\cos\varphi$ описывается формулами:

$$\left.\begin{aligned} Y_{i+1} &= Y_i + \xi_i X_i \cdot 2^{-i}, \\ X_{i+1} &= X_i - \xi_i Y_i \cdot 2^{-i}, \\ \varphi_{i+1} &= \varphi_i - \mathrm{arctg}2^{-i}, \\ \mathrm{sign}\xi_i &= \mathrm{sign}\varphi_i. \end{aligned}\right\} \qquad (5.12)$$

Формулы (5.12) применяются n раз итерационно при начальных значениях координат

$$X_0 = \frac{1}{C_T} = \frac{1}{\sqrt{\prod_{i=0}^{n-1}(1 + 2^{-2i})}}; \quad Y_0 = 0; \ \varphi_0 = \varphi. \qquad (5.13)$$

и в итоге получается $X_n = \cos\varphi$, $Y_n = \sin\varphi$ с точностью до 2^{-n+1}.

С помощью такого алгоритма легко вычислить функцию $\mathrm{arctg}(Y_0/X_0)$. Для этого берется исходный вектор (X_0,Y_0) и поворачивается на такие же стандартные углы таким образом, чтобы "положить" итоговый вектор R_n на ось абсцисс (рис. 5.2), Когда это достигнуто, то накопленная к концу алгоритма алгебраическая сумма углов поворотов равна $\mathrm{arctg}(Y_0/X_0)$. Вычисления выполняются по формулам:

$$\left.\begin{aligned} Y_{i+1} &= Y_i - \xi_i X_i \cdot 2^{-i}, \\ X_{i+1} &= X_i + \xi_i Y_i \cdot 2^{-i}, \\ \varphi_{i+1} &= \varphi_i - \xi_i \mathrm{arctg}2^{-i}, \\ \mathrm{sign}\xi_i &= \mathrm{sign}Y_i. \end{aligned}\right\} \qquad (5.14)$$

и в итоге получается

$$\varphi_n = \mathrm{arctg}\frac{Y_0}{X_0}; \qquad X_n = C_T\sqrt{X_0^2 + Y_0^2}\ . \qquad (5.15)$$

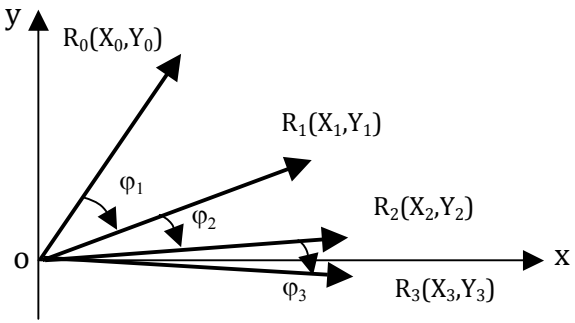


Рис 5.2. К вычислению функции arcfg Y_0/X_0

Вычисление гиперболических функций аналогично описаным выше и имеет схожую геометрическую интерпретацию. Как известно, множество точек с координатами x = chφ, y = shφ для всевозможных действительных φ функция с гиперболическими координатами образует на плоскости XY гиперболическую кривую (рис. 5.3).

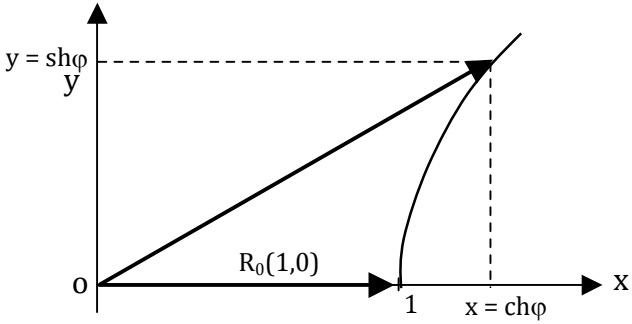


Рис. 5.3. Геометрическая иллюстрация алгоритмов вычисления гиперболических функций.

Поэтому для получения функций chφ_0, shφ_0 требуется найти точку на гиперболе, соответствующую значению аргумента φ_0. Для этого берут исходный вектор $R_0(1,0)$ и вращают его, одновременно изменяя длину вектора таким образом, чтобы конец вектора скользил по гиперболе. Одновременно с перемещением вектора подсчитывают значения аргумента, соответствующие его текущему положению. Когда текущее значение аргумента достигнет заданного φ_0, координаты

вектора будут равны $x = ch\varphi_0$, $y = sh\varphi$. При изменении аргумента на величину L координаты вектора, скользящего по гиперболе, изменятся согласно известным формулам:

$$\left.\begin{aligned} sh(\varphi + L) &= sh\varphi \cdot chL + ch\varphi \cdot shL, \\ ch(\varphi + L) &= ch\varphi \cdot chL + sh\varphi \cdot shL. \end{aligned}\right\} \qquad (5.16)$$

Для данного случая эти формулы удобно записать следующим образом:

$$\left.\begin{aligned} Y_{i+1} &= chL_i(Y_i + X_i\, thL_i), \\ X_{i+1} &= chL_i(X_i + Y_i\, thL_i). \end{aligned}\right\} \qquad (5.16)$$

где X_{i+1}, Y_{i+1} – координаты вектора, после приращения аргумента φ_i на величину L_i. В качестве констант L_i в двоичной системе счисления выбирают следующие константы $arth2^{-1}$, $arth2^{-2}$,..., $arth2^{-n}$. Для того, чтобы описанный процесс перемещения вектора сходился в требуемую точку, необходимо использовать каждую из констант дважды. Аналогично тригонометрическим функциям, множитель thL_i в формулах (5.17) отбрасывают, в результате чего длина вектора будет дополнительно изменяться. В результате 2^n приращений аргумента на указанные константы вектор изменится в C_h раз

$$C_h = \prod_{i=0}^{n-1} (1 + 2^{-2i}) \qquad (5.18)$$

где C_h – коэффициент деформации гиперболического вектора. Деформация компенсируется аналогично тригонометрическим функциям.

С учетом вышеизложенного, формулы вычисления функций $ch\varphi$, $sh\varphi$ принимают вид:

$$\left.\begin{aligned} Y_{i+1} &= Y_i + \xi_i X_i \cdot 2^{-K_i}, \\ X_{i+1} &= X_i + \xi_i Y_i \cdot 2^{-K_i}, \\ \varphi_{i+1} &= \varphi_i - \xi_i arcth2^{-K_i}, \\ sign\xi_i &= sign\ \varphi_i. \end{aligned}\right\} \qquad (5.19)$$

$$K_i = \begin{cases} \dfrac{i}{2}, & \text{если i – четное,} \\ \dfrac{i+1}{2}, & \text{если i – нечетное.} \end{cases}$$

Задав начальные условия $X_0 = 1/C_h$, $Y_0 = 0$, $\varphi_0 = \varphi$ и применив формулы (5.19) итерационно 2n раз, получим $X_{2n} = ch\varphi$, $Y_{2n} = sh\varphi$. Для сходимости вычисления достаточно, чтобы

$$|\varphi| \le 2arth2^n.$$

Вычисление функции $\operatorname{arth}\frac{Y}{X}$ заключается в следующем. Исходный вектор (x,y) перемещается так, чтобы его конец скользил по гиперболе и попал в точку пересечения гиперболы с осью абсуисс. Когда это достигнуто, то алгебраическая сумма всех констант L_i (i = 0, 1,...), соответствующих выполненным преобразованиям исходного вектора, равна искомому значению функции $\operatorname{arth}\frac{Y}{X}$. Этот процесс описывается формулами:

$$\left.\begin{aligned} Y_{i+1} &= Y_i - \xi_i X_i \cdot 2^{-K_i}, \\ X_{i+1} &= X_i - \xi_i Y_i \cdot 2^{-K_i}, \\ \varphi_{i+1} &= \varphi_i + \xi_i \operatorname{arcth} 2^{-K_i}, \\ \operatorname{sign}\xi_i &= \operatorname{sign} Y_i. \end{aligned}\right\} \qquad (5.20)$$

которые применяются 2^n раз итерационно при начальных значениях $Y_0 = Y$, $X_0 = X$, $\varphi = 0$ и дают в итоге

$$\varphi_{2n} = \operatorname{arth}\frac{Y}{X},\ X_{2n} = C_h\sqrt{X_0^2 - Y_0^2} \qquad (5.21)$$

с точностью до n двоичный разрядов.

В описанных алгоритмах вычисления прямых тригонометрических и гиперболических функций происходит "разложение" значения аргумента в набор констант $\pm\operatorname{arctg}2^{-i}$; или $\pm\operatorname{arcth}2^{-i}$ т.е. аргумент как бы преобразуется в систему счисления с цифрами ±1 и с весами разрядов $\operatorname{arctg}2^{-i}$ или $\operatorname{arcth}2^{-i}$ соответственно, а функция вычисляется по набору $\xi_0, \xi_1,..., \xi_n$. В алгоритмах вычисления соответствующих обратных функций получают "цифра за цифрой" значение функции в виде $\xi_0\ \xi_1...$ в той же промежуточной системе счисления, а затем преобразуют значение функции в двоичную систему счисления путем сложения соответствующих констант. Аналогично для вычисления логарифмической и экспоненциальной функций в двоичной системе применяются константы $\lg(1 \pm 2^i)$.

Для вычисления функции $x = e^{\varphi}$ аргумент φ преобразуют в сумму констант вида $\lg(1 + \xi_i \cdot 2^{-i})$ (i = 1, 2,...). При этом знак $\xi_i = \pm 1$ следует выбирать на каждом шаге так, чтобы текущее промежуточное значение аргумента φ_i стремилось к заданному φ_0, т.е.

$$\operatorname{sign}\xi_i = \operatorname{sign}\left(\varphi_0 - \sum_{j=1}^{i-1}\ln(1 + \xi_i 2^{-j})\right) \qquad (5.22)$$

Для того, чтобы процесс представления аргумента суммой констант сходился, необходима дважды использовать каждую из констант $\ln(1 \pm 2^{-i})$. Видно, что приращению текущего значения φ_i на величину $\ln(1 + \xi_i \cdot 2^{-i})$ соответствует увеличение текущего значения функции $X_i = e^{\varphi_i}$ в $(1 + \xi_i \cdot 2^{-K_i})$ раз, т.е. $X_{i+1} = X_i + X_i \cdot \xi_i \cdot 2^{-K_i}$. Таким образом, алгоритм вычисления e^{φ} описывается следующими итерационными формулами:

$$\left.\begin{aligned} X_{i+1} &= X_i + \xi_i X_i \cdot 2^{-K_i}, \\ \varphi_{i+1} &= \varphi_i - \ln(1 \pm \xi_i \cdot 2^{-i}), \\ \operatorname{sign}\xi_i &= \operatorname{sign}\varphi_i. \end{aligned}\right\} \qquad (5.23)$$

которые при начальных значениях $X_0 = 1$, $\varphi_0 = \varphi$ дают после 2n итераций $X_{2n} = e^{\varphi}$.

Для вычисления функции lnX аргумент X преобразуют в произведение

$$X = \frac{1}{\prod_{i=0}^{2n-1} (1 + \xi_i \cdot 2^{-K_i})}. \qquad (5.24)$$

Чтобы добиться такого разложения аргумента, т.е. найти $\xi_0, \xi_1, \ldots, \xi_{2n-1}$, берут заданное значение аргумента и последовательно умножают его на $(1 + \xi_0 \cdot 2^{-1})$, $(1 + \xi_1 \cdot 2^{-1})$, $(1 + \xi_3 \cdot 2^{-2})$,..., каждый раз выбирая знак ξ_i таким образом, чтобы получаемое произведение стремилось к единице. Когда X представлено формулой (5.24), логарифм lnX находится как

$$\ln X = -\prod_{i=0}^{2n-1} \ln\left(1 + \xi_i \cdot 2^{-K_i}\right) \qquad (5.25)$$

Таким образом, алгоритм вычисления функций lnX описывается формулами

$$\left.\begin{aligned} X_{i+1} &= X_i + \xi_i X_i \cdot 2^{-K_i}, \\ \varphi_{i+1} &= \varphi_i - \ln(1 \pm \xi_i \cdot 2^{-i}), \\ \operatorname{sign}\xi_i &= -\operatorname{sign}(1 - X_i). \end{aligned}\right\} \qquad (5.26)$$

которые при начальных значениях $X_0 = 1$, $\varphi_0 = \varphi$ дают после 2n итераций $\varphi_{2n} = \ln X$.

Чтобы этот алгоритм был похожим на алгоритмы многих других функций, можно одновременно с вычислением величины Xi вести подсчет величины $Y_i = 1 - X_i$. Для этого достаточно установить начальное значение $Y_0 = 1 - X_0$ и прибавлять к Y_i приращения, противоположные приращениям величины X_i. Тогда формулы алгоритма вычисления lnX примут вид

$$\left.\begin{aligned} X_{i+1} &= X_i + \xi_i X_i \cdot 2^{-K_i}, \\ Y_{i+1} &= Y_i - \xi_i X_i \cdot 2^{-K_i}, \\ \varphi_{i+1} &= \varphi_i - \ln(1 \pm \xi_i \cdot 2^{-i}), \\ \operatorname{sign}\xi_i &= -\operatorname{sign} Y_i. \end{aligned}\right\} \qquad (5.27)$$

где $X_0 = 1$, $Y_0 = 1 - X_0$, $\varphi_0 = 0$, $\varphi_{2n} = \ln X$.

Другой способ вычисления функции e^{φ} и $\ln X$ использует алгоритмы вычисления функций $ch\varphi$, $sh\varphi$, arthZ и известные тождества

$$\left.\begin{aligned} e^{\varphi} &= ch\varphi + sh\varphi, \\ \ln X &= 2\operatorname{arth}\frac{X-1}{X+2}. \end{aligned}\right\}$$

Аналогично можно вычислять функцию $\sqrt{X}$ с помощью алгоритма функции $\operatorname{arth}\frac{Y}{X}$ и тождества

$$\sqrt{X} = C_h\sqrt{\left(X + \frac{1}{4C_h^2}\right)^2 - \left(X - \frac{1}{4C_h^2}\right)^2} \qquad (5.29)$$

В самом деле, согласно формуле (5.21) в результате вычисления $\operatorname{arth}\frac{Y}{X}$ получим итоговое значение координаты $X_{2n} = C_h\sqrt{X_0^2 - Y_0^2}$. Если установить начальные значения $X_0 = X + \frac{1}{4C_h^2}$, $Y_0 = X - \frac{1}{4C_h^2}$, то по алгоритму (5.20) согласно формуле (5.29) получим $x_{2n} = \sqrt{X}$.

Методом "цифра за цифрой" можно вычислять не только натуральный, но и любой другой логарифм, например, двоичный или десятичный. Для этого необходимы соответственно константы вида $\log_2(1 \pm 2^{-i})$ и $\lg(1 \pm 2^{-i})$.

Вычисление функции arcsinY сводится к следующему: вектор R с координатами $X_0 = 1$, $Y_0 = 0$ поворачивается на набор стандартных углов так, что ордината принимает значения аргумента. Тогда алгебраическая сумма всех выполненных углов поворота равна искомой функции arcsinY. Для реализации такого алгоритма используется те же стандартные углы $\operatorname{arctg}2^{-i}$ и та же операция поворота вектора с одновременным его удлинением, что и в алгоритмах (5.12) и (5.14). Чтобы скомпенсировать итоговое удлинение вектора, в качестве начального берут укороченный вектор. При этом длина вращаемого вектора в процессе выполнения алгоритма растет от начального значения до единицы и

соответственно этому претерпевают деформацию обе координаты X_i, Y_i. А так как в качестве критерия направления векторов в этом алгоритме используется знак разности $(Y - Y_i)$, то указанная деформация приводит к возможности выбора ошибочного направления поворота. Поэтому в отличие от алгоритмов (5.12) и (5.14) необходимо дважды использовать каждый из стандартных углов $\mathrm{arctg}2^{-i}$ и, кроме того, учитывать возможность выхода вектора во второй или третий квадранты плоскости XY. С учетом изложенного алгоритм вычисления arcsinY описывается следующими формулами:

$$\left.\begin{aligned} X_{i+1} &= X_i - \xi_i Y_i \cdot 2^{-K_i}, \\ Y_{i+1} &= Y_i + \xi_i X_i \cdot 2^{-K_i}, \\ \varphi_{i+1} &= \varphi_i + \xi_i \mathrm{arcth}2^{-K_i}, \\ \mathrm{sign}\xi_i &= -\mathrm{sign}(Y - Y_i)\cdot \mathrm{sign}X_i. \end{aligned}\right\} \qquad (5.30)$$

где

$$K_i = \begin{cases} \dfrac{i}{2}, & \text{если i – четное,} \\ \dfrac{i-1}{2}, & \text{если i – нечетное.} \end{cases}$$

При начальных значениях $X_0 = \dfrac{1}{C_h^2}$, $Y_0 = 0$, $\varphi = 0$ получим $\varphi_{2n} = \mathrm{arcsin}Y$. Как видим, алгоритм отличается от алгоритмов вычисления других тригонометрических функций наличием дополнительной операции для определения ξ_i причем эту i-ю операцию необходимо выполнять после i-го применения других формул системы (5.30). Чтобы избежать этого, можно параллельно с вычислением вести подсчет величины $Z_i = Y - Y_i$. При этом алгоритм видоизменяется следующим образом:

$$\left.\begin{aligned} X_{i+1} &= X_i - \xi_i Y_i \cdot 2^{-K_i}, \\ Y_{i+1} &= Y_i + \xi_i X_i \cdot 2^{-K_i}, \\ Z_{i+1} &= Z_i + \xi_i X_i \cdot 2^{-K_i}, \\ \varphi_{i+1} &= \varphi_i + \xi_i \mathrm{arcth}2^{-K_i}, \\ \mathrm{sign}\xi_i &= -\mathrm{sign}\, Z_i \cdot \mathrm{sign}X_i. \end{aligned}\right\} \qquad (5.31)$$

При начальных условиях $X_0 = \dfrac{1}{C_h^2}$, $Y_0 = 0$, $Z_0 = Y$, $\varphi = 0$ получим $\varphi_{2n} = \mathrm{arcsin}Y$.

Другой способ вычисления функции $\mathrm{arcsin}\dfrac{Y}{R}$ основан на использовании тождества

$$\arcsin\frac{Y}{R} = \operatorname{arctg}\frac{Y}{\sqrt{R_0^2 - Y_0^2}} = \operatorname{arctg}\frac{C_h Y}{C_h\sqrt{R_0^2 - Y_0^2}} \qquad (5.32)$$

Сначала вычисляют $\operatorname{arcth}\frac{Y}{R}$ и находят величину $C_h\sqrt{X_0^2 - Y_0^2}$ (см. выражение (5.21)). Затем вычисляют C_hY с помощью операции умножения или алгоритма вычисления $ch\varphi$ при $X_0 = Y$, $Y_0 = 0$, и $\varphi_0 = 0$ и наконец, к найденным величинам применяют алгоритм (5.14) (вместо умножения Y на C_h можно выполнять деление $C_h\sqrt{X_0^2 - Y_0^2}$ на C_h).

Функцию arccosX можно вычислять аналогично функции arcsinY, т.е. взять вектор $R_0(1,0)$ и повернуть его так, чтобы координата X_i приняла значение аргумента. Другие способы нахождения arccosX сводятся к вычислению функций на основе тождества.

$$\arccos X = \operatorname{arctg}\frac{\sqrt{1 - X^2}}{X}. \qquad (5.33)$$

Вычисление функций arcshX, arcchX аналогично описанным выше для соответствующих обратных тригонометрических. Они отличаются тем, что вектор в процессе преобразования должен двигаться по гиперболе, а тождества имеют вид

$$\operatorname{arcsh} X = \operatorname{arcth}\frac{X}{\sqrt{1 - X^2}}; \quad \operatorname{arcch} X = \operatorname{arcth}\frac{\sqrt{1 + X^2}}{X}. \qquad (5.34)$$

Причем $\sqrt{1 - X^2}$ вычисляется в алгоритме функции arctgX. Остальные ЭФ вычисляются с помощью перечисленных выше алгоритмов на основе известных тождеств тригонометрии. Следует отметить, что операцию деления, используемую в этих тождествах, можно выполнять по подобному же алгоритму. Для этого делитель представляется в виде

$$X = \frac{1}{\prod_{i=1}^{n-1}(1 + \xi_i \cdot 2^{-i})} \qquad (5.35)$$

и тогда функция φ/X вычисляется по формуле

$$\varphi/X = \varphi\prod_{i=1}^{n-1}(1 + \xi_i \cdot 2^{-i}) \qquad (5\text{-}36)$$

При этом алгоритм деления записывается так:

$$\left.\begin{aligned} X_{i+1} &= X_i + \xi_i Y_i \cdot 2^{-K_i}, \\ Y_{i+1} &= Y_i - \xi_i X_i \cdot 2^{-K_i}, \\ \varphi_{i+1} &= \varphi_i + \xi_i \operatorname{arcth} 2^{-K_i}, \\ \operatorname{sign}\xi_i &= \operatorname{sign} Y_i, \end{aligned}\right\} \qquad (5.31)$$

где устанавливают начальные $X_1 = X$, $\varphi_1 = \varphi$, $Y_1 = 1 - X$ и получают в результате $\varphi_n = \varphi/X$.

Контрольные вопросы к главе 5

1. Сформулируйте алгоритм извлечения корня квадратного и обоснуйте его корректность.
2. Как можно вычислять суммы парных произведений?
3. Как сравнительно быстро можно умножить и разделить два комплексных числа? Укажите получаемый при этом выигрыш.
4. В чем состоит суть метода "цифра за Цифрой"?
5. Как вычисляются $\sin\varphi$ и $\cos\varphi$ в соответствии с методом "цифра за цифрой"?
6. Как вычисляются arctg Y/X и модуль вектора в соответствии с методом "цифра за цифрой"?

Глава 6 ДВОИЧНО-ДЕСЯТИЧНАЯ АРИФМЕТИКА

Как уже указывалось в главе 2, в двоично-кодированном представлении десятичного числа каждая десятичная цифра i-го разряда изображается тетрадой двоичных символов: $a_i = \alpha_4^i\alpha_3^i\alpha_2^i\alpha_1^i$, где α_j^i – двоичная цифра i-й тетрады. Полученный таким образом десятичный код, кодированный двоичными символами, для краткости называют D-кодом.

Имеется некоторое множество D-кодов. Оно обусловливается наличием всего десяти разрешенных из шестнадцати возможных комбинаций, которые допускает тетрада. Наличие запрещенных комбинаций в D-кодах отличает их от обычных позиционных систем счисления, в которых все комбинации – разрешенные. Из всего множества известных D-кодов наибольшее распространение в вычислительной технике получили код D1 прямого замещения (система 8421) и код D2 с избытком 3 (система 8421+3).

6.1. Сложение в прямых D-кодах

Вследствие наличия запрещенных комбинаций, при сложении чисел в любом из D-кодов возникает необходимость в коррекции результата и трудности в формировании десятичного переноса в следующую тетраду. Особенности сложения чисел в каждом из D-кодов различны, поэтому будем рассматривать из раздельно. При этом мы считаем, что заданы числа $A = a_n a_{n-1}...a_1a_0$ и $B = b_n b_{n-1}...b_1b_0$, где a_i, b_i – двоично-кодированные десятичные цифры (тетрады).

Необходимо получить: $A + B = C = c_{n+1}c_n...c_1c_0$, причем

$$c_i = a_i + b_i + \Pi_{i-1} - \Pi_i \cdot p; \quad C_{n+1} = \Pi_n ,$$

где $\Pi_i = \{0,1\}$, $\Pi_{i-1} = \{0,1\}$ – десятичные переносы; p=10 - основание системы счисления.

Так как наибольшее десятичное одноразрядное число равно 9, то с учетом переноса в данный разряд, значение результата разрядного суммирования лежит в пределах от 0 до 19. При этом единица во втором разряде представляет собой десятичный перенос в следующую тетраду, а сумма получается в двоичном коде, отличном от требуемого двоично-десятичного представления, т.е. она требует коррекции.

6.1.1. Код D1

При сложении чисел в коде D1 могут возникать следующие случаи:

1. Если $a_i + b_i + П_{i-1} < 10$, то при выполнении действий над разрядами тетрады по правилам двоичной арифметики сразу получается правильный результат.

2. Если $a_i + b_i + П_{i-1} \geq 10$, то должен быть десятичный перенос. Поэтому сумма в данной тетраде должна быть равна:

$$a_i + b_i + П_{i-1} - 10 \cdot П_i, \text{ где } П_i = 1.$$

При этом признаком неправильного результата является, в одном случае, возникновение потетрадного переноса $П'_i = 1$, во втором – появление запрещенной комбинации, если $15 > a_i + b_i + П_{i-1} \geq 10$. В любом из этих случаев необходимо скорректировать результат в данной тетраде введением поправки +0110, что приведет к возникновению потетрадного переноса и во втором случае. Коррекция обусловлена тем, что каждый перенос уносит с собой из данной тетрады 16 единиц, а приносит в следующую только 10 единиц.

Пример: Сложить тетрады $a_i = 1000$ и $b_i = 1001$ при $П_{i-1} = 1$.

$$c'_i = a_i + b_i + П_{i-1} = 1\ 0010.$$

Так как $\cdot П'_i = 1$, требуется коррекция результата.

$$c_i = 0010 + 0110 = 1000; \quad П_i = \cdot П'_i = 1.$$

Пример: Сложить $a_i = 1000$ и $b_i = 0110$ при $П_{i-1} = 1$.

$$c'_i = a_i + b_i + П_{i-1} = 0\ 1111.$$

Так как величина $c'_i = 1111$ принадлежит к запрещенным комбинациям, то необходимо ввести поправку вида 0110:

$$c_i = 1111 + 0110 = 10101, \text{ т.е. } П_i = 1;\ c_i = 0101.$$

Таким образом, если в 1-й тетраде сумма цифр с переносом из $(i-1)$-й тетрады меньше 10, то сложение производится без поправок. Если же сумма цифр с переносом равна или больше 10, то производится коррекция результата тетрады введением поправки +0110, а возникающий при этом перенос прибавляется к содержимому $(i-1)$-й тетрады.

При этом, если в нескольких тетрадах, начиная с i-й, разрядная сумма равна 1001, то перенос приведет к формированию запрещенной комбинации в i-й тетраде. В результате этого потребуется коррекция, которая приведет к запрещенной комбинации в $(i+1)$-й тетраде и т.д. Следовательно, из-за последовательного распространения потетрадных

переносов время сложения в коде D1 составит в худшем случае n тактов, где n – количество тетрад. Обычно схемы строят таким образом, чтобы перенос, возникающий при прибавлении тетрадной поправки, проходил сквозь тетрады, в которых предварительная сумма равна $9_{10}=1001_2$ и сбрасывал их в 0. При этом сумма всегда формируется за два такта.

Пример: Сложить числа A = 248_{10} = 0010 0100 1000 и
B = 355_{10}=0011 0101 0101.

$$\begin{array}{r} 0010\ 0100\ 1000 \\ +0011\ 0101\ 0101 \\ \hline 0101\ 1001\overset{\leftarrow}{}1101 \\ +\qquad\qquad 0110 \\ \hline 0101\overset{\leftarrow}{}1010\ 0011 \\ +\qquad 0110 \\ \hline 0110\ 0000\ 0011 \end{array}$$

Стрелкой показаны единицы потетрадных переносов.

6.1.2. Код D2

При сложении чисел в коде D2 возможны следующие случаи с учетом того, что $a'_i = a_i + 3$, b'_i $b_i + 3$, где a'_i, b'_i – тетрады для кода D2. Если $a_i + b_i + \Pi_{i-1} \le 15$, то $c_i = (a_i + b_i + \Pi_{i-1} + 3) = c'_i + 3$, т.е. результат необходимо скорректировать на величину – 0011. Если же $a_i + b_i + \Pi_{i-1} > 15$, то $c_i = (a_i + b_i + \Pi_{i-1} + 3) + 3 - \Pi'_i = c'_i - 3$, поскольку $\Pi'_i = 16$, т.е. по условию здесь возникает потетрадный перенос, который «уносит» с собой шесть избыточных комбинаций. Пдагому в этом случае требуется поправка резуль-< тата на величину +0011.

Пример: Заданы в коде D2 A = 46 = 0111 1001; B = 37 = 0110 1010.
Найти их сумму.

$$\begin{array}{r} 0111\ 1001 \\ +0110\ 1010 \\ \hline 1110\overset{\leftarrow}{}0011 \\ {}^{-}0011{}^{+}0011 \\ \hline 1011\ 0110 \end{array}$$

При сложении в коде D2 не возникает проблемы сквозного переноса и операция сложения выполняется в два такта. Это объясняется тем, что в нескольких тетрадах сумма цифр слагаемых равна $9_{10} = 1111_2$, то при поступлении переноса из (i – 1)-й тетрады в i-ю, ее содержимое сбрасывается в 0 и перенос пойдет в (i + 1)-ю тетраду в этом же такте

суммирования. В i-ю же тетраду во втором такте суммирования будет прибавлена поправка +0011. Поэтому коррекция результата может выполняться потетрадно с блокировкой цепей потетрадных переносов.

Таким образом, в коде D2 всегда производится коррекция промежуточного результата, полученного путем сложения цифр слагаемых по правилам двоичной арифметики. При этом, если при сложении i-х тетрад не возникает перенос, т.е. $П'_i = 0$, то поправка равна –0011; если же возникает потетрадный перенос $П'_i = 1$, то поправка в i-й тетраде равна +0011.

6.2. Сложение чисел в инверсных D-кодах

D-коды могут быть представлены в разрядной сетке машины в форме либо с фиксированной, либо с плавающей запятой. При этом отрицательные числа могут быть представлены в прямом, обратном и дополнительном кодах. Поэтому, если $A = -0,a_1a_2...a_n$, где a_i – тетрады, то

$$[A]_{пр} = -1,a_1a_2...a_n;$$
$$[A]_o = 1,\overline{\overline{a}}_1\,\overline{\overline{a}}_2...\,\overline{\overline{a}}_n;$$
$$[A]_д = 1,\overline{\overline{a}}_1\,\overline{\overline{a}}_2...\,\overline{a}_n;$$

где $\overline{\overline{a}}_i$ – дополнение до p–1 во всех тетрадах; $\overline{a}_n$ – дополнение до p = 10 в младшей тетраде.

Следовательно, $\overline{a}_n + a_n = p$; $\overline{\overline{a}}_i + a = p - 1$, $i = \overline{1,n-1}$. Обратный код D2 получается простым инвертированием цифр тетрад.

Пример: Найти обратный и дополнительный коды числа A = –0,136 для кода D2.

$$A_{d2} = -0,\ 0100\ 0110\ 1001$$
$$[A_{d2}]_o = 1,1011\ 1001\ 0110$$

На основании соотношения $[A]_д = [A]_o + p^{-n}$ получим дополнительный код числа A.

$$[A_{d2}]_д = 1,1011\ 1001\ 0111.$$

В коде D1 прямое инвертирование цифр тетрад означает получение дополнения до $2^4 - 1 = 15$. Поэтому для получения обратного D1 кода во все тетрады числа прибавляют вначале +0110, после чего производят инвертирование цифр тетрад.

Пример: Для числа A = –0,148 = – 0,0001 0100 1000 получить обратный и дополнительный коды D1.

0,0001 0100 1000
+0110+0110+0110
0,0111 1010 1110

Откуда $[A_{d1}]_o$ = 1, 1000 0101 0001, $[A_{a'}]_д$ = 1,1000 0101 0010.

При сложении в инверсных D-кодах, как и при сложении в прямых D-кодах, необходима коррекция результата, которая может осуществляться либо программными, либо аппаратными средствами. На практике обычно применяют второй способ. При этом в процессе выполнения коррекции цепи межтетрадного переноса в сумматорах блокируются для кода D2, а при сложении в коде D1 не блокируются.

Пример: Сложить в обратном и дополнительном кодах D1 и D2 числа A = –0,926 и B = +0,845.

Для кода D1:

A = –0,1001 0010 0110 и B = +0,1000 0100 0101

$[A]_o$ = 1,0000 0111 0011
$[B]_o$ = 0,1000 0100 0101
1,1000 1011 1000
+0110
$[C]_o = [A]_o + [B]_o$ = 1,1001←0001 1000
$[C]_{пр}$ = 1,0000 1000 0001

$[A]_д$ = 1,0000 0111 0100
$[B]_д$ = 0,1000 0100 0101
1,1000 1011 1001
+0110
$[C]_д$ = 1,1001←0001 1000
$[C]_{пр}$ = 1,0000 1000 0001

Для кода D2:

A = –0,1100 0101 1001; B = +0,1011 0111 1000

$[A]_o$ = 1,0011 1010 0110
+ $[B]_o$ = 0,1000 0100 0101
1,1111←0001 1110
1101 0011 1101
$[C]_o$ = 1,1001 0001 1000
$[C]_{пр}$= 1,0011 1011 0100

Отрицательные поправки вносятся в виде дополнения: 1101.

Таким образом, код D2, т.е. код с избытком 3, позволяет просто формировать поразрядные дополнения до p – 1 = 9 и также выполнять алгебраическое сложение. Однако, как и код D1, он требует коррекции результата, что увеличивает время реализации операций. Код D1 обладает рядом полезных свойств. Например, цифры десятичной системы счисления, значения которых больше или равны пяти, отличаются тем, что в старшем разряде кода D2 всегда присутствует единица. Благодаря этому свойству по значению только одного старшего разряда тетрады определяется необходимость округления. Этим же свойством можно воспользоваться для упрощения реализации ускоренного умножения.

6.3. Сдвиг D-кодов

При сдвиге D-кода числа на один десятичный разряд необходимо предусмотреть сдвиг двоичных цифр сразу на 4 разряда. Правила сдвига остаются теми же, что и для двоичных дробей, т.е. сдвиг вправо ограничивается величиной допустимой ошибки, которую можно уменьшить путем округления сдвинутого числа. Для этого вводят дополнительный десятичный разряд (ДДР). При этом положительные дроби округляют добавлением числа 5 в ДДР, если цифра в нем больше или равна 5. Округление отрицательной дроби, представленной в обратных D-кодах, не производится. Округление в дополнительных D-кодах производился только тогда, когда «хвост» числа больше $0{,}5 \cdot 10^{-n}$, где n – число основных десятичных разрядов. Операция сдвига D-кода влево корректна до тех пор, пока не произойдет переполнение разрядной сетки.

При выполнении операций умножения и деления D-кодов часто возникает необходимость умножения или деления чисел на 2. Для умножения D1 кода на 2 необходимо сдвинуть заданное число на один двоичный разряд влево. Поправки вносятся как и при сложении в коде D1, т.е. если в i-й тетраде в результате сдвига возникла запрещенная комбинация или из i-й тетрады в (i+1)-ю была выдвинута единица, то к i-й тетраде прибавляется поправка «0110» и цепи межтетрадных переносов не блокируются. В остальных случаях коррекция результата не производится.

Пример: Задано в коде D1 число $A = 0{,}159_{10} = 0{,}0001\ 0101\ 1001$. Сдвинуть его на один двоичный разряд влево.

$$\begin{array}{r} \bar{A} = 0{,}0010\ 1011\ 0010 \\ +0110+0110 \\ \hline 0{,}0011^{-}0001\ 1000 \end{array}$$

Таким образом, $\vec{A}$ = 0, 0011 0001 1000 = $0{,}318_{10}$.

При делении кода D1 на 2 необходимо сдвинуть его на один двоичный разряд вправо и вычесть поправку «0011» из тех тетрад, в которые сдвинулась единица из предшествующей, поскольку она имеет вес 8 единиц, а не 10/2 = 5.

Пример: Задано в коде D1 число A = 0,158 = 0,0001 0101 1000,
сдвинуть его на один двоичный разряд вправо.

$$\begin{array}{l}\vec{A} = 0{,}0010\ 1010\ 1100 \\ \qquad\quad {}^{-}0011{}^{-}0011 \\ \hline 0{,}0011\ 0111\ 1001\end{array}$$

Таким образом, $\vec{A}$ = 0,0000 0111 1001 = $0{,}079_{10}$.

При умножении кода D2 на 2 необходимо сдвинуть его на один двоичный разряд влево. Коррекция вносится как и при сложении, т.е. если из i-й тетрады сдвигается единица в (i+1)-ю, то к i-й тетраде прибавляется «0011», если же единица в (i+1)-ю тетраду не сдвигается, то к i-й тетраде прибавляется «1101». При этом цепи межтетрадных переносов блокируются.

Пример: Задано в коде D2 число A = $0{,}159_{10}$=0,0100 1000 1100,
сдвинуть его на один разряд влево.

$$\begin{array}{l}\vec{A} = 0{,}1001\ 0001\ 1000 \\ \quad {}^{+}1101{}^{+}0011\ \ 0011 \\ \hline 0{,}0110\ \ 0100\ 1011\end{array}$$

Таким образом, $\vec{A}$ = 0,0110 0100 1011=$0{,}318_{10}$.

При делении кода D2 на 2 необходимо сдвинуть его на один двоичный разряд вправо и прибавить поправку согласно табл. 6.1. Таблица 6.1 получается на основании следующих соображений.

Таблица 6.1

Цифра, сдвигаемая в данную тетраду	Цифра, выдвигаемая из данной тетрады	Поправка
-	0	$П_0$ = 0001
-	1	$П_1$ = 0010
0	0	$П_{00}$ = 0110
0	1	$П_{01}$ = 0111
1	0	$П_{10}$ = 1001
1	1	$П_{11}$ = 1010

Если цифра a, записанная в данной тетраде, четная, т.е. a = 2b, то ее двоичный код будет нечетным, так как он равен a+3 = 2b+3. Если a – нечетное, т.е. a = 2b + 1, то двоичный код в данной тетраде будет четным, так как он равен a+3 = 2b+4.

В старшую тетраду в прямом коде ничего не сдвигается, но из нее в соседнюю справа тетраду сдвигается либо 0, либо 1. Если сдвигается 0, значит a = 2b+1, т.е. код в ней четен и равен 2b+4. После сдвига он станет равен b+2, а должен быть равным b+3, следовательно, необходима поправка на величину $П_0$ = 0001. Если из старшей тетрады сдвигается 1, то a четно (a = 2b) и после сдвига должен быть код b + 3. Сдвигается же код 1/2(a+3) = (2b+3) = b+1+1/2. Здесь 1/2 переходит в виде 1 в тетраду справа, а поправка, очевидно, будет равна $П_1$ = 0010.

Теперь рассмотрим сдвиг любой тетрады. Если в нее вдвигается 0, то это значит, что в нее должна попасть 1 с весом 5 (10/2). Если из тетрады сдвигается 0, то в ней цифра нечетна, т.е. a = 2b+1, а код ее четный, т.е. равен 2b+1+3 = 2b+4. После, сдвига в тетраде будет код b+2, а должно быть b+3. Поэтому поправка в этом случае составит $П_{00}$ = = $(5+1)_{10}$ = 0110. Если же из данной тетрады выдвигается код 1, то в ней число четно, т.е. a = 2b, а код нечетный и равен 2b+3. После сдвига в тетраде получим (2b+3)·1/2 = b+1+1/2. При этом 1/2 в виде 1 сдвигается в соседнюю справа тетраду, а в данной остается код b+1, но должно быть b+3. Отсюда поправка равна $П_{01} = (5+2)_{10} = 0111_2$.

Если в данную тетраду сдвигается код 1, то это значит, что в нее должен попасть 0, т.к. десятичная цифра в предыдущей тетраде четная (цифра, а не ее код!). Эта 1 имеет вес 8, поэтому в поправку необходимо ввести отрицательную величину (–8). Если из данной тетрады выдвигается 0, то код в ней четен и равен 2b+4. После сдвига он равен b+2, т.е. результирующая поправка должна составить 1–8 = –7. Отрицательные поправки обычно вносятся в виде дополнений до 16-ти, поэтому $П_{10}$ = 1001**.** Если из данной тетрады сдвигается 1, значит a четно, т.е. a = 2b, а код в ней нечетен и равен 2b+3. После сдвига получим код b + 1 + 1/2. При этом 1/2 переходит в виде 1 в соседнюю справа тетраду, а в данной остается b+1, т.е. необходима поправка кода тетрады на величину 2–8 = –6, которая также задается в виде дополнения до 16-ти, т.е. $П_{11}$ = 1010. Необходимо помнить, что в коде D2 при реализации поправок цепи межтетрадных переносов блокируются.

Если данная тетрада младшая, то выдвигаемая из нее вправо двоичная цифра будет представлять собой инверсный код остатка от деления числа на 2: если цифра кода остатка 0, то цифра остатка числа равна 1 и наоборот.

Пример: Задано в коде D2 A=526_{10}=1000 0101 1001, сдвинуть его на один двоичный разряд вправо.

$\vec{A}$ = 0100 0010 1100

$^{+}$0001 0111 1010

0101 1001 0110

Таким образом, A=0101 1001 0110=526 : 2=263_{10}.

6.4. Умножение чисел в D-кодах

Умножение чисел в D-кодах сводится к последовательному суммированию частных произведений, получаемых при умножении множимого на очередную цифру множителя. При этом умножение сопровождается расшифровкой значения очередной i-й тетрады множителя, которая представляется в виде $(\beta_4\beta_3\beta_2\beta_1)_i^j$ и сдвигом множителя на четыре, разряда сразу.

Самым простым способом расшифровки тетрады является последовательное вычитание из значения тетрады до получения нуля и соответственно прибавление множимого в сумматор в каждом такте. Так как при умножении множимого на тетраду возможно переполнение разрядной сетки сумматора (вследствие того, что множимое прибавляется к сумме частных произведений столько раз, сколько единиц содержится в данном десятичном разряде множителя), то в нем необходимо предусмотреть дополнительную тетраду для учета возникающих переносов. Из четырех возможных способов умножения в D-кодах целесообразно применять только один: умножение младшими разрядами множителя со сдвигом суммы частных произведений вправо. Умножение обычно производится в прямом коде, Т.е. знак результата определяется суммой знаковых цифр сомножителей по модулю 2.

Пример: умножить в прямом D2 коде числа A = –0,45;

$[A]_{пр}$ = 1,0111 1000

и B = +0,24; $[B]_{пр}$ = 0,0101 0111.

Знак результата определяется так же, как и в случае двоичной системы: 1⊕0=1. Для выполнения операции умножения необходим регистр сдвига множимого вправо с вычитанием единицы из младшей тетрады, а также сумматор частных произведений с десятичной коррекцией.

РгВ→		СМ→
0101 0111		0011 0011 0011
0001	+A	0111 1000
0110	поправка	1010 1011
0001		1101 1101
0101	+A	0111 1000
0001		0111 1000
0100	поправка	1111 0000
0001		1101 0011
Конец анализа	+A	1100 0011
		0111 1000
	поправка	0100 0011 1011
		0011 1101
	+ A	0100 0110 1000
		0111 1000
	поправка	0100 1110 0000
		1101 0011
0011 0101	Сдвиг на 4 разряда	0100 1011 0011
0001		0011 0100 1011 0011
0100	+A	0111 1000
0001	поправка	1100 0011
		1101 0011
0011	+A	1001 0110
Конец анализа		0111 1000
	поправка	0100 0000 1110
		0110 1101
		0100 0011 0111 0011

Таким образом, $[A \times B]_{пр} = -1{,}0100\ 0011\ 0111\ 0011$.

Если сомножители имеют по n десятичных разрядов, то в регистре множимого должно быть n разрядов (тетрад) справа от запятой, а в сумматоре должно быть n основных разрядов, одна тетрада переполнений и один (n+1)-й дополнительный десятичный разряд (ДДР), который предназначен для округления результата (в коде D2 для округления достаточно иметь один дополнительный двоичный разряд). При необходимости сохранить все 2n разрядов произведения справа от сумматора должен быть еще сдвиговый регистр для младших разрядов произведения. Регистр множителя должен иметь n тетрад справа от запятой. Причем, если младшая тетрада этого регистра выполнена в виде реверсивного счетчика, т.е. счетчика, который может прибавлять или вычитать единицу, то операцию умножения можно ускорить, в среднем, почти в

два раза. В этом случае необходимо иметь специальный разряд для записи 1, если при суммировании в счетчике-тетраде появится код 10_{10}.

Такой состав оборудования объясняется следующим образом. Если очередная цифра множителя, находящаяся в младшей тетраде (счетчике) регистра множителя, равна или меньше 5_{10}, то производится многократное прибавление множимого к сумме частных произведений сумматора. При каждом суммировании множимого вычитается 1 из счетчика. Такие действия выполняются до тех пор, пока в счетчике не появится код 0. Если же очередная цифра множителя равна или больше 6_{10}, то производится многократное вычитание (сложение в дополнительном D-коде) множимого из суммы частных произведений сумматора. При каждом вычитании множимого к содержимому счетчика прибавляется 1. Вычитания продолжаются до тех пор, пока в счетчике не появится код 10_{10}. При этом в специальный разряд записывается 1.

По окончании анализа текущей тетрады множителя производится сдвиг на одну тетраду вправо суммы частных произведений и множителя. При этом з счетчик регистра множителя сдвигается следующая тетрада множителя, к которой прибавляется содержимое специального разряда, т.е. 1, если она была там записана. Операция умножения заканчивается сдвигом множителя и суммы частных произведений. Всего должно быть выполнено n сдвигов. На последнем сдвиге в счетчик сдвигается нуль множителя. Если при этом в него добавляется 1 из специального разряда, то операция умножения заканчивается прибавлением множимого к сумме частных произведений. В результате получается 2n-разрядное произведение.

Таким образом, описанный алгоритм позволяет сократить время умножения практически без изменения состава оборудования. Если необходим n-разрядный результат, то произведение округляется путем прибавления к ДДР, т.е. к (n+1)-й тетраде числа 5_{10}.

Пример: Задано $A=0{,}367_{10}$, $B=0{,}583_{10}$. Найти произведение $C = A \times B$ и округлить до 3-х разрядов.

Все действия для простоты выполняем в десятичном коде. Квадратной рамкой обведен счетчик в регистре множителя (РгВ), счетчик тактов (разрядов) умножения (СчТ) и специальный разряд (СпР). На сумматоре (СМ) самый левый разряд (от запятой) – двоичный (знаковый), а второй слева (разряд переполнений) – десятичный.

СМ		РгВ	СпР	СчТ
00, 000000		,58 [3]	[0]	[0]
+00, 367	$[+a_{10}]_д$	–1		
00, 367000		,58 [2]		
+00, 367	$[+a_{10}]_д$	–1		
00, 734000		,58 [1]		
+00, 367	$[+a_{10}]_д$	–1		
01, 101000		,58 [0]		
00, 110100	Сдвиг	,05 [8]		+1 [1]
+19, 633	$[-a_{10}]_д$	+1		
19, 743100		,05 [9]		
+19, 633	$[-a_{10}]_д$	+1		
19, 376100		,05 [10] ⟶	[0]	
	Сдвиг	,00 [5] +1 ⟵↓		+1 [2]
19, 937610		[6]	[0]	
+19, 633	$[-a_{10}]_д$	+1		
19, 570610		,00 [7]		
19, 570610		,00 [7]	[0]	
+19, 633	$[-a_{10}]_д$	+1		
19, 203610		,00 [8]		
+19, 633	$[-a_{10}]_д$	+1		
18, 836610		,00 [9]		
+19, 633	$[-a_{10}]_д$	+1		
18, 469610		,00 [10] ⟶	[1]	
	Сдвиг	,00 [0] +1 ⟵↓		+1 [3]
19, 846961		[1]	[0]	
+00, 367	$[-a_{10}]_д$	–1		
00, 213961		,00 [0]		
+ 5 окр.				
00, 214				

Таким образом, $C = A \times B = 0{,}214_{10}$.

Дальнейшим развитием описанного способа ускорения операции умножения в D-кодах является формирование кратных множимого A. Например, при наличии кратных ±2A и ±4A очередное частное произве-

дение формируется следующим образом в зависимости от текущей цифры множителя B (табл. 6-2). При этом, если предыдущая цифра B была больше пяти, то действие на очередном шаге надо увеличить на +1A. Как видно из таблицы, в худшем случае для данного способа потребуется всего 2 действия алгебраического сложения при получении одного частного произведения, а в среднем, для этого необходимо 1,2 сложений-вычитаний. Способ позволяет анализировать очередную цифру множителя простыми аппаратными средствами. Для его реализации потребуется еще два регистра для хранения 2A и 4A или коммутатор для формирования 4A и 2A путем сдвига A, соответственно, на 2 и 1 разряд.

Таблица 6.2.

Текущая цифра множителя	Вид операции	Текущая цифра множителя	Вид операции
0	0	5	+A+4A
1	+A	6	−4A
2	+2A	7	−A−2A
3	+A+2A	8	−2A
4	+4A	9	−A

Другой прием, применяемый для ускоренного умножения, сводится к следующим пошаговым преобразованиям произведения:

$$A \times B = 2A \times \frac{B}{2}, \text{ если B – четное число;}$$

$$A \times B = 2A \times \frac{B-1}{2} + A, \text{ если B – нечетное число.}$$

Пример: Задано A = 35, B = 34. Найти C = A×B.

35×34 = 2×35×34/2 = 70×17 = 140×8 + 70 = 280×4 + 70 = 560×2 + 70 =

= 1120×1 + 70 = 1190.

Таким образом, C = A×B = 1190.

При этом удвоение числа означает его сдвиг влево, а деление на 2 – сдвиг вправо. Так как двоичные сдвиги производятся над D-кодами, то требуется коррекция тетрад на каждом шаге, выполнение которой для каждого D-кода производится в соответствии с правилами, изложенными в предыдущем параграфе. Как видим, сложения здесь производятся

только, при нечетном множителе. Наибольшее их количество возможно при $B = 2^n - 1$ и равно (n–1), а при $B = 2^n$ производятся только сдвиги множимого и множителя.

Некоторую качественную картину об эффективности рассмотренных способов умножения дает сравнение числа сложений, которое необходимо для получения полного произведения по каждому из способов, например, при B = 0,583.

1. Только суммирование множимого. Количество сложений: 5+8+3 = 16.

2. Суммирование и вычитание множимого, множитель при этом фактически имеет вид: $B = 1,\overline{4}\overline{2}3$, где черта над цифрой означает вычитание. Тогда количество сложений составит: 1+4+2 = 10.

3. Суммирование и вычитание кратных множимого. Количество сложений: 1+1+1+2 = 5.

4. Пошаговое формирование произведения путем двоичного сдвига множимого и множителя. Количество сложений: 1+1+1+1 = 4.

Таким образом, если учесть, что в реальных ЭВМ количество разрядов значительно больше 3-х, то выигрыш в быстродействии при выполнении умножения ускоренными методами окажется еще более существенным.

6.5. Деление чисел в D-кодах

Деление десятичных чисел обычно выполняется в прямых D-кодах. Знак частного определяется так же, как и при двоичном делении, поэтому в дальнейшем будет рассматриваться деление положительных десятичных дробей. Из двух возможных способов деления двоичных чисел для деления десятичных чисел целесообразно применять только способ при неподвижном делителе и сдвиге остатков и чретного влево. При двоичном делении в каждом такте получалась очередная цифра частного, которая принимала одно из двух возможных значений (0 или 1). Для определения этой цифры делитель вычитался из делимого на нулевом шаге и сдвинутых влево на один разряд очередных остатков на последующих шагах. При десятичном делении количество вычитаний делителя из сдвинутого очередного остатка, как правило, больше 1, так как разряды частного могут принимать значения от 0 до 9. Так же как и при умножении, в сумматоре слева от запятой должно быть пять двоичных разрядов: самый левый знаковый, где фиксируется знак остатка, и еще одна тетрада, в которую может сдвигаться старшая цифра текущего остатка при его сдвиге влево.

Справа от запятой в сумматоре должна быть (n+1)-я тетрада, младшая из которых используется только для округления частного. В регистре делителя справа от запятой должно быть n тетрад. Регистр частного должен быть с левым сдвигом и иметь (n+1) тетраду. Причем младшая тетрада должна представлять собой реверсивный счетчик, в котором формируется очередная десятичная цифра частного при выполнении очередного цикла деления. Всех циклов деления должно быть n+2, так как в первом цикле получается цифра слева от запятой, а справа от запятой необходимо сформировать n+1 цифру с учетом разряда округления. Если в первом цикле цифра частного не равна 0, то частное вышло за пределы разрядной сетки, т.е. операция не корректна. При выполнении каждого шага в каждом цикле деления к счетчику тактов прибавляется 1, а операция завершается после выполнения (n+1) циклов.

В исходном состоянии делимое находится в сумматоре, делитель – в регистре делителя, а регистр частного установлен в состояние 0. Деление начинается с вычитания делителя из делимого в нулевом цикле и из сдвинутых остатков – в последующих, циклах. Вычитание на каждом шаге заменяется сложением в дополнительном D-коде и производится до тех пор, пока не получится отрицательный остаток. При этом каждый раз при получении положительного остатка добавляется единица в специальный счетчик, где накапливается очередная цифра частного. Затем осуществляется сдвиг остатка на четыре двоичных разряда и прибавление к нему делителя до тех пор, пока не получится положительный остаток. Количество сложений (без последнего) является дополнением соответствующей цифры частного до 9, что заносится в счетчик очередной цифры частного.

Таким образом, процесс деления состоит из ряда последовательно чередующихся циклов сложения и вычитания, которые выполняются в сумматоре, работающем по правилам алгебраического сложения в соответствующем D-коде, и завершаются сдвигами частного и остатков влево на одну тетраду. При этом в тетраду сумматора слева от запятой сдвигается код из тетрады справа от запятой, а содержимое знакового разряда не меняется.

Пример: Разделить число A = 0,154675 на число B = 0,55.

Так как A < B, на нулевом шаге остаток должен получиться отрицательным, т.е. операция корректна. Далее будет рассматриваться только цифровая часть чисел, причем для удобства сдвигаться будет делитель.

Шаг 1:

154675	СЧ=0	$Y_1 = 0$
–55000		
99675	СЧ=0+1	$Y_1 = 1$
–55000		
44675	СЧ=1+1	$Y_1 = 2$
–55000		
–89675		$Y_1 = 2$

Остаток меньше 0. Сдвиг частного влево и делителя вправо (или остатка влево) на одну тетраду.

Шаг 2:

–89675	СЧ=9	$Y_2 = 9$
+ 5500		
95175	СЧ=9–1	$Y_2 = 8$
+ 5500		
00675		$Y_2 = 8$

Остаток больше 0. Сдвиг частного влево и делителя вправо на 4 двоичных разряда.

Шаг 3:

00675	СЧ=0	$Y_3 = 0$
– 550		
00125	СЧ=0+1	$Y_3 = 1$
– 550		
99575		$Y_3 = 1$

Остаток меньше 0. Сдвиг частного влево и делителя вправо на 4 двоичных разряда. И так далее.

Таким образом, $Y = A/B \approx 0,281$.

По другому способу при делении чисел, заданных в D-кодах, результат получается в двоичной системе счисления. Допустим при делении A на B получено частное вида $Y = 0, y_1 y_2 \ldots y_n$.

Тогда

$$A = B(y_1 \cdot 2^{-1} + y_2 \cdot 2^{-2} + \ldots + y_n \cdot 2^{-n}) + R_n ,$$

где R_n – остаток от деления.

Положим $\cdot\ y_1 = 1, y_2 = 1, \ldots y_n = 0$. Тогда остаток на первом шаге

$$R_1 = A - 2^{-1} \cdot B.$$

Если $R_1 > 0$, то $y_1 = 1$, если $R_1 < 0$, то $y_1 = 0$.

В последнем случае восстанавливается предыдущий положительный остаток. Затем полагаем, что $y_2 = 1$, а остальные $y_i = 0$ и т.д. Остаток на любом шаге составит $R_i = A - B \sum_i y_i \cdot 2^i$.

Следует учитывать, что при сдвиге чисел, представленных в D-коде, необходимо вводить поправки. Поэтому для хранения делителя желательно иметь самостоятельный сумматор, в котором при передаче единицы из одной тетрады в другую автоматически вносится поправка.

Пример: Заданы A = 0,2425; B = 0,5200. Найти Y = A/B.

Делитель B на i-ом шаге	Сумматор (СМ)	Примечание	Цифры
0,5200	0, 2425	$B\cdot 2^{-1}$; СМ := [СМ] – $B\cdot 2^{-1}$	$y_1 = 0$
0,2600	–0, 2600	R<0	
	9, 9825		
	+0, 2600	Восстановление R_1, т.е.	
0,1300	0, 2425	$R_1 + B\cdot 2^{-1}$	
	–0, 1300	$B\cdot 2^{-2}$; СМ := [СМ] – $B\cdot 2^{-2}$	$y_2 = 1$
	0, 1125	$R_2 > 0$	
0,0650	–0, 0650	$B\cdot 2^{-3}$; $R_2 – B\cdot 2^{-3}$	$y_3 = 1$
	0, 0475	$R_3 > 0$	
0,0325	–0, 0325	$B\cdot 2^{-4}$; $R_3 – B\cdot 2^{-4}$	$y_4 = 1$
	0, 0150	$R_4 > 0$	
0,0162	–0, 0162	$B\cdot 2^{-5}$; $R_4 – B\cdot 2^{-5}$	$y_5 = 0$
	9, 9938	$R_5 < 0$	
	+0, 0162	Восстановление R_5	
	0, 0150	$B\cdot 2^{-6}$; $R_5 – B\cdot 2^{-6}$	$y_6 = 1$
0,0081	–0, 0081	R6 > 0	
	0, 0069		
		и т.д.	

Таким образом, Y = 0, 011101.

6.6. Перевод чисел в D-кодах

Вначале рассмотрим перевод целых десятичных чисел, представленных в D-коде, в двоичную систему счисления.

Пусть задано десятичное число $A = a_n a_{n-1} \ldots a_2 a_1$, где a_i – десятичная цифра, которая представлена в D-коде в виде $a_i = \alpha_4^i\alpha_3^i\alpha_2^i\alpha_1^i$. Используя равенство $10 = 8 + 2 = (2^3+2^1)$, любое десятичное целое число можно записать так:

$$A = \Big(\ldots\Big(\alpha_4^n\alpha_3^n\alpha_2^n\alpha_1^n(2^3+2^1) + \alpha^{n-1}_4\alpha^{n-1}_3\alpha^{n-1}_2\alpha^{n-1}_1\Big)\cdot(2^3+2^1)+\ldots$$
$$+ \alpha_4^2\alpha_3^2\alpha_2^2\alpha_1^2\Big)\cdot(2^3+2^1)+\ldots\ \alpha_4^1\alpha_3^1\alpha_2^1\alpha_1^1\ .$$

Умножение на 2^k означает сдвиг двоичного кода на k разрядов влево. Следовательно, перевод сводится к двоичным сдвигам соответствующих тетрад и их последующему суммированию.

Пример: Заданы A = 254 или в коде D1 A = 0010 0101 0100. Найти A_2.

$A_2 = (0010\cdot(2^3 + 2^1)+ 0101)(2^3 + 2^1)+ 0100 =$

$= (10000 + 100 + 101)(2^3 + 2^1)+ 0100 =$

$= 10.000.000+100.000+101.000 + 100.000 + 1000 + 1010 + 100=$

$= 11.101.000 + 10.110 = 11.111.110.$

Для перевода правильных дробей можно применить следующий прием. Заданную k-разрядную десятичную дробь вначале рассматривают как целое число и переводят по описанному алгоритму, а затем делят на 10, записанное двоичными символами (10 можно перевести в двоичный код по тому же алгоритму).

Пример: Задано $A_{10} = 0{,}37$ или в коде D1 A = 0,0011 0111. Найти A_2.

$A_2 = A'_2/B$, причем

$A'_2 = 0011(2^3+2^1)+0111 = 11110+0111 = 100101$, $B = 10^2 =$.

$A'_2 = 1001010{,}0 \mid 1100100$

$-\ 110010\ 0 \mid 0{,}0101111010 = A_2$

$0011000\ 000$

$-\ 01100\ 100$

$1011\ 1000$

$-\ 110\ 0100$

$101\ 01000$

$-\ 11\ 00100$

$10\ 001000$

$-\ 1\ 100100$

$0\ 1001000$

$-\ 1100100$

$1{,}1100100$

Перевод чисел из двоичной системы счисления в D-код может осуществляться разными способами. Например, делением целых двоичных чисел на число 1010. При этом десятичные цифры получаются последовательно одна за другой. При дробных числах эта операция выполняется таким образом, что при пбследовательном умножении дроби на число 1010 получаются соответствующие цифры десятичных дробей.

Пример: Задано $A_2 = 0{,}0101111010 = 0{,}37_{10}$.

Найти двоично-десятичный код этого числа.

При определении кода каждой десятичной цифры числа умножение A на 1010_2 заменяем сложением $A\cdot 2^3$ и $A\cdot 2^1$.

0		0|0101111010	
		0|1011110100	$\times 2^1$
	010	1111010000	$\times 2^3$
0011 ←	011	1101100010	
		0|101100010	
		1|01100010	
	0101	10001000	
0110 ←	0110	11101010	
		0|11101010	
		1|1101010	
	0111	0101000	
1001 ←	1001	0010010	

Таким образом, $A_{(2-10)} = 0{,}0011\ 0110\ 1001 = 0{,}369_{10}$.

Перевод из D-кода в двоичную систему счисления и наоборот можно упростить следующим образом. Пусть число A задано в коде D1, т.е.

$$A = \sum_{i=-k}^{n} p^i \cdot \sum_{j=1}^{4} \alpha_j^i \cdot 2^{j-1}.$$

Обозначив через b_j^i константы вида $p^i \cdot 2^{j-1}$, получим

$$A = \sum_{i=-k}^{n} \sum_{j=1}^{4} \alpha_j^i \cdot b_j^i .$$

Следовательно, если хранить двоичные коды констант b_j^i в таблице, то перевод из D-кода в двоичную систему сводится к формирование частных произведений вида $\alpha_j^i \cdot b_j^i$, где $\alpha_j^i \in \{0,1\}$, и их последующему суммированию.

Пример: В коде D1 задано число $A_{10} = 59$, т.е. $A_{(2-10)} = 0101\ 1001$.

Найти двоичный код этого числа.

Константы b_j^i в этом случае следующие: $b_1^0 = 1$, $b_2^0 = 2_{10} = 10_2$, $b_3^0 = 4_{10} = 100_2$, $b_4^0 = 8_{10} = 1000_2$, $b_1^1 = 10_{10} = 1010_2$, $b_2^1 = 20_{10} = 10100_2$; $b_3^1 = 40_{10} = 101000_2$,. $b_4^1 = 80_{10} = 1010000$.

Поэтому $A_2 = 0\cdot 1010000 + 1\cdot 101000 + 0\cdot 10100 + 1\cdot 1010 + 1\cdot 1000 + 0\cdot 100 + 0\cdot 10 + 1\cdot 1 = 111011$.

Эти же константы b_j^i позволяют упростить перевод числа A из двоичной системы счисления в D-код. При этом вначале из числа A вычита-

ется наибольшая константа b_4^n. Если разность положительна, то $\alpha_4^n = 1$, в противном случае $\alpha_4^n = 0$. На следующем шаге из остатка вычитается следующая константа b_3^n, а значение α_3^n определяется знаком нового остатка. Эта процедура повторяется до тех пор, пока не будут найдены все α_j^i. Следует отметить, что при отрицательном текущем остатке к нему на следующем шаге прибавляется новая константа.

Пример: Задано $A = 59_{10} = 111011_{(2)}$

Найти представление числа A в коде D1.

В данном случае n = 1, поэтому потребуются только константы $b_1^0 \div b_4^0$ и $b_1^1 \div b_4^1$, с помощью двоичных кодов которых за 8 шагов получим все восемь коэффициентов α_j^i.

```
          111011
шаг 1   −1010000
        ---------
         −010101  → α4^1 = 0
шаг 2    + 101000
         ---------
           10011  → α3^1 = 1
шаг 3    −  10100
         ---------
          −00001  → α2^1 = 0
шаг 4    +    1010
         ---------
            1001  → α1^1 = 1
шаг 5    −    1000
         ---------
            0001  → α4^0 = 1
шаг 6    −     100
         ---------
             −011  → α3^0 = 0
шаг 7    +      10
         ---------
              −01  → α2^0 = 0
шаг 8    +       1
         ---------
               00  → α1^0 = 1
```

Таким образом, $A_{(2-10)} = 0101\ 1001$.

Достоинством данного способа перевода является использование одних и тех же констант для прямого и обратного преобразований, что упрощает его аппаратурную реализацию. При этом перевод в D-код сводится, по сути дела, к выполнению операции деления с переменным делителем, а перевод из D-кода – к выполнению операции умножения с переменным множимым.

Контрольные вопросы к главе 6

1. Что такое код D1 и код D2?
2. Чем отличается сложение чисел в прямом коде D1 от сложения в прямом коде D2?
3. В чем состоят особенности сложения чисел в инверсных D-кодах?
4. Чем отличается двоичный сдвиг влево кода D1 от сдвига влево кода D2?
5. Как выполняется двоичный сдвиг кодов D1 и D2?
6. Какие способы умножения чисел в D-кодах Вы знаете?
7. Оцените время умножения чисел в D-кодах различными способами?
8. Какие способы деления чисел в D-кодах Вы знаете? В чем состоит их суть?
9. Оцените различные способы перевода чисел в D-кодах.

Глава 7 ВЫПОЛНЕНИЕ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ В СИСТЕМАХ СПЕЦИАЛЬНОГО НАЗНАЧЕНИЯ

В поисках путей упрощения алгоритмов арифметических операций либо ускорения их выполнения целесообразно рассмотреть некоторые другие системы счисления, такие, например, как системы с основанием –2, с основанием +2 и цифрами 1, $\overline{1}$, а также непозиционную систему счисления - систему остаточных классов (СОК) [53, 92].

7.1. Арифметические операции р системе счисления с цифрами 1, $\overline{1}$

Во второй- главе мы рассмотрели перевод прямого кода двоичных чисел с цифрами 0,1 в систему с цифрами 1, $\overline{1}$. Для выполнения арифметических операций необходимо уметь переводить дополнительный код двоичного числа в систему 1, $\overline{1}$ и наоборот.

Поэтому будем считать, что исходные числа заданы в обычной двоичной системе счисления дополнительными кодами в форме с естественной запятой. Необходимо найти их изображения в двоичной системе счисления с цифрами 1, $\overline{1}$. Будем сначала считать, что исходное число А положительно и его целая часть содержит n+1 разряд, а дробная - к разрядов. Если исходное число имеет меньше разрядов, то вначале дополняется нулями целая часть до n+1 разряда, а дробная – до k–1 разряда. Затем приписывается слева еще один разряд со значением 1, в результате чего количественный эквивалент нового числа станет равным $A' = 2^{n-1} + A$. Теперь все нули в изображении нового числа А' заменяются на $\overline{1}$, что равносильно вычитанию из числа А' числа $B = 2^{n-1} – A\cdot 2^{-k+i}$. В результате получим новое число А", такое, что $A'' = A' – B = 2A + 2^{-k+1}$. Для завершения перевода необходимо сдвинуть число А" на один разряд вправо, т.е. разделить на 2. Окончательно получим $C = A + 2^{-k}$. Именно такой количественный эквивалент имеет исходное число.

Пример: Задано $A = 10{,}1_2$.

Найти изображение А в системе счисления с цифрами 1, $\overline{1}$ при К = 3 и п = 3.

Вначале дописываем слева и справа необходимые нули и слева еще один разряд со значением 1. Получаем А' = 10010,10. Теперь заменяем

все нули в изображении A' на $\overline{1}$: A" = $1\overline{1}\overline{1}1\overline{1},1\overline{1}$. Сдвигаем число A" на один разряд вправо и получаем результат A = $1\overline{1}\overline{1}1,\overline{1}1\overline{1}$.

Так как дополнительные и прямые коды положительных чисел совпадают, то рассмотренный алгоритм может быть применен и для перевода прямых кодов.

Когда отрицательное число задано дополнительным кодом, его количественный эквивалент определяется как

$$A = -2^{n+1} + \sum_{i=-k}^{n} a_i \cdot 2^i = -2^{n+1} + A^*,$$

где a_i – значение основных (цифровых) разрядов дополнительного кода. При записи числа A дополнительным кодом будем считать, что целая часть исходного числа имеет столько же разрядов, сколько и искомого. Тогда для перевода числа A в двоичную систему счисления с цифрами 1, $\overline{1}$ вначале, по необходимости дополним дробную часть числа нулями до k–1 разрядов и знаковый разряд заменим на ноль. В результате этих действий получим новое число A' = A*. Затем все нули в числе A', включая знаковый разряд, заменим на $\overline{1}$, в результате чего получим число A":

$$A'' = -2^{n+1} + A^* - (2^{n+1} - A^* - 2^{-k+1}) - 2(-2^{n+1} + A^*) + 2^{-k+1} - 2A + 2^{-k+1}.$$

Теперь число A" сдвинем на один разряд вправо и получим окончательный результат B, т.е. исходное число, представленное в системе счисления с цифрами 1, $\overline{1}$ и имеющее значение A + 2^{-k}, количественный эквивалент которого равен A [53, 92].

Пример: В двоичной системе счисления с цифрами 0, 1 дополнительным кодом задано число –2,5 в форме с естественной запятой при k = 3 и n = 3, т.е. A = 11101,1. Найти изображение этого числа в системе счисления с цифрами 1, $\overline{1}$.

Вначале трансформируем исходную запись, заменяя знаковый разряд на 0 и дополняя справа запись нулями до k – 1 разрядов дробной части: A' = 01101,10. Затем все нули в записи этого числа заменим на $\overline{1}$, получим A" = $\overline{1}11\overline{1}1,1\overline{1}$. Далее уменьшим в два раза число A" и окончательно найдем, что искомое число B есть B = $\overline{1}11\overline{1},11\overline{1}$. Количественный эквивалент этого числа есть –2,5.

Таким образом, перевод чисел, заданных дополнительным кодом в двоичной системе счисления с цифрами 0,1, в двоичную систему с цифрами 1, $\overline{1}$ осуществляется путем выполнения логических преобразований над исходной записью в соответствии с одним из двух рассмот-

ренных алгоритмов. Обратный перевод, т.е. перевод чисел, изображенных в двоичной системе счисления с цифрами 1, $\overline{1}$, в обычную двоичную систему с положительной базой, может быть реализован путем выполнения логических преобразований, отвечающих прямому переводу, в обратном порядке.

Пример: В двоичной системе счисления с цифрами 1, $\overline{1}$ задано число – 2,5 в форме с естественной запятой при k = 3 и n = 3. Найти изображение этого числа в обычной двоичной системе счисления дополнительным кодом.

Итак, исходная запись есть B = $\overline{1}11\overline{1},11\overline{1}$. Для перевода вначале сдвинем влево исходную запись, получим число B' = $\overline{1}11\overline{1}1,1\overline{1}$. Затем все $\overline{1}$ заменим на нули, тогда B" = 01101,10. Теперь, поскольку исходное число имело знак "–", знаковому разряду припишем значение 1, окончательно получим A = $11101,1_2$. Количественный эквивалент получившегося числа есть –2,5 и оно представлено дополнительным кодом в двоичной системе счисления.

7.1.1 Алгебраическое сложение в системе счисления с цифрами 1, $\overline{1}$

Пусть операнды заданы в двоичной системе счисления с цифрами 1, $\overline{1}$ и требуется найти их сумму. Будем считать, что количественный эквивалент слагаемых определен соотношением (2+20). Следовательно, если мы найдем сумму двух слагаемых, то согласно (2.20) она будет увеличена по сравнению с ее реальным значением на 2^{-k}. Отсюда ясна необходимость коррекции результата сложения. При определении разрядной суммы и переноса надо иметь в виду, что слагаемые имеют только ненулевые значения, а перенос может иметь значение как +1, так и –1. Если считать, что для значений поразрядной суммы и переноса допустимы значения 1, 0, $\overline{1}$, то таблица сложения в двоичной системе счисления с цифрами 1, $\overline{1}$ примет вид табл. 7.1.

Таблица 7.1.

a_i	b_i	$П_{i-1}$	S_i	$П_i$	a_i	b_i	$П_{i-1}$	S_i	$П_i$	a_i	b_i	$П_{i-1}$	S_i	$П_i$
$\overline{1}$	$\overline{1}$	0	0	$\overline{1}$	$\overline{1}$	$\overline{1}$	$\overline{1}$	$\overline{1}$	$\overline{1}$	$\overline{1}$	$\overline{1}$	1	$\overline{1}$	0
$\overline{1}$	1	0	0	0	$\overline{1}$	1	$\overline{1}$	$\overline{1}$	0	$\overline{1}$	1	1	1	0
1	$\overline{1}$	0	0	0	1	$\overline{1}$	$\overline{1}$	$\overline{1}$	0	1	$\overline{1}$	1	1	0
1	1	0	0	1	1	1	$\overline{1}$	1	0	1	1	1	1	1

Характерной особенностью сложения является то, что значение разрядной суммы отлично от нуля только в том случае, если в предыдущем разряде был выработан перенос. Если же в предыдущем разряде перенос не был выработан, то разрядная сумма обязательно есть нуль. Легко заметить, что необходимая коррекция может быть произведена как присвоение переносу в младший разряд сумматора значения $\overline{1}$.

Пример: Найти сумму двух чисел A и B, представленных в двоичной системе счисления с цифрами 1, $\overline{1}$ в форме с естественной запятой при к = 3 и п = 3, если A = 2 и B = –0,75.

Имеем A = $1\overline{1}\overline{1},\overline{1}\overline{1}\overline{1}$ и B = $\overline{1}111,1\overline{1}1$. Сложим эти числа по правилам табл. 7.1. Учитывая, что $\Pi_0 = \overline{1}$, получим A + B = 0010, $\overline{1}0\overline{1}$.

Результат сложения представлен в двоичной системе счисления с цифрами 1, 0, $\overline{1}$, поэтому необходимо еще перевести его в систему счисления с цифрами 1, $\overline{1}$. Сделаем этот перевод в соответствии с ранее рассмотренным в главе 2 алгоритмом – выделим вначале в изображении исходного числа конструкции вида 0...01 и 0...0$\overline{1}$, а затем заменим их на конструкции $1\overline{1}$...$\overline{1}\overline{1}$ и $\overline{1}$1...11. Например, если A + B = 0010, $\overline{1}0\overline{1}$, то после перевода получим A + B = $1\overline{1}\overline{1}\overline{1},1\overline{1}1$, что в соответствии с (2.20) отвечает числу 1,25. Для ускорения операции сложения следует избавиться от этапа перевода, т.е. сразу получать результат в окончательном виде. Очевидно, для этого следует изменить таблицу сложения таким образом, чтобы исключить получение нулевых значений разрядной суммы.

Вне зависимости от того, как получен результат, его надо еще проанализировать для выполнения возможной коррекции. Это связано с тем, что наличие переноса из старшего разряда сумматора еще не означает переполнения сумматора. Только в том случае, если знаки переноса со старшего разряда сумматора и значения суммы в этом разряде совпадают, переполнение реально существует. Когда же знаки этих величин разные, го переполнения фактически нет и для коррекции результата надо отбросить значение переноса, а знак старшего разряда суммы изменить на противоположный.

Пример: Найти сумму двух величин A = 4 и B = 8,5, которые заданы в двоичной системе счисления с цифрами 1, $\overline{1}$ в форме с естественной запятой при k = 3 и n = 3.

Действуя по правилам (7.1), получим, что

$$S = A + B = 1\overline{1}1\overline{1},\overline{1}\overline{1}\overline{1} + 11\overline{1}\overline{1},\overline{1}1\overline{1} = 1\overline{1}11\overline{1},\overline{1}1\overline{1}.$$

Поскольку знаки переноса и старшего разряда суммы не совпадают, то необходима коррекция суммы, и после ее выполнения получим окончательно $S = 11\bar{1},\bar{1}1\bar{1}$.

Для осуществления операции вычитания чисел, представленных в двоичной системе счисления с цифрами 1, $\bar{1}$, надо вначале у вычитаемого числа поменять знаки во всех разрядах на противоположные, а затем сложить это число с другим операндом. Будем считать, что количественный эквивалент числа определен соотношением (2.20), тогда после инверсии знака у всех разрядов числа B и сложения его с числом A получим новое число S, численное значение которого меньше истинной величины A – B на 2, поэтому, как и при сложении, необходимо значение переноса в младший разряд сумматора сделать ненулевым, а именно $\Pi_0 = 1$.

Пример: Даны числа A = 3,25 и B = 12,75, представленные в двоичной системе счисления с цифрами 1, $\bar{1}$ и в форме с естественной запятой k = 3 и n = 3, т.е. $A = 1\bar{1}\bar{1}1,1\bar{1}1$ и $B = 11\bar{1},\bar{1}11$. Надо найти разность S = A – B.

Вначале инвертируем знаки во всех разрядах числа B, тогда $B' = \bar{1}\bar{1}\bar{1}1,1\bar{1}\bar{1}$. Затем сложим A и B' по правилам (7.1) с учетом $\Pi_0 = 1$, получим $S = \bar{1}\bar{1}11,\bar{1}1\bar{1}$. Количественный эквивалент результата согласно (2.20) есть C = A – B = –9,5. Основное достоинство сложения чисел в этой двоичной системе счисления состоит в отсутствии анализа знаков операндов. В то же время, здесь усложнены определение переполнения сумматора, операции сдвига чисел и округления.

Хотя технически циклический сдвиг осуществляется точно так же, как и в случае обычной системы счисления, реализации нециклического сдвига имеет свои особенности. При сдвиге вправо старший разряд инвертируется, затем все число сдвигается вправо на t разрядов. Разряды, вышедшие за разрядную сетку, теряются, а освободившиеся слева разряды заполняются новым (инвертированным) значением старшей цифры исходного числа. После завершения сдвига значение старшего разряда вновь инвертируется.

Пример: Число $A = 11\bar{1}\bar{1}\bar{1},\bar{1}\bar{1}\bar{1}$ ($A = 16_{10}$) сдвинуть на три разряда вправо, сохранив положение запятой.

Вначале получаем $A = \bar{1}1\bar{1}\bar{1}\bar{1},\bar{1}\bar{1}\bar{1}$, затем $A'' = \bar{1}\bar{1}\bar{1}\bar{1}1,\bar{1}\bar{1}\bar{1}$ и, наконец, окончательно $A''' = 1\bar{1}\bar{1}\bar{1}1,\bar{1}\bar{1}\bar{1}$. Полученное значение A''' есть 2.

Сдвиг влево должен быть произведен с учетом возможной свертки разрядов, выходящих за разрядную сетку. С этой целью анализируется

очередной разряд, вышедший за разрядную сетку, и если он совпадает со старшим из оставшихся разрядов, то никаких изменений не производится, если же он имеет противоположный знак, то старший разряд сдвинутого числа инвертируется. Освобождающиеся справа разряды должны быть заполнены цифрами $\overline{1}$. Действительно, при сдвиге влево на t разрядов из равенства (2.20) получим

$$A' = 2^t A = 2^t \sum_{i=-k}^{n} a_i \cdot 2^i + 2^{t-k}.$$

Если на место освободившихся справа разрядов записать t цифр $\overline{1}$, то значение A" станет равным $A' - (2^t - 1)2^{-k}$, т.е. $A = 2^t A + 2^k$.

Пример: Число $A = 1\overline{1}\overline{1}1,\overline{1}\overline{1}\overline{1}$ сдвинуть влево на три разряда.

После 1-го сдвига имеем $A' = 1\overline{1}\overline{1}\overline{1}1,\overline{1}\overline{1}\overline{1}$, следовательно, в пределах заданного количества разрядов сдвига число A' должно иметь вид $A'' = 1\overline{1}\overline{1}\overline{1},\overline{1}\overline{1}\overline{1}$. Действуя подобным образом и далее, получим окончательно $A'' = 11\overline{1}\overline{1}\overline{1},\overline{1}\overline{1}\overline{1}$. Количественный эквивалент числа A" есть, таким образом, 16, а численный эквивалент исходного числа был 2.

Хотя выполнение сложения в двоичной системе счисления и встречает определенные трудности, все же ради справедливости следует сказать, что эти затруднения преодолеваются весьма просто и в целом операция сложения оказывается близкой к этой же операции в обычной системе счисления при использовании дополнительных кодов.

7.1.2. Умножение чисел в системе счисления с цифрами 1, $\overline{1}$

Умножение чисел, представленных в двоичной системе счисления с цифрами 1, $\overline{1}$ и удовлетворяющих соотношению (2.20), не вызывает принципиальных трудностей, но следует иметь в виду, что полученный результат $C = A \cdot B$ будет иметь величину $C = A \cdot B + 2^{-k}(A + B) + 2^{-2k}$. Приведение его к принятой для операндов форме потребует дополнительных действий, что усложняет алгоритм умножения и увеличивает время его реализации. В то же время, не видно никаких особых преимуществ операции умножения, когда оба операнда заданы в системе счисления с цифрами 1 и $\overline{1}$. Однако совсем иначе обстоит дело, когда запись в системе с цифрами 1, $\overline{1}$ используется в качестве промежуточного представления. В главе 4 мы видели, что выполнение умножения в дополнительных и особенно, обратных кодах усложнено по сравнению с умножением в прямых кодах. Эти усложнения вынуждают переводить обратные коды в прямые и в них выполнять операцию, а затем вновь переходить к обратным кодам. Для облегчения выполнения операции

умножения в дополнительных и обратных кодах имеет смысл рассмотреть варианты с предварительным преобразованием множителя, в систему счисления с цифрами 1, $\overline{1}$. При этом фактически вычисленное значение произведения окажется $A \cdot B + 2^{-k}A$, но вычитание числа $2^{-k}A$ из произведения осуществить просто, так как реализация умножения практически всегда производится как операция $A \cdot B + D$. Отсюда достаточно перед началом операции в сумматор поместить величину –A. Обычное выполнение операции по первой схеме умножения приведет к автоматическому вычитанию величины $2^{-k}A$ из произведения, т.е. обеспечит получение правильного результата. При новом представлении множителя процесс умножения обычен и на каждом шаге производится либо прибавление множимого к сумме частных произведений (когда цифра множителя +1), либо его вычитание (цифра множителя –1). Результат получается сразу же в дополнительном коде [53, 92].

Пример: Даны величины B = –3/8 и A = 5/8, представленные в двоичной системе счисления модифицированными дополнительными кодами, т.е. $[B]_м$= 11,101 и $[A]_м$ = 00,101. Найти их произведение C = AB с помощью промежуточного представления множителя в системе счисления с цифрами 1, $\overline{1}$.

Вначале представим множитель в системе с цифрами 1, $\overline{1}$. Действия по обычным правилам, найдем, что B = $\overline{1}$,11$\overline{1}$1. 1-й шаг 1,011 - начальное значение суммы частных произведений (загрузка –A) СМо,

1,011 – учет веса цифры множителя b_{-4}
+ 0,101+ множимое A
0,000 CM_1,
2-й шаг 0,000 – учет веса цифры множителя b_{-3}, $p_{-7} = 0$
– 0,101 – множимое A
1,011 CM_2,
3-й шаг 1,101 – учет веса b_{-2}, $p_{-6} = 1$
+ 0,101 + множимое A
0,010 CM_3,
4-й шаг 0,001 – учет веса b_{-2}, $p_{-5} = 0$
+ 0,101 + множимое A
0,110 CM_4,
5-й шаг 0,011 - учет веса $b_{зн}$, $p_{-4} = 0$
– 0,101 - множимое A
1,110 CM_5, – старшие разряды произведения.

Окончательно имеем $[C]_д$= 1,1100010.

Таким образом, преобразование множителя, представленного дополнительным кодом, в системе счисления с цифрами 1, $\overline{1}$ позволяет умножение дополнительных кодов произвести точно так же, как умножение прямых кодов. При этом результат сразу же получается в исходной форме, а преобразование множителя может быть произведено в процессе анализа его цифр, т.е. пет необходимости предварительно осуществлять преобразование, достаточно множитель проинтерпретировать в двоичной системе с цифрами 1, $\overline{1}$.

7.1.3. Деление чисел в системе счисления с цифрами 1, $\overline{1}$

Прямая реализация операции деления чисел, представленных в двоичной системе счисления с цифрами 1, $\overline{1}$, не имеет смысла, ибо частное от деления числа A на число B будет иметь значение $Y = \frac{A + 2^k}{B + 2^{k'}}$, которое при малых A и B может быть весьма далеким от истинной величины A/B. Наиболее рациональным представляется применение рассматриваемой системы счисления для изображения частного от деления чисел, представленных в обычной двоичной системе счисления прямыми, дополнительными или обратными кодами. В этом случае цифра частного на каждом шаге деления есть 1 (если уменьшаемое имеет знак "+") либо $\overline{1}$ (если оно имеет знак "–"). Перевод частного из системы счисления с цифрами 1, $\overline{1}$ в исходную производится по известным правилам [92].

Промежуточное представление частного в системе счисления с цифрами 1, $\overline{1}$ имеет ряд особенностей. Во-первых, цифра частного определяется знаком уменьшаемого, а не остатка, как обычно. Во-вторых, результат сразу получается в дополнительном коде и представлен в системе счисления с цифрами 1, $\overline{1}$.

Таким образом, исследование особенностей двоичной системы счисления с цифрами 1, $\overline{1}$ показывает, что наиболее рационально использовать эту систему как промежуточную при выполнении операций умножения и деления в дополнительных и обратных кодах. В то же время возможна и прямая реализация арифметических действий в системе счисления с цифрами 1, $\overline{1}$, которая, однако, оказывается сложнее, особенно при сложных операциях, таких как умножение, деление.

7.2. Арифметические операции в минус-двоичной системе счисления

Особенностью этой системы счисления является представление положительных чисел единым кодом. Знак числа при этом определяется четностью или нечетностью старшей значащей цифры в его изображении, а количественный эквивалент следующим соотношением

$$A = \sum_{i=-k}^{n} a_i \cdot (-2^i).$$

При количестве разрядов m = n + 1 + k абсолютная величина числа A не может быть больше, чем [92]:

$$\max|A| = 3^{-1} \cdot 2^{-k}[2^{m+1} + \delta(m) - 2], \qquad (7.2)$$

где $\delta(x) = 1 - (-1)^x/2$ – для целого x. Наибольшее значение абсолютной величины числа противоположного знака A определяется при подстановке m = m – 1 в (7.2). Тогда, учитывая, что $\delta(m-1) = 1 - \delta(m)$, получим

$$\max|A| = 3^{-1} \cdot 2^{-k}[2^m - \delta(m) - 1]. \qquad (7.3)$$

Как следует из выражений (7.2) и (7.3), диапазон представления положительных и отрицательных чисел при фиксированной длине машинных слов отличается приблизительно в два раза.

Прежде чем приступить к выполнению арифметических действий, необходимо научиться переводить числа из двоичной в минус-двоичную систему счисления и наоборот. При переводе из двоичной системы счисления можно использовать тот факт, что количественный эквивалент числа A в минус-двоичной системе счисления есть разность сумм количественных эквивалентов разрядов с четными (A’) и нечетными (A”) номерами, т.е. A = A’ – A”, а количественный эквивалент числа в двоичной системе составит A = A’ + A”. Если рассматривать исходное двоичное положительное число B в качестве нового минус-двоичного числа A, то получим A = B’ – B”. Так как количественные эквиваленты исходного и переведенного чисел должны быть равны, то к значению A = B’ – B” надо прибавить в минус-двоичном коде величину 2B”. Таким образом, для перевода положительных двоичных чисел в минус-двоичную систему счисления необходимо из исходного слова B сформировать новое слово B’, у которого разряды с нечетными номерами совпадают с исходным, а разряды с четными номерами нули. Затем из слова B’ путем сдвига на один разряд влево получить слово B” и прибавить его в минус двоичной системе счисления к исходному слову B.

Пример: Задано $B = 11{,}125_{10} = 1011{,}001_2$. Найти его представление в минус-двоичной системе счисления.

Формируем B^1, у которого все четные разряды нули, а нечетные совпадают со значением нечетных разрядов числа В, т.е. В’ = 1010,001. Сдвигаем В’ на один разряд влево и получаем В” = 10100,010. После сложения В’ и В” получим А = 11111,011. Для проверки переводим в десятичную систему счисления:

$$A = 1(-2)^4+1(-2)^3+1(-2)^2+1(-2)^1+1(-2)^0+0(-2)^{-1}+1(-2)^{-2}+1(-2)^{-3} = 11{,}25.$$

При переводе отрицательного числа, заданного прямым кодом, необходимо из исходного сформировать новое число, у которого все нечетные разряды есть нули, а четные разряды совпадают с четными разрядами исходного кода. Затем это число надо сдвинуть на один разряд влево и сложить с исходным по правилам минус-двоичной арифметики.

Для отрицательных чисел, заданных в дополнительном коде, перевод основных разрядов производится по алгоритму перевода положительных чисел, а знакового – по правилу перевода отрицательных чисел, заданных прямым кодом. Это обстоятельство связано с тем, что количественный эквивалент дополнительного кода числа, как отмечалось в главе 3, есть сумма положительного и отрицательного чисел.

Пример: Задано $[B]_д$ = 1,0011. Перевести его в минус-двоичную систему счисления.

Формируем из знакового и основных разрядов два слова В’ = 1,0000 и В” = 0,0011. Переводим раздельно эти числа по правилам перевода, соответственно, отрицательных и положительных чисел и получаем А’ = 11,0000 и А” = 0,0111. Затем определяем А = 11,0111 путем сложения чисел А’ и А”. Количественный эквивалент этого числа равен –13/16 переведенного числа, как и исходного, равен –13/16.

Следует отметить, что для представления двоичного числа, имеющего n основных и один знаковый разряд, в минус-двоичной системе счисления на основании (7.2) и (7.3) потребуется n+2 разряда.

Для обратного перевода в двоичную систему необходимо исходное слово А представить в виде двоичного слова А’, у которого четные разряды совпадают с четными разрядами, а нечетные равны 0, и двоичного слова А”, у которого нечетные разряды совпадают с нечетными разрядами исходного, а четные есть нули. Затем из слова А’ вычесть слово А” и получить двоичный эквивалент числа А т.е. В = А’ – А”.

Если действия необходимо выполнять в минус-двоичной системе, то исходное положительное число А вначале рассматривают как готовый результат перевода его в двоичную систему. Очевидно, что

тогда B = A’ + A”, в то время, как A = A’ – A”, Чтобы количественные эквиваленты A и B были одинаковы, необходимо из A вычесть удвоенное значение суммы всех нечетных разрядов, т.е. B = A – 2A”.

Пример: Задано A = 10111 = 19_{10}, перевести его в двоичную систему счисления.

Вначале из нечетных разрядов A формируем A’ = 00010, сдвигаем его на один разряд влево и получаем A” = 00100. Затем из A вычитаем A”, т.е. B = 10111 – 00100=10011 = 19_{10}.

Если исходное число A отрицательно, то для получения прямого кода двоичного числа необходимо из A вычесть в минус-двоичной системе счисления удвоенное значение четных разрядов. Для получения дополнительного кода надо из исходного числа A вычесть удвоенное значение всех нечетных разрядов, кроме самого старшего, определяющего знак.

Пример: Задано A = 111111 = -21_{10}. Найти его изображение дополнительным кодом в двоичной системе.

Формируем слово A’ из всех нечетных разрядов, кроме старшего: A’ = 001010, сдвигаем A’ на разряд рлево: A” = 010100. Вычитаем A” из A и получаем $[B]_д$ =1.01011, т.е. B = -21_{10}.

7.2.1. Алгебраическое сложение в минус-двоичной системе счисления

Специфика сложения в минус-двоичной системе обусловлена тем, что веса соседних разрядов имеют разные знаки. Поэтому единица переноса из предыдущего разряда должна вычитаться из значения суммы данного разряда, а не прибавляться к нему. Если при этом данные разряды обоих слагаемых равны 0_f то значению разрядной суммы присваивается величина 1, а значению переноса — величина +1. Добавление по единице в два соседних разряда равносильно вычитанию единицы у младшего из них, так как знаки весов соседних разрядов противоположны. С учетом изложенного поразрядное сложение в минус-двоичной системе счисления будет иметь такой вид, как в таблице 7.2.

Как видно из таблицы, при сложении в минус-двоичцой системе необходимо определять значение переноса двух типов: переноса –1 ($П_i^-$) и +1 ($П_i^+$), что усложняет операцию. Перенос –1 в i-й разряд можно заменить на два переноса со значением +1 в i-й и (i+1)-й разряды, так как добавление по единице в два соседних разряда числа в минус-двоичной системе счисления эквивалентно вычитанию 1 из младшего разряда.

Таблица 7.2.

a_i	b_i	$П_{i-1}$	S_i	$П_i$
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	−1
0	0	−1	1	+1
0	1	−1	0	0
1	0	−1	0	0
1	1	−1	1	0
0	0	+1	1	0
0	1	+1	0	−1
1	0	+1	0	−1
1	1	+1	1	−1

В этом случае в каждом разряде необходимо формировать два переноса $П_i$ и d_i с учетом которых образуются значения разрядных сумм. Таблица 7.3 сложения примет при этом следующий вид:

Таблица 7.3.

a_i	b_i	$П_{i-1}$	d_{i-2}	S_i	$П_i$	d_i
0	0	0	0	0	0	0
0	1	0	0	1	0	0
1	0	0	0	1	0	0
1	1	0	0	0	1	1
0	0	1	0	1	0	0
0	1	1	0	0	1	1
1	0	1	0	0	1	1
1	1	1	0	1	1	1
0	0	0	1	1	0	0
0	1	0	1	0	1	1
1	0	0	1	0	1	1
1	1	0	1	1	1	1
0	0	1	1	0	1	1
0	1	1	1	1	1	1
1	0	1	1	1	1	1
1	1	1	1	0	0	1

При вычитании в минус-двоичной системе счисления из числа А числа В можно вначале поменять знак числа В на противоположный, а затем прибавить его к числу А. Так как все разряды числа В при этом должны иметь противоположный знак, то для сохранения количественного эквивалента необходимо к исходному числу В прибавить 2В, т.е. число В, сдвинутое на один разряд влево. В результате получим исходное число В, но с противоположным знаком.

Пример: Задано $A = 0{,}10101_{(-2)} = -21/32_{(10)}$ Найти –А в минус-двоичной системе.

Вначале удваиваем число А и получаем А’ = 1,01010. Складываем А и А’ и получаем –А =1,11111, количественный эквивалент которого составляет +21/32.

Таким образом, операция вычитания в минус-двоичной системе счисления при отмеченном способе изменения знака операнда выполняется за два такта сложения, что является недостатком. Недостаток также в более сложной по сравнению с двоичной системой процедуре определения переполнения разрядной сетки и округления результата.

7.2.2. Умножение и деление чисел в минус-двоичной системе счисления

Умножение чисел в минус-двоичной системе счисления выполняется та кже, как и умножение прямых кодов положительных чисел в двоичной системе счисления, т.е. путем сложения всех частных произведений вида $A \cdot b_i \cdot (-2)^i$, где b_i – цифры множителя. При этом учитывается, что сдвиг числа в минус-двоичной системе счисления не только изменяет вдвое его величину, но и знак на противоположный. Произведение получается с правильным знаком, так как знаки операндов определены весом старшей значащей цифры, т.е. анализ знака операндов не требуется. При этом, если операнды имели по n разрядов, то произведение согласно (7.2) будет иметь 2n + 3 разряда, что обусловлено спецификой представления чисел в минус-двоичной системе счисления.

Пример: Заданы $A = 111_{(-2)} = 3_{10}$; $B = 1101_{(-2)} = -3_{10}$. Найти А·В.

1-й шаг	0000	СМ
	+ 1101	$A \cdot b_0$
	1101	СМ
2-й шаг	01101	СМ →
	+ 1101	$A \cdot b_1$
	00111	СМ
3-й шаг	000111	СМ →
	+ 1101	$A \cdot b_2$
	001011	СМ

Результат равен 1011, его количественный эквивалент составляет –9. Как видно из примера умножение в минус-двоичной системе даже проще, чем в обычной двоичной за счет отсутствия этапа формирования знака результата.

Операция деления в минус-двоичной системе счисления является наиболее сложной, так как на четных и нечетных шагах алгоритма должны формироваться цифры частного разных знака. Для упрощения алгоритма вначале формирует частное в обычной двоичной системе с помощью алгоритма деления без восстановления остатка. Затем, по мере получения цифр частного, производят его перевод из двоичной в минус-двоичную систему счисления [92]. Однако даже в этом случае выполнение операции деления в минус-двоичной системе счисления существенно сложнее, чем в обычной двоичной и поэтому рассматриваться не будет.

Таким образом, в минус-двоичной системе счисления просто выполняются операции сложения и умножения, а реализация операций вычитания и деления вызывает определенные трудности. Правда, трудности в реализации операции деления могут быть преодолены, например, путем использования итерационных схем деления, базирующихся на умножении.

7.3. Арифметические операции в системе остаточных классов

При однозначном изображении диапазон допустимых в СОК чисел определяется соотношением [1]:

$$0 \le A \le M-1,$$

где $M = S_1 \times S_2 \ldots S_{n-1} \times S_n = \prod_{i=1}^{n} S_i$ – произведение всех весов разрядов, называемое периодом СОК

Для доказательства этого положения возьмем взаимно простые числа S_1, S_2,...S_g, S_p, S_r, S_t, при этом из трех последних образуем модули, имеющие общий делитель: $S_{pr} = S_p \cdot S_r$ и $S_{rt} = S_r \cdot S_t$. Для двух чисел A и B, представленных в СОК можно тогда записать:

$$A = m_1S_1+a_1 = m_2S_2+a_2 = \ldots = m_gS_g+a_g = m_{pr}S_pS_r+a_{pr} = m_{rt}S_rS_t + a_{rt}.$$
$$B = n_1S_1+b_1 = n_2S_2+b_2 = \ldots = n_gS_g+b_g = n_{pr}S_pS_r+b_{pr} = n_{rt}S_rS_t + b_{rt}.$$

Предположим, что числа A и B записаны в СОК одинаково, т.е. представлены одинаковыми цифрами. Тогда $a_1 = b_1$; $a_2 = b_2$; … $a_g = b_g$; $a_{pr} = b_{pr}$; $a_{rt} = b_{rt}$. Следовательно:

$$A - B = (m_1 - n_1)\cdot S_1 = (m_2 - n_2)\cdot S_2 = ... = (m_g - n_g)\cdot S_g = \\ = (m_{pr} - n_{pr})\cdot S_p S_r = (m_{rt} - n_{rt})\cdot S_r S_t.$$

Так как все S_i есть взаимно простые числа, то эти равенства могут быть удовлетворены только при условии:

$$\begin{aligned} m_1 - n_1 &= K_1 S_1 S_2 ... S_i ... S_g S_p S_r S_t, \\ m_2 - n_2 &= K_2 S_1 S_2 ... S_i ... S_{g-1} S_p S_r S_t, \\ &\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ m_g - n_g &= K_g S_1 S_2 ... S_i ... S_g S_p S_r S_t, \\ m_{pr} - n_{pr} &= K_{pr} S_1 S_2 ... S_i ... S_g S_t, \\ m_{rt} - n_{rt} &= K_{rt} S_1 S_2 ... S_i ... S_g S_p, \end{aligned}$$

где $K_1, K_2, ..., K_g, K_{pr}, K_{rt}$ – какие-то целочисленные коэффициенты.

Отсюда следует:

$$A - B = K_1 S, S_2 ... S_i ... S_g S_r S_p S_t = K_2 S_1 S_2 ... S_i ... S_g S_r S_p S_t = \\ = .. . = K_g S_1 S_2 .. . S_i ... S_g S_r S_p S_t = K_{rp} S_1 S_2 .. . S, .. . = S_g S_r S_p S_t = \\ = K_{rt} S_1 S_2 ... S_i ... S_g S_r S_p S_t = K_1 \prod_{i=1}^{n} S_i = K_2 \prod_{i=1}^{n} S_i = ... = K_g \prod_{i=1}^{n} S_i = \\ = K_{pr} \prod_{i=1}^{n} S_i = K_{rt} \prod_{i=1}^{n} S_i = ... = K_g \prod_{i=1}^{n} S_i ,$$

так как очевидно, что $K_1 = K_2 = ... = K_g = K_r = K_p = K_t = K$.

Поэтому $A - B = K \prod_{i=1}^{n} S_i$ или $A = B + K \prod_{i=1}^{n} S_i$. При $K = 0$ имеем $A = B$. При $K = 1$, $A = B + \prod_{i=1}^{n} S_i$, если при этом и $B = 0$, то $A = \prod_{i=1}^{n} S_i = M$. Таким образом, произведение взаимно простых чисел (модулей) S_i, т.е. $\prod_{i=1}^{n} S_i$ определяет диапазон однозначно представимых чисел от 0 до $\prod_{i=1}^{n} S_i = M$. Числа, отличающиеся друг от друга на величину $M = \prod_{i=1}^{n} S_i$, представляются в СОК одинаково. Далее, не имеет смысла брать модули, имеющие общий делитель, так как при этом не происходит увеличения диапазона представимых чисел. Количество разрядов, однако, уменьшится, но эти разряды будут иметь большее основание, что не выгодно из-за усложнения схем.

Следует отметить, что в СОК необязательно в качестве модулей брать простые числа в их естественной последовательности – 2, 3, 5, 7 и т.д. Для обеспечения достаточно большего диапазона представимых

чисел необходимо выбирать модули — основания достаточно большие по величине, причем только один из них может быть четным. Например, если взять модули 38, 41, 43, 53, 59, то будет обеспечено представление чисел в диапазоне 0 : 209 490 237.

СОК допускает расширение или сокращение набора оснований, не искажая при этом исходное число. Так, если в СОК с весами S_1, S_2, S_3 число A = (a_1, a_2, a_3), то введя новые веса S_4 и S_5, то же A изображается в виде

$$A = (a_1, a_2, a_3, a_4, a_5).$$

Аналогичным образом можно сократить набор весов. При этом, естественно, уменьшится диапазон представимых чисел. При расширении набора весов диапазон представления увеличивается.

Знак числа в СОК может быть задан несколькими способами. Обычно отрицательное число –A представляют как дополнение его модуля до M, т.е, –A = M –|A|. Так как M = O(modS_i), то знак числа можно изменить путем вычитания заданного числа из нуля, т.е. путем вычитания i-го вычета из нуля с последующим его дополнением до S_i по mod S_i. Тогда в качестве положительных чисел можно взять числа натурального ряда, лежащие на обрезке [1, M/2–1], а в качестве отрицательных - на отрезке [M/2, M–1], т.е. интервал представления целых чисел со знаком составляет:

$$-M/2 \le A < M/2.$$

Пример: Для S_1 = 2, S_2 = 3, S_3 = 5 требуется представить числа A = +12 и B = –10.

Так как M = 2·3·5 = 30, то диапазон представления положительных чисел есть 1…14, а отрицательных — –15…–1. Число –B представляется в виде дополнения |B| до M, т.е. B* = M –|B|. Тогда A = (0, 0, 2), B = (0, 2, 0). Если A = 12, т.е. (0, 0, 2), то –A получим следующим образом; (0, 0; 0) – (0, 0, 2) = (0, 0, –2) = (0, 0, 3).

В СОК также как и счет поразрядными являются арифметические операции сложения, вычитания и умножения. Они базируются на следующих свойствах сравнений.

1. Сравнения можно почленно складывать. Если

$$A_1 \equiv a_1 (\mathrm{mod}\ S),$$
$$\vdots$$
$$A_{to} \equiv a_n (\mathrm{mod}\ S),$$

то с учетом того, что $A_i = K_i \cdot S + a_i$, получим: —

$$A_1 + A_2 + ... + A_n \equiv a_1 + a_2 + ... + a_n (\text{mod } S).$$

Отсюда следует, что слагаемое, стоящее в какой-либо части сравнения, можно переносить в другую часть, поменяв при этом его знак, т.е.

$$A + B \equiv C (\text{mod } S),$$

или

$$A_1 \equiv a_1 (\text{mod } S),$$

2. Два числа, сравнимые с третьим числом, сравнимы и между собой если

$$A \equiv B (\text{mod } S),$$
$$B \equiv C (\text{mod } S),$$

то

$$A \equiv C (\text{mod} S).$$

3. Сравнения можно почленно перемножать. Пусть

$$A_1 \equiv a_1 (\text{mod } S),$$
$$A_2 \equiv a_2 (\text{mod } S).$$

Это можно представить в алгебраической форме:

$$A_1 = a_1 + K_1 \cdot S,$$
$$A_2 = a_2 + K_2 \cdot S.$$

После умножения получаем:

$$A_1 \times A_2 = a_1 \times a_2 + NS,$$

где $N = a_1K_2 + a_2K_1 + K_1K_2S$ — частное от деления произведения $A_1 \times A_2$ на S. В общем случае имеем:

$$A_1 A_2 ... A_n \equiv a_1 a_2 ... a_n (\text{mod } S).$$

Отсюда следует, что обе части сравнения можно умножать на одно и то же целое число.

Если $A \equiv a (\text{mod } S)$, $C \equiv c (\text{mod } S)$, то $AC \equiv ac (\text{mod } S)$.

4. Из свойства 3 следует также, что сравнения можно возводить в степень. Если $A \equiv a (\text{mod } S)$, то $A^n \equiv a^n (\text{mod } S)$.

Значение этих свойств состоит в том, что при рассмотрении различных арифметических выражений входящие в эти выражения числа можно заменять на другие, сравнимые с ними по данному модулю S. В частности, каждое число может быть заменено своими вычетами. Поэтому в СОК арифметические операции сложения, вычитания и умножения являются поразрядными.

При этом

$$A + B(\text{mod}S) \equiv \begin{cases} a + b, & \text{если } a+b < S, \\ a + b - S, & \text{если } a+b \geq S, \end{cases}$$

$$A + B(\text{mod}S) \equiv \begin{cases} a - b, & \text{если } a-b < S, \\ a - b + S, & \text{если } a-b \geq S, \end{cases}$$

$$A \times B(\text{mod } S) \equiv ab(\text{mod } S).$$

т.е. необходимо сначала произвести умножение вычетов, а затем деление произведения на S. Таким образом, остаток от деления суммы, разности и произведения двух чисел на S равен соответственно сумме, разности и произведению по модулю S остатков от деления на S исходных чисел.

Пример: Перевести числа $A = 5_{10}$ и $B = 2_{10}$ в СОК с $S_1 = 2$; $S_2 = 3$; $S_3 = 5$ и найти их сумму, разность и произведение.

по модулю 2: $a_1 = 5 - [5/2] \cdot 2 = 1$, $b_1 = 2 - [2/2] \cdot 2 = 0$,

по модулю 3: $a_2 = 5 - [5/3] \cdot 3 = 2$, $b_2 = 2 - [2/3] \cdot 3 = 2$,

по модулю 5: $a_3 = 5 - [5/5] \cdot 5 = 0$, $b_3 = 2 - [2/5] \cdot 5 = 2$,

т.е. $A_{СОК} = 120$; $B_{СОК} = 022$.

Сложение чисел: A+B = C

по модулю 2: $a_1 + b_1 = 1 + 0 = 1$,

по модулю 3: $a_2 + b_2 = 2 + 2 = 4 = 1$,

по модулю 5: $a_3 + b_3 = 0 + 2 = 2$,

те. $A_{СОК} + B_{СОК} = 120+022 = 112_{СОК} = 7_{10}$.

Вычитание чисел: A–B = C

по модулю 2: $a_1 - b_1 = 1 - 0 = 1$,

по модулю 3: $a_2 - b_2 = 2 - 2 = 0$,

по модулю 5: $a_3 - b_3 = 0 - 2 + 5 = 3$,

т.е. $A_{СОК} - B_{СОК} = 120 - 022 = 103_{СОК} - 3_{10}$.

Умножение чисел: Ax B = C

по модулю 2: $a_1 \cdot b_1 = 1 \cdot 0 = 0$,

по модулю 3: $a_2 \cdot b_2 = 2 \cdot 2 = 4 = 1$,

по модулю 5: $a_3 \cdot b_3 = 0 \cdot 2 = 0$,

т.е. $A_{СОК} \cdot B_{СОК} = 120 \cdot 022 = 010_{СОК} = 10_{10}$.

Перевод числа A из однородной позиционной системы счисления с основанием P в систему остаточных классов может производиться различными способами. Один из алгоритмов перевода состоит в том, что число A делится последовательно на модули S_i, каждый из которых представлен в p-ичной позиционной системе счисления. Деление производится по правилам p-ичной арифметики. Полученные значения

остатков a_i представляются также в р-ичной системе счисления. По другому алгоритму перевода все действия выполняются по правилам арифметики остаточных классов. Каждая цифра р-ичной системы счисления и основание р представляются в системе остаточных классов, т.е. $a_i = (\alpha_1^{(i)},\alpha_2^{(i)},..., \alpha_n^{(i)})$, $0 < a \le p-1$ и $p = (\alpha_1, \alpha_2,..., \alpha_n)$. Затем в общее выражение полинома, который задает значение числа А в однородной позиционной системе, подставляются эти значения и полином вычисляется по схеме Горнера по правилам умножения и сложения в системе остаточных классов.

Пример: Для числа А = 21 найти изображение в СОК с $S_1 = 2$; $S_2 = 3$; $S_3 = 5$.

Так как М = 30, то А не выходит из области допустимых чисел. По условию имеем: $a_0 = (1, 1, 1)$, $a_1 = (0, 2, 2)$, $p = 10 = (0, 1, 0)$. Для перевода необходимо вычислить полином вида

$$A_{СОК} = (0, 2, 2)\cdot(0, 1, 0)+(1, 1, 1) = (1, 0, 1).$$

Если последовательно определять вычеты от числа А = 21 по модулям 2, 3, 5, получим тот же результат.

При обратном переводе исходное число $A = (a_1,a_2,...,a_n)$ можно представить как сумму n чисел b_i, каждое из которых имеет только один ненулевой вычет a_i в i-м разряде, т.е. $A = (a_1,0,...,0)+(0,a_2,...,0)+...+(0, 0,...,a_n)$. Числа b представим как произведение а на число b_i, которое имеет только один единственный ненулевой вычет, занимающий i-й разряд. Тогда

$$A = a_1(1,0,...,0)+a_2(0,1,...,0)+...+a_n(0,0,...,1). \qquad (7.4)$$

Векторам, имеющим один единственный вычет, соответствуют определенные численные значения, которые заранее известны и представляются в р-ичной системе счисления также как и a_i. Тогда, выполнив все действия в соответствии с выражением (7.4) по правилам р-ичной арифметики, получим промежуточный результат перевода числа А из СОК в р-ичную систему счисления. Для получения окончательного результата необходимо найти вычет от полученного числа по модулю М.

Пример: Задано число А = (1, 0, 1) и В = (0, 2, 3) в системе вычетов по модулям $S_1 = 2$; $S_2 = 3$; $S_3 = 5$. Перевести это число в десятичную систему счисления.

Для данного случая М = 30, $a_1 = 1$, $a_2 = 0$, $a_3 = 1$, $b_1 = 0$, $b_2 = 2$, $b_3 = 3$. Численный эквивалент вектора $\beta_1 = (1, 0, 0)$ есть 15, вектора $\beta_2 = (0, 1, 0)$ есть 10, вектора $\beta_3 = (0, 0, 1)$ есть 6. Тогда согласно (7.4) получим:

$$A^* = 1\cdot15+0\cdot10+1\cdot6 = 21,$$
$$B^* = 0\cdot15+2\cdot10+3\cdot6 = 38.$$

Так как A* < M и B* > M = 30, то для получения B необходимо еще найти вычет B* по mod M. Тогда окончательно имеем: A = 21, B = 8.

Так как желательно, чтобы M было максимально возможно большим, можно модули выбирать следующим образом: S_1 – наибольшее нечетное число, соответствующее машинному слову, S_2 – наибольшее нечетное число < S_1, взаимно простое с S_1; S_3 –наибольшее нечетное число < S_2, взаимно простое с S_1 и S_2 и т.д. пока не, будет набрано количество S_i, достаточное для образования требуемого M.

В случае машины с двоичной арифметикой предпочитают в качестве модулей числа вида $S_i = 2^{Li} - 1$. Это упрощает выполнение основных арифметических действий и позволяет полностью использовать всю разрядную сетку, так как a_i может быть любым двоичным числом, состоящим из L_i бит. В этих условиях действия алгебраического сложения и умножения по всем mod S_i выполняются следующим образом:

$$a_i + b_i = \begin{cases} a_i + b_i, & \text{если } a_i + b_i < 2^{Li}, \\ (a_i + b_i) \bmod 2^{Li} + 1, & \text{если } a_i + b_i \geq 2^{Li}, \end{cases}$$

$$a_i \cdot b_i = (a_i \cdot b_i) \bmod 2^{Li} + \frac{a_i \cdot b_i}{2^{Li}}.$$

В случае получения отрицательного результата при вычитании можно к нему прибавить S_i. Для работы с модулями вида $2^{Li} - 1$ надо знать, при каких условиях число $2^L - 1$ взаимно просто с числом $2^f - 1$. Для этого существует простое правило, которое утверждает, что $2^L - 1$ и $2^f - 1$ являются взаимно простыми тогда и только тогда,когда L и f взаимно просты.

Достоинство СОК заключается в том, что результат, получаемый при сложении, вычитании и умножении цифр в любом из разрядов, не зависит от результатов аналогичных операций в других разрядах чисел. Следовательно, операции во всех разрядах можно выполнять одновременно и независимо друг от друга, что значительно увеличивает скорость вычислений по сравнению с позиционной системой счисления. Причем, они могут быть реализованы сравнительно простыми техническими средствами (например, таблицами), так как количество возможных остатков при делении числа на небольшое по величине основание S_i ограничено.

К основным недостаткам символических систем счисления вообще и СОК в частности относятся:

а) отсутствует удобный способ сравнения чисел, т.к. их поразрядная операция не дает требуемой информации;

б) затруднен перевод из позиционной системы счисления в СОК, еще сложнее обратный перевод;

в) отсутствует простой алгоритм деления;

г) усложнено выполнение операций, которые требуют округления результата;

д) отсутствует удобный способ определения переполнения разрядной сетки машины, т.к. все операции в СОК являются поразрядными.

СОК применяется в специализированных вычислительных машинах, для которых: 1) одним из основных требований является получение высокого быстродействия, 2) диапазон исходных чисел и промежуточных результатов строго фиксирован, 3) операция деления встречается крайне редко. Кроме того СОК применяется в ЭВМ для контроля цепей передачи и переработки информации.

Контрольные вопросы к главе 7

1. Как переводятся числа в систему 1, $\overline{1}$?
2. Как сложить два числа в системе 1, $\overline{1}$?
3. Как умножить два числа с использованием системы 1, $\overline{1}$?
4. Как переводятся числа в минус-двоичную систему счисления?
5. Какие особенности сложения чисел в минус-двоичной системе Вы знаете?
6. Как определить диапазон допустимых в СОК чисел?
7. Как представить в СОК отрицательные числа?
8. По каким правилам складываются, вычитаются и умножаются числа в СОК?
9. Какие способы перевода чисел в СОК и обратно Вы знаете? Оцените их трудоемкость.
10. Какие модули предпочтительнее для машин с двоичной арифметикой и почему?

Глава 8 АППАРАТНЫЙ КОНТРОЛЬ ВЫЧИСЛЕНИЙ В ЭВМ

Ошибки в работе ЭВМ обусловлены: 1) погрешностями задания исходных данных; 2) методическими погрешностями; 3) нарушениями функционирования ЭВМ.

Ошибка в работе ЭВМ третьего вида может быть систематической, возникающей в результате неисправностей, и случайной, возникающей в результате сбоев. Сбоями называют кратковременные нарушения правильной работы машины.

Появление неисправностей и сбоев практически неизбежно. В этих условиях задача повышения надежности работы ЭВМ может быть решена двумя путями: 1) увеличением надежности отдельных элементов, узлов и устройств ЭВМ; 2) посредством выявления и исправления ошибок.

Первый путь эффективен, но всегда ограничен достигнутыми на данный момент времени возможностями технологии.

Второй, путь связан с введением избыточности в перерабатываемую информацию. Избыточность может быть либо временной, либо пространственной. Временная избыточность связана с увеличением времени решения задачи (в частном случае задача может быть решена дважды) и вводится программным путем. Она является основой программного способа обнаружения и исправления ошибок.

Пространственная избыточность заключается в удлинении кодов чисел, в которые вводятся дополнительные (контрольные) разряды, и является основой схемного (аппаратного) способа обнаружения и исправления ошибок. Следовательно, аппаратный контроль основан на введении определенного вида избыточности при кодировании как рабочей, так и вспомогательной машинной информации. Рабочая информация – это та информация, которая непосредственно участвует в вычислительном процессе для получения окончательного результат. Вспомогательная информация – это вся командная информация, которая способствует автомата-ческому ведению вычислительного процесса (код операции, адреса чисел, признаки разного рода).

Рабочая и вспомогательная информация представляет собой тот минимум, который необходим для получения правильною результата на идеальной ЭВМ. Поэтому она называется основной машинной инфор-

мацией в отличие от избыточной, которая специально вводится только для обнаружения ошибок в работе ЭВМ.

Избыточная (контрольная) информация строится по отношению к основной по определенному закону. Основным логическим законом организации аппаратного контроля узлов ЭВМ является сравнение по модулю. Для его реализации в схему ЭВМ необходимо ввести дополнительное оборудование следующих 3-х видов:

1) дополнительные разряды для представления, хранения и передачи избыточной информации;

2) преобразователи для выполнения операций над контрольными кодами;

3) схемы проверки соответствия друг другу основной и избыточной информации.

Из-за большого разнообразия функций, выполняемых ЭВМ, не удается найти такой единый закон кодирования избыточной информации, который подходил бы для организации контроля всех узлов ЭВМ. Однако в ЭВМ имеются большие группы блоков и узлов, характеризующиеся общностью информационных процессов, контроль которых может быть организован по единому принципу. Эти группы следующие:

1. Пассивные цепи передачи и хранения информации, у которых ин формация на входе блока (до выполнения операции) во всех разрядах совпадает с выходной (носители информации ЗУ, регистры числового и командного трактов ЭВМ и т.д.).

2. Цепи переработки основной информации, выполняющие арифметические и логические операции в числовом и командном трактах (арифметические устройства, индексные регистры, счетчики, часть цепей центрального и местного устройства управления).

3. Цепи функциональных преобразований вспомогательной информации (обработки сигналов управления вычислительным процессом).

8.1. Контроль пассивных цепей

Контроль пассивных цепей производится с помощью посылочных кодов, которые по корректирующей способности делятся на две группы: коды, обнаруживающие ошибки, и коды, исправляющие ошибки.

Для четкого разграничения этих кодов рассмотрим несколько понятий. Кодовым весом W некоторой двоичной комбинации называется количество единиц, содержащееся в ней. Количество разрядов, в которых не совпадают двоичные цифры двух кодовых комбинаций, называют кодовым расстоянием d между этими комбинациями. В ЭВМ

кодовое расстояние определяется обычно поразрядным Сложением по mod2 исходных комбинаций с последующим определением кодового веса полученной суммы.

Минимальным кодовым расстоянием d_{min} двоичного кода, применяемого для обработки информации, называют самое малое кодовое расстояние, возможное между двумя любыми комбинациями этого кода.

В обычном двоичном n-разрядном коде возможны кодовые расстояния от 1 до n, т.е. $d_{min} = 1$.

Любой корректирующий код должен иметь минимальное кодовое расстояние $d_{minКОР} \geq 2$. Например, при d = 2 одиночная ошибка приведет к комбинации, запрещенной в данном коде, и, таким образом, будет выявлена.

В общем случае, когда в кодовой комбинации могут появиться ошибки любой кратности $i \leq n$ (обычная, двойная и т.д.) для обнаружения этой ошибки потребуется корректирующий код с минимальным кодовым расстоянием $d_{minОБН} = i+1$, а для исправления этой ошибки $d_{minИСПР} = 2i + 1$.

В машинах параллельного действия более вероятными являются одиночные ошибки, так как разрядные тракты узлов независимы, надежны и поэтому, одновременное нарушение работы в нескольких трактах маловероятно.

Обнаружение одиночной ошибки можно обеспечить добавлением к слову одного контрольного разряда. Диапазон кодовых комбинаций в этом случае увеличивается в два раза, и все комбинации можно разбить так, чтобы минимальное кодовое расстояние меаду любыми комбинациями было не менее двух. При этом в контрольный разряд записывается единица, если вес исходной комбинации является четным числом и нуль в противном случае. Тогда вес любой избыточной комбинации всегда является нечетным числом. Можно также записывать двоичную цифру в контрольный разряд, исходя из достижения четности веса избыточной комбинации. Оба варианта избыточного кода называют кодом с проверкой по паритету или на четность, так как исчезновение либо появление в комбинации ложной единицы в результате одиночной ошибки приводит к нарушению его четности.

Достоинства кода с проверкой по паритету:

1) относительно высокая корректирующая способность, так как он позволяет обнаружить все нечетные групповые ошибки, в том числе одиночные.

2) небольшая относительная избыточность.

Однако код с $d_{min} = 2$ не может исправить даже одиночную ошибку, так как любая запрещенная комбинация является равноотстоящей (на ± единицу) от двух разрешенных комбинаций.

Если считать, что в кодовых комбинациях возможны только одиночные ошибки и невозможны ошибки более высокой кратности, то методику исправления одиночной ошибки можно построить, развивая идею ею обнаружения. Методика состоит в следующем. Если имеется не один, а два контрольных разряда, то можно локализовать ошибку с точностью до полуслова, т.е. указать в какой половине кодовой комбинации она находится. Если же имеется несколько контрольных разрядов, то каждый из них можно использовать для обнаружения ошибки только в одном полуслове проверяемой кодовой комбинации. При этом полуслова формируются таким образом:

а) первое полуслово строится из второй половины разрядов проверяемого слова;

б) второе полуслово образуется из второй половины первого полуслова и из второй половины тех разрядов, которые не вошли в первое полуслово;

в) третье полуслово формируется из четверти разрядов, принадлежащих только первому полуслову, из четверти разрядов, принадлежащих только второму полуслову, из четверти разрядов, принадлежащих первому и второму полусловам и из четверти разрядов, не вошедших ни в первое, ни во второе полуслово. И так далее до тех пор, пока в каждом полуслове не останется по одному разряду, принадлежащему только этому полуслову.

Количество разбиений n-разрядного слова, т.е. количество полуслов определяется по формуле:

$$K \geq \log_2(n+1), \qquad (8.1)$$

(знак > вместо = учитывает возможность такого случая, когда log2(n+1) является нецелым числом).

Пример: Задано n = 15, поэтому К = 4. Разбить 15-разрядное слово на полуслова.

Результат разбиения приведен на рис. 8.1.

При правильном подборе полуслов контрольными будут разряды, имеющие номер 2^x. Число контрольных разрядов К при заданном количестве числовых разрядов m можно найти из соотношений (8.1) и (8.2.)

$$n = k+m, \qquad (8,2)$$

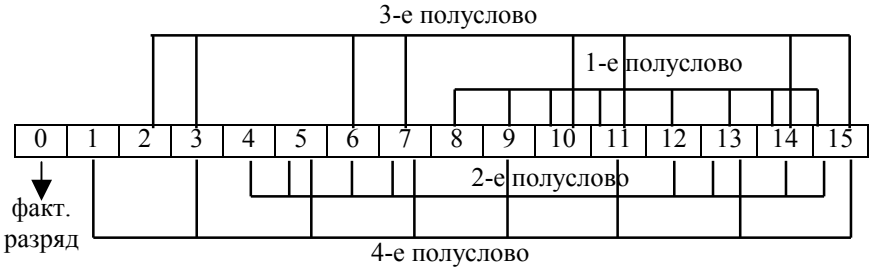


Рис. 8. 1. Разбиение слова на полуслова.

В результате их совместного решения получим:

$$2^k \geq m + 1.$$

Таким образом, если выделить отдельный контрольный К-разрядный регистр, то появившееся в нем в результате проверок на четность всех полуслов двоичное число определяет номер ошибочного разряда. Для коррекции возникшей ошибки необходимо выполнить сложение по mod 2 содержимого этого разряда с единицей. Если, например, проверка на четность показала наличие ошибок в 1-м, 3-м, 4-м полусловах, то ошибка произошла в том единственном разряде, который входит во все эти полуслова, т.е. в 11-м. Этот код был введен в 1950 г. Хеммингом. Поэтому его часто называют кодом Хемминга.

По методу Хемминга могут быть построены коды разной длины. При этом чем больше длина кода, тем меньше относительная избыточность. Например, для контроля числа, имеющего 54 двоичных разрядов, потребуется только шесть дополнительных (избыточных) разрядов. Коды Хемминга используют, в основном, для контроля накопителей ЗУ и при передаче информации по каналам связи (телеобработка, распределенные ВС, вычислительные сети и т.п.).

8.2. Контроль цепей переработки информации

Посылочные корректирующие кеды применяют для контроля правильности выполнения арифметических операций, так как у них не существует однозначной связи между контрольными разрядами исходных операндов и контрольными разрядами результата арифметической операции. Для этой цели служат арифметические корректирующие коды, которые, как и посылочные, подразделяются на две группы: коды, обнаруживающие ошибки, и коды, исправляющие ошибки.

Различают два метода получения контрольного кода: числовой и цифровой. При числовом методе контроля контрольный код заданного числа определяется как наименьший положительный остаток от деления числа А на выбранный модуль Р:

$$r = A - \{A/p\}p; \tag{8.3}$$

где $\{A/p\}$ – целая часть от деления числа А на р.

Величина модуля р существенно влияет на качество контроля. Если при числовой контроле р = q, т.е. оснований системы счисления, в которой представлено число, то контролируется только младший разряд числа и контроль как таковой не имеет смысла. Для $p = q^m$ справедливы аналогичные соображения, так как опять не все разряды числа (при $m < n$) участвуют в контроле и ошибки в разрядах старше m вообще не воспринимаются. Для числового метода контроля по modP справедливы основные свойства сравнений. Поэтому, если

$$A \equiv r_A (\text{mod } P);$$
$$B \equiv r_B (\text{mod } P),$$

где $0 \le r_A \le P-1$; $0 \le r_B \le P-1$, то

$$A+B \equiv r_A + r_B (\text{mod } P).$$

Откуда

$$r_{A+B} \equiv r_A + r_B (\text{mod } P). \tag{8.4}$$

Аналогичным образом доказывается справедливость и следующих соотношений:

$$r_{A-B} \equiv r_A - r_B (\text{mod } P). \tag{8.5}$$
$$r_{A.B} \equiv r_A \cdot r_B (\text{mod } P). \tag{8.6}$$

Пример: Заданы А = 125, В = 89, Р = 11. Найти r_A (mod P) и r_B(mod P).

$$r_A = 125 - \{125/11\}\cdot 11 \equiv 4 (\text{mod } 11),$$
$$r_A = 89 - \{89/11\}\cdot 11 \equiv 1 (\text{mod } 11).$$

Недостатком числового метода контроля по mod P является использование операции деления для определения остатка, что требует больших затрат машинного времени.

При цифровом методе контроля контрольный код числа образуется делением суммы цифр числа на выбранщлй модуль при выполнении условий

$$r'_A = \sum_i a_i - \left\{ \frac{\sum_i a_i}{p} \right\} \cdot P. \tag{8.7}$$

или

$$r'_A = \sum_i a_i (\mathrm{mod}\ P).$$

Возможны два пути получения контрольного кода: 1) непосредственное деление суммы цифр на модуль P; 2) суммирование цифр по mod P, Второй путь значительно проще, так как если $a_i < P$, то контрольный код получается только операцией суммирования. В этом заключается преимущество цифрового метода контроля перед числовым.

Пример: Заданы A = 159, B = 41, P = 11. Найти r'_A и r'_B.

$$r'_A = \sum_i a_i \equiv 4 (\mathrm{mod}\ 11),\ r'_B \equiv 5 (\mathrm{mod}\ 11),$$

Однако при цифровом методе свойства сравнений не всегда справедливы. Происходит это из-за наличия переносов (заемов) при выполнении арифметических действий над числами. Поэтому получение контрольного кода результата операции должно сопровождаться его коррекцией.

Допустим, заданы числа A и B и их контрольные коды

$$r'_A = \sum_i a_i (\mathrm{mod}\ P),\ r'_B = \sum_i b_i (\mathrm{mod}\ P),$$

а также C = A+B. Требуется найти контрольный код r'_C. При наличии результата C код r'_C находится по общему правилу:

$$r'_C = \sum_i c_i (\mathrm{mod}\ P).$$

Для сравнения контрольных кодов необходимо получать r'_C через контрольные коды слагаемых.

Сумму цифр результата можно найти, зная цифры a_i и b_i и количество переносов в каждом разряде. Каждый перенос уносит из одного разряда q единиц, где q – основание системы счисления, и добавляет одну 1 в следующий, разряд, т.е. сумма цифр уменьшится на величину (q–1) на каждый перенос. Тогда

$$\sum_i c_i = \sum_i a_i + \sum_i b_i - K(q-1),. \qquad (8.8)$$

где K – количество переносов, возникших при сложении. Так как

$$r'_A = \sum_i a_i (\mathrm{mod}\ P),$$

$$r'_B = \sum_i b_i (\mathrm{mod}\ P),$$

то

$$r'_C = \sum_i c_i (\mathrm{mod}\ P).$$

Подставив эти значения *в* (8.8), получим

$$r_C = [r'_A + r'_B - K(q-1)] (\mathrm{mod}\ P).$$

Аналогичные рассуждения можно привести и для разности чисел C = A – B.

Пример: Заданы A = 565; B = 278; P = 11. Найти r'_C и r'_{A+B}.

$\sum_i a_i = 5 \oplus 6 \oplus 5 \equiv 5 (\text{mod } 11)$; $r'_A \equiv 5 (\text{mod } 11)$;

$\sum_i b_i = 2 \oplus 7 \oplus 8 \equiv 6 (\text{mod } 11)$; $r'_B \equiv 6 (\text{mod } 11)$;

C = A + B = 843; K = 2,

$r'_C = 8 \oplus 4 \oplus 3 \equiv 4 (\text{mod } 11)$;

$r'_{A+B} = 5 \oplus 6 \oplus (2 \cdot 2) \equiv 4 (\text{mod } 11)$.

Для организации контроля логических и арифметических операций, например, сложения в ЭВМ, помимо обычного АУ для операндов, нужно иметь еще следующее дополнительное оборудование: сумматор для выполнения действий над остатками операндов, схему быстрого определения остатка от деления суммы на модуль и схему сравнения суммы остатков операндов с остатком их суммы.

8.3. Выбор модуля для контроля

Как было отмечено выше, справедливость свойств сравнений для контрольных кодов распространяется только на числовой метод контроля, В этом состоит его достоинство. Достоинство цифрового метода контроля состоит в простоте получения контрольных кодов. Чтобы сохранить оба эти достоинства, необходимо выполнить условие

$$r_A = r'_A.$$

Так как $r_A \equiv A (\text{mod } P)$; $r'_A = \sum_i a_i (\text{mod } P)$, то в этом случае должно выполняться условие:

$$\sum_i a_i q^i \equiv \sum_i a_i (\text{mod } P) \qquad (8.10)$$

Это возможно только тогда, когда почленно обе части выражения сравнимы:

$$a_i q^i \equiv \sum_i a_i (\text{mod } P),$$

или $$q^i \equiv 1 (\text{mod } P).$$

Отсюда следует, что $q^i \equiv 1 (\text{mod } P)$,

или $$q = mP + 1,$$

где m – целое число. Откуда получаем:

$$P = (q - 1)/m, \qquad (8.11)$$

т.е. для сохранения условия $r_A = r'_A$ необходимо наложить определенные ограничения на модуль P. К величине модуля обычно предъявляются следующие требования:

1) величина модуля P должна быть по возможности небольшой, с тем, чтобы остатки r от деления на него любых чисел не требовали большого объема оборудования для хранения и обработки,

2) величина модуля должна обеспечивать обнаружение любой одиночной арифметической или логической ошибки и как можно большего числа ошибок более высокой кратности,

3) образование контрольного кода должно осуществляться наиболее простым и быстрым способом, т.е. простыми средствами.

Анализ (8.11) с учетом изложенных требований показывает, что для двоичной системы счисления нет целочисленного решения. Тогда, с учетом того, что цифровая информация в ЭВМ представляется двоичными символами, для контроля целесообразно перейти к системам счисления с основанием $q = 2^s$, где s – некоторое целое положительное число ($s \geq 2$). Этот переход осуществляется разбиением исходной двоичной информации на группы но s разрядов с последующим суммированием этих групп по модулю $P = (2^s - 1)/m$ или при m = 1, $P = 2^s - 1$. Этот процесс называется свертыванием числа по модулю P.

На практике содержимое контрольных разрядов, т.е. свертка двоичного числа, определяется следующим образом:

1) все число, начиная с младших разрядов, разбивается на i = n/s групп по s разрядов каждая,

2) все группы последовательно, начиная с младшей, суммируются на обычном двоичном s-разрядном сумматоре по модулю $P = 2^s - 1$, т.е. с циклическим переносом единицы переполнения в младший разряд (сумматор обратного кода). Полученная в сумматоре свертка и будет искомым остатком в двоичной системе счисления.

Рассмотрим частные случаи образования сверток при различных значениях модуля P.

1. Контроль по mod 3(m = 1, s = 2, P = 3). Так как $2^2 \equiv 1(\text{mod } 3)$, то контролируемая информация предстарляется символами четверичной системы, а свертки образуются суммированием диад по модулю 3. Для этого требуется двухразрядный двоичный сумматор с целью циклического переноса из старшего разряда в младший.

Пример: Заданы $A = 53_{10} = 110101_2$; и $B = 30_{10} = 011110_2$; P = 3.

$r_A = 11 \oplus 01 \oplus 01 \equiv 10(\text{mod } 3)$;

$r_в = 01 \oplus 11 \oplus 10 \equiv 11(\text{mod } 3)$.

3. Контроль по mod 7(m = 1, s = 3, p = 7). Так как $2^3 = 1(\text{mod } 7)$, то контролируемая информация представляется символами восьмеричной системы, а для получения свертки необходим трехразряный двоичный сумматор с цепью циклического переноса.

Пример: Заданы $A = 249_{10} = 01111100_2$; P = 7.

$r_A = 011 \oplus 111 \oplus 001 \equiv 100(\text{mod } 7)$.

Можно показать, что с ростом модуля повышается корректирующая способность кода, однако при этом растет объем контрольного оборудования.

8.4. Контроль логических операций

К логическим операциям относятся операции сдвига, логического сложения и умножения.

Пусть задано число $A = a_n a_{n-i} \ldots a_1$, имеющее контрольный код $r_a = a_{ks} \ldots a_{k1}$.

Обозначим код числа А, сдвинутый влево, через $\overleftarrow{A}$ (без циклического переноса) и через $\overleftarrow{A}_ц$ – с циклическим переносом (при сдвиге вправо стрелка в обозначении будет повернута направо). Тогда контрольные коды обозначить соответственно: *

$$A \equiv r_A \ (\text{mod P}); \quad \overleftarrow{A} \equiv r_{\overleftarrow{A}} \ (\text{mod P});$$

$$\overrightarrow{A} \equiv r_{\overrightarrow{A}} \ (\text{mod P}); \ \overleftarrow{A}_ц \equiv r_{\overleftarrow{A}_ц} \ (\text{mod P}); \ \ \overrightarrow{A}_ц \equiv r_{\overrightarrow{A}_ц} \ (\text{mod P}).$$

Сдвиг влево на один разряд двоичного числа эквивалентен умножению его на два. Так как при сдвиге числа происходит потеря некоторых его разрядов, то можно предполагать, что контрольный код сдвинутого числа изменится на величину Л:

$$r_{\overleftarrow{A}} = r_{\overleftarrow{A}} + \Delta(\text{mod P}); \qquad (8.12)$$

где $r_{\overleftarrow{A}} = 2\, r_A$ – сдвинутый влево контрольный код.

Величина Δ зависит от значений a_n и a_{ks}, которые при сдвиге выходят за пределы разрядной сетки. Выход старшей 1 за пределы разрядной сетки при сдвиге n-разрядного числа эквивалентен вычитанию по mod P $a_n \sigma_{n+1}$ единиц из контрольного кода сдвинутого числа, где σ_{n+1} – вес (n+1)-го разряда.

Выход за пределы разрядной сетки разряда a_{ks} при сдвиге контрольного кода влево эквивалентен его уменьшению на $2^s \equiv 1(\text{mod } 2^s - 1)$, т.е. эту потерю можно восстановить прибавлением 1 к контрольному коду. Тогда можно записать:

$$r_{\overleftarrow{A}} \equiv (r_{\overleftarrow{A}} - a_n\sigma_{n+1} + a_{ks}) \bmod (2^s - 1) \qquad (8.13)$$

Веса разрядов числа, представленного в системе с основанием 2^s, назначаются следующим образом, например при s = 3:

	a_n	a_{n-1}	a_{n-2}	a_{n-3}	a_{n-4}	a_{n-5}	a_3	a_2	a_1
веса σ_i	2^2	2^1	2^0	2^2	2^1	2^0	2^2	2^1	2^0

Вследствие этого значения поправок Δ для контроля выполнения левого сдвига по модуль $(2^s - 1)$ будут (табл. 8.1):

Таблица 8.1.

Значения a_n	0	1	0	1
Значения a_{ks}	0	0	1	1
Поправка Δ	0	−1	+1	0

Значение поправки Δ = −1 обычно заменяют ее дополнением до модуля P.

Для выполнения сдвига влево с циклическим переносом из старшего разряда в младший необходимо уменьшить контрольный код на величину $a_n(\sigma_{n+1} - 1)$. Так как $\sigma_{n+1} - 1 \equiv (\text{mod } P)$, то этот член равен 0. Тогда можно записать:

$$r_{\overleftarrow{A_ц}} \equiv (r_{\overleftarrow{A}} + a_{ks}) (\bmod (2^s - 1)). \qquad (8.14)$$

Пример: Заданы A = 1,01011010; s = 3; P = 7. Найти $r_{\overleftarrow{A}}$ и $r_{\overleftarrow{A_ц}}$.

$r_A = 101 \oplus 011 \oplus 010 = 0110 (\text{mod} 7)$;

$\overleftarrow{A}$ = 1,10110100; s = 3; P 7. $r_{\overleftarrow{A}} = 110$.

Так как $a_n = 1$, $a_{ks} = 0$, то

$r_{\overleftarrow{A}} = 110 - 1 = 000 \equiv 101 (\text{mod } P)$

Таким образом, для того, чтобы не нарушалось соответствие основным и контрольным кодом при циклическом сдвиге на один разряд влево, необходимо к сдвинутому контрольному коду прибавить значение старшего его разряда, которое он имел до сдвига. Этой цели служит цепь циклического переноса из старшего разряда в младший в сумматоре контрольного кода.

При сдвиге вправо происходит потеря младших разрядов числа и его контрольного кода. Контрольный код сдвинутого вправо числа можно получить по формуле:

$$r_{\overrightarrow{A}} = (r_{\overrightarrow{A}} + \Delta) \bmod (2^s - 1), \qquad (8.15)$$

только поправка Δ будет при этом иной.

В случае простого сдвига поправка Δ к контрольному коду зависит от величины модуля P и для P = 7 принимает следующие значения (табл. 8.2):

Таблица 8.2.

Значения a_1	0	0	1	1
Значения a_{k1}	0	1	0	1
Поправка Δ	000	100	011	000

При модифицированном сдвиге вправо, который выполняется по правилу A = 1, $a_{n-1}...a_2a_1$; $\vec{A}_м$ = 1,1 a_{n-1} ... a_3a_2, происходит также потеря младших разрядов кодовой комбинации числа и контрольного кода. Для этого случая контрольный код вычисляется также по (8.15), но поправки Δ примут вид (табл. 8.3) для P = 7:

Таблица 8.3.

Значения a_1	0	0	1	1
Значения a_{k1}	0	1	0	1
Поправка Δ	100	001	000	100

Пример: Задано A = 1,01110111101; P = 7. Найти $r_{\vec{A}}$ и $r_{\vec{A}_M}$.

$r_A = 101 \oplus 011 \oplus 111 \oplus 101 = 011 (\mathrm{mod} 7)$;

$\vec{A}$ = 0,101110111100; $r_{\vec{A}} = 001$.

Так как $a_1 = 1$, $a_{k1} = 0$, то $\Delta = 011$.

Тогда

$r_{\vec{A}} = 001 \oplus 011 \equiv 101 (\mathrm{mod}\ 7)$

Модифицированный сдвиг числа A:

$A_м$ = 1,10111011110.

При $a_1 = 1$; $a_{k1} = 0$, поправка $\Delta = 000$. Тогда

$r_{\vec{A}_M} = 001 \oplus 000 \equiv 001 (\mathrm{mod}\ 7)$.

Таким образом, одним из способов реализации соответствия между основным и контрольным кодом при сдвиге вправо является коррекция контрольного кода в соответствии с приведенной выше таблицей 8.3 и 8.2.

Вторым способом реализации соответствия между основным и контрольным кодом при сдвиге вправо является удлинение регистра основного кода на n разрядов, с тем, чтобы при сдвиге вправо не пропадали его младшие разряды. При этом осуществляется контроль по модулю P всего 2n-разрядного регистра основного кода.

Операцию сложения по mod 2 можно выразить через другие операции, например, $A \oplus B = A + B - 2(A \wedge B)$.

Это выражение можно представить иначе:

$$A \oplus B = (A + B) + \overline{A \wedge B_{сдв}}$$

Тогда используя переход от арифметических выражений к сравнениям, получим следующую формулу для образования контрольного кода:

$$r_{\oplus} = r_{A+B} + \overline{r_{\wedge}} \pmod{P}, \qquad (8.16)$$

где r_{A+B} – контрольный код суммы двух чисел, $\overline{r_{\wedge}}$ – инверсия контрольного кода логического произведения двух чисел со сдвигом влево на один разряд.

Пример: Заданы A = 010000111, B = 101110011, P= 7. Найти $r_{\oplus}$.

$r_A = 010$; $r_B = 000$; $r_{A+B} = 010$.

Затем получим:

$A \wedge B = 0000000011$; $r_{\wedge} = 011$;

$(A \wedge B)_{сдв}$ - 0000000110; $r_{\wedge} = 110$.

После этого определим $\overline{r_{\wedge}} = 001$. И по формуле (8.15) получим:

$r_{\oplus} = 010 \oplus 001 \equiv 011 (\mathrm{mod} 7)$.

8.5. Контроль арифметических операций

Как известно, арифметические операции выполняются в ЭВМ в прямом, обратном и дополнительном кодах. Если изображения чисел хранятся в машине в каком-либо коде, то их можно рассматривать как единые кодовые комбинации, к которым можно применять все сформулированные выше правила получения свертки. При этом требуется обязательная кратность общего числа разрядов избранному модулю.

При сложении в прямом коде складываются только цифровые части изображений чисел, а знаки обрабатываются отдельно. Поэтому в этом случае возможен контроль двумя способами:

1) раздельный контроль знаковой и цифровой частей изображений.

2) обобщенный контроль всего изображения.

При раздельном способе контроль знаковых разрядов производят обычно с помощью схемы, выявляющей переполнения, так как в модифицированном коде появление одиночной ошибки в знаковых разрядах приводит к несовпадению информации в них. Для цифровых частей изображений имеем:

$$r_{|[A\pm B]пр|} \equiv (r_{|A|} \pm r_{|B|}) \bmod P.$$

При обращенном способе контроля требуется коррекция контрольного кода результата вследствие того, что знак результата при сложении чисел с равными знаками повторяет знак слагаемых, т.е. контрольный код суммы будет равен

$$r_{(A\pm B)пр} \equiv (r_A \pm r_B - \mathrm{Sign}\sigma_S) \bmod P. \qquad (8.18)$$

где Sign – значение знакового разряда операндов; σ_S – вес старшего разряда свертки.

Пример: Заданы $[A]_{пр} = 1{,}01101011$, $[B]_{пр} = 1{,}00110010$, P= 7.

Найти $r_{(A+B)пр}$

$r_A = 110$; $r_B = 101$; $r_{(A+B)пр} = 1{,}10011101$;

$r_{(A+B)пр} \equiv 111$.

На основании (8.18) получим:

$$r_{(A+B)пр} \equiv 110 + 101 - 1{,}100 \equiv 111 (\bmod 7).$$

Обобщенный способ может быть применен и для контроля выполнения операции сложения в обратном коде. В этом случае:

$$r_{(A+B)_o} \equiv (r_{(A)_o} + r_{(B)_o}) \bmod P. \qquad (8.19)$$

Пример: Заданы $[A]_o = 1{,}011001001$, $[B]_o = 0{,}110001111$, P = 3.

Найти $r_{(A+B)_o}$.

$r_{(A)_o} = 10$; $r_{(B)_o} = 11$; $[A+B]_o = 0{,}001011001$;

$r_{(A+B)_o} = 10$.

Проверка:

$$r_{(A+B)_o} = r_{(A)_o} + r_{(B)_o} \equiv 10 (\bmod 3).$$

При сложении чисел в дополнительном коде необходима коррекция контрольного кода результата в случае, если возникает единица переноса из знакового разряда. Т.е. контрольный код суммы будет равен:

$$r_{(A+B)_д} = r_{(A)_д} + r_{(B)_д} - \alpha \qquad (8.20)$$

где α – коррекция (α = 1, если возник перенос из знакового разряда и α = 0 – если переноса нет).

Пример: Заданы $[A]_д$ = 1,00110001110, $[B]_д$ = 1,11101110111, P = 15.

Найти $r_{(A+B)_д}$.

$r_{(A)_д}$ = 0001; $r_{(B)_д}$ = 1110; $[A+B]_д$ = 1,00100000101;

$r_{(A+B)_д}$ = 1110(mod 15).

Проверка:

$$r_{(A+B)_д} = r_{(A)_д} + r_{(B)_д} - \alpha \equiv 0001 + 1110 - 0001 \equiv 1110(\text{mod } 15).$$

В связи с алгебраическим сложением чисел в инверсных кодах возникает вопрос о преобразовании контрольного кода при преобразовании прямого кода чисел в инверсный и обратный. Для конкретности будем считать что отрицательные числа изображаются в обратном коде и P = 3.

Пусть до преобразования в регистре хранился код

$$[A]_{пр} = 2^n + |A|.$$

т.е. в знаковом разряде была записана 1 (числа условно целые); После преобразования этого кода в обратный в регистре получится код

$$[A]_o = 2^n + (2^n - 1 - |A|).$$

Выразив $[A]_o$ через $[A]_{пр}$, получим

$$[A]_o = 3 \cdot 2^n - 1 - [A]_{пр} \equiv 2[A]_{пр} + 2(\text{mod } 3).$$

При преобразований прямого кода в обратный (или наоборот) необходимо выполнить такие же преобразования с контрольным кодом, т.е.

$$r_{A2} \equiv 2\, r_{A1} + 2(\text{mod } 3).$$

где r_{A1} – контрольный код до преобразования, r_{A2} – контрольный код после преобразования.

Таким образом, при выполнении операций алгебраического сложения в обратном и дополнительном кодах обработка основных и контрольных кодов может производиться в двух разных сумматорах параллельно, что позволяет в значительной степени распараллелить процессы в основном и контрольном АУ, упростить алгоритмы арифметических операций и уменьшить время их реализации. При сложении в обратном коде сумматоры работают независимо друг от друга, при сложении в дополнительном коде единица переноса из

знакового разряда суммы используется для коррекции контрольного кода последней.

При использовании позиционной системы счисления все арифметические операции сводятся к последовательности сложений кодов. Поэтому достаточно проверять только правильность всех сумм, получаемых в машине (окончательных при алгебраическом сложении и промежуточных при умножении и делении). Если формируется 2n-разрядный результат, то операцию умножения можно проверять также и прямым способом – сравнением по mod P произведения остатков сомножителей с остатком их произведения.

Сравнимость основного и контрольного кодов результата проверяется только после окончания операции. При этом в конце операции проверяется также сравнимость основного и контрольного кодов операндов. Это обусловлено тем, что одиночная ошибка в операнде может оказаться источником многократных ошибок в результате операции. Например, ошибка, исказившая один разряд множимого при его сдвиге, вызовет серию неправильных его передач в сумматор. Суммарный эффект всех этих неправильных передач может не нарушить соответствия между основным и контрольным кодами произведения. Поэтому эту ошибку можно обнаружить либо только при проверке всех частных произведений, либо путем проверки соответствия основного и контрольного кода множимого после окончания операции.

Контрольные вопросы к главе 8

1. Чем отличается цифровой метод контроля от числового?
2. Как выбирают модуль для контроля?
3. Как можно контролировать логические операции и операции сдвига чисел?
4. В чем состоят особенности контроля: операции сложения чисел в прямом, обратном и дополнительном кодах?
5. Как контролируют операцию преобразования прямого кода числа в инверсный и наоборот?

СПИСОК ЛИТЕРАТУРЫ

1. Поспелов Д. А. Арифметические основы вычислительных машин дискретного действия. – М.: Высшая школа. –1970. – 308 с.
2. Шауман А. М. Основы машинной арифметики. –Л.: Изд-во Ленингр. ун-та. –1979. –312 с.
3. Попов Б. А., Теслер Г. С. Вычисление функций на ЭВМ. –Киев: Наукова думка. –1984. –599 с.
4. Соренков Э. И., Телига А. И., Шаталов А. С. Точность вычислительных устройств и алгоритмов. –М.: Машиностроение. –1976. – 200 с.