ОГЛАВЛЕНИЕ

Подготовка специалистов по вычислительной технике, в общем плане — по электронной обработке данных, немыслима без углубленного изучения микропроцессорной техники. Микропроцессорная техника является основой для компьютеризации общества. Благодаря микропроцессорам родились удивительные технические творения наших дней — профессиональные персональные компьютеры, возможности которых близки, а иногда и превосходят возможности больших и средних компьютеров недавнего прошлого. Внедрение персональных компьютеров в сферу инженерной деятельности повысит производительность труда в несколько раз; станет более дешевой, массовой и надежной, а ее использование будет экономически эффективным практически во всех областях народного хозяйства.

В нашей стране в развитии микропроцессорной техники объективно сложились два основных направления. Первое из них образуют микропроцессоры и микрокомпьютеры, имеющие систему команд СМ ЭВМ. Второе направление составляют однокристальные микропроцессоры КР580ИК80, К1810ВМ86 и арифметический (математический) сопроцессор К1810ВМ87. Литературы по этому направлению издано недостаточно. Изучение программирования для микропроцессора К1810ВМ86 и сопроцессора К1810ВМ87 приобретает особое значение, так как они применяются в профессиональных персональных компьютерах ЕС1840/1/2, Искра-1030 и Нейрон-И9.

Пособие состоит из четырех глав. Содержание гл. 1 соответствует разделу по арифметическим основам компьютеров. Наибольший интерес представляет § 1.5, в котором рассматривается принятый за рубежом стандарт на арифметику с плавающей точкой, и § 1.6, где показаны особенности программирования арифметических операций. Гл. 2—4 построены по единому принципу: краткая общая характеристика микропроцессора, его программная (регистровая) модель, система команд (с более подробным изложением арифметических команд) и заключительный параграф по алгоритмам и программам вычислительных задач. Завершают каждую главу контрольные вопросы и упражнения, помогающие закрепить изучаемый материал.

Авторы выражают благодарность сотрудникам кафедры вычислительной техники Московского института электронной техники — зав. кафедрой чл.-кор. Л. Н. Преснухину, проф. Б. М. Кагану за ряд ценных замечаний, способствовавших улучшению материала книги, и инж. О. Ф. Куприяновой за помощь в подготовке рукописи.

Свои замечания и пожелания о книге направлять по адресу: *101430, Москва, ГСП-4, Неглинная ул., д. 29/14, издательство «Высшая школа».*

Авторы

При подготовке данного учебного пособия авторы поставили цель познакомить студентов вузов, специализирующихся в области электронной обработки данных, с основами программирования арифметических операций в современных однокристальных микропроцессорах. Достижение этой цели и желание сделать пособие автономным потребовали кратко изложить двоичную систему счисления, двоичную арифметику, форматы машинных чисел и особенности производства арифметических операций в микропроцессорах с ограниченной длиной слова. Этот материал представлен в первой главе пособия. Остальные три главы содержат конкретные алгоритмы и программы арифметических операций для наиболее распространенных сейчас микропроцессоров КР580ИК80 и К1810ВМ86, а также арифметического сопроцессора К1810ВМ87. Наряду с алгоритмами и программами в этих главах имеются необходимые сведения о самих микропроцессорах: краткая техническая характеристика, программная модель, режимы адресации и система команд.

Эволюция микропроцессоров прошла этапы от сравнительно слабого по вычислительным возможностям 8-битного микропроцессора КР580ИК80 к гораздо более мощному 16-битному микропроцессору К1810ВМ86 и далее к арифметическому сопроцессору К1810ВМ87, ориентированному исключительно на инженерно-технические и научные расчеты. Сопроцессор обеспечивает диапазон и точность представления чисел, характерные для средних и больших компьютеров недавнего прошлого, и имеет сравнимую с ними производительность. Благодаря ему сфера применений микропроцессоров будет значительно расширена, а это потребует увеличения числа специалистов, ориентирующихся в численных расчетах.

Программы арифметических операций сопровождается графическими иллюстрациями и даются на языке Ассемблер. Выбор этого языка объясняется несколькими причинами, в частности эффективностью ассемблерных программ, наибольшей близостью языка Ассемблера к архитектуре микропроцессора и доступностью для программиста всех ресурсов микропроцессора. Изучение программ на языке Ассемблер поможет студентам уяснить реализацию арифметических операторов на языках программирования высокого уровня и освоить некоторые общие приемы программирования.

Рассмотренные в данном учебном пособии вопросы программирования арифметических операций охватывают наиболее распространенные 8- и 16-битные однокристальные микропроцессоры и арифметический сопроцессор. По-видимому, необходимость знания точных численных расчетов в настоящее время будет все более настоятельной благодаря повышению вычислительной мощности микропроцессоров и реализуемых на их основе качественно новых систем. В ближайшем будущем появятся 32-битные микропроцессоры и совместимые с ними арифметические сопроцессоры. Наибольший интерес для студентов представляет материал по арифметическому сопроцессору К1810ВМ87 из-за недостатка посвященной ему технической литературы и архитектурной совместимости его с будущими разработками.

Изучение схем алгоритмов и программ арифметических операций поможет студентам глубже разобраться в особенностях программирования микропроцессоров. Используемые в них приемы программирования применимы к решению других практических задач.

ГЛАВА 1

ФОРМАТЫ ЧИСЕЛ И ОСНОВЫ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

Данная глава носит вводный характер и знакомит читателей с двоичной системой счисления и основами двоичной арифметики, форматами числовых данных, принятыми в современных компьютерах, общими алгоритмами выполнения арифметических операций и особенностями их реализации в микропроцессорах с ограниченной длиной машинного слова. Рассмотрены также диапазоны и точность представления чисел в компьютерах, стандарт на числа с плавающей точкой, принятый в новейших разработках, а также особые случаи (нарушение нормализации, переполнение и антипереполнение), возникающие при производстве арифметических операций над числами с плавающей точкой. Изложение материала сопровождается иллюстративными примерами.

1.1. БИТЫ, БАЙТЫ, СЛОВА

Компьютеры оперируют данными, имеющими исключительно двоичное представление или кодирование. Независимо от того, как изображает входные данные пользователь (десятичные числа в различных формах, шестнадцатеричные числа, символьные цепочки и др.), они аппаратно и (или) программно преобразуются в цепочки (последовательности) двоичных цифр — единиц и нулей. При выводе данных осуществляется обратное преобразование двоичных цепочек в удобную для пользователя форму, например десятичные числа.

Бит. Двоичная цифра, имеющая всего два значения 1 и 0, называется *битом* (BInary digiT). С помощью двух битов можно представить четыре значения (кода) — 00, 01, 10 и 11; с помощью трех битов — восемь значений от 000 до 111 и т. д. Группа из n бит позволяет представить 2^n значений или комбинаций — от 00 ... 00 до 11 ... 11.

Единицы данных. Во всех современных компьютерах важную роль играет представление данных группами по 8 бит, называемых *байтами* (byte — слог) и содержащими любую из $2^8 = 256$ ком-

бинаций. По существу, байт стал стандартной базовой единицей, из которой образуются все остальные единицы машинных данных. В зависимости от того, как интерпретируется содержимое байта, оно может быть: кодированным представлением символа внешнего алфавита, целым знаковым или беззнаковым числом, частью команды или более сложной единицы данных и т. д. Другими словами, интерпретацию байта определяет программист в зависимости от контекста своей программы.

Биты в байте нумеруются справа налево, начиная с нуля (см. рис. 1.1). Такая нумерация принята во всех компьютерах, только в моделях ЕС ЭВМ биты нумеруются слева направо.

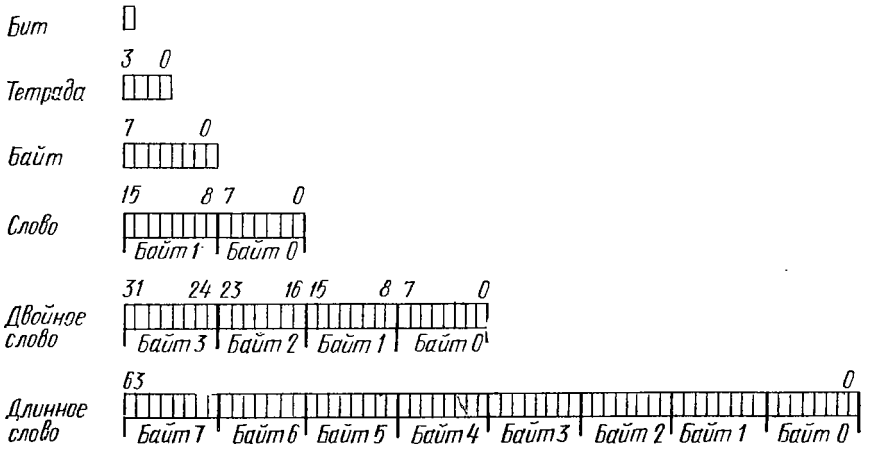


Рис. 1.1. Машинное представление данных

Трехбитная единица данных называется *триадой.* Она может содержать 8 комбинаций от 000 до 111. Триады используются только в восьмеричной системе счисления.

Четырехбитная единица данных называется *тетрадой;* она может содержать 16 различных комбинаций от 0000 до 1111. Применение тетрад ограничено упакованными десятичными числами (см. § 1.3).

Единица данных, состоящая из 16 бит или двух байт, называется *словом.* Слово может содержать любую из $2^{16}=65536$ комбинаций. Для краткой записи больших ступеней числа два число $2^{10}=1024$ обозначается «К» и читается как приставка «кило-», или, просто, как буква «к». По аналогии с нумерацией бит байты в слове также нумеруются справа налево, начиная с нуля: байт 0 является младшим, а байт 1 — старшим.

Следующая единица данных состоит из четырех байтов и называется *двойным (длинным) словом*. Число возможных комбинаций в двойном слове составляет 2^{32}, что очень близко к 4 млрд. Число $2^{20}=1\,048\,576$, близкое к миллиону, обозначают «М» и читают как приставку «мега», или как букву «м», а число $2^{30}=1\,073\,741\,824$, близкое к миллиарду, обозначают «Г» и читают как приставку «гига-», или как букву «г».

Последняя из рассматриваемых нами единица данных состоит из 64 бит или 8 байт и называется *счетверенным словом* (возможно, появится более короткий термин «тетраслово»). Число комбинаций в этой единице данных составляет 2^{64} или более 10^{19}.

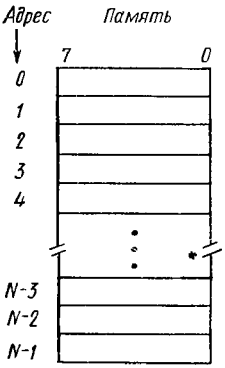


Рис. 1.2. Организация логической памяти

Машинное слово. Основная или базовая единица данных, которой оперирует микропроцессор, называется *машинным словом*. Практически во всех микропроцессорах длина машинного слова кратна байту. Длина слова является важнейшей характеристикой микропроцессора и в соответствии с ней микропроцессоры подразделяются на 8-, 16- или 32-битные. В 16-битных микропроцессорах всегда есть команды обработки байтов, а в 32-битных микропроцессорах — команды операций с байтами и словами.

Хранение данных в памяти. Программы и данные, к которым процессор имеет непосредственный доступ, хранятся в *основной памяти*, иногда называемой также оперативной памятью (запоминающим устройством).

Различают логическую и физическую основную память. *Логическая память*, т. е. та память, которую «видит» и к которой обращается процессор, организована в последовательность из N байт, образующих пространство логической памяти. Байты нумеруются от 0 до $N-1$, и порядковый номер байта называется его *адресом* (см. рис. 1.2). Для обращения к любому байту необходимо указать его адрес (адреса представляются в двоичной форме и оказываются еще одной единицей данных, которой должен оперировать процессор). В языках высокого уровня адреса обычно называются *указателями* (pointer). Длина адреса m связана с числом N байт (емкостью памяти) простыми соотношениями:

$$N=2^m,\ m=\log_2 N.$$

● *Примечание.* Обычно адреса записываются в шестнадцатеричной системе счисления и в таком же виде даются в листингах объектных программ. Мы будем идентифицировать шестнадцатеричные числа заключительной буквой H (от Hexadecimal или Hex — шестнадцатеричный).

Формируемый процессором адрес памяти всегда является адресом байта в пространстве логической памяти: он называется эффективным, виртуальным или логическим адресом. Ниже используется термин *логический адрес.*

Длина логического адреса может совпадать или не совпадать с длиной машинного слова. Например, в 8-битных микропроцессорах длина логического адреса составляет 16 бит, определяя пространство логической памяти 64К байт. В 16-битных микропроцессорах длина логического адреса обычно совпадает с длиной машинного слова и пространство логической памяти также составляет 64К байт.

На практике наблюдается тенденция постоянного увеличения емкости основной памяти, что позволяет хранить в ней программы большего размера, несколько программ в мультипрограммных системах и значительные объемы обрабатываемых данных. Требование увеличения емкости памяти связано с расширением адресов и создает трудности манипуляций адресами, не вписывающимися в размер арифметико-логического устройства (АЛУ) процессора. Как компромиссное решение в ряде процессоров увеличение емкости памяти достигается путем преобразования (отображения) логических адресов в более длинные *физические адреса,* по которым и производятся реальные обращения к памяти. Специальное устройство, которое осуществляет такое преобразование, называется *устройством управления памятью или диспетчером памяти.* Устройство управления памятью реализуется в виде внешней микросхемы, но может находиться и в составе микропроцессора. Включение устройства управления памятью в системе показано на рис. 1.3.

Принцип действия устройства управления памятью основывается на страничной либо сегментной организации памяти. У обоих способов есть много общих моментов, но размер страницы фиксирован, а размер сегмента может быть переменным. В составе устройства управления памятью есть несколько регистров, которые процессор загружает специальными командами. Регистры содержат начальные (базовые) адреса и атрибуты страниц (сегментов). Логический адрес, выдаваемый процессором, считается смещением байта в странице (сегменте), т. е. расстоянием его от начала страницы (сегмента), и получение физического адреса байта сводится к суммированию, а иногда к сцеплению (конкатенации) содержимого одного из регистров и логического адреса. При этом длина n физического адреса больше длины m логического адреса.

Физическая память состоит из *ячеек,* в каждой из которых хранится одно *слово памяти.* При каждом обращении (доступе) к памяти, т. е. выполнении считывания или записи, в операции участвует вся ячейка памяти. Обычно длина слова памяти совпадает с длиной машинного слова, хотя это условие не является обязательным.

Таким образом, логический адрес однозначно определяет байт в пространстве логической памяти, а физический адрес — байт в пространстве физической памяти. Рассмотрим, как адресуются более крупные единицы данных, состоящие из нескольких байтов, каждый из которых имеет свои логический и физический адреса. В подавляющем большинстве современных компьютеров (за исключением ЕС ЭВМ) адресация данных подчиняется двум простым правилам:

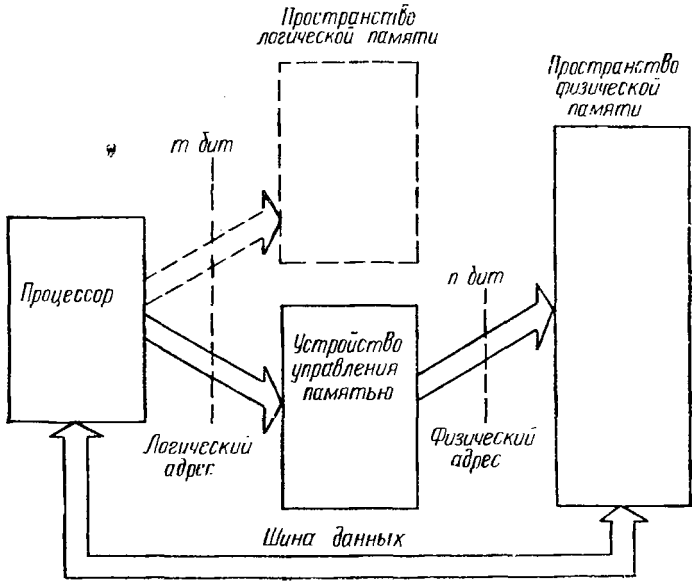


Рис. 1.3. Логическая и физическая память

адресом любой единицы данных считается наименьший из адресов тех байтов, которые образуют эту единицу;

по этому адресу находится младший байт единицы данных, а остальные байты следуют в порядке возрастания адресов.

На рис. 1.4 показано, что адресом двойного слова является 1000H и по этому адресу находится его младший байт. Принятые правила несколько неудобны для пользователей, так как мы привыкли читать слева направо — «от старшего к младшему». Это неудобство проявляется только при анализе содержимого памяти, выведенного на дисплей или принтер в шестнадцатеричном формате. При программировании на любом языке (кроме машинного языка) учитывать «обратный» порядок байтов в памяти не требуется.

В тех случаях, когда длина слова памяти превышает байт, на

размещение данных могут накладываться некоторые ограничения. Например, обычно требуется, чтобы 16-битные слова размещались только по четным адресам (или по так называемым границам слов), двойные слова — по адресам, кратным 4 (по границам двойных слов) и т. д. Такое требование называется *выравниванием ад-*

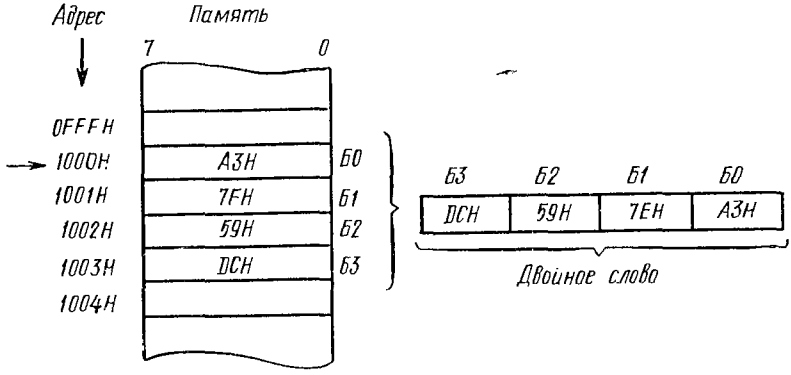


Рис. 1.4. Хранение в памяти двойного слова

ресов по целочисленным границам. На рис. 1.5 показаны два варианта размещения слова в памяти с длиной слова 16 бит. Когда слово выравнено (рис. 1.5, *а*), для его считывания достаточно одного обращения к памяти по адресу 1000H. Если же слово не вы-

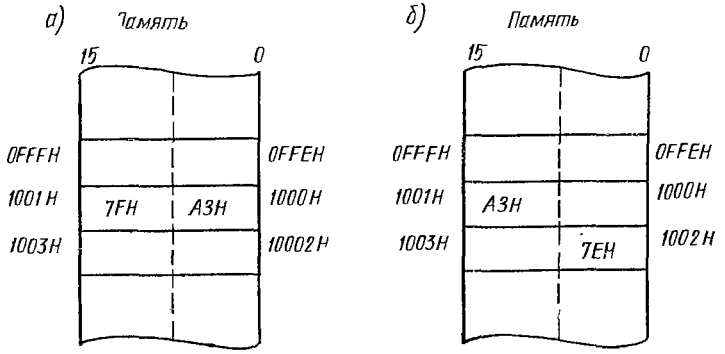


Рис. 1.5. Размещение слова в памяти:
а — с выравниванием, *б* — без выравнивания

равнено, считывание его потребует двух операций в памяти (по адресам 1000H и 1002H), а процессор должен учитывать, по каким линиям шины данных передаются байты слова.

1.2. ФОРМАТЫ ЦЕЛЫХ ДВОИЧНЫХ ЧИСЕЛ

Система вещественных десятичных чисел, применяемая в ручных расчетах, предполагается бесконечной и непрерывной, т. е. здесь не накладывается никаких ограничений на диапазон используемых чисел и на точность (количество значащих разрядов или цифр) их представления. Для любого вещественного числа существует бесконечно много чисел, которые больше и меньше его, и между любыми двумя вещественными числами находится также бесконечно много чисел. Реализовать такую систему чисел в технических устройствах невозможно. В компьютерах размеры регистров и ячеек памяти фиксированы, что накладывает ограничения на систему представимых чисел. Ограничения касаются диапазона допустимых чисел и точности их представления. Другими словами, система машинных чисел оказывается конечной и дискретной, образуя подмножество системы вещественных чисел.

В любом компьютере имеется максимальное представимое число $Z_{макс}$ и минимальное представимое число $Z_{мин}$, между ними находится конечное множество допустимых чисел (рис. 1.6). Если

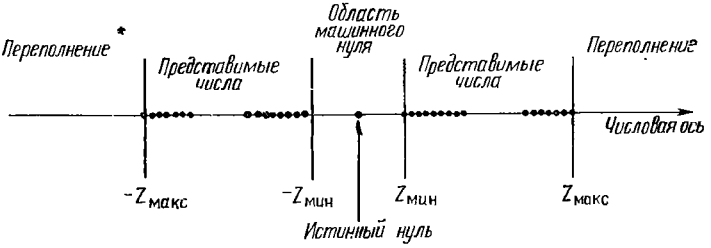


Рис. 1.6. Система машиных чисел

результат операции превышает $Z_{макс}$, возникает *переполнение* и дальнейшее продолжение выполнения программы не имеет смысла. Если результат операции оказывается меньше $Z_{мин}$, фиксируется *антипереполнение;* большинство компьютеров при антипереполнении возвращают как результат операции нуль. Область чисел от $-Z_{мин}$ до $+Z_{мин}$, за исключением истинного нуля, называют *машинным нулем.*

В компьютерах применяется исключительно двоичная позиционная система счисления — любое число представляется последовательностью из 1 и 0.

● *Примечание.* В соответствии с правилами употребления точки и запятой в языках высокого уровня и в машинных листингах отделяют целую часть числа от дробной точкой и говорят о форматах с *фиксированной точкой* и с *плавающей точкой.*

Позиционные системы счисления. В позиционных системах счисления, к которым относится и общепринятая десятичная система

счисления, числовое значение цифры зависит от ее местоположения (позиции) в последовательности цифр, изображающей число. Например, в числе 636.96 одна и та же цифра 6 обозначает (слева направо) шесть сотен, шесть единиц и шесть сотых. Это представление есть сокращенная запись следующей суммы:

$$639.96 = 6\times10^2 + 3\times10^1 + 6\times10^0 + 9\times10^{-1} + 6\times10^{-2}.$$

Любое десятичное число

$$X = a_{n-1}a_{n-2}\ldots a_1a_0.a_{-1}a_{-2}\ldots a_{-m},$$

где $a \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ — десятичные цифры, n — число разрядов целой части, m — число разрядов дробной части, можно развернуть в сумму:

$$X = a_{n-1}\times10^{n-1} + a_{n-2}\times10^{n-2} + \ldots + a_0\times10^0 +$$

$$+ a_{-1}\times10^{-1} + \ldots + a_{-m}\times10^{-m} = \sum_{i=1}^{n} a_i\times10^i. \qquad (*)$$

Здесь значение i называется разрядом, величина 10^i — *весом* i-го разряда, а $n+m$ — *длиной* числа. Число десять называется *основанием* системы счисления; оно равно отношению весов двух соседних разрядов.

Принципы построения десятичной системы счисления распространяются и на другие позиционные системы счисления. Выберем в качестве основания целое положительное число $q>1$. За цифры рассматриваемой системы счисления естественно принять первые порядковые десятичные числа 0, 1, 2, ..., $q-1$; основание q будет иметь вид 10_q (нижний индекс показывает основание системы счисления; в десятичных числах он будет опускаться). Если q больше десяти, придется вводить специальные символы, соответствующие цифрам десять, одиннадцать и т. д. Если $q=16$, цифрами будут 0, 1, ... 9, A, B, C, D, E и F. Для данной системы счисления справедливо приведенное соотношение (*).

Пример 1.1. Найти десятичный эквивалент восьмеричного числа 376.2_8.

$$X = 376.2_8 = 3\times8^2 + 7\times8^1 + 6\times8^0 + 2\times8^{-1} = 254.25.$$

В двоичной системе счисления имеется всего две цифры: 1 и 0.

Пример 1.2. Найти десятичный эквивалент двоичного числа 100110.101_2.

$$X = 100110.101 = 1\times2^5 + 0\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 +$$

$$+ 0\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} = 38.625.$$

Двоичная система счисления получила исключительное распространение в вычислительной технике благодаря возможности представления цифры каждого разряда электронной схемой с двумя устойчивыми состояниями и простоте выполнения арифметических операций.

Перевод чисел из одной системы счисления в другую. Поскольку люди и компьютеры пользуются различными системами счисления, следует познакомиться с принципами перевода чисел из одной системы счисления в другую. Рассмотрим перевод десятичных чисел в системы с произвольными основаниями q, так как преобразование числа в десятичное осуществляется по соотношению $(*)$ и никаких трудностей не вызывает. Разберем перевод чисел вручную (машинный перевод покажем далее).

Пусть задано *целое* десятичное число X и его необходимо перевести в систему счисления с основанием q, т. е. найти цифры a_i в записи $a_{n-1}a_{n-2}\ldots a_1a_0$. Воспользуемся соотношением $(*)$ и запишем в общем виде:

$$X = a_{n-1}q^{n-1} + a_{n-2}q^{n-2} + \ldots + a_1q^1 + a_0.$$

Разделим обе части этого равенства (а фактически только исходное число X) на q:

$$X/q = a_{n-1}q^{n-2} + a_{n-2}q^{n-3} + \ldots + a_1 \qquad (a_0).$$

Как результат получается целое частное

$$X^{(1)} = a_{n-1}a_{n-2}\ldots a_2a_1,$$

а остаток от деления равен цифре a_0. Частное $X^{(1)}$ имеет такой же вид, как и исходное число X, поэтому для нахождения a_1 необходимо разделить $X^{(1)}$ на q. В результате получается новое частное

$$X^{(2)} = a_{n-1}a_{n-2}\ldots a_3a_2,$$

а остаток от деления равен a_1. Повторяющееся выполнение указанных действий позволяет найти все цифры a_i. Это приводит к следующему правилу:

■ Правило первое. Для перевода целого десятичного числа X в систему счисления с основанием q необходимо последовательно делить исходное число X и образующиеся частные на q до получения частного, равного нулю. Искомое представление есть последовательность остатков от операций деления, причем первый остаток дает младшую цифру.

Пример 1.3. Перевести число 236 в восьмеричную систему счисления ($q=8$).

Выполняются операции деления:

$-\frac{236}{16}\ \Big\vert\ \frac{8}{29}$	$-\frac{29}{24}\ \Big\vert\ \frac{8}{3}$	$-\frac{3}{0}\ \Big\vert\ \frac{8}{0}$
$-\frac{76}{72}$	$a_1 = 5$	$a_2 = 3$
$a_0 = 4$		

Следовательно, восьмеричная запись числа 236 имеет вид 354_8.

Пример 1.4. Перевести число 236 в двоичную систему счисления.

Простые операции деления на 2 производятся в уме и получается короткая запись:

Частные		Остатки
236	↑	$0 = a_0$
118	|	$0 = a_1$
59	|	$1 = a_2$
29	|	$1 = a_3$
14	|	$0 = a_4$
7	|	$1 = a_5$
3	|	$1 = a_6$
1	|	$1 = a_7$
0		

Записывая снизу вверх, получим $236 = 11101100_2$.

Рассмотрим перевод правильных дробей. Правильную десятичную дробь X требуется перевести в систему счисления с основанием q; иначе говоря, необходимо найти цифры a_{-i} в записи $(0.a_{-1}a_{-2}\ldots a_{-m})$. Воспользуемся соотношением $(*)$:

$$X = a_{-1}q^{-1} + a_{-2}q^{-2} + a_{-3}q^{-3} + \ldots + a_{-m}q^{-m}.$$

Как и при переводе целых чисел, цифры a_{-i} находятся последовательно. Умножим обе части приведенного равенства (а фактически только исходную дробь X) на q:

$$X \times q = a_{-1} + a_{-2}q^{-1} + a_{-3}q^{-2} + \ldots + a_{-m}q^{-m+1}.$$

Цифра a_{-1} равна целой части полученного произведения, а дробная часть его соответствует новой правильной дроби $X^{(1)}$:

$$X^{(1)} = a_{-2}q^{-1} + a_{-3}q^{-2} + \ldots + a_{-m}q^{-m+1}.$$

При умножении $X^{(1)}$ на q целая часть произведения дает цифру a_{-2}, а дробная часть соответствует новой правильной дроби

$$X^{(2)} = a_{-3}q^{-2} + \ldots + a_{-m}q^{-m+2}.$$

Отсюда вытекает следующее правило перевода.

Правило второе. Для перевода правильной десятичной дроби X в систему счисления с основанием q необходимо последовательно умножать исходную дробь и дробные части получаю-

щихся произведений на q. Искомые цифры нового представления есть последовательность целых частей произведений, причем первая из них дает старшую цифру a_{-1}.

Пример 1.5. Перевести десятичную дробь 0.8125 в восьмеричную систему счисления.

Выполняем операции умножения:

$$\begin{array}{rr} \times\ 0.8125 & \times\ 0.5000 \\ 8 & 8 \\ \hline a_{-1} = 6.5000 & a_{-2} = 4.0000 \end{array}$$

Восьмеричное представление десятичной дроби 0.8125 есть 0.64_8.

Пример 1.6. Перевести десятичную дробь 0.3 в двоичную систему счисления. Производим операции умножения:

$$\begin{array}{cccccc} \times\ 0.3 & \times\ 0.6 & \times\ 0.2 & \times\ 0.4 & \times\ 0.8 & \times\ 0.6 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ \hline 0.6 & 1.2 & 0.4 & 0.8 & 1.6 & 1.2 \\ a_{-1} & a_{-2} & a_{-3} & a_{-4} & a_{-5} & a_{-6} \end{array}$$

Как видно, при умножении никогда не получится нулевая дробная часть, поэтому десятичная дробь не имеет точного двоичного представления, а с точностью до шестого двоичного разряда равна 0.010011_2.

Сформулированные правила перевода справедливы не только для перевода десятичных чисел в систему счисления с основанием q, но и для перевода чисел из системы счисления с основанием q_1 в систему счисления с основанием q_2. В этом случае операции умножения и деления нужно производить по правилам системы счисления с основанием q_1. Так как легко оперировать только десятичными числами, обычно такое преобразование осуществляется через промежуточную десятичную систему счисления, что условно можно записать в виде $q_1 \to 10 \to q_2$.

Рассмотрим элементарные приемы взаимного преобразования двоичных и восьмеричных (а также шестнадцатеричных) чисел. Так как $8 = 2^3 = 1000_2$ и $16 = 2^4 = 10\,000_2$, эти преобразования выполняются без каких-либо вычислений.

■ Правило третье. Для перевода двоичного числа в систему счисления с основанием 8 (16) необходимо исходное число влево и вправо от точки сгруппировать по три (четыре) бита, а затем каждую группу записать одной восьмеричной (шестнадцатеричной) цифрой.

Пример 1.7. Перевести 1111010.1011_2 в системы счисления с основаниями 8 и 16.

$$1111010.\ 1011_2 = 001\ \ 111\ \ 010.\ \ 101\ \ 100_2 = 172.54_8$$

$$1111010.1011_2 = 0111\ \ 1010.\ \ 1011_2 = 7A.B_{16}$$

■ Правило четвертое. Для перевода восьмеричного (шестнадцатеричного) числа в двоичное необходимо каждую цифру ис-

ходного числа записать в виде эквивалентного ей трехбитного (четырехбитного) двоичного числа.

Пример 1.8. Перевести числа 273.4_8 и $5AF18_{16}$ в двоичную систему счисления.

$$273.4_8 = 010\ \ 111\ \ 011.\ \ 100_2 = 10111011.1_2$$

$$5AF.18_{16} = 0101\ \ 1010\ \ 1111.0001\ \ 1000_2 = 10110101111.00011_2$$

Целые беззнаковые двоичные числа. Формат целых двоичных чисел без знака имеет вид, показанный на рис. 1.7. Здесь значок вставки ($\wedge$) обозначает местоположение двоичной точки. Как видно из этого рисунка, все разряды числа являются значащими, а двоичная точка находится справа или, как говорят, *фиксирована* после младшего значащего разряда. Отсюда появляется еще одно название этого формата и других аналогичных форматов — *формат*

Рис. 1.7. Формат целых беззнаковых двоичных чисел

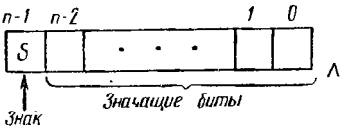


Рис. 1.8. Общий формат целых знаковых чисел

с фиксированной точкой. Следовательно, в этом формате представимы только целые числа 0, 1, 2, ..., 2^n-1 и любая комбинация битов (двоичный набор) является допустимой.

Целые беззнаковые числа при программировании используются для представления тех числовых объектов, которые принципиально не могут быть отрицательными. Примерами таких объектов служат адреса ячеек памяти, номера строк исходной программы, счетчики повторений циклов и т. п. В различных языках программирования для указания типа беззнаковых целых чисел применяются объявления (спецификаторы) типов BYTE ($n=8$), WORD ($n=16$), ADDRESS ($n=16$), UNSIGNED ($n=16$) и др.

Целые знаковые двоичные числа. Чтобы компьютеры могли оперировать положительными и отрицательными числами, один из разрядов необходимо отвести для изображения знака чисел S. Обычно им является старший (левый) бит, а стандартное кодирование знака имеет такой вид:

$$S = \begin{cases} 0 — \text{число положительное,} \\ 1 — \text{число отрицательное.} \end{cases}$$

Знаковые числа длиной n бит представляются в формате, показанном на рис. 1.8. Имеется несколько способов кодирования знаковых чисел.

Прямой код. Наиболее естественное кодирование знаковых чисел заключается в том, чтобы поместить в бит S знак числа,

а остальные биты использовать для абсолютного значения числа. В англоязычной литературе для этого способа принят выразительный термин «знак и модуль», а в нашей литературе закрепился термин «прямой код». Отображение двоичных n-битных наборов на числовую ось для прямого кода показано на рис. 1.9. Диапазон представимых чисел составляет от $-(2^{n-1}-1)$ до $+(2^{n-1}-1)$, а число нуль может быть положительным 00 ... 00 и отрицательным 100 ... 00.

Пример 1.9. Представить в прямом коде числа +108 и −108.

+108	0000	0000	0110	1100	006C
−108	1000	0000	0110	1100	806C

Прямой код наиболее удобен и в нем очень легко реализуется операция изменения знака числа (операция NEGATE). Однако с «компьютерной» точки зрения у прямого кода есть существенные недостатки. Во-первых, неудобно иметь два представления нуля, и, во-вторых, операция алгебраического сложения требует анализа знаков операндов и выбора фактической операции сло-

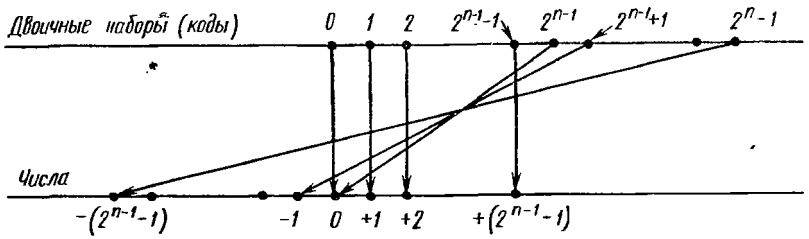
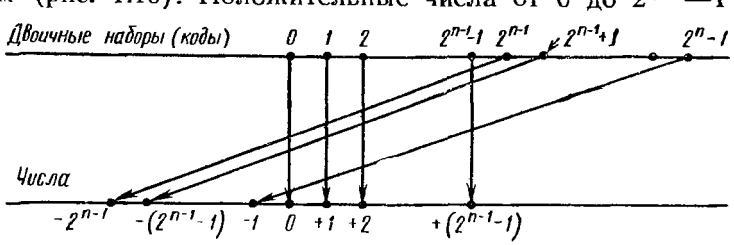


Рис. 1.9. Представление чисел в прямом коде

жения или вычитания. Указанные недостатки привели к тому, что прямой код лишь иногда применяется при вводе и выводе данных.

Дополнительный код. В дополнительном коде сохраняется общий формат знаковых целых чисел, показанный на рис. 1.8, но отображение двоичных наборов на числовую ось становится другим (рис. 1.10). Положительные числа от 0 до $2^{n-1}-1$ пред-



1 10

Рис. 1.10. Представление чисел в дополнительном коде

ставляются без всяких изменений, как в прямом коде, а для отрицательных чисел введено специальное кодирование. Так, число —1 кодируется набором 11 ... 11, а число -2^{n-1} — 100 ... 00. Еще одно наглядное изображение чисел в дополнительном коде показано на рис. 1.11. Вне круга показаны все 4-битные наборы, а внутри — представляемые ими числа в дополнительном коде.

Получение дополнительного кода отрицательного числа $-X$ осуществляется по следующему правилу:

$$X_{\text{дк}} = 2^n - |X|,$$

где n — длина машинного слова.

Пример 1.10. Образовать дополнительный код числа —1429 ($n=16$).

—1429		—0000	0101	1001	0101	—0595H
		10000	0000	0000	0000	2^{16}
	—	0000	0101	1001	0101	
$(-1429)_{\text{дк}}$		1111	1010	0110	1011	FA6BH

Стандартное получение дополнительного кода несколько неудобно из-за операции вычитания, поэтому обычно дополнительный код отрицательного числа $-X$ образуют по следующему правилу.

■ Правило пятое. Для получения дополнительного кода отрицательного числа $-X$ необходимо записать n-битный модуль этого числа. Затем все биты инвертируются, т. е. заменяются на противоположные, и к полученному числу прибавляется 1.

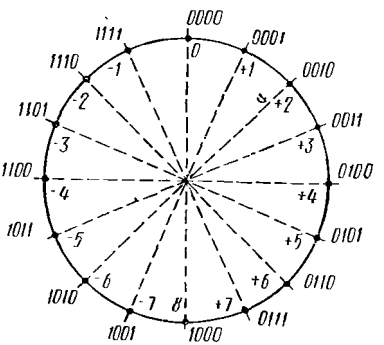


Рис. 1.11. Круговая диаграмма дополнительного кода

Пример 1.11. Найти дополнительный код числа —1429 ($n=16$).

—1429	—0000	0101	1001	0101	—0595
Модуль	0000	0101	1001	0101	0595
Инверсия	1111	1010	0110	1010	FA6A
+					
Плюс 1				1	
$(-1429)_{\text{дк}}$	1111	1010	0110	1011	FA6B

Это же правило можно сформулировать по-другому.

■ Правило шестое. Для получения дополнительного кода числа $-X$ необходимо записать n-битный модуль этого числа, затем вычесть единицу и инвертировать все биты.

Оба правила (пятое и шестое) элементарно доказываются при условии, что инвертирование n-битного числа X эквивалентно вычитанию его из числа 11 ... 11, равного 2^n—1. Тогда, например, по пятому правилу имеем:

инвертировать все биты	$2^n-1-\|X\|$
прибавить 1	$2^n-1-\|X\|+1=2^n-\|X\|$

Действия по шестому правилу принимают вид:

вычесть 1	$\|X\|-1$
инвертировать все биты	$(2^n-1)-(\|X\|-1)=2^n-\|X\|$

В обоих случаях результатом оказывается дополнительный код числа $-X$.

Когда получение дополнительного кода осуществляется вручную, удобно воспользоваться следующим правилом.

■ Правило седьмое. Для получения дополнительного кода числа $-X$ необходимо записать n-битный модуль этого числа. Затем следует просматривать число справа налево, сохранить все младшие нули и первую встретившуюся 1, а остальные биты инвертировать.

Пример 1.12. Найти дополнительный код числа —1408 ($n=16$).

—1408	—0000	0101	1000	0000	—0580
			↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	Сохранить
—1408	0 0 0 0	0 1 0 1	1 0 0 0	0 0 0 0	
	↑ ↑ ↑ ↑	↑ ↑ ↑ ↑			Инвертировать
$(-1408)_{дк}$	1111	1010	1000	0000	FA8

Аналогичные правила (пятое и шестое) справедливы и для перехода от дополнительного кода отрицательного числа к симметричному положительному числу. Поэтому, если исходить из дополнительного кода числа, пятое, шестое и седьмое правила превращаются в правила изменения знака числа. Обычно эта операция имеет мнемоническое обозначение NEG (от negate — изменить знак).

Отметим, что в дополнительном коде:

число нуль имеет единственное представление 00 ... 00;

максимальное по модулю отрицательное число -2^{n-1} не имеет симметричного положительного числа, поэтому применительно к нему в операции изменения знака фиксируется особый случай;

инкремент (увеличение на 1) максимального положительного числа $+(2^{n-1}-1)$ как результат дает 2^{n-1}, т. е. максимальное по модулю отрицательное число, поэтому операция инкремента (и декремента, т. е. уменьшения на 1) применима только к беззнаковым числам.

Очень широкое применение дополнительного кода в современных компьютерах объясняется следующими двумя основными причинами. Во-первых, при сложении дополнительных кодов слагае-

мых как беззнаковых целых чисел получается дополнительный код суммы (если, конечно, переполнение исключено):

$$X_{дк} + Y_{дк} = (X + Y)_{дк}.$$

При этом знаковые биты суммируются обычным образом, а возникающий при их сложении перенос игнорируется.

Пример 1.13. Сложить в дополнительном коде числа +2500 и —1200 (n=16).

$(+2500)_{дк}$	0000	1001	1100	0100	09C4
—1200	—0000	0100	1011	0000	—04B0
$(-1200)_{дк}$	1111	1011	0101	0000	B50
Сложение	+ 0000	1001	1100	0100	
	1111	1011	0101	0000	
$(+1300)_{дк}$	1 0000	0101	0001	0100	0514

Игнорируется

Во-вторых, любое число в дополнительном коде можно считать младшими битами («хвостом») числа любой длины, если содержимое знакового бита копировать влево.

Пример 1.14. Найти десятичный эквивалент числа $X_{дк}$=A2.

$X_{дк}$	1010	0010	A2
X	—0101	1110	—5E

Получилось число —94.

Скопируем знаковый бит 8 раз влево, т. е. сделаем длину числа равной 16, и найдем десятичный эквивалент полученного числа.

$X_{дк}$	1111	1111	1010	0010	FFA2
X	—0000	0000	0101	1110	—005E

В результате получается такое же число —94.

Таким образом, знаковый бит числа, представленного в дополнительном коде, разрешается копировать влево произвольное число раз, не изменяя при этом значение числа. Эта операция называется *расширением знака*, и она необходима в тех случаях, когда исходные операнды в операциях сложения и вычитания имеют различную длину.

В заключение отметим, что понятие дополнительного кода распространяется на системы счисления с любым основанием q. При этом во всех выкладках вместо двух будет фигурировать число q:

$$X_{дк} = q^n - |X|.$$

Операция инвертирования цифры превращается здесь в нахождение дополнений до $q-1$.

Пример 1.15. Найти десятичный дополнительный код числа —638 (n=5).

—638	—00638
|—638|	00638
Дополнить до 9	99361
Прибавить 1	99362 ← десятичный дополнительный код

Пример 1.16. Найти шестнадцатеричный дополнительный код числа —3A7 (n=4).

—3A7	—03A7
|—3A7|	03A7
Дополнить до F	FC58
Прибавить 1	FC59 ←— шестнадцатеричный дополнительный код.

Целые знаковые числа применяются в программах для представления тех числовых объектов, которые принципиально не могут иметь дробной части, но могут быть положительными и отрицательными. В языках программирования высокого уровня такие числа должны записываться без десятичной точки и объявляются спецификаторами типа SINT (n=8), INTEGER или INT (n=16), LONG или INTEGER4 (n=32).

1.3. ДЕСЯТИЧНЫЕ ЧИСЛА

Как уже отмечалось, существует противоречие между машинным представлением чисел (двоичная система счисления) и представлением чисел в нашей повседневной жизни (десятичные числа). Преобразования между ними, в случае большого объема входных данных и выходных результатов, ведет к заметным потерям процессорного времени. Вместе с тем имеется обширный круг задач, характеризующихся значительными объемами числовых данных и сравнительно простой их обработкой (экономические задачи, статистические расчеты, бухгалтерские выкладки и т. п.). Поэтому потребовалось разработать такие формы представления чисел, в которых каким-либо образом совмещались бы двоичная и десятичная системы счисления. Такие формы получили общее название *двоично-кодированного десятичного* (Binary — Coded Decimal) представления или BCD-кодирования. Общее свойство этих форм заключается в том, что за основу берется десятичное число и каждая его цифра изображается тем или иным двоичным эквивалентом. К настоящему времени из многочисленных систем двоично-десятичного кодирования практическое применение находят только две: упакованные десятичные числа и неупакованные десятичные числа.

Упакованные десятичные числа. В упакованном формате, который часто называют BCD-представлением десятичных чисел, байт содержит две десятичные цифры. Младшая цифра занимает правую тетраду (биты 3 : 0), старшая — левую тетраду (биты 7 : 4). Обе цифры представлены своими двоичными эквивалентами, называемыми также кодом 8421 (по двоичным весам). Размещение десятичных цифр в байте показано на рис. 1.12, *а*.

Многоразрядные упакованные десятичные числа занимают несколько смежных байтов. При необходимости работы со знаковыми числами старшая тетрада (иногда — младшая) старшего байта отводится для знака числа (рис. 1.12, *б*). Для кодирования знака

можно использовать шесть запрещенных тетрад 1010—1111 (или A—F), не представляющих десятичных цифр. Обычно для изображения знака плюс применяется код 1100 (C), а знака минус — код 1101 (D). Из-за необходимости изображения знака многобайтные упакованные десятичные числа имеют нечетное число разрядов.

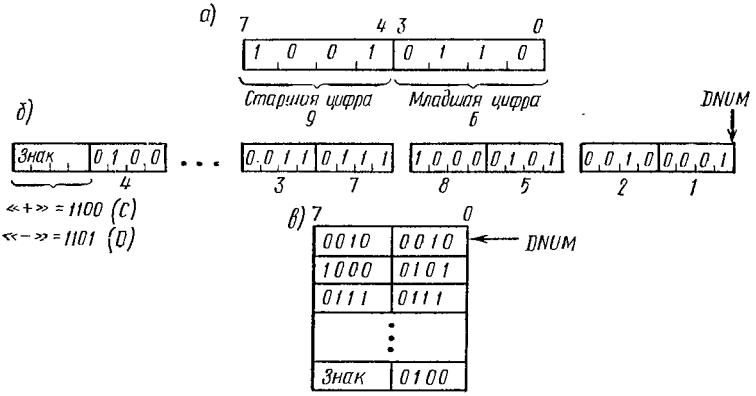


Рис. 1.12. Формат упакованных десятичных чисел:

а — кодирование байта. *б* — многоразрядное число. *в* — размещение его в памяти

При программировании упакованные десятичные числа обычно определяются начальным адресом DNUM (это адрес младшего байта) и числом байтов N. Размещение упакованного десятичного числа в памяти показано на рис. 1.12, *в*.

Неупакованные десятичные числа. Неупакованный десятичный формат, называемый также символьным кодом и кодом ASCII (в переводной литературе), приведен на рис. 1.13. Здесь байт содержит одну десятичную цифру, представленную в коде КОИ-7 (рис. 1.13, *а*). В этом коде десятичным цифрам соответствуют двоичные наборы от 0011 0000 (цифра 0) до 0011 1001 (цифра 9), или в шестнадцатеричной системе от 30H до 39H. Следовательно, можно считать, что собственно значение десятичной цифры занимает младшую тетраду и дается ее двоичным эквивалентом, а в старшей тетраде находится комбинация 0011.

Многоразрядные неупакованные десятичные числа занимают смежные байты (рис. 1.13, *б*). Для знака числа отводится старший байт, в котором комбинация 0010 1011 (2BH) обозначает знак плюс, а комбинация 0010 1101 (2DH) — знак минус. Такие числа определяются в программах их начальным адресом DNUM и числом разрядов (байт) или длиной N. Размещение неупакованного десятичного числа в памяти показано на рис. 1.13, *в*.

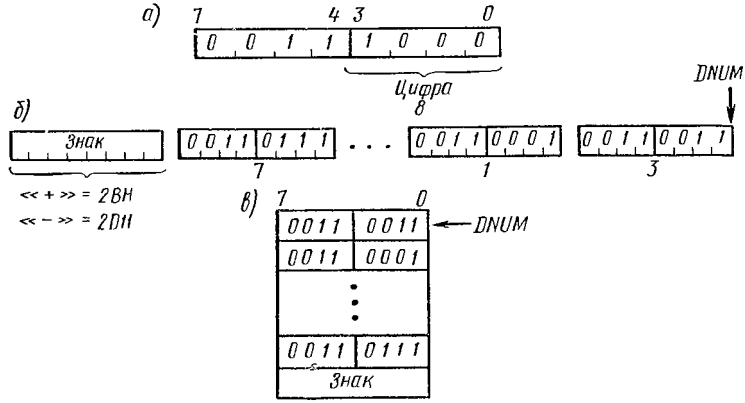


Рис. 1.13. Формат неупакованных десятичных чисел:
а — кодирование байта, *б* — многоразрядное число, *в* — размещение его в памяти

1.4. ФОРМАТЫ ЧИСЕЛ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Рассмотренный выше формат чисел с фиксированной точкой — не единственный из-за своих ограничений на диапазон чисел и представимые числовые значения (только целые числа). Очень мало задач, для решения которых достаточно оперировать только целыми числами. Поэтому для компьютеров разработан еще один формат представления чисел — *формат с плавающей точкой*, называемый *полулогарифмическим* или *экспоненциальным форматом* или же *научной записью*.

В вычислениях с большими и малыми числами применяется следующая запись чисел:

$$X = +0.000035 = +3.5 \times 10^{-5} = 0.35 \times 10^{-4} = +35 \times 10^{-6} = \ldots$$

$$X = -19000000 = -19 \times 10^{+6} = -1.9 \times 10^{+7} = -0.19 \times 10^{+8} = \ldots$$

Здесь число X изображается как произведение некоторого другого числа на степень 10, т. е. основания системы счисления. Так как основание системы счисления известно, его можно не указывать, а взять от второго сомножителя только показатель степени. Другими словами, первый сомножитель записывается как есть, а от второго сомножителя $10^{\pm N}$ записывается только $\pm N$, т. е. его десятичный логарифм, что объясняет термин «полулогарифмический формат». Чтобы не путать знак показателя степени со знаком арифметической операции (сложение или вычитание), перед показателем степени принято записывать букву E (от слова Exponent — показатель степени, экспонента). Следовательно, число записывается в виде

$$\pm \text{M} \text{E} \pm \text{П}.$$

Число ±М называется *мантиссой* (или *значащей частью*); мантисса может быть любым знаковым числом: целым, правильной дробью или смешанным числом. Нетрудно заметить, что знак мантиссы определяет знак всего числа. Число ±П называется *порядком* (*экспонентой*); порядок может быть только целым знаковым числом. Буква Е — разделитель мантиссы и порядка. Чтобы сообщить о представлении числа с повышенной или двойной точностью, вместо буквы Е записывается буква D (от слова Double — двойной). В этом случае для мантиссы отводится больше разрядов.

Порядок ±П определяет истинное или фактическое положение десятичной точки вместо положения, которое она занимает в изображении мантиссы. Если порядок положительный, точка перемещается («плывет» — отсюда и название «плавающая точка») вправо на число разрядов, равное значению порядка.

Пример 1.17. Найти значение числа —0,0956 Е+2.

$$-0.0956\text{E} + 2 = -0.0956 \times 10^{+2} = -009.56 = -9.56$$

Если порядок отрицательный, точка перемещается («плывет») влево.

Пример 1.18. Найти значение числа +1.289 Е—3.

$$+1.289\text{E} - 3 = +1.289 \times 10^{-3} = +0.001289 = +0.001289$$

Переход к изображению чисел с плавающей точкой в двоичной системе счисления не вызывает никаких трудностей: порядок и мантисса становятся двоичными числами, а основанием системы счисления служит число 2:

$$\pm M \times 2^{\pm П} = \pm M\text{E} \pm П.$$

Как и в десятичной системе счисления, порядок определяет истинное положение двоичной точки, а перемещение ее осуществляется по двоичным разрядам.

Пример 1.19. Найти десятичные эквиваленты двоичных чисел с плавающей точкой.

$$-0.000101\ \text{E}+100 = -0.000101 = -1\ 1/4$$
$$+10.111\text{E} - \ 11 = +0.010111 = +23/64$$

Рассмотренное представление чисел имеет один существенный недостаток: запись числа оказывается неоднозначной.

Пример 1.20. Эквивалентные формы чисел с плавающей точкой.

$$-35.5 = -0.353\ \text{E}+2 = -0.00353\ \text{E}+4 = -35300\ \text{E}-2 = \ldots$$
$$+101_2 = +0.101_2\ \text{E}+11_2 = +0.001101_2\ \text{E}+101_2 = +1010.0_2\ \text{E}-1 = \ldots$$

Чтобы исключить неоднозначность записи чисел, во всех компьютерах принято *нормализованное* представление чисел с плавающей точкой, требующее, чтобы мантисса была правильной дробью

и старшая цифра ее отличалась от нуля. Из приведенного выше примера этим требованиям удовлетворяют только числа — 0.353 E+2 и 0.101_2 E+11_2.

Кроме однозначности нормализованное представление обеспечивает также сохранение максимального количества значащих цифр мантиссы в результатах операций.

Пример 1.21. Пусть длина мантиссы составляет 6 бит. Определить нормализованное и ненормализованное представления числа +43 ($+101011_2$).

Нормализованное представление: $+0.101011_2$ E+110_2

Ненормализованное представление: $+0.000101\,011_2$ E+1001_2

Во втором случае три младших бита мантиссы оказываются потерянными и истинное значение числа равно $+\frac{5}{64}\times 2^{+9}=40$, а не 43!

Классический формат с плавающей точкой. Формат чисел с плавающей точкой, который применялся в компьютерах первого и второго поколений, представлен на рис. 1.14. Он состоит из 4 полей:

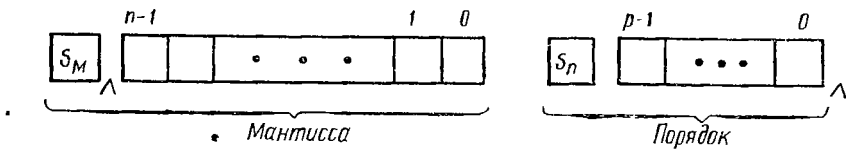


Рис. 1.14. Классический формат чисел с плавающей точкой

знак мантиссы S_M (он совпадает со знаком всего числа), n — битная мантисса, являющаяся правильной дробью, знак порядка $S_П$ и p-битный порядок, который является целым числом. Мантисса и порядок представлены в прямом коде — знак и абсолютное значение.

Пример 1.22. Пусть $n=5$ и $p=3$. Найти десятичные эквиваленты чисел с плавающей точкой.

1	11011	0	101	мантисса = —27/32, порядок = +5, число = $-27/32\times 2^{+5}=-27$
0	11000	1	100	мантисса = +3/4, порядок = —4, число = $+3/4\times 2^{-4}=+3/64$
0	11111	0	111	мантисса = 31/32, порядок = +7, число = $+31/42\times 2^{+7}=+124$
0	10000	1	111	мантисса = +1/2, порядок = —7, число = $+1/2\times 2^{-7}=+1/256$

Пример 1.23. Произвести обратное преобразование для того же формата с $n=5$ и $p=3$.

Число = —3/128, двоичное представление -0.0000011_2 мантисса = = —0.11000 = —3/4, порядок = -101_2 = —5 1 11000 1 101

Число = +47.5, двоичное представление $+101111.1_2$ мантисса = +0.10111 11_2 = +23/32, порядок = $+1.10_2$ = +6 0 10111 0 110

Последний пример показывает, что число +47.5 невозможно точно представить с принятым количеством битов мантиссы. Вместо него (без учета округления) представляется число +46.

Определим диапазон нормализованных чисел в формате, имеющем p бит порядка и n бит мантиссы. Поскольку для прямого кода диапазон симметричен относительно нуля, производятся расчеты только для положительных чисел. Минимальное число получается, если взять минимальную мантиссу и максимальный (по абсолютному значению) отрицательный порядок, т. е.

$$X_{\text{мин}} = 0\ 100\ldots00\ 1\ 11\ldots11$$

Здесь мантисса равна $+1/2$, а порядок $-(2^p-1)$; поэтому

$$X_{\text{мин}} + 1/2 \times 2^{-(2^p-1)} = 2^{-2^p}.$$

Чтобы получить максимальное число, необходимо взять максимальную мантиссу и максимальный положительный порядок:

$$X_{\text{макс}} = 0\ 111\ldots11\ \ 0\ 11\ldots11$$

Нетрудно установить, что

$$X_{\text{макс}} = +(1-2^{-n}) \times 2^{2^p-1}.$$

Если в мантиссе пренебречь членом 2^{-n}, то

$$X_{\text{макс}} \approx 2^{2^p-1}.$$

Найденные для $X_{\text{мин}}$ и $X_{\text{макс}}$ соотношения показывают, что диапазон представимых чисел определяется только количеством p бит порядка. С увеличением p диапазон очень быстро расширяется как в область очень больших, так и в область очень малых чисел. Воспользовавшись приближенным равенством $2^{10} \approx 10^3$, покажем, насколько быстро расширяется диапазон:

$$\begin{aligned}
p=6 \quad & X_{\text{мин}} = 2^{-2^6} = 2^{-64} \approx 10^{-19}, \\
& X_{\text{макс}} \approx 2^{2^6-1} = 2^{63} \approx 10^{19}, \\
p=7 \quad & X_{\text{мин}} = 2^{-2^7} = 2^{-128} \approx 10^{-38}, \\
& X_{\text{макс}} \approx 2^{2^7-1} = 2^{127} \approx 10^{38}, \\
p=8 \quad & X_{\text{мин}} = 2^{-2^8} = 2^{-256} \approx 10^{-76}, \\
& X_{\text{макс}} \approx 2^{2^8-1} = 2^{255} \approx 10^{76}.
\end{aligned}$$

Увеличение длины порядка всего на один бит удваивает показатель степени у десяти для минимального и максимального чисел.

Рассмотрим, на что влияет количество битов мантиссы n. Возьмем любое число X в формате с плавающей точкой $X = M \times 2^{\Pi}$. На числовой оси ему соответствует определенная точка.

Очевидно, ближайшие к нему представимые числа $X^-_{бл}$ и $X^+_{бл}$ отличаются на единицу младшего бита мантиссы (рис. 1.15):

$$X^-_{бл} = (M - 2^{-n}) \times 2^{П}.$$

$$X^+_{бл} = (M + 2^{-n}) \times 2^{П}.$$

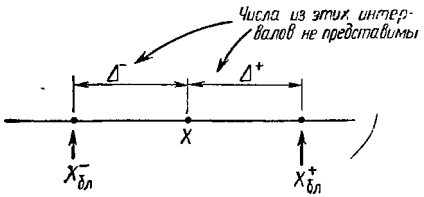


Рис. 1.15. Соседние числа с плавающей точкой на числовой оси

Примечание. Для простоты предполагается, что мантисса числа X не равна 100...00 или 11...11, так как любые уменьшение или увеличение таких значений мантиссы связаны с модификацией порядка. Это ограничение оказывается несущественным.

Определим значение разности между соседними представимыми числами:

$$\Delta^- = X^-_{бл} - X = -2^{-n} \times 2^{П},$$

$$\Delta^+ = X^+_{бл} - X = +2^{-n} \times 2^{П}.$$

Значения Δ^- и Δ^+ равны по абсолютной величине и показывают абсолютную ошибку (погрешность) представления чисел. Обозначим ее через Δ:

$$\Delta = 2^{-n} \times 2^{П}.$$

В вычислениях важнее относительная ошибка δ, равная

$$\delta = \frac{\Delta}{X} = \frac{2^{-n} \times 2^{П}}{M \times 2^{П}} = \frac{2^{-n}}{M}.$$

Чтобы получить максимальное значение δ, необходимо взять минимальную мантиссу, т. е. 1/2:

$$\delta_{макс} = \frac{2^{-n}}{1/2} = 2^{-n+1}.$$

Полученное выражение показывает, что количество битов мантиссы (или *длина* мантиссы) определяет ошибку представления чисел. Рассмотрим несколько примеров:

$$n = 24 \quad \delta_{макс} = 2^{-23} \quad (\approx 10^{-7})$$

$$n = 32 \quad \delta_{макс} = 2^{-31} \quad (\approx 10^{-9})$$

$$n = 56 \quad \delta_{макс} = 2^{-55} \quad (\approx 10^{-16})$$

$$n = 64 \quad \delta_{макс} = 2^{-63} \quad (\approx 10^{-19})$$

Про относительную ошибку 10^{-N} говорят, что числа представимы с точностью до N-го десятичного разряда или точность пред-

ставления чисел составляет N десятичных разрядов (цифр). Поэтому, например, длина мантиссы $n=56$ соответствует точности представления чисел до 16-го десятичного разряда.

Для формата с плавающей точкой из выражения для абсолютной ошибки $\Delta=2^{-n}\times 2^{\Pi}$ следует, что величина Δ зависит от порядка числа и при изменении порядка на единицу изменяется вдвое, но отношение Δ к числу остается примерно постоянным. Таким образом, представимые числа «сгущаются» в области малых значений и «разрежаются» в области больших значений (рис. 1.16). Количество представимых чисел между соседними целыми степенями двух одно и то же. Например, между 1/4 и 1/2 находится столько же представимых чисел, сколько между 4 и 8, между 16384 и 32768 и т. д. Объясняется это просто: при фиксированном порядке количество представимых чисел определяется длиной мантиссы n и равно 2^{n-1} (с учетом нормализации).

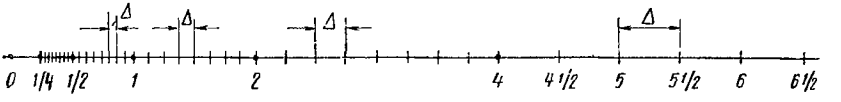


Рис. 1.16. Расположение на числовой оси чисел с плавающей точкой

В рассматриваемом формате имеется одно особое число — нуль. Оно представляется нулями во всех разрядах и называется *истинным* (или *ненормализованным*) *нулем*. В отличие от него иногда вводится так называемый *машинный* (или *нормализованный*) *нуль*, под которым понимается минимальное представимое число $X_{\text{мин}}$. Машинный нуль может быть положительным и отрицательным.

Как бы ни был огромен диапазон представимых чисел в формате с плавающей точкой, в программах могут возникать ситуации, когда получающийся результат выходит за пределы диапазона представимых чисел. Такие ситуации называются *особыми случаями*.

Получение результата, который больше максимального представимого числа $X_{\text{макс}}$, называется *переполнением* или переполнением порядка (порядок результата больше 0.11 ... 11). При возникновении переполнения продолжение программы не имеет смысла, поэтому в процессоре генерируется внутреннее прерывание и операционная система прекращает выполнение программы, выдавая сообщение о переполнении.

Если результат операции оказывается меньше минимального представимого числа $X_{\text{мин}}$, возникает особый случай *антипереполнения* или *исчезновения порядка* (порядок результата меньше 1 11 ... 11). Особый случай антипереполнения не является столь катастрофическим, как переполнение. В большинстве компьютеров при возникновении антипереполнения как результат операции возвращается истинный или машинный нуль.

В некоторых операциях, например сложения или вычитания, как результат операции может получиться число, мантисса которого равна нулю. Очевидно, независимо от значения порядка такое число равно нулю (его иногда называют *псевдонулем*). Такая ситуация называется потерей значимости и естественная реакция на ее возникновение — возвращение результата, равного истинному или машинному нулю.

Формат с плавающей точкой ЕС ЭВМ. В форматах чисел с плавающей точкой Единой Системы ЭВМ, которые показаны на рис. 1.17, введены два существенных изменения по сравнению с рассмотренным выше форматом. Во-первых, в поле порядка (биты 1—7) содержится не истинное значение показателя степени, а так называемый *смещенный порядок* E или *характеристика*. Характеристика представляет собой целое беззнаковое число из диапазона 0—127, равное истинному порядку, увеличенному на 64. Поэтому минимальное значение характеристики 0000000 соответствует истинному порядку —64, а максимальное 1111111_2 (127) — истинному порядку +63. Благодаря такому кодированию становится ненуж-

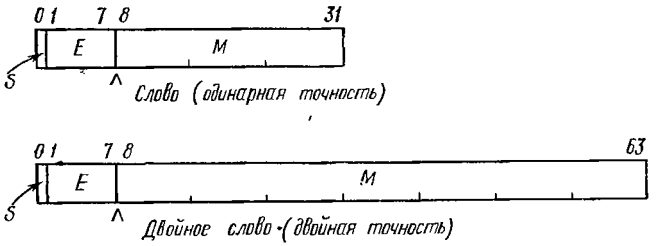


Рис. 1.17. Форматы чисел с плавающей точкой ЕС ЭВМ

ным специальный бит знака порядка (знак порядка совпадает с инвертированным старшим битом характеристики) и можно сравнивать нормализованные числа с плавающей точкой как целые числа. Операция сравнения, реализуемая в большинстве компьютеров путем вычитания сравниваемых операндов, выполняется для целых чисел намного быстрее, чем стандартная операция вычитания чисел с плавающей точкой.

Во-вторых, при определении величины числа за основание, возводимое в степень (равную истинному порядку), взято число 16, а не 2. Как будет показано далее, такой прием значительно расширяет диапазон представимых чисел и несколько увеличивает скорость выполнения арифметических операций. Переход к основанию 16 означает, что перемещать точку в зависимости от значения порядка приходится не через отдельные биты, а через группы из четырех битов (тетрады).

Левый бит отведен для единственного знакового бита, содержащего знак S мантиссы и всего числа. Биты 8—31 или 8—63

занимает мантисса М, которая по-прежнему считается правильной дробью. Условием нормализации становится отличие от нуля четырех старших битов мантиссы, т. е. в битах 8—11 не разрешается комбинация 0000. Таким образом, число становится равным

$$X=(-1)^S \times M \times 16^{E-64}.$$

Определим диапазон и точность представления чисел в формате ЕС ЭВМ. Для нормализованных чисел в формате слова, называемом также коротким форматом или форматом с одинарной точностью, имеем:

Минимальное число $X_{мин}$ 0 0000000 0001 0000 ... 0000

Характеристика равна 0, порядок равен —64 и мантисса равна $^1/_{16}$:

$$X_{мин}=+1/16 \times 16^{-64}=2^{-260}\approx+10^{-78}.$$

Максимальное число $X_{макс}$ 0 1111111 1111 1111 ... 1111.

Характеристика равна 127, порядок равен +63 и мантисса равна $(1-16^{-6})$:

$$X_{макс}=+(1-16^{-6}) \times 16^{+63}\approx+2^{+252}\approx\pm 10^{+76}.$$

У нормализованных чисел в формате двойного слова, называемом также длинным форматом или форматом с двойной точностью, диапазон получается практически таким же:

$$X_{мин}=+1/16\times16^{-64}=+2^{-260}\approx+10^{-78},$$

$$X_{макс}=+(1-16^{-14}) \times 16^{+63}\approx+2^{+252}\approx+10^{+76}$$

Оба формата различаются только точностью представления чисел: точность короткого формата соответствует 6—7 десятичным разрядам, а длинного формата — 16—17 десятичным разрядам.

Для того чтобы расширить диапазон в области малых чисел, в ЕС ЭВМ допускаются ненормализованные числа. Минимальное представимое число в коротком формате имеет вид

$$X_{мин}=0\ 0000000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001$$

$$X_{мин}=+16^{-6}\times 16^{-64}=+16^{-70}=2^{-230}\approx+10^{-84}.$$

Минимальное представимое число в длинном формате:

$$X_{мин}=+16^{-14} \times 16^{-64}=+16^{-78}=2^{-312}\approx+10^{-84}.$$

Разумеется, при переходе к ненормализованным числам ухудшается точность их представления, так как сокращается количество значащих цифр мантиссы из-за появления слева нулей.

Формат с плавающей точкой СМ ЭВМ. В этом формате, показанном на рис. 1.18, также имеются отличия от классического формата. Левый бит отведен для знака числа. Затем следует 8-битный смещенный порядок, смещение равно 128 (1000 0000).

Следовательно, диапазон истинного порядка составляет от —128 (характеристика равна 0000 0000) до +127 (характеристика равна $1111\ 1111_2=255$). Основанием, которое возводится в степень, равную истинному порядку, вновь стало 2. Наиболее интересное отличие рассматриваемого формата от классического формата и от формата ЕС ЭВМ — представление мантиссы. Мантисса по-прежнему считается правильной дробью и должна быть нормали-

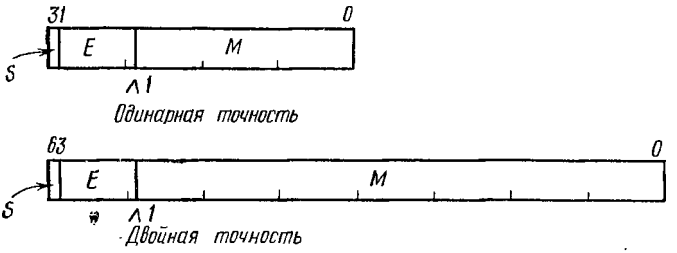


Рис. 1.18. Форматы чисел с плавающей точкой СМ ЭВМ

зованной, т. е. старший бит ее для всех представимых чисел содержит единицу 1: .1XX ... XXX. В целях экономии памяти и повышения точности представления чисел старшая «1» явно не фигурирует, образуя так называемый *скрытый бит*. Когда число передается из памяти в процессор, скрытый бит превращается в явный и в операции участвует 24- или 56-битная мантисса. При записи результата в память старший бит мантиссы не передается и она становится 23- или 55-битной. Таким образом, число равно

$$X=(-1)^S \times .1(M) \times 2^{E-128}.$$

Определим диапазон и точность представления чисел в формате СМ ЭВМ. Для нормализованных чисел в коротком формате имеем

$X_{мин}$ 0 00000000 000 0000 0000 0000 0000 0001

$X_{мин} = +(1/2+2^{-24})\times 2^{-128} \approx +2^{-129} \approx +10^{-39}$

$X_{макс}$ 0 11111111 111 1111 1111 1111 1111 1111

$X_{макс} = +(1-2^{-24})\times 2^{+127} \approx +2^{+127} \approx +10^{+38}$

У нормализованных чисел в длинном формате диапазон представимых чисел практически такой же. Короткий и длинный форматы различаются только точностью представления чисел: точность короткого формата соответствует 6—7 десятичным разрядам, а длинного формата — 16—17 десятичным разрядам.

1.5. СТАНДАРТ НА АРИФМЕТИКУ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Выше были рассмотрены действующие форматы чисел с плавающей точкой. Наличие нескольких форматов создает трудности в обеспечении мобильности программ, т. е. возможности переноса программ, разработанных для одного компьютера, на другие. Кроме того, компьютеры разных семейств по-разному реагируют на особые случаи, возникающие при выполнении программ. Эти и другие причины потребовали упорядочения форматов чисел с плавающей точкой, унификации реакций на особые случаи и точного определения результатов операций, что привело к принятию за рубежом стандарта на арифметику с плавающей точкой. Он был введен как стандарт IEEE-754 и получил широкое распространение. Вновь разрабатываемые системы, как правило, подчиняются требованиям стандарта. Например, представление чисел и правила выполнения операций в арифметическом сопроцессоре К1810ВМ87 удовлетворяют стандарту (см. гл. 4). По-видимому, стандарт получит международное признание и будет способствовать мобильности вычислительных программ. В стандарте определены четыре формата чисел с плавающей точкой.

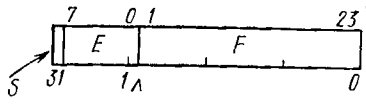


Рис. 1.19. Базовый одинарный формат

Базовый одинарный формат. Поля этого 32-битного формата для двоичного числа X с плавающей точкой приведены на рис. 1.19. Формат содержит знаковый бит S, 8-битный смещенный порядок E и 23-битную дробь F. Значение v числа X определяется по следующим правилам:

если E=255 и $F \neq 0$, то v является не-числом,
если E=255 и F=0, то $v=(-1)^S \times \infty$ (знаковая бесконечность),
если $0<E<255$, то $v=(-1)^S \times 2^{E-127} \times (1.F)$,
если E=0 и $F \neq 0$, то $v=(-1)^S + 2^{-126} \times (0.F)$,
если E=0 и F=0, то $v=(-1)^S \times 0$ (знаковый нуль).

Таким образом, в этом формате смещение равно 127, используется скрытый бит F_0 целой части мантиссы, а минимальный (E=0) и максимальный (E=255) смещенные порядки зарезервированы для представления специальных чисел. Введено, в частности, специальное кодирование для знаковой бесконечности. Диапазон представимых чисел базового одинарного формата составляет $\pm 10^{\pm 38}$, а точность — 6—7 десятичных разрядов.

Базовый двойной формат. Поля этого 64-битного формата для числа X с плавающей точкой показаны на рис. 1.20. Он содержит знаковый бит S, 11-битный смещенный порядок E и 52-битную дробь F. Значение v числа X определяется по следующим правилам:

если F=2047 и $F \neq 0$, то v является не-числом,
если E=2047 и F=0, то $v=(-1)^S \times \infty$ (знаковая бесконечность),

если $0<E<2047$, то $v=(-1)^S\times 2^{E-1023}\times(1.F)$,
если $E=0$ и $F\neq 0$, то $v=(-1)^S\times 2^{-1022}\times(0.F)$,
если $E=0$ и $F=0$, то $v=(-1)^S\times 0$ (знаковый нуль)

Здесь смещение равно 1023, используется скрытый бит F_0 целой части мантиссы, а минимальный ($E=0$) и максимальный ($E=2047$) смещенные порядки отведены для представления специальных чисел. Этот формат аналогичен предыдущему, но диапазон и точность представления чисел значительно увеличены. Диапазона $\pm 10^{\pm 308}$ и точности в 16—17 десятичных разрядов этого формата достаточно для подавляющего большинства практических применений.

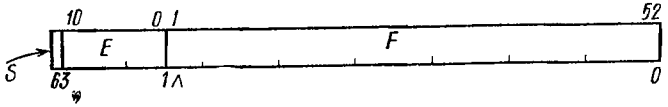


Рис. 1.20. Базовый двойной формат

Расширенный одинарный формат. Расширенные форматы зависят от реализации, т. е. их параметры жестко не фиксируются. Двоичное число X с плавающей точкой в расширенном одинарном формате имеет четыре поля: знаковый бит S, смещенный порядок E (диапазон и смещение зависят от реализации), один явный или скрытый бит F_0 целой части мантиссы и дробь F длиной не менее 31 бит. Диапазон порядка должен находиться между минимальным значением $m\leqslant -1023$ и максимальным значением $M\geqslant 1024$. Значение v числа X определяется по следующим правилам:

если $E=M$ и $F\neq 0$, то v является не-числом,
если $E=M$ и $F=0$, то $v=(-1)^S\times\infty$ (знаковая бесконечность),
если $m<E<M$, то $v=(-1)^S\times 2^{E'}\times(F_0.F)$,
если $E=m$ и F_0 или F ненулевые, то $v=(-1)^S\times 2^{E'}\times(F_0.F)$, где E′ равно m или $m+1$ в зависимости от реализации,
если $E=m$ и $F_0=F=0$, то $v=(-1)^S\times 0$ (знаковый нуль).

Расширенный двойной формат. Этот формат аналогичен предыдущему, но диапазон порядка должен составлять от $m\leqslant -16383$ до $M\geqslant +16384$, а дробь F должна иметь минимум 63 бит.

Все реализации, удовлетворяющие рассматриваемому стандарту, должны, как минимум, поддерживать базовый одинарный формат. Рекомендуется также поддержка одного из расширенных форматов. Подробнее об этом стандарте см. гл. 4.

1.6. ОСОБЕННОСТИ ВЫПОЛНЕНИЯ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ В МИКРОПРОЦЕССОРАХ

Любое действие в микропроцессорах, в том числе и арифметические операции, осуществляются специальными командами. Система команд содержит группу команд, реализующих арифметические операции и определяющих вычислительные возможности. Ко-

нечно, команды этой группы для микропроцессоров разных классов существенно различаются. Например, в 8-битном микропроцессоре КР580ИК80 есть только команды сложения и вычитания, а для умножения и деления приходится разрабатывать подпрограммы. 16-битный микропроцессор К1810ВМ86 имеет команды всех арифметических операций, а в сопроцессоре К1810ВМ87 есть команды таких сложных операций, как извлечение квадратного корня, возведение в степень, логарифмирование и др. В этом параграфе рассмотрены принципы выполнения арифметических операций над целыми числами, числами с плавающей точкой и десятичными числами.

1.6.1. ОПЕРАЦИИ НАД ЦЕЛЫМИ ЧИСЛАМИ

Арифметические флажки. В арифметических операциях важная роль отводится так называемым *признакам* или *флажкам,* показывающим особенности полученного результата операции. Любой микропроцессор имеет несколько арифметических флажков, состояния которых после выполнения команды дают общую характеристику результата. Наиболее широко применяются следующие флажки.

1. *Флажок переноса,* обозначаемый C, CY (CarrY) и CF (Carry Flag), фиксирует цифру переноса из старшего бита при сложении чисел, т. е. он представляет собой расширение результата на один бит влево. В операции вычитания флажок переноса превращается во флажок заема (borrow) и устанавливается в 1, когда уменьшаемое меньше вычитаемого при интерпретации операндов как целых беззнаковых чисел. Очевидно, состояние флажка переноса при сложении беззнаковых чисел можно считать признаком переполнения: 0 — переполнения нет, 1 — возникло переполнение. При сложении и вычитании знаковых чисел состояние флажка переноса самостоятельного значения не имеет. Этот флажок играет важную роль и интенсивно используется в программах, так как позволяет на малоразрядном микропроцессоре обрабатывать числа любой длины, обеспечивая связь между отдельными частями операндов.

2. *Флажок вспомогательного* или *дополнительного переноса,* обозначаемый A (Auxiliary — дополнительный) или AF, показывает при сложении перенос, а при вычитании заем из младшей тетрады (т. е. бита 3) результата. Применение этого флажка, недоступного программисту, ограничено командами десятичной арифметики.

3. *Флажок нуля,* обозначаемый Z (Zero) и ZF, своим состоянием показывает получение нулевого или ненулевого результата операции: $Z=0$ — результат не равен нулю, $Z=1$ — получен нулевой результат.

4. *Флажок знака* обозначается S (Sign), SF и N (Negative —

отрицательный) и своим состоянием повторяет значение старшего бита результата. Так как в дополнительном коде старший бит является знаковым, то S=0, если получен положительный результат, и S=1, если результат операции отрицательный.

5. *Флажок переполнения* обозначается O, OV (OVerflow — переполнение) или OF. В операциях сложения и вычитания знаковых чисел этот флажок показывает, находится ли результат внутри диапазона представимых чисел: V=0 — результат правильный, переполнения нет, V=1 — возникло переполнение. Флажок переполнения относится только к операциям над знаковыми числами. Кроме операций сложения и вычитания флажок переполнения действует и в операции деления, показывая, что частное слишком велико для отведенного ему места. В операции умножения состояние V=1 означает, что старшая половина произведения содержит значащие разряды.

Сложение. Во всех микропроцессорах обязательно имеется команда сложения, обычно с мнемоникой ADD, осуществляющая сложение двоичных чисел, длина которых равна длине машинного слова. Операнды команды ADD считаются целыми беззнаковыми числами, т. е. она единообразно суммирует все биты операндов. Особенность дополнительного кода заключается в том, что операнды команды ADD можно интерпретировать и как целые знаковые числа — результат будет правильным (конечно, с учетом возможного переполнения). Различие между формами представления операндов приводит к различной интерпретации состояний флажков. Покажем это на примерах, используя операнды длиной в один байт.

Пример 1.24	Беззнаковые числа	Знаковые числа
а) 0110 1010	106	(+106)
+ 0010 1110	+ 46	+ (+46)
1001 1000	152	−104(?)
C=0, S=1, Z=0, V=1		
б) 1011 0111	183	(−73)
+ 0010 1110	+ 46	+ (+46)
1110 0101	229	−27
C=0, S=1, Z=0, V=0		
в) 0100 0111	71	(+71)
+ 1110 0011	+ 227	+ (−29)
0010 1010	42(?)	+42
C=1, S=0, Z=0, V=0		

г) 1100 0000 + 1010 1101 = 0110 1101

C=1, S=0, Z=0, V=1

192 + 173 = 109(?)

(—64) + (—83) = +109(?)

д) 1110 0011 + 0001 1101 = 0000 0000

C=1, S=0, Z=1, V=0

227 + 29 = 0(?)

(—29) + (+29) = 0

Флажок переполнения устанавливается в 1, если значения переноса в старший бит и из старшего бита не совпадают. Действительно, при сложении переполнение может возникнуть, когда знаки операндов одинаковы, а само переполнение заключается в том, что при сложении положительных чисел получается отрицательная сумма (есть перенос в старший бит, но нет переноса из старшего бита) или при сложении отрицательных чисел получается положительный результат (есть перенос из старшего бита, но нет переноса в старший бит).

Когда длина операндов превышает длину машинного слова, сложение приходится выполнять в несколько приемов, для чего организуется программный цикл. Операцию необходимо начать с младших частей операндов и «продвигаться» затем в сторону старших частей. При этом следует учитывать переносы, возникающие при каждом сложении и фиксируемые во флажке переноса. Покажем выполнение операции на примере сложения 24-битных операндов в микропроцессоре, длина слова которого составляет 8 бит (байт).

Пример 1.25. Сложить операнды 11001000 01110101 11101111 и 00010010 11111111 11001011.

Начинается сложение с младших байтов:

```
    11101111
  + 11001011
  ----------
1<-- 10111010      флажок C = 1
```

Суммируются средние байты с учетом единицы переноса:

```
        1          перенос от сложения младших байтов
    01110101
  + 11111111
  ----------
1<-- 01110101      флажок C = 1
```

Наконец, суммируются старшие байты чисел:

```
        1          перенос от сложения средних байтов
    11001000
  + 00010010
  ----------
0<-- 11011011      флажок C = 0
```

Получен результат 11011011 01110101 10111010, флажок переноса находится в состоянии 0.

Из примера видно, что в системе команд весьма желательно иметь команду, которая не только суммирует операнды, но и прибавляет в младший бит значение флажка переноса (0 или 1). Такая команда есть во всех микропроцессорах, она называется сложением с переносом и обычно имеет мнемонику ADC (ADd with Carry). Ее очень удобно использовать в программном цикле сложения длинных операндов, обеспечив, чтобы при сложении младших частей флажок переноса находился в состоянии 0. Команда ADC воздействует на арифметические флажки точно так же, как и команда ADD.

К командам сложения обычно относят и однооперандную команду инкремента (INCrement) с мнемоникой INC или INR, осуществляющую увеличение значения своего единственного операнда на 1. Отличительная особенность этой команды заключается в том, что она воздействует на все арифметические флажки, за исключением флажка переноса, который при ее выполнении сохраняет свое текущее состояние. Объясняется эта особенность тем, что наиболее часто команда инкремента применяется для модификации указателей памяти и значение переноса здесь не играет никакой роли.

Вычитание. В системах команд всех микропроцессоров есть двухоперандная команда вычитания с мнемоникой SUB (SUBtract), в которой определяются уменьшаемое и вычитаемое, а разность обычно помещается на место уменьшаемого. Операнды этой команды, как и операнды команды сложения, можно интерпретировать как целые беззнаковые или знаковые числа при условии представления знаковых чисел в дополнительном коде. Команда вычитания воздействует на все арифметические флажки, причем флажок переноса становится флажком заема.

Во всех микропроцессорах выполнение команды вычитания производится в два этапа: вначале образуется дополнительный код вычитаемого, а затем уменьшаемое и модифицированное вычитаемое суммируются. При этом проявляется особенность: состояние флажка переноса (т. е. заема) противоположно фактическому значению переноса при сложении уменьшаемого и дополнительного кода вычитаемого. Рассмотрим действие команды вычитания на нескольких примерах.

Пример 1.26.

а) Уменьшаемое равно 0101 0111, вычитаемое равно 0100 1000. Обычное вычитание:

— 0101 0111	— 87	— (+87)
0100 1000	72	(+72)
0←—0000 1110	15	+15

Дополнительный код вычитаемого равен 1011 1000. Сложение его с уменьшаемым даст:

$$\begin{array}{rll} & 0101 & 0111 \\ + & 1011 & 1000 \\ \hline 1\leftarrow\!\!- & 0000 & 1111 \end{array}$$

$C = 0,\ S = 0,\ Z = 0,\ V = 0$

б) Уменьшаемое равно 0111 0111, вычитаемое равно 1011 0101. Дополнительный код вычитаемого равен 0100 1011 и при сложении его с уменьшаемым получим:

$$\begin{array}{rll} & 0111 & 0111 \\ + & 0100 & 1011 \\ \hline 0\leftarrow\!\!- & 1100 & 0010 \end{array} \qquad \begin{array}{rl} + & 119 \\ & 181 \\ \hline & 194(?) \end{array} \qquad \begin{array}{rl} - & (+119) \\ & (-\ 75) \\ \hline & -62\ (?) \end{array}$$

$C = 1,\ S = 1,\ Z = 0,\ V = 1$

При интерпретации операндов как целых беззнаковых чисел возник заем ($119 < 181$), а если операнды считать знаковыми числами, возникло переполнение.

в) Уменьшаемое равно 1000 1111, вычитаемое равно 1111 1111. Дополнительный код вычитаемого равен 0000 0001. Сложив его с уменьшаемым, получим:

$$\begin{array}{rll} & 1000 & 1111 \\ + & 0000 & 0001 \\ \hline 0\leftarrow\!\!- & 1001 & 0000 \end{array} \qquad \begin{array}{rl} - & 143 \\ & 255 \\ \hline & 144(?) \end{array} \qquad \begin{array}{rl} - & (-113) \\ & (-1) \\ \hline & -112 \end{array}$$

$C=1,\ S=1,\ Z=0,\ V=0.$

Когда длина операндов больше длины машинного слова, вычитание, как и сложение, осуществляется в несколько приемов в программном цикле. Операция начинается с младших частей операндов с последовательным переходом к старшим частям. При этом необходимо учитывать заемы, которые фиксируются во флажке переноса. Рассмотрим вычитание 24-битных чисел в микропроцессоре с длиной слова 8 бит.

Пример 1.27. Уменьшаемое равно 11001000 01110101 11101111, а вычитаемое 00010010 11111111 11001011. Сначала производится традиционное вычитание:

$$\begin{array}{rlll} & 1 & 0 & \leftarrow\!\!-\!\!-\text{межбайтные заемы} \\ - & 11001000 & 01110101 & 11101111 \\ & 00010010 & 11111111 & 11001011 \\ \hline & 10110101 & 01110110 & 00100100\leftarrow\!\!- \text{ разность} \end{array}$$

В микропроцессоре эта операция выполняется следующим образом. Вычитаются младшие байты операндов, для чего находится дополнительный код байта вычитаемого (00110101) и складывается с байтом уменьшаемого:

$$\begin{array}{rll} & & 11101111 \\ + & & 00110101 \\ \hline 1\leftarrow\!\!- & 00100100 & \text{флажок } C = 0 \end{array}$$

Вычитаются средние байты операндов; дополнительный код среднего байта вычитаемого равен 00000001 и при сложении его со средним байтом уменьшаемого имеем:

```
  01110101
+ 00000001
----------
0←—01110110      флажок C = 1
```

Флажок C будет установлен в 1 и заем необходимо учесть (т. е. вычесть) при вычитании старших байтов операндов. Дополнительный код старшего байта вычитаемого равен 11101110 и сложение его со старшим байтом уменьшаемого дает:

```
  11001000
+ 11101110
----------
1←—10110110      флажок C = 0
```

Теперь необходимо учесть единицу заема, т. е. вычесть 1, которая в дополнительном коде имеет вид 11111111:

```
  10110110
+ 11111111
----------
1←—10110101      флажок C = 0
```

Полученный результат совпадает с разностью, найденной выше прямым вычитанием.

Вычитание единицы заема можно осуществить и по-другому: когда флажок переноса находится в состоянии 1, следует брать не дополнительный код вычитаемого, а его обратный код, получающийся простым инвертированием всех битов. Поэтому при сложении старших байтов уменьшаемого и обратного кода вычитаемого получим:

```
  11001000
+ 11101101
----------
1←—10110101      флажок C = 0
```

Рассмотренное показывает, что для вычитания чисел с длиной, большей длины машинного слова, удобно иметь команду, которая не просто вычитает два числа, но и автоматически учитывает состояние флажка переноса (заема). Такая команда называется вычитанием с заемом и имеет мнемонику SBB (SuBtract with Borrow). Действие ее описываем следующим образом, полагая, что разность замещает уменьшаемое:

уменьшаемое←уменьшаемое — вычитаемое — состояние заема

Команду SBB легко встраивать в программные циклы вычитания. Она воздействует на арифметические флажки так же, как команда вычитания SUB.

К командам вычитания относится однооперандная команда

декремента (DECrement) с мнемоникой DEC и DCR. Она производит уменьшение на 1 значения своего единственного операнда. Команда декремента воздействует на все арифметические флажки, за исключением флажка переноса, сохраняющего текущее состояние. Объясняется эта особенность команды декремента тем, что наиболее часто она применяется для модификации счетчиков циклов и указателей памяти. В таких применениях команды декремента знать значение переноса (возник он или нет) не требуется.

Еще одна команда, относящаяся к вычитанию, называется изменением знака (NEGate) и обычно имеет мнемонику NEG. По существу она эквивалентна вычитанию из нуля значения своего единственного операнда. При использовании дополнительного кода эта команда может вызвать особый случай (обычно интерпретируемый как перевыполнение), когда ее операнд имеет вид 100 ... 00. Такое отрицательное число -2^{n-1} не имеет равного по абсолютному значению положительного числа.

К вычитанию обычно относится и двухоперандная команда сравнения (COMpare) с мнемоникой CMP и COM. В этой команде допускается такая же спецификация операндов, как и в команде вычитания, и она повторяет все действия команды вычитания, за исключением одного: результат вычитания не замещает уменьшаемое и вообще нигде не сохраняется. Следовательно, позитивным итогом команды сравнения являются состояния арифметических флажков, показывающие отношение между операндами, а ни один из операндов не изменяется.

При использовании команды сравнения необходимо отчетливо представлять, что ее операнды допускают интерпретацию как беззнаковых, так и знаковых чисел в дополнительном коде, что влияет на содержательный смысл состояний флажков. Наиболее просто выявляется равенство (Z=1) и неравенство (Z=0) операндов: об этом сигнализирует флажок нуля Z независимо от интерпретации операндов. Отношение «больше — меньше» в случае беззнаковых чисел показывает флажок переноса: если первый операнд (уменьшаемое) больше второго (вычитаемого), то флажок переноса устанавливается в состояние 0, и наоборот. Следующие примеры показывают, что отношение «больше — меньше» для знаковых чисел проверяется суммой по модулю 2 состояний флажков S и V: если первый операнд больше второго, то $S \oplus V = 0$.

Пример 1.28.

	X	Y	−Y	X−Y
а)	0010 0010	0000 0100	1111 1100	+ 0010 0010
	+ 34	+ 4		1111 1100
				1 ← 001 1110

$C = 0,\ S = 0,\ Z = 0,\ V = 0,\ S \oplus V = 0$

б) 0010 0010 1111 1100 0000 0100

+34 −4

$$\begin{array}{rr} & 0010\ 0010 \\ + & 0000\ 0100 \\ \hline 0\leftarrow & 0010\ 0110 \end{array}$$

$C = 1,\ S = 0,\ Z = 0,\ V = 0,\ S \oplus V = 0$

в) 1111 1100 1101 1110 0010 0010

−4 −34

$$\begin{array}{rr} & 1111\ 1100 \\ + & 0010\ 0010 \\ \hline 1\leftarrow & 0001\ 1110 \end{array}$$

$C = 0,\ S = 0,\ Z = 0,\ V = 0,\ S \oplus V = 0$

г) 0111 1111 1111 1100 0000 0100

+127 −4

$$\begin{array}{rr} & 0111\ 1111 \\ + & 0000\ 0011 \\ \hline 0\leftarrow & 1000\ 0011 \end{array}$$

$C = 1,\ S = 1,\ Z = 0,\ V = 1,\ S \oplus V = 0$

д) 0000 0100 1000 0001 0111 1110

+4 −127

$$\begin{array}{rr} & 0000\ 0100 \\ + & 0111\ 1110 \\ \hline 0\leftarrow & 1000\ 0010 \end{array}$$

$C = 1,\ S = 1,\ Z = 0,\ V = 1,\ S \oplus V = 0$

Умножение. Если в системе команд микропроцессора имеется команда умножения, никаких трудностей операция умножения, естественно, не вызывает. Но если команда умножения отсутствует или формат операндов не удовлетворяет требованиям имеющейся команды умножения, приходится разрабатывать программную реализацию умножения. Рассмотрим основные принципы программирования умножения на примере беззнаковых чисел. Пусть сомножителями X и Y являются n-битные беззнаковые числа:

$$\text{множимое } X = x_{n-1}x_{n-2}\ldots x_1x_0,$$

$$\text{множитель } Y = y_{n-1}y_{n-2}\ldots y_1y_0.$$

Представим множитель Y в развернутой форме:

$$Y = y_{n-1}2^{n-1} + y_{n-2}2^{n-2} + \ldots + y_1 2^1 + y_0 2^0$$

и найдем произведение:

$$Z = XY = (Xy_{n-1})\,2^{n-1} + (Xy_{n-2})\,2^{n-2} + \ldots + (Xy_1)\,2^1 + (Xy_0)\,2^0.$$

Произведение $z_i = (Xy_i)2^i$ множимого на один бит множителя называется *частичным произведением,* а сумма k первых или последних частичных произведений называется *k-й суммой частичных произведений.* При $k = n$ сумма частичных произведений превращается в полное произведение Z. Произведение n-битных сомножителей имеет длину $2n$ бит и уменьшить ее невозможно.

Нетрудно заметить, что частичное произведение z_i равно либо нулю (когда $y_i=0$), либо множимому X с учетом веса 2^i (когда $y_i=1$). Умножение реализуется циклическим процессом, на каждом шаге которого:

анализируется очередной бит y_i множителя;

в зависимости от его значения происходит ($y_i=1$) или нет ($y_i=0$) прибавление множимого к предыдущей сумме частичных произведений;

производится изменение взаимного положения множимого X и суммы частичных произведений с учетом веса 2^i.

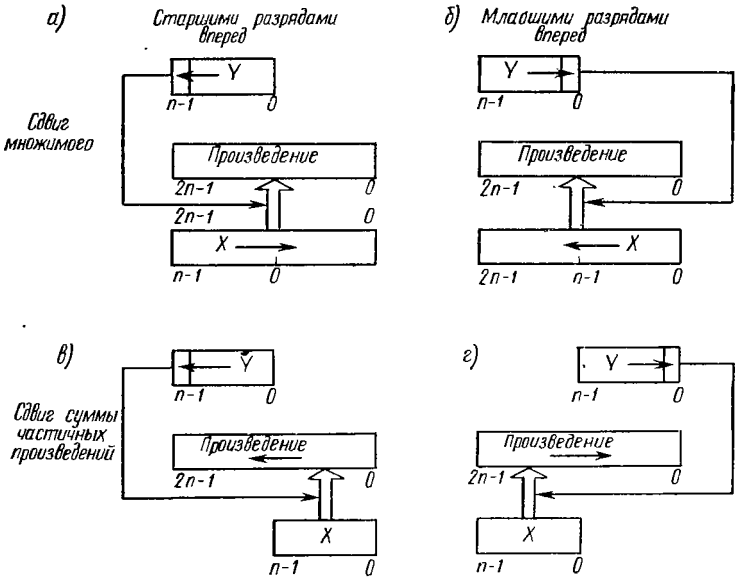


Рис. 1.21. Варианты умножения целых двоичных чисел

В соответствии с выражением для произведения Z существует четыре возможных вариаита умножения, показанных на рис. 1.21. Они различаются тем, с каких разрядов множителя Y (младшего или старшего) начинается умножение и что сдвигается — множимое или сумма частичных произведений. Вариант умножения, начиная с младших или старших разрядов множителя, называется еще *умножением младшими* или *старшими разрядами вперед.*

Когда сомножители представлены в дополнительном коде, ни один из приведенных вариантов не дает правильного произведения в дополнительном коде. В этом случае стандартный прием выполнения операции заключается в реализации следующего алгоритма:

по знаковым битам сомножителей образовать (сложением по модулю 2) и временно сохранить знак произведения;

образовать абсолютные значения сомножителей и умножить их, пользуясь любым вариантом;

с учетом знака произведения представить результат в дополнительном коде.

Для сомножителей, представленных в дополнительном коде, используется еще один алгоритм умножения:

проанализировать знак множителя и при $Y<0$ умножить сомножители на -1, т. е. сделать Y положительным;

выполнить умножение по любому из вариантов с такими изменениями:

а) при сдвиге X вправо знаковый бит сохраняет свое значение и копируется в соседний правый бит (арифметический сдвиг вправо);

б) во всех битах старшей половины X первоначально должна находиться копия знакового бита;

в) при суммировании множимого значение X должно быть дополнено старшими копиями знакового бита;

г) изменений нет.

При программировании операции умножения можно реализовать любой вариант, но для получения эффективных программ приходится учитывать особенности микропроцессора и системы команд (см. гл. 2).

Деление. Операция деления является обратной по отношению к умножению и реализуется похожими циклическими действиями. Обозначим через X — делимое, Y — делитель и $Z=X/Y$ — частное, считая их целыми беззнаковыми числами. При делении целых чисел принято как дополнительный результат формировать еще и остаток R. Для операции деления характерен случай деления на нуль ($Y=0$). Обычно он фиксируется как переполнение, но иногда учитывается отдельно.

Рассматривают две разновидности операции деления:

делимое, делитель и частное имеют одну и ту же длину n,

делимое имеет двойную длину по сравнению с делителем и частным.

Схема выполнения операции в обоих случаях одна и та же, но в первом переполнение возможно только при нулевом делителе, а во втором — кроме того, когда старшая половина делимого больше делителя.

Если умножение, по существу, сводится к последовательности сложений, то деление — к последовательности вычитаний делителя вначале из делимого, а затем из остатков. Деление двоичных чисел по сравнению с десятичными оказывается намного проще благодаря тому, что цифрами частного могут быть только 0 и 1. Следовательно, цифра z_{n-i} частного определяется просто: если текущий остаток r_{i-1} больше или равен делителю, цифра частного рав-

на 1, а если $r_{i-1}<Y$, то цифра частного равна 0. Сравнение в компьютерах приводится к вычитанию, поэтому общий алгоритм деления включает в себя следующие шаги:

вычесть делитель Y из остатка r_{i-1}, полученного на предыдущем шаге (за начальный остаток r_0 принимается делимое X), и образовать остаток r_i;

если $r_i<0$, то $z_{n-i}=0$ и необходимо вернуться к r_{i-1}, прибавив к r_i делитель (это действие называется *восстановлением остатка*); если же $r_i\geqslant 0$, то $z_{n-i}=1$ и за очередной остаток принимается r_i;

скорректировать взаимное положение остатка r_i и делителя Y (что-то одно сдвигается на один бит) и повторить действия по определению следующей цифры z_{n-i-1} частного.

Так как частное можно определить только со старших разрядов, то существует два варианта деления, показанных на рис. 1.22. В первом варианте на каждом шаге производится сдвиг делителя вправо, а во втором — сдвиг остатка влево.

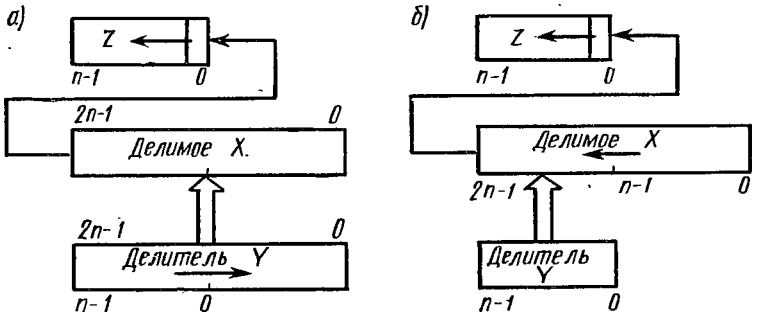


Рис. 1.22. Варианты деления целых двоичных чисел

Двоичная система счисления обладает замечательным свойством, позволяющим во втором варианте деления устранить действия по восстановлению остатка. Пусть на i-м шаге получился отрицательный остаток — r_i. Восстановление его заключается в получении предыдущего остатка $r_{i-1}=-r_i+Y$. После этого r_{i-1} сдвигается влево с образованием $2r_{i-1}=-2r_i+2Y$, а затем осуществляется вычитание делителя Y: $r_{i+1}=(2r_i+2Y)-Y=-2r_i+Y$. Таким образом, при получении отрицательного остатка r_i необходимо сдвинуть r_i влево и прибавить делитель Y. Этот прием легко реализуется схемно, но его программное воплощение громоздко.

Когда делимое и делитель представлены в дополнительном коде, можно применить алгоритм деления, в котором выбор на каждом шаге сложения или вычитания основывается на анализе знаков остатка и делителя. Однако его программная реализация малоэффективна и обычно применяется такой же способ, как и в

умножении: отдельно определяется знак частного, затем производится деление абсолютных значений операндов и в заключение образуется дополнительный код частного.

1.6.2. ОПЕРАЦИИ НАД ЧИСЛАМИ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Машинное представление числа с плавающей точкой состоит из двух частей, имеющих совершенно различный смысл: мантисса, являющаяся правильной дробью, содержит значащие разряды числа, а порядок (целое знаковое число) показывает фактическое положение двоичной точки в цифрах мантиссы. Следовательно, в арифметических операциях обработка мантисс и порядков должна производиться по разным правилам. В микропроцессорах широкого назначения нет команд, оперирующих числами с плавающей точкой. Поэтому при необходимости производства вычислений с такими числами приходится разрабатывать подпрограммы соответствующих операций. В арифметическом сопроцессоре (см. гл. 4) формат чисел с плавающей точкой является единственным внутренним форматом.

Сложение. Предположим, что требуется сложить два числа X и Y, представленных в формате с плавающей точкой:

$$X = m_x П_x, \quad Y = m_y П_y.$$

Поскольку порядки $П_x$ и $П_y$ показывают положение двоичной точки, а складывать можно только одноименные разряды слагаемых, первым действием операции сложения должно быть так называемое *выравнивание* (или *уравнивание*) *порядков*. Другими словами, необходимо представить числа X и Y в такой форме, в которой их порядки одинаковы. В случае $П_x = П_y$ выравнивать порядки не нужно. Из двух исходных порядков $П_x$ и $П_y$ один будет меньше другого. Пусть для определенности $П_y < П_x$ и разность порядков $\Delta П = П_x - П_y$ положительна.

Выравнивание порядков осуществляется двумя способами: либо сделать общим порядком меньший из них ($П_y$), либо считать общим порядком больший из них ($П_x$). В любом случае для сохранения значения числа придется трансформировать мантиссу того числа, порядок которого будет изменяться. Для уменьшения большего порядка ($П_x$) до меньшего ($П_y$) сдвигается мантисса m_x на $\Delta П$ разрядов влево, т. е. в направлении старших разрядов. Такой подход потребует для хранения мантисс регистров двойной длины, так как ни один выдвигаемый слева бит отбросить нельзя из-за его большой значимости. Поэтому остается только второй вариант, который и принят во всех компьютерах: меньший порядок ($П_y$) увеличивается до большего $П_x$. При этом для сохранения значения числа Y его мантисса сдвигается вправо на $\Delta П$ разрядов. При этом происходит выдвигание младших битов мантиссы m_y, которые при ограниченной длине регистров мантисс теряются. Их по-

теря, однако, существенного значения не имеет, поскольку выдвигаются и теряются младшие биты мантиссы. Отсюда следует вывод о том, что операция сложения, будучи абсолютно точной для целых чисел, принципиально связана с ошибками (погрешностями) для чисел с плавающей точкой.

Для выравнивания порядков необходимо найти разность $\Delta\Pi$ порядков (как целых знаковых чисел), по ее знаку определить больший из порядков Π_x или Π_y и сдвинуть мантиссу числа с меньшим порядком на $\Delta\Pi$ разрядов вправо. Порядком результата оказывается общий (больший) порядок, а двоичная точка в представлениях обеих мантисс будет в одном и том же месте.

Рассмотрим примеры выравнивания порядков для классического формата с плавающей точкой (в дальнейшем покажем особенности этого процесса для чисел в форматах ЕС ЭВМ и СМ ЭВМ). Предполагаем, что длина мантиссы $n=5$, а длина порядка $p=3$.

Пример 1.29. Даны числа

X: $m_x = 1$ 11011 $\quad \Pi_x = 0$ 101 $\quad$ (X = — 27)

Y: $m_y = 0$ 10110 $\quad \Pi_y = 0$ 010 $\quad$ (Y = +2 3/4)

Разность порядков $\Delta\Pi = \Pi_x - \Pi_y = (+5) - (+2) = +3$. Знак плюс показывает, что порядок Π_x больше порядка Π_y и, следовательно, мантиссу m_y необходимо сдвигать на три бита вправо:

$$m_y = 0 \;\; 00010 \;\; (110) \;\; \Pi_y = 0 \;\; 101 \;\; (Y = +2).$$

При сдвиге выдвигаются и теряются младшие биты мантиссы m_y, содержащие 110. Чтобы повысить точность операции сложения, обычно регистры мантисс «удлиняют» вправо на несколько битов.

Пример 1.30. Даны числа:

X: $m_x = 0$ 10011 $\quad \Pi_x = 1$ 001 $\quad$ (X = +19/64)

Y: $m_y = 1$ 11101 $\quad \Pi_y = 0$ 011 $\quad$ (Y = — 7 1/4)

Разность порядков $\Delta\Pi = \Pi_x - \Pi_y = (-1) - (+3) = -4$. Знак минус показывает, что $\Pi_y > \Pi_x$, поэтому мантиссу m_x потребуется сдвигать на 4 бита вправо:

$$m_x = 0 \;\; 00001 \;\; (0011) \;\; \Pi_x = 0 \;\; 011 \;\; (X = +1/4).$$

Общий порядок слагаемых стал равным 0 011, т. е. +3.

Очевидно, если $|\Delta\Pi| \geqslant n$, где n — длина мантисс, мантисса числа с меньшим порядком вся выдвинута вправо и сумма определяется сразу: она равна числу с большим порядком.

Второй этап сложения чисел с плавающей точкой заключается в том, чтобы сложить мантиссы как числа с фиксированной точкой, так как в обеих мантиссах двоичная точка находится в одном месте. Для сложения мантисс можно применить те же способы, что и рассмотренные в п. 1.6.1. Продолжим предыдущие примеры 1.28 и 1.29. Введем для удобства слева дополнительный бит.

Пример 1.31. Имеем $m_x = 1$ 11011, $m_y = 0$ 00010.

Образуем дополнительные коды мантисс и сложим их:

$$\begin{array}{ll} m_x & 11\ \ 00101 \\ m_y & +\ 00\ \ 00010 \\ \hline m_z & 11\ \ 00111 \end{array}$$

В результате получена отрицательная мантисса m_z, которая представлена в дополнительном коде. Прямой код: $m_z = 1\ \ 11001$.
Результат сложения:

$$m_z = 1\ \ 11001 \qquad \Pi_z = 0\ \ 101 \qquad (Z = -25)$$

Пример 1.32. Имеем $m_x = 0\ \ 00001$, $m_y = 1\ \ 11101$.
Образуем дополнительные коды мантисс и складываем их:

$$\begin{array}{ll} m_x & 00\ \ 00001 \\ m_y & +\ 11\ \ 00011 \\ \hline m_z & 11\ \ 00100 \end{array}$$

Здесь также получена отрицательная мантисса, поэтому окончательный результат равен

$$m_z = 1\ \ 11100 \qquad \Pi_z = 0\ \ 011 \qquad (Z = -7)$$

При сложении мантисс может возникнуть нарушение нормализации вправо и влево. Нарушение нормализации вправо происходит при сложении мантисс с разными знаками и близких по абсолютному значению; оно заключается в том, что старший бит мантиссы результата оказывается нулевым. Такое нарушение нормализации ликвидируется простым действием сдвига мантиссы влево до тех пор, пока в ее старшем бите не будет 1. При каждом сдвиге мантиссы для сохранения значения числа уменьшается порядок на 1.

Пример 1.33. Даны числа:

$$X: m_x = 1\ \ 10011 \qquad \Pi_x = 0\ \ 101 \qquad (X = -19)$$
$$Y: m_y = 0\ \ 10001 \qquad \Pi_y = 0\ \ 101 \qquad (Y = +17)$$

Выравнивать порядки не требуется, так как $\Pi_x = \Pi_y$. Складываем мантиссы:

$$\begin{array}{ll} m_x & 11\ \ 01101 \\ m_y & +\ 00\ \ 10001 \\ \hline m_z & 11\ \ 11110 \end{array}$$

Мантисса результата $m_z = 1\ \ 00010$ и имеет нарушение нормализации вправо. Для устранения его m_z необходимо три раза сдвинуть влево и уменьшить порядок Π_z на 3:

$$m_z = 1\ \ 10000 \qquad \Pi_z = 0\ \ 010 \qquad (Z = -2)$$

При ликвидации нарушения нормализации вправо может оказаться, что порядок результата достиг своего минимального значения (у нас оно равно 1 111), а процедура нормализации требует его дальнейшего уменьшения. Такая ситуация называется *исчезновением порядка* или *антипереполнением;* она свидетельствует о том, что результат меньше минимального представимого нормализованного числа. Здесь можно предпринять два дейст-

вия. Простейшая и традиционно предпринимаемая реакция — возвратить как результат операции нуль. Второй подход, обеспечивающий большую точность,— оставить результат ненормализованным и разрешить ему в таком виде участвовать в дальнейших вычислениях.

При сложении чисел с плавающей точкой следует учитывать и случай, когда сложение мантисс дает нуль (мантиссы одинаковы по абсолютному значению, но имеют разные знаки). Этот случай называется *потерей значимости.* Независимо от значения порядка результат будет равен нулю (его иногда называют *псевдонулем*), поэтому обычно предусматривается превращение такого числа в истинный нуль.

При сложении мантисс с одинаковыми знаками может возникнуть и нарушение нормализации влево максимум на один разряд. Оно устраняется путем сдвига мантиссы вправо и увеличения порядка на 1.

Пример 1.34. Даны числа:

X: $m_x = 0$ 10010 $\quad \Pi_x = 0$ 100 $\quad$ (X = +9)

Y: $m_y = 0$ 11001 $\quad \Pi_y = 0$ 100 $\quad$ (Y = +12 1/2)

Выравнивать порядки не нужно, так как $\Pi_x = \Pi_y$; суммируем мантиссы:

$$\begin{array}{lll} m_x & + & 00\ 10010 \\ m_y & & 00\ 11001 \\ \hline m_z & & 01\ 01011 \end{array}$$

Возникло нарушение нормализаций влево, поэтому сдвигаем мантиссу вправо и производим инкремент порядка:

$m_z = 0$ 10101 (1) $\quad \Pi_x = 0$ 101 $\quad$ (Z = +21)

При нарушении нормализации влево с отрицательными мантиссами в знаковых битах будет комбинация 10. Старший бит показывает знак мантиссы результата (отрицательный) и его нужно сохранить в окончательном результате. Младший бит после сдвига вправо и преобразования в прямой код даст 1 в старшем бите мантиссы.

Если порядок результата с нарушением нормализации влево равен максимальному (в нашем случае 0 111), то возникло *переполнение.* Обычно оно ведет к прекращению выполнения программы.

При сложении чисел с плавающей точкой в форматах ЕС ЭВМ необходимо учитывать следующие обстоятельства. Во-первых, вычитание характеристик автоматически убирает смещение и разность характеристик ΔX равна разности порядков $\Delta\Pi$. Во-вторых, при получении $\Delta\Pi = N$ необходимо сдвигать мантиссу числа с меньшим порядком на N 16-ричных разрядов, т. е. $4 \times N$ бит.

В третьих, о нарушении нормализации вправо свидетельствует получение 0000 в четырех старших разрядах мантиссы. Сдвиг мантиссы влево осуществляется на 4 бита с декрементом порядка на 1. При нарушении нормализации влево приходится сдвигать мантиссу на 4 бита вправо с инкрементом порядка на 1.

При использовании чисел с плавающей точкой в форматах СМ ЭВМ собственно операция сложения дополняется двумя действиями. Первое действие заключается в восстановлении скрытых единиц мантисс, а второе действие превращает «истинный» результат в формат со скрытым битом мантиссы.

Вычитание. Операция вычитания чисел с плавающей точкой, т. е. получение Z=X—Y, элементарно приводится к операции алгебраического сложения: Z=X+(—Y). Следовательно, вычитание реализуется изменением знака вычитаемого и последующего сложения чисел со всеми особенностями операции сложения.

Умножение. Умножение чисел с плавающей точкой принципиальных трудностей не вызывает, так как из представления чисел в форме

$$X = m_x \times 2^{П_y}, \quad Y = m_y \times 2^{П_y}$$

следует, что

$$Z = X \times Y = (m_x \times m_y) 2^{П_x + П_y}.$$

Другими словами, мантисса произведения равна произведению мантисс сомножителей, а порядок произведения равен сумме их порядков. Для умножения мантисс, как чисел с фиксированной точкой, можно применить любой из рассмотренных в п. 1.6.1 вариантов умножения. Знак произведения определяется отдельным действием путем сложения по модулю 2 знаков сомножителей.

При умножении нормализованных мантисс может возникнуть только нарушение нормализации вправо максимум на один разряд, так как произведение минимальных мантисс, равных 1/2, дает 1/4 (или 0.01_2). Оно устраняется обычным образом: мантисса сдвигается влево на один бит, а порядок уменьшается на 1.

В операции умножения могут возникать как переполнение, так и антипереполнение. Переполнение обнаруживается, когда сумма порядков сомножителей больше максимального допустимого порядка, а антипереполнение — после того, как произведен декремент порядка при устранении нарушения нормализации вправо и порядок оказался меньше минимального допустимого.

Когда умножаются числа с плавающей точкой в форматах ЕС ЭВМ и СМ ЭВМ, учитывается, что сложение характеристик дает результат с удвоенным смещением и для перехода к характеристике произведения из суммы характеристик необходимо вычесть смещение. Кроме того, в формате СМ ЭВМ требуется учитывать скрытый бит мантиссы.

Деление. Из представления делимого X и делителя Y в форме

$$X = m_x 2^{П_x}, \quad Y = m_y 2^{П_y}$$

следует, что частное Z равно

$$Z = X/Y = (m_x/m_y) \times 2^{П_x - П_y}.$$

Таким образом, мантисса частного равна результату деления мантисс операндов как чисел с плавающей точкой, а порядок частного равен разности их порядков. Для деления мантисс применим любой вариант, но обычно применяется вариант со сдвигом остатков влево. Если $m_x > m_y$, деление мантисс дает целую часть, равную 1. Обычно такая ситуация при делении чисел с фиксированной точкой считается переполнением, но в данном случае этого не происходит, так как результирующую мантиссу всегда можно сдвинуть на один бит вправо и соответственно скорректировать (инкремент) порядок. Нарушения нормализации вправо при делении нормализованных чисел не происходит.

В операции деления, как и при умножении, могут возникать переполнение и антипереполнение. Переполнение имеет место, когда после инкремента порядка (для устранения нарушения нормализации влево) получается число, большее максимального допустимого порядка, а антипереполнение — когда разность порядков оказалась меньше минимального допустимого порядка.

При делении чисел с плавающей точкой в форматах ЕС ЭВМ и СМ ЭВМ разность характеристик представляет собой истинный порядок частного, так как смещение уничтожается. Поэтому для получения характеристики частного к разности характеристик требуется прибавить смещение. В формате СМ ЭВМ необходимо учитывать скрытый бит мантиссы. Для операции деления в формате ЕС ЭВМ характерна особенность, связанная с использованием основания 16. Максимальная мантисса делимого равна 1111 1111 ... 1111, а минимальная мантисса делителя равна 0001 0000 ... 0000. Следовательно, при «обычном» делении частное имеет 4 бита целой части и стандартный алгоритм деления не годится. В такой ситуации ($m_x > m_y$) целесообразно сдвинуть мантиссу делимого вправо на 4 бита и произвести инкремент порядка частного.

1.6.3. ОПЕРАЦИИ НАД ДЕСЯТИЧНЫМИ ЧИСЛАМИ

Выше были рассмотрены два основных формата десятичных чисел — упакованный и неупакованный. Как правило, в современных микропроцессорах нет команд, явно оперирующих числами, представленными в этих форматах. Даже в сопроцессоре К1810ВМ87 десятичная «арифметика» представлена всего двумя командами: команда загрузки FBLD передает десятичное число из памяти в регистр сопроцессора, преобразуя его в двоичный формат с плавающей точкой, а команда запоминания FBSTP передает число из внутреннего регистра в память с преобразованием его в десятичный упакованный формат.

В большинстве микропроцессоров ради упрощения их внутренней организации реализовано двухэтапное выполнение арифметических операций с десятичными числами. На первом этапе операнды обрабатываются как целые двоичные числа командами двоичной арифметики, а на втором этапе специальная команда коррекции преобразует промежуточный двоичный результат в десятичный формат. Наиболее полно команды коррекции для обоих десятичных форматов представлены в микропроцессоре К1810ВМ86 (см. гл. 3).

Контрольные вопросы и упражнения

1. Сколько двоичных комбинаций можно представить в 8, 16, 24 и 32 бит?

2. Оцените десятичные эквиваленты следующих степеней двух: 2^{24}, 2^{32}, 2^{40}, 2^{64}.

3. Найдите двоичные веса 2^i для $i=0, 1, 2, \ldots, 15$.

4. Определите десятичные эквиваленты чисел: 3333_4, 274_5, 7777_8, FF_{16}, 1000_{16}, $FFFF_{16}$.

5. Переведите следующие десятичные целые числа в двоичную, восьмеричную и шестнадцатеричную системы счисления: 255, 512, 1023, 2049, 4000.

6. Преобразуйте следующие правильные десятичные дроби в двоичные, восьмеричные и шестнадцатеричные: 0.125, 0.875, 17/64, 2/3, 4/19. Всегда ли в ответе периодическая двоичная дробь?

7. Найдите десятичные эквиваленты указанных двоичных чисел: 10110.1, 11111111, 101010101010, 0.11011, 111111111.111.

8. Найдите двоичные и десятичные эквиваленты шестнадцатеричных чисел: AA, 100, FFF, 2000, FFFF.

9. Представьте в прямом коде ($n=8$) числа +28, —28, +112, —112, +127, —127, —128.

10. Представьте в дополнительном коде те же числа, что и в упражнениях 9.

11. Байт содержит A0H. Дайте десятичные эквиваленты, рассматривая содержимое как беззнаковое число и как знаковое число в прямом и дополнительном кодах. Повторите упражнение для содержимого байта 10H, FFH 80H.

12. Покажите кодирование чисел 375, 1024, 3825 в упакованном и неупакованном десятичных форматах.

13. Какие десятичные числа представляют собой упакованные коды: 00110101, 10000011011001, 0110100000100001010l?

14. Какие десятичные числа представляют собой неупакованные коды: 0011010100111001, 0011100000110010001101000011000?

15. Представьте десятичные числа 4.5, 31, 1023, 4096 в классическом формате с плавающей точкой ($n=12$, $p=5$).

16. Представьте числа 8.25, 511.5, 1024, 4095 в форматах ЕС ЭВМ и СМ ЭВМ (с одинарной и двоичной точностью).

17. Укажите все достоинства представления чисел с плавающей точкой в нормализованной форме.

18. Укажите преимущества использования смещенных порядков для чисел с плавающей точкой.

19. Приведите достоинства и недостатки использования скрытого бита мантиссы для чисел с плавающей точкой.

20. Для базовых форматов стандарта на арифметику с плавающей точкой:
найдите минимальные и максимальные представимые числа;
определите кодирование бесконечностей и не-чисел;
определите минимальные представимые числа, если допускаются ненормализованные числа.

21. Пусть X=01001010, Y=01110000, Z=10000000, W=11110000. Выполните следующие операции в дополнительном коде и определите состояния флажков переноса, знака, нуля и переполнения: X+Y, Y—X, X—Z+W, Y+W, X+Y+ +Z+W.

22. Разработайте граф-схемы программ сложения и вычитания многобайтных чисел в дополнительном коде.

23. Выполните умножение чисел по всем вариантам умножения (n=8): 5× ×7, 12×15, 200×30, 255×4 (сомножители преобразовать в двоичную форму).

24. Осуществите деление чисел по двум вариантам деления (n=8): 12/4, 200/30, 128/35.

25. Разработайте граф-схемы алгоритмов для всех арифметических операций над числами с плавающей точкой.

26. Какой из этапов сложения (вычитания) чисел с плавающей точкой в наихудшем случае потребует максимума элементарных действий?

ГЛАВА 2

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ В МИКРОПРОЦЕССОРЕ КР580ИК80

В этой главе подробно рассмотрено производство арифметических операций в 8-битном однокристальном микропроцессоре КР580ИК80. Вначале даны общие характеристики микропроцессора и его программная (регистровая) модель, содержащая доступные программисту внутренние регистры. Специально выделен материал по аппаратному стеку, который организуется в оперативной памяти и служит удобным расширением внутренних регистров. Приведены минимально необходимые сведения по языку Ассемблер, который применяется для реализации алгоритмов арифметических операций. Рассмотрена система команд микропроцессора с классификацией их по функциональному назначению.

Большую часть гл. 2 составляют программы арифметических операций, операнды и результаты которых представлены в различных форматах. Кроме того, затронуты вопросы преобразований форматов чисел. Настоящая глава играет в пособии ключевую роль, потому что в ней алгоритмы и программы арифметических операций изложены наиболее подробно с приведением необходимого иллюстративного материала.

2.1. ОБЩАЯ ХАРАКТЕРИСТИКА МИКРОПРОЦЕССОРА

Микросхема КР580ИК80 (далее сокращенно К580) представляет собой однокристальный центральный процессор с фиксированными длиной машинного слова (8 бит) и системой команд. Характеристика ее как центрального процессора предполагает, что для разработки функционально законченного изделия — специализированной микропроцессорной системы (микросистемы) или микрокомпьютера широкого назначения к микропроцессору (МП) необходимо подключить память и средства ввода-вывода (ВВ). МП К580 рассчитан на разнообразные применения в качестве ядра системы, выполняющего обработку цифровых данных. Он изготовляется по NMOП-технологии и выпускается в 40-контактном кор-

пусе с двусторонним расположением выводов (типа DIP). На кристалле МП располагается около 5000 транзисторов. Напряжения питания равны +12 В±5%, +5 В±5% и —5 В±5%, потребляемая мощность не превышает 1 Вт, рабочий диапазон температур составляет от —10 до +70°С. МП синхронизируется двухфазными сигналами с частотой до 2 МГц от микросхемы генератора КР580ГФ24.

При разработке прикладных программ в первую очередь интересуются ресурсами системы, находящимися в распоряжении программиста. Внутренние ресурсы МП, входящие в состав его программной или регистровой модели, рассматриваются в § 2.2. Ниже кратко рассматриваются внешние ресурсы системы, представленные подсистемами памяти (запоминающими устройствами) и ввода-вывода.

Обычно память состоит из программного энергонезависимого постоянного запоминающего устройства (ПЗУ), допускающего только считывание хранимой информации, и полупроводниковой энергозависимой оперативной памяти (или запоминающего устройства с произвольной выборкой — ЗУПВ), выполняющей операции считывания и записи. Если тип операции не играет роли, говорят об обращении к памяти или о доступе к ней.

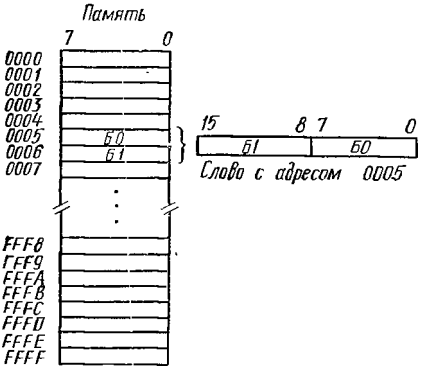


Рис. 2.1. Адресное пространство памяти микропроцессора КР580ИК80

При обращении к памяти МП выдает на внешнюю шину адреса A15—A0 16-битный адрес и отдельный управляющий сигнал, идентифицирующий тип операции. Старшие 8 бит адреса A15—A8 часто называются *адресом страницы*, а младшие 8 бит A7—A0 — *адресом в странице* (или *строкой*). Следовательно, *адресное пространство*, или *поле памяти*, состоит из 64К ячеек (байт); оно показано на рис. 2.1. Можно также считать, что адресное пространство состоит из 256 страниц по 256 строк в каждой. Два любых соседних байта образуют *слово*; адресом слова считается меньший из двух адресов байта, причем по этому адресу хранится младший байт слова. В системах на базе МП К580 понятия логического и физического адресов совпадают, а в других микропроцессорах они могут различаться.

Различные области адресного пространства группируются в блоки из последовательных ячеек, образующие так называемую *карту памяти*. Блоки относятся к аппаратным устройствам (блоки ПЗУ и ЗУПВ) или к программным элементам, например основной

программе, процедуре обработки прерываний или драйверам (подпрограммам) ввода-вывода.

Подсистема ВВ представлена *входными портами* (портами ввода) и *выходными портами* (портами вывода). Данные от устройств ввода подаются во входные порты и считываются микропроцессором по шине данных D7—D0. Передача данных в МП называется вводом или считыванием (загрузкой). Выходные порты воспринимают данные от МП (с шины данных) и передают их в устройства вывода. Передача данных из МП называется выводом или записью. В простейшем случае входные и выходные порты представляют собой буферные регистры с конкретными номерами (адресами) и осуществляют взаимодействие с периферийными устройствами. Многие микросхемы программируемых контроллеров периферийных устройств для программиста выглядят как набор входных/выходных портов.

В системах на базе МП К580 можно организовать один из двух основных способов ВВ.

Первый способ, называемый изолированным ВВ, предполагает наличие специальных команд IN ввода и OUT вывода. Эти команды содержат прямой адрес входного или выходного порта, к которому производится обращение. Длина адреса составляет один байт, поэтому пространство ВВ, показанное на рис. 2.2, состоит из 256 входных и 256 выходных портов. Оно изолировано от пространства памяти в том смысле, что в системе может быть ячейка памяти с адресом А, а также входной и выходной порты с этим же адресом А. Обращения к пространствам памяти и ВВ идентифицируют различные управляющие сигналы. В самом МП для вводимых и выводимых данных выделен специальный регистр (аккумулятор А), поэтому изолированный ВВ называется также *аккумуляторным*.

Рис. 2.2. Адресное пространство ввода-вывода микропроцессора КР580ИК80

Второй способ, называемый ВВ, *отображенным на память*, не требует специальных команд и управляющих сигналов, предназначенных для ВВ. Входные и выходные порты считаются ячейками адресного пространства памяти. Следовательно, все команды с обращением к памяти, содержащие адреса портов, превращаются в команды ВВ: команды загрузки осуществляют ввод, а команды запоминания — вывод.

При детальном анализе способов ВВ оказывается, что ни один из них не имеет явных преимуществ по сравнению с другим, однако ВВ, отображенный на память, считается более гибким и универсальным. К его достоинствам относят большой набор команд для ВВ (причем некоторые из них могут совмещать ВВ с преобразованием данных), почти неограниченное число входных и выходных портов (конечно, за счет сокращения рабочей памяти) и отсутствие специальных управляющих сигналов ВВ.

2.2. ПРОГРАММНАЯ МОДЕЛЬ МИКРОПРОЦЕССОРА

На кристалле МП расположена очень сложная схема из нескольких тысяч транзисторов с многочисленными буферными элементами, арифметико-логическим устройством, блоками синхронизации, схемами реакции на прерывания и т. д. Знать подробное внутреннее устройство МП (его *микроархитектуру*) пользователям не обязательно. Во-первых, оно является недоступным и, во-вторых, по мере увеличения степени интеграции становится чрезмерно сложным. Для программиста достаточно знать *программную модель* МП, показывающую его ресурсы, которые доступны на уровне машинных команд. Другими словами, программная модель показывает, чем может распоряжаться программист. Ответ на вопрос о том, как можно распоряжаться ресурсами МП, дают режимы адресации операндов и система команд. Разработчику аппаратных средств микропроцессорных систем необходимо знать функциональное назначение всех входных и выходных сигналов МП, их нагрузочную способность, многочисленные временные характеристики и временную диаграмму работы. Все это объединяется в понятие *внешнего интерфейса* и учитывается при объединении отдельных компонентов в законченную систему.

Основу программной модели любого МП образуют регистры, поэтому иногда ее называют регистровой моделью. Такая модель МП К580 показана на рис. 2.3. Часто при разработке программ привлекаются регистры данных или регистры общего назначения (РОН), находящиеся в полном распоряжении программиста. В эту группу входит 7 регистров с мнемоническими обозначениями A, B, C, D, E, H, L; эти обозначения применяются для идентификации регистров в командах при программировании на языке Ассемблер. Если говорить об «идеальных» РОН, все эти регистры должны обладать одинаковыми функциональными возможностями. Однако в МП К580 этого нет и РОН оказываются довольно специализированными, что приходится учитывать при программировании.

Наиболее важным и интенсивно используемым в программах является регистр A, называемый *аккумулятором*. Особая роль аккумулятора заключается в том, что команды МП позволяют прибавить операнд только к содержимому аккумулятора, вычесть значение только из содержимого аккумулятора, осуществить ввод

и вывод только через аккумулятор и т. д. При программировании очень быстро выясняется, что аккумулятор оказывается «узким местом» в МП.

Функциональные возможности регистров B, C, D, E примерно одинаковы, но регистры D и E обладают большей гибкостью благодаря команде XCHG, осуществляющей обмен содержимого регистров HL и DE. Об особой роли регистров H и L, считающихся основным указателем памяти в МП, говорится ниже.

Особенность МП К580 заключается в том, что РОН допускается программно объединять в так называемые регистровые пары

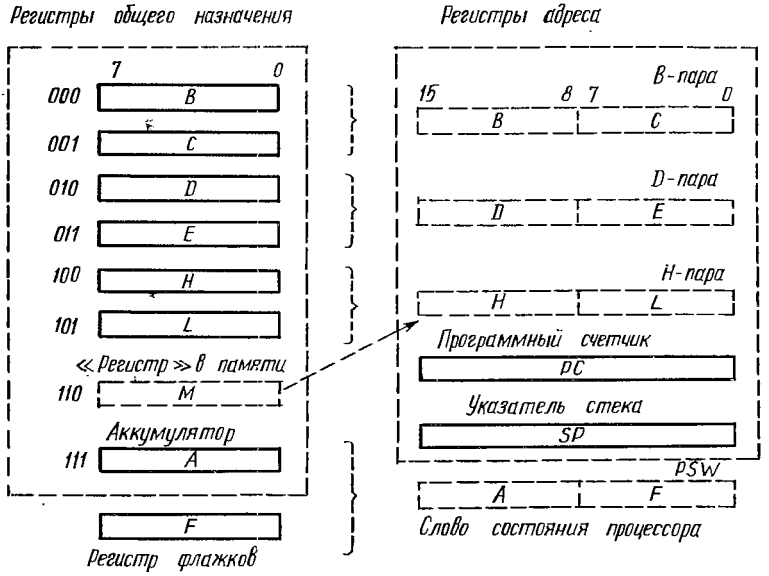


Рис. 2.3. Программная модель микропроцессора КР580ИК80

(register pair), что показано на рис. 2.3 пунктиром. При таком объединении (его называют также сцеплением и конкатенацией) из 8-битных регистров образуются 16-битные регистры, которые можно привлекать для адресации памяти, а также хранения 16-битных данных. Таким образом, содержимое регистровой пары допускает двойную интерпретацию: адрес памяти и 16-битные данные, формат которых определяет программист. Если регистр или, как в данном случае, регистровая пара применяются для адресации памяти, их называют *указателями памяти.*

Наиболее гибкой из всех регистровых пар является H-пара. Объясняется это тем, что адресуемый ею байт в памяти считается «регистром» M микропроцессора (от Memory — память). «Регистр» M имеет трехбитный номер (адрес) 110, и все команды (в которых

фигурируют явные регистры МП) допускают и указание «регистра» М. Например, его содержимое можно передать в любой другой регистр МП, прибавить к аккумулятору или загрузить в «регистр» М константу и т. д. Регистровые пары DE и BC используются для косвенной адресации памяти, но при этом допускаются только операции загрузки (из памяти) и сохранения (в памяти) аккумулятора. Косвенная адресация памяти имеет несколько преимуществ. Во-первых, команды с обращением к памяти становятся короткими и, во-вторых, изменение содержимого регистровой пары позволяет одной и той же командой обращаться к различным ячейкам памяти, что очень удобно для обработки регулярных структур данных.

Как видно из программной модели, в МП есть еще три специализированных регистра. Два из них — программный счетчик PC и указатель стека SP — имеют длину 16 бит и функционируют только как указатели памяти, а 8-битный регистр флажков (F-регистр) предназначен для регистрации некоторых особенностей или признаков результатов операций. Остановимся вкратце на функциях этих регистров.

При выполнении программы необходимо «следить» за ее отдельными командами так, чтобы к моменту окончания текущей команды в специальном регистре МП был образован адрес следующей команды. Такую функцию «слежения» за командами и выполняет *программный счетчик* PC, называемый также *счетчиком команд* и *счетчиком адреса команды*. Всякий раз, когда МП считывает из памяти очередной байт команды, производится увеличение содержимого PC на 1 (эту операцию называют инкрементом). Следовательно, к окончанию выборки из памяти всей текущей команды, т. е. даже до начала собственно ее выполнения, в PC образуется адрес следующей по порядку команды. Если МП после окончания текущей команды обращается за очередной командой именно по этому адресу, то говорят, что реализуется естественный порядок выполнения команд. Однако время от времени по логике программы необходимо нарушать естественный порядок. Такую функцию выполняют специальные команды передачи управления (их иногда образно называют командами управления программой), которые сами задают адрес очередной команды, называемый *адресом перехода*. Все команды передачи управления в МП К580 определяют полный 16-битный адрес перехода. Очевидно, для передачи управления команда должна просто загрузить адрес перехода в PC и МП обратится за следующей командой по этому адресу.

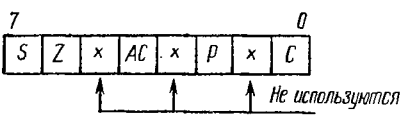


Рис. 2.4. Формат регистра флажков микропроцессора КР580ИК80

Регистр флажков, формат которого показан на рис. 2.4, имеет

длину 8 бит, но используются в нем только 5 бит. Команды воздействуют на флажки по-разному, но, в общем, они показывают следующие признаки результата:

C — флажок переноса (Carry) в операции сложения (вычитания) устанавливается в 1 при наличии переноса (заема) из старшего бита результата; по существу, этот флажок можно считать расширением результата на один бит влево;

S — флажок знака (Sign) своим состоянием повторяет значение старшего (знакового) бита результата;

Z — флажок нуля (Zero) устанавливается в 1 при получении нулевого результата 00000000;

P — флажок паритета (Parity), или четности, устанавливается в 1, если результат содержит четное число единиц;

AC — флажок вспомогательного, или дополнительного, переноса (Auxiliary Carry) устанавливается в 1 при наличии переноса (заема) из младшей тетрады результата в старшую.

Как будет видно далее, наиболее интенсивно в вычислительных алгоритмах используется флажок C. Именно благодаря ему малоразрядный МП может оперировать числами произвольной разрядности. Флажок P в основном предназначен для связных применений МП, а флажок AC позволяет работать с десятичными числами.

В МП К580 — обширный набор команд условных передач управления, осуществляющих выбор одного из двух направлений в ходе программы по состоянию проверяемого флажка. Выполнение таких команд часто называют *принятием решения* или *разветвлением.*

Регистр флажков и аккумулятор объединяются в регистровую пару, содержащую слово состояния процессора PSW (Processor Status Word) только в двух стековых командах: PUSH PSW и POP PSW.

Указатель стека SP (Stack Pointer) предназначен для адресации так называемой вершины стека TOS (Top Of Stack) или ST (Stack Top). Стек играет важную роль в функционировании МП и в разработке программ. Рассмотрим работу стека подробнее.

Стек. Стек, называемый также буфером LIFO (Last In — First Out: последний пришел — первый ушел), обратной магазинной памятью и даже пуш-даун списком, представляет собой область оперативной памяти (ЗУПВ) со своеобразной дисциплиной работы. По существу он, как это будет видно далее, как бы расширяет регистры МП в память, т. е. намного (практически неограниченно, но, разумеется, в пределах выделенной ему области) увеличивает их число.

Содержимое любого регистра МП можно поместить в стек с помощью операции включения (push), а из стека можно извлечь (pop) последние включенные в него данные в регистр МП. При включении данные как бы «кладутся» сверху ранее занятых ячеек стека, а при извлечении «берутся» из верхней ячейки стека TOS. Операции push и pop в литературе называются еще «проталкиванием» и «выталкиванием», но эти термины оказываются неточными, так как они подразумевают «движение» всех элементов стека, чего на самом деле нет.

В МП К580 обе стековые операции выполняются с 16-битными словами,

т. е. в стек можно включать только содержимое регистровых пар и извлекать из стека можно только в регистровые пары. Например, команда PUSH H включает в стек содержимое регистров H и L, а команда POP PSW извлекает из стека данные в регистр флажков и аккумулятор. На рис. 2.5 показано состояние стека при выполнении нескольких команд. Из рисунка нетрудно составить реальную последовательность действий МП при выполнении команды PUSH H (учитывая, что МП может передавать только байты):

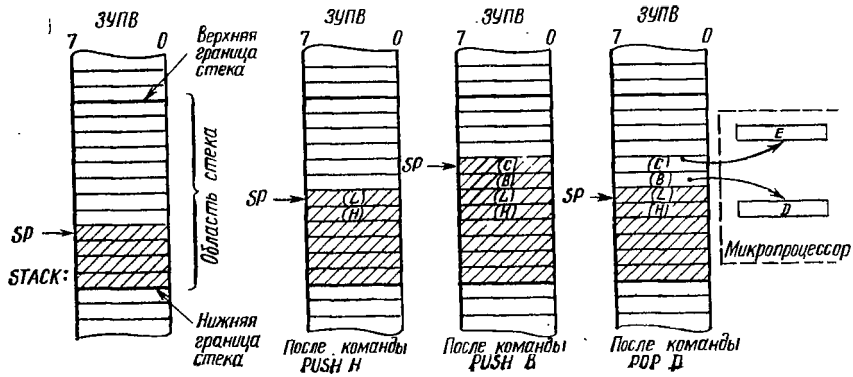


Рис. 2.5. Принцип действия стека

произвести уменьшение на 1 (декремент) указателя стека: SP←(SP)—1,
записать в TOS содержимое регистра H: TOS←(H),
произвести декремент SP: SP←(SP)—1,
записать в TOS содержимое регистра L: TOS←(L).

Аналогичные действия реализуются и при выполнении команды PUSH B, которая включает в стек содержимое регистров B и C. Таким образом, занятые элементы стека по мере включения «продвигаются» в область меньших адресов памяти или, как еще говорят, стек «растет» вверх.

Команда POP D извлечения из стека в D-пару, т. е. регистры D и E, состонт из следующих действий:

считать содержимое TOS и передать его в регистр E: E←(TOS),
произвести инкремент указателя стека: SP←(SP)+1,
считать содержимое TOS и передать его в регистр D: D←(TOS),
произвести инкремент указателя стека: SP←(SP)+1.

Извлечение из стека не разрушает в памяти считываемые данные и, произведя специальными командами декремеит SP на 2, можно вновь извлечь из стека ранее считанные данные. Такой прием иногда используется при программировании.

Необходимо до выполнения первой стековой операции правильно инициализировать указатель стека. Если, например, для стека выделяется область ЗУПВ с конечным адресом STACK, то в указатель стека необходимо загрузить значение STACK+1 (напомним, что первое действие при включении в стек — декремент SP). При программировании следует тщательно следить за использованием стека, в частности, каждой команде PUSH должна соответствовать команда POP. Микропроцессор К580 не имеет никаких средств аппаратного контроля за поведением стека. Ошибки при работе со стеком приводят к двум особым ситуациям. Первая из них, называемая *переполнением* (overlow) стека, возникает, когда SP достиг верхней граннцы области стека и делается попытка произвести дополнительное включение в стек. МП «послушно» выполнит команду включения в стек, хотя за верхней границей ЗУПВ может не быть или оно есть, но занято нужными данными. Ошибочная запись на нужные дан-

ные обычно называется «перезаписью» (overwrite). Вторая ошибочная ситуация называется *антипереполнением* (underflow) стека; она возникает, когда указатель стека находится на нижней границе области стека и делается еще одна попытка извлечения из стека. МП выполнит команду POP, но при этом считываются не те предполагаемые данные, которые планировал получить программист. За нижней границей области стека ЗУПВ может отсутствовать, но МП этого не заметит. Как видно, возникновение любой особой ситуации ведет к тому, что поведение системы становится непредсказуемым (говорят, что возникла «фатальная» ошибка).

Стек наиболее часто применяется для автоматического запоминания адресов возврата при вызовах подпрограмм и для передачи параметров подпрограммам. Передача параметров через стек принята практически во всех языках высокого уровня. Перед вызовом параметры включаются в стек, а затем вызывается подпрограмма. Она может обращаться к переданным параметрам, используя смещения относительно SP. При возврате из подпрограммы параметры удаляются из стека путем соответствующего (числу параметров) инкремента SP. Стек действует при этом как динамическая область памяти, которая выделяется для каждой подпрограммы, а по окончании подпрограммы освобождается.

К стеку удобно прибегать и для временного сохранения любых данных при нехватке внутренних регистров МП. Если, например, при программировании потребовались регистры H и L, а в них находятся нужные данные, содержимое HL следует включить в стек, т. е. «освободить» эти регистры для использования. Через некоторое время сохраненные данные можно вернуть в регистры H и L командой POP H. С помощью стека упрощается решение многих «мелких» задач, встречающихся при программировании. Например, командами

```
PUSH   H        ; Включить содержимое HL в стек
POP    B        ; Извлечь из стека в BC
```

содержимое HL передается в регистры BC.

В МП К580 есть специфическая команда XTHL, которая производит обмен содержимого регистра HL и двух верхних ячеек стека, т. е. последних включенных в стек данных; при этом указатель стека не изменяется. С помощью команд

```
PUSH   B        ; Включить содержимое BC в стек
XTHL            ; Обменять TOS и HL
POP    B        ; Извлечь из стека в BC
```

осуществляется обмен содержимого регистров HL и BC.

Стек играет огромную роль при обработке прерываний. Реагируя на внешние или внутренние прерывания, МП должен переключиться с текущей программы на другую программу, называемую обработчиком прерывания (interrupt handler). Такое контекстное переключение необходимо осуществить так, чтобы после обслуживания прерывания возобновить прерванную программу «как ни в чем не бывало». Для этого требуется где-то временно сохранить содержимое используемых обработчиком прерывания регистров МП (в предельном случае — содержимое всех регистров МП) на время его работы. Удобным временным «хранилищем» состояния МП является стек.

Формат программ на языке Ассемблер. Далее приводятся фрагменты программ на языке Ассемблер, поэтому необходимо иметь минимальные сведения об этом языке.

Исходная программа на языке Ассемблер выглядит как последовательность строк (называемых операторами), допускающих несколько интерпретаций. Строка может быть преобразована в машинную команду, использована для размещения данных в памяти или содержать указания (директивы) только для программы-ассемблера.

Независимо от функций строк исходной программы формат их представления неформально стандартизован и состоит из четырех частей или *полей*:

[*Метка*:] *Операция* [*Операнды*] [; Комментарий]

Здесь квадратные скобки показывают поля, содержимое которых может быть пустым; другими словами, в каждой строке (за естественным исключением целой строки-комментария) только поле операции должно что-то содержать.

Каждое поле состоит из одной или нескольких лексем. Под лексемой понимается наименьшая осмысленная единица информации, используемая программой-ассемблером. В качестве лексем выступают идентификаторы (имена) и числовые константы. Примерами идентификаторов служат символические имена (мнемоники) машинных команд. Для указания конца лексемы применяется разделитель, которым чаще всего является пробел, но может применяться и табуляция. Кроме того, разделитель отмечает конец одного поля и начало следующего. Обычно ассемблер допускает использование любого числа пробелов в тех местах, где разрешен один пробел (так называемый свободный формат строки).

```
                ; Программа вводит 10 чисел, выбирает среди них
                ; наибольшее и выводит его на дисплей.
                ;
COUNT   EQU     10          ; Количество вводимых чисел
                ;
        MOV     CX,COUNT    ; Инициализировать цикл
        MOV     MAX,0       ; Начальное значение максимума
NEXT:   CALL    INDEC       ; Ввести число в регистр AX
        CMP     AX,MAX      ; Сравнить с максимумом
        JLE     LESS        ; Оно меньше или равно максимуму
        MOV     MAX,AX      ; Число больше текущего максимума
LESS:   LOOP    NEXT        ; Повторять до завершения
        MOV     AX,MAX      ; Передать максимум для вывода
        CALL    OUTDEC      ; Вывести максимальное число
        HLT                 ; Остановить микропроцессор
        ...
        ...
INDEC:                      ; Подпрограмма ввода
        ...
        ...
OUTDEC:                     ; Подпрограмма вывода
        ...
        ...
MAX     DW      0           ; Ячейка для максимума
        ...
```

Ограничителями называются специальные символы, которые могут отмечать конец лексем. Примерами ограничителей служат такие знаки пунктуации, как запятая, точка, двоеточие и др. В большинстве случаев для улучшения читабельности программы вместе с ограничителем можно использовать разделитель.

Рассмотрим типичный фрагмент ассемблерной программы для микропроцессора К1810ВМ86, который позволит разобраться в формате строк. Эта программа осуществляет последовательный ввод десяти чисел, выбирает из них наибольшее и выводит его, например, на дисплей. Ввод чисел производится подпрограммой INDEC, а вывод — подпрограммой OUTDEC.

Как видно, программа состоит, в основном, из символьных последовательностей, которые напоминают английские слова, разделенные знаками пунктуации. Такие последовательности называются *идентификаторами* или *символическими именами*. Образование идентификаторов обычно подчиняется следующим правилам:

первым символом должна быть буква;

остальными символами могут быть буквы и цифры, а также некоторые специальные символы, например знак подчеркивания (_);

ассемблер распознает первые 6, 8 или 31 символов (число зависит от конкретной реализации), а остальные игнорирует.

Поле метки. В поле метки находится символическое имя, которое определяет программист. Метка применяется для ассоциирования имени с некоторым численным значением, которое может представлять собой адрес переменной в памяти (строка с меткой MAX), константу (строка с меткой COUNT) или адрес команды (строки с метками NEXT и LESS). В последнем случае мы имеем дело с истинной меткой, которая используется в командах передачи управления и освобождает программиста от необходимости работать с численными адресами команд. Такие метки должны заканчиваться двоеточием.

Поле операции. В поле операции содержится мнемоника либо машинной команды, либо директивы ассемблера. Каждая машинная команда имеет уникальную мнемонику, передающую основную функцию команды, например, ADD — сложить, SUB (tract) — вычесть и т. д. Мнемоники считаются зарезервированными или ключевыми словами ассемблера и программистам не разрешается использовать их для других целей. Ассемблер заменяет мнемоники байтами машинных команд, привлекая для этого встроенную таблицу. Директивы ассемблера служат для распределения и инициализации памяти, определения символических имен, управления листингом ассемблирования и т. д.

Поле операнда. Большинство машинных команд и директив ассемблера требуют задания одного или нескольких операндов, разделяемых запятыми. Отдельные операнды могут быть представлены константами, переменными или специальными символическими именами, обозначающими, например, регистры микропроцессора. Кроме того, из них можно образовывать выражения. Ниже используются только простые арифметические выражения вида COUNT+5, ADDR—NUM и т. п. Поле операнда является наиболее сложным полем ассемблерных строк и синтаксис его зависит от особенностей реализации конкретного ассемблера.

Поле комментария. Это поле, начинающееся точкой с запятой, предназначено только для пояснений в программе и полностью игнорируется ассемблером. Комментарии должны отражать не общие функции команд (их описывают мнемоники), а действия команд в контексте конкретной программы. Комментарием может быть целая строка, начинающаяся точкой с запятой.

2.3. СИСТЕМА КОМАНД МИКРОПРОЦЕССОРА

Из 79 базовых команд МП К580 наибольший интерес представляют команды арифметических операций, показывающие его вычислительные возможности. Вместе с тем придется вкратце рассмотреть и остальные команды, так как для успешного программирования на языке Ассемблер необходимо знать всю систему команд. Далее применяются следующие условные обозначения:

r, r1, r2 — регистры микропроцессора A, B, C, D, E, H, L, M;

rp — регистровые пары B, D, H, SP, PSW;

data8 — байт непосредственных данных в команде (константа);

data16 — слово непосредственных данных в команде (константа);

addr — 16-битный адрес памяти в команде;

port — 8-битный номер (адрес) входного или выходного порта (константа);

src — источник (source); операнд, который не изменяется при выполнении команды;

dst — получать (destination); операнд, который изменяется при выполнении команды, так как он замещается результатом.

Форматы команд. В МП К580 имеются команды длиной один, два или три байта; их форматы показаны на рис. 2.6.

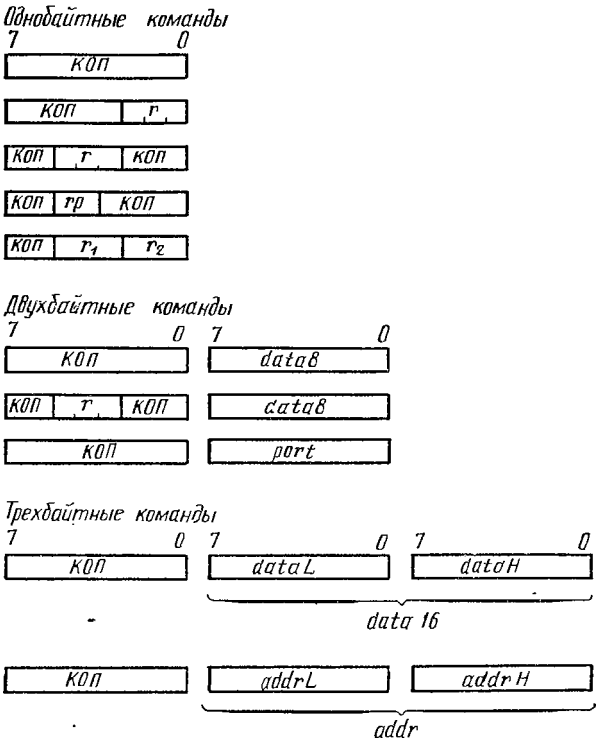


Рис. 2.6. Форматы команд микропроцессора КР580ИК80

Однобайтные команды отличаются наибольшим разнообразием. В простейшем случае байт команды состоит из одного кода операции, который неявно определяет операнд команды. В следующих двух форматах байт содержит поле (возможно, разделенное) кода операции и номер регистра *r*. Здесь неявным по-

лучателем (*dst*) в двухоперандных командах, например сложения, выступает аккумулятор А.

Такой же формат имеют унарные команды, выполняющие, например, инкремент или декремент регистра. В командах с косвенной адресацией памяти байт содержит поле *rp*, определяющее используемый указатель памяти. Наконец, команда MOV передачи данных между регистрами содержит два трехбитных поля r_1 и r_2, определяющих получатель *dst* и источник *src*. В языке Ассемблер МП-К580 первым указывается получатель, а вторым — источник; например, команда MOV B, E передает содержимое регистра E в регистр B. В некоторых других МП принят обратный порядок: первым указывается источник, а вторым — получатель.

В двухбайтных командах первый байт содержит код операции и, возможно, операнд-получатель *dst*, представленный регистром *r*. Вторым байтом является либо непосредственный операнд *data8*, либо адрес входного/выходного порта *port*.

В трехбайтных командах первый байт отведен для кода операции, а второй и третий байты содержат либо непосредственный операнд *data16*, либо абсолютный адрес памяти *addr*. В этих командах 16-битное слово размещено «наоборот» — сначала младший, а затем старший байт, что соответствует общему принципу «младшее — по меньшему адресу».

Команды арифметических операций. Команды, образующие группу команд арифметических операций, приведены в табл. 2.1.

Как видно из этой таблицы, вычислительные возможности МП К580 определяют команды сложения ADD, ADC, ADI, ACI и соответствующие им команды вычитания SUB, SBB, SUI, SBI. Во всех командах допускается двойная интерпретация 8-битных операндов: как беззнаковых целых и как знаковых целых чисел в дополнительном коде. Все восемь команд воздействуют на флажки, причем в командах вычитания флажок переноса превращается во флажок заема, т. е. он устанавливается в 1, если уменьшаемое меньше вычитаемого (операнды считаются беззнаковыми целыми). Команды ADC сложения с переносом и SBB вычитания с заемом удобны для операций с многобайтными числами: сначала складываются (вычитаются) их младшие байты (это можно сделать командой ADC или SBB, обеспечив нулевое состояние флажка переноса C), а затем — последующие байты с учетом межбайтных переносов (заемов).

В командах ADI, ACI, SUI, SBI с непосредственной адресацией операндом-источником служит второй байт команды, содержащий константу *data8*.

Команды инкремента INR и декремента DCR увеличивают (уменьшают) содержимое адресуемого регистра на 1. Отличительная особенность этих команд заключается в том, что они не влияют на состояние флажка переноса C, т. е. он сохраняет свое прежнее состояние.

Таблица 2.1. Команды арифметических операций

Название	Мнемоника	Функция	Примечание
Команды сложения			
Сложение	ADD *r*	A←(A)+(*r*)	Воздействует на все флажки
Сложение с переносом	ADC *r*	A←(A)+(*r*)+(C)	Воздействует на все флажки
Сложение с непосредственным операндом	ADI *data8*	A←(A)+*data8*	То же
Сложение с непосредственным операндом и переносом	ACI *data8*	A←(A)+*data8*+(C)	»
Инкремент регистра	INR *r*	*r*←(*r*)+1	Не модифицирует флажок переноса
Двойное сложение	DAD *rp*	HL←(HL)+(*rp*) *rp*=B, D, H, SP	Воздействует только на флажок переноса
Инкремент регистровой пары	INX *rp*	*rp*←(*rp*)+1	Не воздействует на флажки
Десятичная коррекция аккумулятора	DAA	Двоичное число в аккумуляторе преобразуется в упакованное десятичное. Воздействует на флажок переноса	
Команды вычитания			
Вычитание	SUB *r*	A←(A)—(*r*)	Воздействует на все флажки
Вычитание с заемом	SBB *r*	A←(A)—(*r*)—(C)	То же
Вычитание с непосредственным операндом	SUI *data8*	A←(A)—*data8*	»
Вычитание с непосредственным операндом и заемом	SBI *data8*	A←(A)—*data8*—(C)	»
Декремент регистра	DCR *r*	*r*←(*r*)—1	Не модифицирует флажок переноса
Декремент регистровой пары	DCX *rp*	*rp*←(*rp*)—1 *rp*=B, D, H, SP	Не воздействует на флажки
Команды сравнения			
Сравнение	CMP *r*	(A)—(*r*)	Воздействует на все флажки
Сравнение с непосредственным операндом	CPI *data8*	(A)—*data8*	То же

Команды арифметического сравнения CMP и CPI вычитают из содержимого аккумулятора значение адресуемого операнда, модифицируют по результату все флажки, но не изменяют содержимое аккумулятора. Такая операция называется неразрушающим сравнением.

МП К580 выполняет простейшие операции с 16-битными операндами, считающимися беззнаковыми целыми. Команды инкремента INX и декремента DCX регистровой пары, не влияющие на флажки, очень удобны для продвижения указателей регулярных структур данных. При выполнении команды двойного сложения DAD в соответствии с результатом модифицируется состояние флажка переноса C. Отметим, что команда DAD H удваивает содержимое H-пары.

В группу арифметических команд включена команда десятичной коррекции аккумулятора DAA. Она рассчитана на то, что в аккумуляторе находится двоичная сумма упакованных десятичных операндов, полученная любой из команд: ADD, ADC, ADI, ACI. Команда DAA анализирует содержимое тетрад аккумулятора и состояния флажков C и AC. Как итог этого анализа, в аккумуляторе формируется правильный десятичный результат, а флажок C отражает значение десятичного переноса. Команда DAA не корректирует результат двоичного вычитания упакованных десятичных операндов.

Команды передач данных. Обычно в самую многочисленную группу команд передач данных входят те команды, которые просто передают данные из источника *src* в получатель *dst*: *dst*←(*src*). На них в прикладных программах приходится до трети всех команд. Команды передач данных МП К580, показанные в табл. 2.2, не модифицируют состояний флажков (за единственным и естественным исключением команды POP PSW), т. е. они не проверяют

Таблица 2.2. **Команды передач данных**

Название	Мнемоника	Функция	Примечание
Межрегистровая передача	MOV r_1, r_2	r_1←(r_2)	Команда MOV M, M запрещена
Передача непосредственного операнда	MVI *r, data8*	*r*←*data8*	
Загрузка непосредственного 16-битного операнда	LXI *rp, data16*	*rp*←*data16*	*rp*=B, D, H, SP
Загрузка аккумулятора из памяти	LDA *addr*	A←(*addr*)	
Сохранение аккумулятора в памяти	STA *addr*	*addr*←(A)	
Косвенная загрузка аккумулятора из памяти	LDAX *rp*	A←((*rp*))	*rp*=B, D

Название	Мнемоника	Функция	Примечание
Косвенное сохранение аккумулятора в памяти	STAX *rp*	(*rp*)←(A)	*rp*=B, D
Загрузка H-пары из памяти	LHLD *addr*	L←(*addr*) H←(*addr*+1)	
Сохранение H-пары в памяти	SHLD *addr*	*addr*←(L) *addr*+1←(H)	
Обмен регистровых пар	XCHG	(HL)↔(DE)	
Передача из HL в SP	SPHL	SP←(HL)	
Включение в стек	PUSH *rp*	TOS←(*rp*)	*rp*=B, D, H, PSW
Извлечение из стека	POP *rp*	*rp*←(TOS)	*rp*=B, D, H, PSW
Обмен вершины стека и HL	XTHL	(HL)↔(TOS)	
Ввод из входного порта	IN *port*	A←((*port*))	
Вывод в выходной порт	OUT *port*	*port*←(A)	

особенностей передаваемых данных. Если все же передаваемый операнд необходимо проверить, приходится применять специальную команду.

Наиболее интенсивно в командах передач данных привлекаются аккумулятор A и регистровая пара HL. В частности, только аккумулятор фигурирует в командах загрузки и запоминания (сохранения) с абсолютной адресацией (LDA и STA) и с косвенной адресацией (LDAX и STAX), а H-пара участвует в командах LHLD, SHLD, XCHG, SPHL и XTHL. Кроме того, ввод и вывод осуществляются командами IN и OUT чеерз аккумулятор.

Целесообразно обратить особое внимание на команды передач между внутренними регистрами МП, так как они часто встречаются в программах. Пусть, например, потребовалось передать содержимое D-пары в указатель стека: SP←(DE). Специальная команда такой передачи отсутствует. Нетрудно убедиться, однако, что требуемую передачу осуществляет последовательность из трех команд: XCHG, SPHL, XCHG.

Команды логических операций. Команды этой группы являются поразрядными, т. е. выполняются независимо для отдельных битов операндов. Неадресумый операнд находится в аккумуляторе, куда загружается и результат операции. Флажки C и AC никогда не могут быть установлены в 1, поэтому они принудительно сбрасываются в 0. Команды логических операций МП К580 приведены в табл. 2.3.

Таблица 2.3. **Команды логических операций**

Название	Мнемоника	Функция
Логическое И	ANA *r*	A←(A)∧(*r*)
Логическое ИЛИ	ORA *r*	A←(A)∨(*r*)
Логическое исключающее ИЛИ (сложение по модулю 2)	XRA *r*	A←(A)⊕(*r*)
Логическое И с непосредственным операндом	ANI *data8*	A←(A)∧*data8*
Логическое ИЛИ с непосредственным операндом	ORI *data8*	A←(A)∨*data8*
Логическое исключающее ИЛИ с непосредственным операндом	XRI *data8*	A←(A)⊕*data8*
Инвентирование аккумулятора	CMA	A←$\bar{A}$

Команды логических операций применяются для проверки отдельных битов операнда, установки их в 1 и сброса в 0, инвертирования битов, объединения нескольких полей в одно и т. д. Например, команды ORA A и ANA A часто применяются только для сброса флажка переноса, так как специальной такой команды нет. Команда XRA A осуществляет сброс аккумулятора и флажка переноса.

В настоящую группу традиционно включаются команды сдвигов, принцип действия которых показан на рис. 2.7. Эти команды

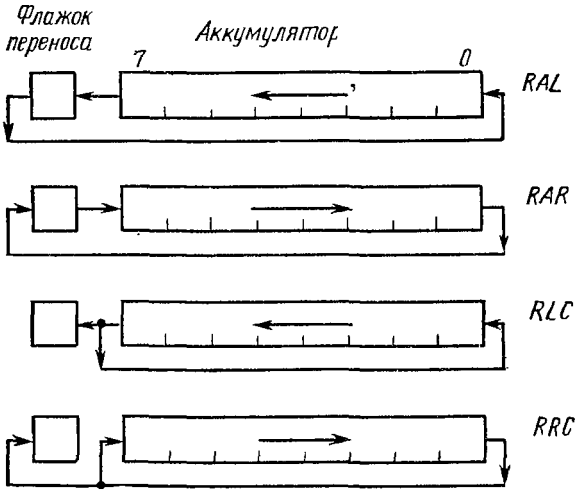


Рис. 2.7. Команды сдвигов микропроцессора КР580ИК80

сдвигают только содержимое аккумулятора и только на один бит влево или вправо. Все команды реализуют циклический сдвиг, причем первые две RAL и RAR — через флажок переноса. Выдвинутый из аккумулятора бит всегда находится во флажке переноса.

Команды передачи управления. Все команды передачи управления в МП К580 содержат во втором и третьем байтах адрес перехода, который загружается в программный счетчик. Исключение составляет только однобайтная команда возврата из подпрограммы, в которой адрес перехода автоматически берется из двух верхних ячеек стека, и команда PCHL, в которой адресом перехода служит содержимое Н-пары. Команды безусловной передачи управления МП К580 приведены в табл. 2.4.

Таблица 2.4. **Команды безусловной передачи управления**

Название	Мнемоника	Функция	Примечание
Безусловный переход	JMP *addr*	PC←*addr*	
Вызов подпрограммы	CALL *addr*	TOS←(PC) PC←*addr*	Адрес возврата сохраняется в стеке
Возврат из подпрограммы	RET	PC←(TOS)	
Переход по Н-паре	PCHL	PC←(HL)	
Рестарт	RST *n*	TOS←(PC) PC←0000000000*nnn*000	

Кроме приведенных в табл. 2.4 команд, имеются также команды условных переходов, вызовов подпрограммы и возвратов из подпрограммы (см. табл. 2.5). Каждая из них проверяет заданное

Таблица 2.5. **Команды условной передачи управления**

Команда	Проверяемый флажок							
	C		Z		S		P	
	Условные передачи управления							
	0	1	0	1	0	1	0	1
Переход	JNC	JC	JNZ	JZ	JP	JM	JPO	JPE
Вызов подпрограммы	CNC	CC	CNZ	CZ	CP	CM	CPO	CPE
Возврат из подпрограммы	RNC	RC	RN	RZ	RP	RM	RPO	RPE

условие, представленное состоянием конкретного флажка, и, если условие удовлетворяется, осуществляет передачу управления. Если же условие не удовлетворяется, передача управления не производится, а выполняется следующая по порядку команда.

Команды управления микропроцессором. Последнюю, самую немногочисленную, группу образуют команды управления микропроцессором, показанные в табл. 2.6. Команды EI и DI разреша-

Таблица 2.6. **Команды управления микропроцессором**

Название	Мнемоника	Функция
Разрешение прерываний	EI	IFF←1
Запрещение прерываний	DI	IFF←0
Пустая команда	NOP	Никаких действий
Останов	HLT	Состояние останова
Установка флажка переноса	STC	C←1
Дополнение (инвертирование) флажка переноса	CMC	$C \leftarrow (\overline{C})$

ют и запрещают восприятие внешних прерываний по входу INT посредством установки в 1 и сброса в 0 внутреннего триггера IFF разрешения прерываний. Пустая, или холостая, команда NOP не производит никаких действий, а команда останова HLT переводит МП в состояние останова, в котором он прекращает выполнение программы, ожидая сигнала прерывания. Две последние команды STC и CMC управляют состоянием флажка переноса.

Длина и время выполнения программы. При разработке прикладных программ возникает необходимость оценить их длину и время выполнения. Длина программы определяется просто — каждая из команд имеет фиксированную длину (один, два или три байта), следовательно, длина программы в байтах равна сумме длин составляющих ее команд.

Время выполнения программы определяется несложными расчетами. В отличие от средних и больших компьютеров в микропроцессорах время выполнения большинства команд фиксировано, т. е. не зависит от значений операндов. Оно измеряется в числе тактов (периодов) синхронизации и приводится в справочной литературе. Просуммировав такты выполнения отдельных команд, можно найти число тактов K, приходящееся как на отдельные фрагменты программы, так и на всю программу. Зная частоту синхронизации f, нетрудно получить и время T выполнения программы: $T=K/f$.

Необходимо учитывать, что в МП К580 для команд условных вызовов подпрограммы (CC, CNC, CZ и др.) и условных возвратов из подпрограммы (RC, RNC, RZ и др.) указываются два чис-

ла тактов: 11/17 и 5/11 соответственно. Одно из них (меньшее) получается, когда проверяемое в команде условие не удовлетворяется и передачи управления не происходит, т. е. включения в стек адреса возврата и извлечения из стека адреса возврата нет. Второе значение получается, когда проверяемое условие удовлетворяется и происходит передача управления.

Для примера расчета длины и времени выполнения рассмотрим элементарную подпрограмму, осуществляющую суммирование N-байтных двоичных целых беззнаковых чисел (она подробно рассмотрена в п. 2.4.1).

			Длина команды (байт)	Время выполнения (тактов)
ADDRND:	XRA	A	1	4
LOOP:	LDAX	D	1	7
	ADC	M	1	7
	STAX	D	1	7
	INX	H	1	5
	INX	D	1	5
	DCR	B	1	5
	JNZ	LOOP	3	10
	RET		1	10

Длина программы, равная сумме длин отдельных команд, составляет 11 байт.

Подпрограмма представляет собой простой цикл, число повторений которого N определяется содержимым регистра B, действующим как счетчик цикла. Каждая из команд цикла (от команды LDAX D до команды JNZ LOOP) выполняется N раз. Поэтому общее время выполнения подпрограммы в тактах синхронизации равно

$$K=4+(7+7+7+5+5+5+10)\times N+10=14+46\times N.$$

Если, например, N=8 и частота синхронизации $f=1$ МГц, время выполнения подпрограммы составит 382 мкс.

Время выполнения сложной программы, например оперирующей числами с плавающей точкой, зависит от значений исходных данных. Так, при выравнивании порядков требуется сдвигать мантиссу меньшего числа вправо и количество сдвигов варьируется от 0 (порядки равны) до $m-1$, где m — длина мантиссы. Для таких программ обычно рассчитываются минимальное и максимальное время выполнения.

2.4. АЛГОРИТМЫ И ПРОГРАММЫ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

Как показано в гл. 1, содержимое 8-битного регистра допускает несколько численных интерпретаций: двоичное целое беззна-

ковое число, двоичное целое знаковое число (в дополнительном коде), упакованное целое десятичное число, неупакованное целое десятичное число (цифра). Команды арифметических операций МП К580 обеспечивают сложение и вычитание однобайтных чисел в первых двух форматах. Установленный в 1 флажок переноса С показывает переполнение при сложении целых беззнаковых чисел или получение отрицательной разности (она представлена в дополнительном коде) при вычитании таких же чисел. Переполнение в случае целых знаковых чисел не фиксируется. Благодаря команде DAA можно складывать (но не вычитать!) упакованные десятичные числа с правильной («десятичной») установкой флажка переноса. Все остальные операции (т. е. умножение и деление), а также все операции с числами в других форматах необходимо программировать.

Разрешающей способности (точности) однобайтных чисел обычно недостаточно и приходится вводить многобайтные целые числа и числа с плавающей точкой. В случае 16-битных операндов говорят о *двойной точности*, а когда длина операндов превышает 16 бит, говорят о *многократной точности*. Многобайтные числа хранятся в смежных ячейках памяти, а операции над ними выполняются последовательно, отдельными байтами. При хранении многобайтных чисел в памяти действует принцип «младшее — по меньшему адресу» и адресом всего числа считается адрес его младшего байта. На рис. 2.8 показано размещение четырехбайтного

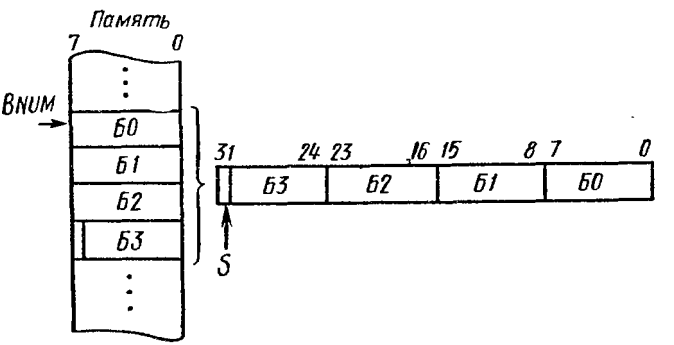


Рис. 2.8. Размещение четырехбайтного целого числа в памяти

целого знакового числа в памяти. Выборка его осуществляется побайтно путем инициализации указателя памяти на BNUM и инкремента указателя по мере обработки каждого байта. Переход от стандартного формата МП к более сложным форматам резко уменьшает его производительность.

Ниже рассматривается несколько алгоритмов и программ операций над многобайтными числами в различных форматах.

Сложение и вычитание. Операции сложения и вычитания очень похожи друг на друга, поэтому они рассматриваются вместе, тем более что флажок переноса C при вычитании автоматически становится флажком заема, позволяя правильно учитывать межбайтные связи.

Подпрограммы сложения и вычитания удобно разделить на две части. Одна из них заключается в выполнении необходимой операции над текущими байтами операндов, начиная с младшего байта. Выборка текущих байтов осуществляется с помощью двух указателей памяти. В качестве указателя памяти можно пользоваться любой из регистровых пар МП, но рекомендуется максимально привлекать регистр HL, затем регистр DE (из-за наличия команды XCHG) и в последнюю очередь регистр BC.

Вторая часть подпрограммы заключается в модификации указателей памяти для перехода к соседним старшим байтам операндов и в определении окончания операции после обработки всех байтов. Учет межбайтных переносов (заемов) легко осуществляется командами ADC и SBB. Практически во всех подпрограммах очень важную роль играет тот факт, что команды передач данных, инкремента и декремента регистров не воздействуют на флажок переноса (заема).

Таким образом, в подпрограмме требуются два указателя памяти и регистр-счетчик оставшихся для обработки операндов, а сама подпрограмма приобретает циклическую структуру. В начале подпрограммы следует произвести начальную установку (*инициализацию*) указателей памяти и счетчика, если этого не делается в вызывающей программе. Для сокращения длины листингов далее предполагается, что инициализацию выполняет вызывающая программа. В заголовках подпрограмм всегда указывается исходное размещение операндов и местонахождение результата при возврате.

Сложение (вычитание) многобайтных чисел начинается с младших байтов, затем суммируются (вычитаются) следующие байты с учетом состояния флажка переноса от предыдущего сложения (вычитания) и т. д. до завершения операции. В приводимой ниже подпрограмме 2.1 предполагается, что операнды находятся в областях памяти, начальные адреса которых передаются в регистрах HL и DE. Сумма замещает операнд, адресуемый регистром DE. Длина операндов в байтах содержится в регистре B, и для простоты нулевая длина операндов исключается.

Отметим для программы 2.1 следующие особенности.

1. Команда XRA A предназначена для сброса флажка переноса (при сложении младших байтов он должен содержать нуль).

2. Ни одна из команд цикла, за исключением команды ADC M, не модифицирует состояния флажка переноса. Следовательно, зна-

Программа 2.1. *Сложение целых беззнаковых чисел произвольной длины:*

```
        ; Начальные адреса операндов в регистрах DE и HL,
        ; длина в регистре B. Сумма замещает операнд,
        ; адресуемый DE.
        ;
ADDRND: XRA     A       ; Вначале перенос должен быть сброшен
LOOP:   LDAX    D       ; Текущий байт первого операнда
        ADC     M       ; Прибавить байт второго операнда
        STAX    D       ; Сохранить текущий байт суммы
        INX     H       ; Продвинуть указатели
        INX     D       ;    на следующие байты слагаемых
        DCR     B       ; Декремент счетчика байт
        JNZ     LOOP    ; Повторять до завершения
        RET             ; Возврат
```

чение переноса от сложения текущих байтов операндов сохраняется до сложения следующих байтов.

3. Перевыполнение после выхода из этой подпрограммы отмечается установленным в 1 флажком переноса.

4. Для превращения рассмотренной программы 2.1 в подпрограмму вычитания целых беззнаковых чисел необходимо заменить команду ADC на команду SBB. Если после возврата флажок переноса установлен в 1, то получена отрицательная разность, представленная в дополнительном коде.

5. Замечательная особенность дополнительного кода заключается в том, что если двоичные операнды представлены в дополнительном коде, то соответствующие подпрограммы применимы для сложения и вычитания целых знаковых чисел. Результат будет представлен также в дополнительном коде. Это утверждение справедливо только при отсутствии переполнения. Необходимость анализа возникновения переполнения несколько усложняет подпрограммы. Обычно для обнаружения переполнения применяется следующее правило: переполнение произошло, если знак суммы отличается от общего знака слагаемых, а знак разности не совпадает со знаком уменьшаемого в случае разных знаков операндов.

В программе 2.1 сумма (или разность) замещает один из операндов. Иногда требуется сложить (вычесть) два многобайтных числа без их разрушения, т. е. здесь приходится размещать результат в третьей области памяти. Для адресации текущих байтов двух операндов и результата потребуются три указателя памяти — регистры HL, DE и BC. Кроме того, аккумулятор необходим для собственно выполнения операции и все регистры МП оказываются занятыми. Для организации счетчика приходится «выходить» в память. Удобно привлечь для временного хранения счетчика стек и использовать специфическую команду XTHL, которая

передает счетчик в регистр H, а вторая команда XTHL возвращает счетчик в стек.

Программа 2.2. *Сложение многобайтных целых беззнаковых чисел с тремя указателями:*

```
                ; Адрес первого операнда в HL, адрес второго операнда
                ; в DE, адрес результата в BC. Число байт операндов
                ; в аккумуляторе A.
                ;
ADDM:   PUSH    PSW         ; Включить счетчик в стек
        XRA     A           ; Сбросить флажок переноса
LOOP:   LDAX    D           ; Сложить текущие байты
        ADC     M           ;    операндов и запомнить
        STAX    B           ;    результат
        INX     D           ; Продвинуть указатели
        INX     H
        INX     B
        XTHL                ; Счетчик в регистре H
        DCR     H           ; Декремент счетчика байт
        XTHL                ; Вернуть счетчик в стек
        JNZ     LOOP        ; Повторять до завершения
        POP     PSW         ; Очистить стек
        RET                 ; Возврат
```

Для этой программы справедливы все примечания к программе 2.1, за исключением того, что здесь флажок переноса сохраняет то состояние, в котором он находился при вызове подпрограммы. Заключительная команда POP PSW «очищает» стек.

Рассмотрим две простые подпрограммы 2.3 и 2.4, показывающие особенности работы с двоичными знаковыми числами. Предположим, что в регистрах HL и DE находятся два 16-битных числа в дополнительном коде. Необходимо образовать в регистре HL их сумму, причем флажки знака S и нуля Z должны показывать соответствующие признаки 16-битного результата и должно фиксироваться переполнение.

Собственно сложение операндов выполняет команда DAD D, а все остальные команды осуществляют проверку возникновения переполнения и правильную установку флажков S и Z. Переполнение фиксируется в том случае, если значение переноса при сложении и знак результата не совпадают.

В подпрограмме вычитания целых знаковых чисел с аналогичными начальными условиями (в регистре HL находится уменьшаемое, в регистре DE — вычитаемое, разность возвращается в регистр HL) необходимо изменить знак вычитаемого и вызвать подпрограмму сложения. При изменении знака вычитаемого возникает особый случай: минимальное отрицательное число 8000H (оно равно —32 768) не имеет равного по модулю положительного числа. При попытке образовать дополнительный код числа

Программа 2.3. *Сложение чисел в дополнительном коде с установкой флажков:*

```
        ; Слагаемые в регистрах HL и DE, сумма в регистре HL.
        ; Флажки S и Z показывают признаки 16-битного результата,
        ; а флажок переноса C сброшен в 0. При переполнении
        ; вызывается подпрограмма OVER.
        ;
ADDCOM: MOV     A,H         ; Проверить, одинаковы или нет
        XRA     D           ;    знаки операндов
        ANI     80H         ; Выделить знаковый бит
        DAD     D           ; Сложить операнды
        JNZ     DIFF        ; Знаки различны, переполнения нет
        RAR                 ; Перенос от сложения в старшем бите A
        XRA     H           ; Сравнить со знаком суммы
        RAL                 ; Признак переполнения
        CC      OVER        ;    во флажке переноса
        ; Установить флажки по результату
DIFF:   XRA     A           ; Сбросить аккумулятор
        ADD     H           ; Установить флажки по старшему байту
        RNZ                 ; Возврат, старший байт не равен 0
        ADD     L           ; Проверить младший байт
        RZ                  ; Результат равен нулю
        XRA     A           ; Отразить положительный
        INR     A           ;    ненулевой результат
        RET                 ; Возврат
```

8000H обычно фиксируется переполнение. Для упрощения подпрограммы исключается это единственное значение вычитаемого и считается его минимальное значение —32767.

Программа 2.4. *Вычитание чисел в дополнительном коде:*

```
        ; Уменьшаемое в HL, вычитаемое в DE, разность в HL.
        ; Флажки S и Z показывают признаки 16-битного результата,
        ; а флажок переноса C сброшен в 0. При переполнении
        ; вызывается подпрограмма OVER.
        ;
SUBCOM: PUSH    D           ; Сохранить вычитаемое в стеке
        MOV     A,D         ; Образовать в регистрах DE
        CMA                 ;    дополнительный код вычитаемого
        MOV     D,A         ;    (изменить знак вычитаемого)
        MOV     A,E
        CMA
        MOV     E,A
        INX     D
        CALL    ADDCOM      ; Теперь можно складывать
        POP     D           ; Вернуть вычитаемое в DE
        RET                 ; Возврат
```

Умножение. Умножение двоичных беззнаковых целых чисел в простейшем случае заключается в суммировании множимого с накоплением, которое производится количество раз, равное значению множителя. Этот способ реализуется элементарной программой 2.5.

Программа 2.5. *Умножение по способу накопления:*

```
        ; Множитель находится в аккумуляторе A, множимое
        ; в регистре E. Произведение возвращается в регистр  HL.
        ;
MULT:   MVI     D,0         ; Подготовить регистры
        LXI     H,0         ;    для умножения
MULT1:  DCR     A           ; Декремент множителя
        RM                  ; Возврат, если умножение закончено
        DAD     D           ; Прибавить множимое
        JMP     MULT1       ; Повторять до завершения
        RET                 ; Возврат
```

Основной недостаток этого способа, исключающий его практическое применение, заключается в невысоком быстродействии.

Имеются четыре способа умножения с анализом отдельных битов множителя с последующим накапливающим суммированием множимого и сдвигом множимого или суммы частичных произведений. Они различаются тем, как анализируются биты множителя (начиная со старших или младших битов) и что сдвигается (множимое или сумма частичных произведений). При желании нетрудно запрограммировать любой вариант умножения, но стремление уменьшить длину программы и (или) время выполнения операции заставляет максимально использовать при программировании особенности программной модели МП и возможности его системы команд.

В любом варианте умножения приходится учитывать, что произведение n-битных целых чисел имеет длину $2n$ бит и отбрасывать младшие биты нельзя. При этом переполнение в операции умножения невозможно, но приходится сдвигать числа двойной длины. В МП К580 сдвиг вправо и 8-битное сложение можно производить только в аккумуляторе. Однако команда DAD *rp* выполняет 16-битное сложение, а ее форма DAD H эквивалентна сдвигу содержимого регистра HL влево на один бит с передачей выдвигаемого старшего бита во флажок переноса. Поэтому наиболее эффективными оказываются алгоритмы умножения, в которых множимое или сумма частичных произведений сдвигается влево. Анализ последовательных битов множителя, как правило, осуществляется путем передачи их во флажок переноса. Из мно-

гочисленных известных программ умножения для МП К580 приведем наиболее оригинальные.

В первой из них (программа 2.6) умножение целых беззнаковых чисел осуществляется младшими разрядами вперед со сдвигом множимого влево. Множитель должен находиться в аккумуляторе А, множимое — в регистре Е, а произведение формируется в регистре HL.

Программа 2.6. *Умножение 8-битных беззнаковых целых чисел:*

```
        ; Множитель в аккумуляторе A, множимое в регистре E.
        ; Произведение возвращается в регистр  HL.
        ;
MUL88:  LXI     H,0         ; Подготовить место для произведения
        MVI     D,0         ;    и сдвига множимого влево
LOOP:   ORA     A           ; Умножение закончено?
        RZ                  ; Да, возврат
        RAR                 ; Бит множителя во флажке переноса
        JNC     NOADD       ; Он равен 0
        DAD     D           ; Он равен 1, прибавить множимое
NOADD:  XCHG                ; Множимое в регистре HL
        DAD     H           ; Сдвинуть его влево на один бит
        XCHG                ; Вернуть множимое в DE
        JMP     LOOP        ; Повторять до завершения
```

В этой подпрограмме используются два интересных приема. Во-первых, анализируемый бит множителя командой RAR передается во флажок переноса. При этом в освобождающийся старший бит аккумулятора «вдвигается» нуль из флажка переноса, сброшенного командой ORA A. Следовательно, об окончании умножения можно судить по нулевому содержимому аккумулятора А и счетчик битов здесь не нужен. Во-вторых, для сдвига множимого влево оно командой обмена XCHG передается в регистр HL и удваивается командой DAD H, а затем еще одной командой XCHG множимое возвращается в регистр DE. Если в подпрограмме MUL88 убрать команду LXI H, 0 (т. е. не сбрасывать регистр HL), то по окончании умножения в регистре HL будет образована сумма первоначального содержимого и произведения (E)×(A).

В программе 2.7 с помощью команды DAD *rp* в МП К580 можно очень короткой подпрограммой осуществить умножение 16-битного множимого на 8-битный множитель. Множимое размещается в регистре DE, а множитель — в аккумуляторе А; операнды считаются целыми беззнаковыми числами. Старшие 8 бит произведения образуются в аккумуляторе, а младшие 16 битов в регистре HL. Регистр В используется в качестве счетчика битов.

В отличие от программы 2.6 здесь умножение выполняется старшими разрядами вперед с «неподвижным» множимым. Сдвиг суммы частичных произведений влево реализуется командой DAD H. Выдвигаемый при этом старший бит через флажок переноса командой RAL передается в младший бит аккумулятора.

Программа 2.7. *Умножение 16-битного множимого на 8-битный множитель:*

```
            ; Множимое в регистре DE, множитель в аккумуляторе A,
            ; произведение образуется в регистрах A-HL.
            ;
DMULT:  LXI   H,0       ; Подготовить место для произведения
        MVI   B,8       ; Образовать счетчик бит
LOOP:   DAD   H         ; Сдвинуть произведение влево
        RAL             ; Сдвинуть множитель влево
        JNC   NOADD     ; Бит множителя равен нулю
        DAD   D         ; Прибавить множимое
        ACI   0         ; Прибавить перенос в младший бит A
NOADD:  DCR   B         ; Декремент счетчика бит
        JNZ   LOOP      ; Повторять до завершения
        RET             ; Возврат
```

Одновременно команда RAL сдвигает очередной бит множителя во флажок переноса. Возникающий при сложении множимого и суммы частичных произведений перенос прибавляется к старшим битам произведения с помощью команды ACI 0.

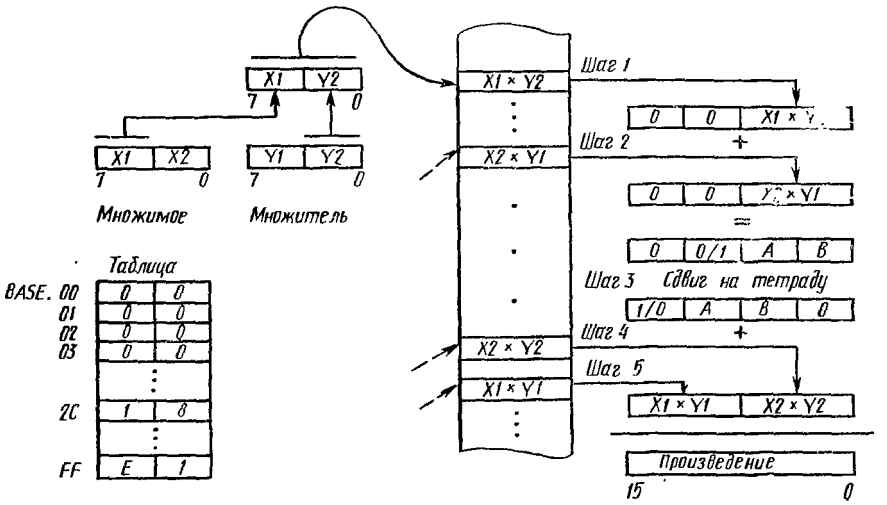


Рис. 2.9. Табличное умножение

Программа 2.8. *Табличное умножение:*

```
        ; Множимое в регистре D, множитель в регистре E,
        ; произведение возвращается в регистр  HL.
        ;
MULT:   LXI     H,0F00FH  ; Маски в регистре HL
        MOV     A,D       ; Передать множимое в аккумулятор
        ANA     L         ; Выделить X2
        MOV     C,A       ; Передать X2 в регистр C
        MOV     A,D       ; Передать множимое в аккумулятор
        ANA     H         ; Выделить X1
        MOV     B,A       ; Передать X1 в регистр B
        MOV     A,E       ; Передать множитель в аккумулятор
        ANA     H         ; Выделить Y1
        ORA     C         ; Образовать в регистре D
        MOV     D,A       ;    (Y1).(X2)
        MOV     A,E       ; Передать множитель в аккумулятор
        ANA     L         ; Выделить Y2
        ORA     B         ; Образовать в регистре L
        MOV     L,A       ;    (X1).(Y2)
        MOV     A,E       ; Передать множитель в аккумулятор,
        RRC               ;    произвести обмен тетрад
        RRC               ;    и сохранить вновь в регистре E
        RRC
        RRC
        MOV     E,A
        ANI     0FH       ; Выделить Y1
        ORA     B         ; Образовать в регистре B
        MOV     B,A       ;    (X1).(Y1)
        MOV     A,E       ; Передать множитель в аккумулятор
        ANA     H         ; Выделить  Y2
        ORA     C         ; Образовать в регистре E
        MOV     E,A       ;    (Y2).(X2)
        MVI     H,BASE    ; Базовый адрес таблицы
        ; Выполнить обращения к таблице
        MOV     C,M       ; Считать X1*Y2 в регистр C
        MOV     L,B       ; Передать (X1).(Y1) в регисто L
        MOV     B,M       ; Считать X1*Y1 в регистр B
        MOV     L,E       ; Передать (Y2).(X2) в регистр L
        MOV     E,M       ; Считать Y2*X2 в регистр E
        MOV     L,D       ; Передать (Y1).(X2) в регистр L
        MOV     L,M       ; Считать Y1*X2 в регистр
        ; Обращения к таблице закончены
        MOV     D,B       ; Передать X1*Y1 в регистр D
        SUB     A         ; Сбросить аккумулятор,
        MOV     H,A       ;    регистр H
```

```
MOV     L,M         ; Считать Y1*X2 в регистр
; Обращения к таблице закончены
MOV     D,B         ; Передать X1*Y1 в регистр D
SUB     A           ; Сбросить аккумулятор,
MOV     H,A         ;    регистр H
MOV     B,A         ;    и регистр B
DAD     B           ; Сложить X1*Y2 и X2*Y1
DAD     H           ; Сдвинуть регистр HL
DAD     H           ;    на одну тетраду влево
DAD     H
DAD     H
DAD     D           ; Образовать произведение
RET                 ; Возврат
```

Как пример, показывающий возможность ускорения операции умножения за счет увеличения объема используемой памяти, рассмотрим подпрограмму FMULT, опирающуюся на табличную реализацию умножения. Разделим 8-битные множимое X и множитель Y на две тетрады: $X=(X_1).(X_2)$, $Y=(Y_1).(Y_2)$. Тогда произведение равно

$$X\times Y=(X_1).(X_2)\times(Y_1).(Y_2)=X_1Y_1 2^8+X_1Y_2 2^4+X_2Y_1 2^4+X_2Y_2.$$

Если использовать каждую пару тетрад $(X_i).(Y_j)$ в качестве индекса таблицы, хранящей произведения тетрад, умножение двух тетрад реализуется с помощью одного обращения к таблице. Элемент таблицы с индексом $(X_i.Y_j)$ содержит 8-битное произведение $X_i\times Y_j$, и общий объем таблицы составляет 256 байт. В операции умножения необходимы четыре обращения к таблице и суммирование считываемых из нее данных с учетом их положения в полном произведении (рис. 2.9).

Перед вызовом подпрограммы FMULT множимое размещается в регистре D, множитель — в регистре E. Произведение образуется в регистре HL. Базовый адрес BASE таблицы имеет вид XX00H.

Эта программа примерно в три раза длиннее программы 2.6, но выполняется она в два раза быстрее.

Когда длина сомножителей составляет 16 бит, а произведения — 32 бит, регистров МП начинает не хватать. Рассмотрим подпрограмму такого умножения, показывающую, как можно «расширить» число регистров МП с привлечением стека. Предполагается, что множитель размещается в регистре DE, множимое — в регистре BC, а произведение образуется в регистрах DE—HL. Аккумулятор A выступает счетчиком битов. Формат сомножителей — целые беззнаковые числа.

В этой подпрограмме младшая часть произведения находится в регистре HL, а «регистром» для старшей части произведения служит вершина стека TOS. Перед входом в цикл умножения, который начинается с метки LOOP, оба регистра произведения очищаются.

Программа 2.9. *Умножение 16-битных сомножителей:*

```
        ; Множитель в регистре DE, множимое в регистре BC,
        ; произведение возвращается в регистрах DE - HL.
        ;
MUL16:  MVI     A,16        ; Образовать счетчик бит
        LXI     H,0         ; Инициализировать регистр HL
        PUSH    H           ;    и стек
LOOP:   XCHG                ; Множитель в HL, младшая часть
                            ;    произведения в DE
        DAD     H           ; Сдвинуть множитель влево
        XCHG                ; Вернуть множитель и произведение
        JNC     NOADD       ; Бит множителя равен нулю
        DAD     B           ; Прибавить множимое
        JNC     NOADD       ; Переноса не возникло
        XTHL                ; Старшая часть произведения в HL
        INX     H           ; Передать 1 в HL
        XTHL                ; Младшая часть произведения в HL
NOADD:  DCR     A           ; Декремент счетчика би
        JNZ     MORE        ; Продолжать умножение
        POP     D           ; Старшая часть произведения в DE
        RET                 ; Возврат
MORE:   DAD     H           ; Сдвинуть младшую часть произведения
        XTHL                ; Старшая часть произведения в HL
        PUSH    PSW         ; Сохранить флажок переноса в стеке
        DAD     H           ; Сдвинуть старшую часть произведения
        POP     PSW         ; Вернуть флажок переноса
        JNC     NOC         ; Единицы в старшую часть не было
        INX     H           ; Передать 1 из младшей части
NOC:    XTHL                ; Обменять части произведения
        JMP     LOOP        ; Повторять умножение
```

В начале цикла старший бит множителя сдвигается во флажок переноса и, если он равен единице, множимое прибавляется к содержимому регистра HL. Чтобы учесть возникающий при этом перенос в старшую половину произведения, командой XTHL она передается в регистр HL и производится его инкремент. Затем обе части произведения возвращаются на свои места. После проверки окончания цикла необходимо сдвигать влево обе части произведения; это действие осуществляется командами, находящимися после метки MORE. При сдвиге командой DAD H младшей части произведения ее старший бит попадает во флажок перено-

са. После передачи старшей части произведения в регистр HL (для этого вновь применяется команда XTHL) состояние флажка переноса приходится временно сохранять в стеке командой PUSH PSW, так как при сдвиге старшей части произведения командой DAD H флажок переноса будет сброшен. Затем запомненное состояние флажка переноса восстанавливается и передается в младший бит сдвинутой старшей части произведения. Командой XTHL обе части произведения возвращаются на свои места и осуществляется переход в начало цикла.

В программе 2.10 умножения целых беззнаковых чисел длиной 16 бит с получением 32-битного произведения применяется тот же способ умножения, что и в предыдущей подпрограмме. Однако вместо использования для старшей части произведения стека последовательно образуемые биты ее сдвигаются в освобождающиеся младшие биты множителя. Здесь умножение выполняется без «выхода» в память, что значительно ускоряет выполнение операции. Предполагается, что множимое находится в регистре BC, множитель — в регистре DE, а произведение образуется в регистрах DE—HL. Аккумулятор A по-прежнему действует как счетчик цикла. Благодаря команде обмена XCHG регистр HL попеременно используется для хранения множителя и суммы частичных произведений, а команда DAD H осуществляет сдвиг его содержимого влево.

Программа 2.10. *Умножение 16-битных чисел в регистрах:*

```
        ; Множимое в регистре BC, множитель в регистре DE,
        ; произведение возвращается в регистрах DE - HL.
        ;
MUL16:  LXI     H,0         ; Подготовить младшую часть произведения
        MVI     A,16        ; Образовать счетчик бит
LOOP:   XCHG                ; Множитель в HL, произведение в DE
        DAD     H           ; Сдвинуть множитель влево
MUL1:   XCHG                ; Множитель в DE, произведение в HL
        JNC     NOADD       ; Бит множителя равен нулю
        DAD     B           ; Прибавить множимое
        JNC     NOADD       ; Переноса в старшую часть нет
        INX     D           ; Передать 1 в младший бит множителя
NOADD:  DCR     A           ; Декремент счетчика бит
        RZ                  ; Умножение закончено
        DAD     H           ; Сдвинуть младшую часть произведения
        JNC     LOOP        ; Переноса нет
        XCHG                ; Множитель в HL, произведение в DE
        DAD     H           ; Сдвинуть множитель влево
        INX     H           ; Передать 1 в младший бит множителя
        JMP     MUL1        ; Повторять умножение
```

Как было показано выше, программирование операций сложения и вычитания целых чисел произвольной длины не вызывает принципиальных трудностей и реализуется простыми циклами. Умножение аналогичных чисел связано с определенными сложностями, а длина подпрограммы значительно увеличивается.

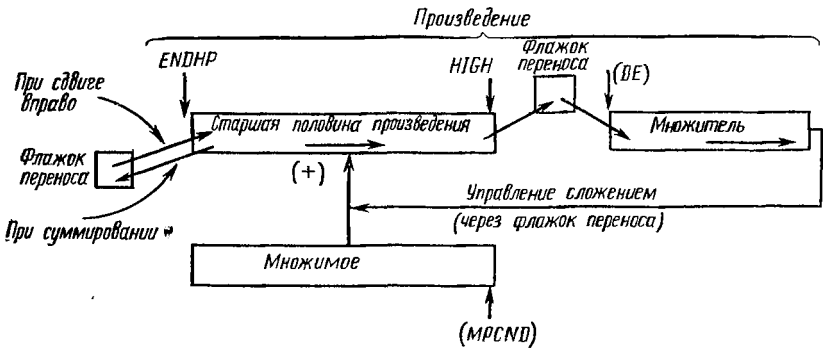


Рис. 2.10. Общая схема умножения чисел произвольной длины

Рассмотрим умножение двоичных целых беззнаковых чисел, имеющих произвольную длину (программа 2.11). Предположим, что начальный адрес (т. е. адрес младшего байта) множителя находится в регистре HL, начальный адрес множимого — в регистре DE, а длина сомножителей — в регистре B. Примем общую схему умножения младшими разрядами вперед со сдвигом суммы частичных произведений вправо. При этом выдвигаемый младший бит передается в старший бит множителя, который также сдвигается вправо (рис. 2.10). Для старшей половины произведения резервируется область памяти, начиная с адреса HIGH. Схема подпрограммы умножения показана на рис. 2.11.

Подпрограмма начинается с проверки длины сомножителей и сразу осуществляет возврат, если она равна нулю. Затем формируются конечные адреса множителя и старшей половины произведения, причем конечный адрес множителя помещается в регистр DE, а конечный адрес старшей половины произведения запоминается в слове ENDHP. Вычисление двух конечных адресов объясняется необходимостью сдвига вправо, а такой сдвиг начинается со старших байтов. Кроме того, начальный адрес множимого сохраняется в слове MPCND. После этого число байтов сомножителей путем умножения на 8 превращается в счетчик битов, который сохраняется в слове COUNT, так как для него регистров МП не хватает. В старшую половину произведения загружаются нули, подготавливая ее для накопления частичных произведений.

Начиная с метки LOOP, реализован цикл умножения на последовательные биты множителя. Первое действие в цикле заключается в сдвиге старшей половины произведения вправо на один бит. При входе в цикл флажок переноса сбрасывается в 0, а затем он будет показывать перенос, возникший при сложении суммы частичных произведений и множимого. Поэтому при сдвиге состояние флажка переноса передается в левый бит старшей половины произведения. Следующее действие заключается в сдвиге множителя вправо с передачей в его освобождающийся старший бит через флажок переноса выдвинутого младшего бита старшей половины произведения. Очередной бит множителя попадает во флажок переноса. Если этот бит содержит 1, происходит суммирование множимого со старшей половиной произведения, а если он содержит 0, сложение не производится. Заключительное действие цикла — декремент счетчика битов и выход из цикла, когда он исчерпывается до нуля. Таким образом, в этой подпрограмме имеется четыре цикла:

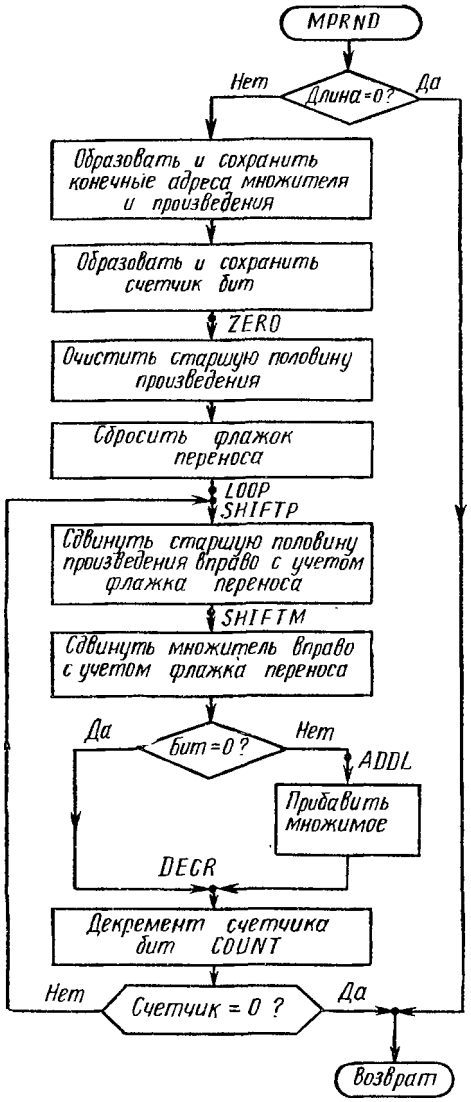


Рис. 2.11. Граф-схема алгоритма умножения чисел произвольной длины

1. «Глобальный» цикл (с метки LOOP) умножения на все биты множителя с числом повторений, равным числу битов в сомножителях. Этот цикл включает в себя все остальные циклы, а счетчик его находится в слове памяти COUNT.

Программа 2.11. *Умножение беззнаковых целых чисел произвольной длины:*

```
        ; Начальный адрес множителя в HL, начальный адрес множимого
        ; в DE, длина сомножителей в регистре B. Старшая половина
        ; произведения начинается с адреса HIGH, а младшая
        ; находится на месте множителя.
        ;
MPRND:  ; Проверить нулевую длину сомножителей.
        MOV     A,B         ; Передать длину в аккумулятор
        ORA     A
        RZ                  ; Возврат, если она равна нулю
        ; Образовать конечные адреса старшей половины
        ; произведения и множителей.
        MOV     C,B         ; Длина в регистрах B и C
        MVI     B,0
        DAD     B           ; Конечный адрес множителя в HL
        MVI     B,0
        DAD     B           ; Конечный адрес множителя в HL
        XCHG                ; В HL - начальный адрес множимого
        SHLD    MPCND       ; Сохранить его в слове MPCND
        LXI     H,HIGH      ; Начальный адрес старшего произведения
        DAD     B           ; Конечный адрес старшего произведения
        SHLD    ENDHP       ; Сохранить его в слове ENDHP
        ; Образовать счетчик бит.
        MOV     L,C         ; Длина сомножителей в HL
        MOV     H,B
        DAD     H           ; Умножить ее на 8
        DAD     H
        DAD     H
        SHLD    COUNT       ; Сохранить счетчик бит в COUNT
        ; Очистить старшую половину произведения
        MOV     B,C         ; Счетчик байт в регистре B
        LXI     H,HIGH      ; Начальный адрес старшего произведения
ZERO:   MVI     M,0         ; Очистить старшую
        INX     H           ;    половину произведения
        DCR     B
        JNZ     ZERO
        ; Подготовка к умножению закончена.
        ANA     A           ; Флажок переноса должен быть сброшен
LOOP:   ; Сдвинуть вправо старшую половину произведения
        MOV     B,C         ; Счетчик байт в регистре B
        LHLD    ENDHP       ; Сдвиг начинается со старшего байта
SHIFTP: DCX     H           ; Продвинуть указатель
        MOV     A,M         ; Произвести сдвиг байта
        RAR
        MOV     M,A
        DCR     B           ; Декремент счетчика байт
        JNZ     SHIFTP      ; Продолжать сдвиг
```

```
        ; Сдвинуть вправо множитель.
        MOV     L,E         ; В HL - конечный адрес множителя
        MOV     H,D
        MOV     B,C         ; Счетчик байт в регистре B
SHIFTM: DCX     H           ; Продвинуть указатель
        MOV     A,M         ; Произвести сдвиг байта
        RAR
        MOV     M,A
        DCR     B           ; Декремент счетчика байт
        JNZ     SHIFTM      ; Продолжать сдвиг
        ; Анализ очередного бита множителя во флажке переноса.
        JNC     DECR        ; Бит множителя равен нулю
        ; Прибавить множимое к произведению.
        PUSH    D           ; Сохранить адрес множителя
        LHLD    MPCND       ; Начальный адрес множимого
        XCHG                ;    в регистре DE
        LXI     H,HIGH      ; Адрес произведения в регистре HL
        MOV     B,C         ; Счетчик байт в регистре B
        ANA     A           ; Сбросить флажок переноса
ADDL:   LDAX    D           ; Очередной байт множимого
        ADC     M           ; Прибавить к произведению
        MOV     M,A
        INX     H           ; Продвинуть указатели
        INX     D
        DCR     B           ; Декремент счетчика байт
        JNZ     ADDL        ; Продолжать сложение
        POP     D           ; Восстановить адрес множителя
        ; Декремент и анализ счетчика бит.
DECR:   LDA     COUNT       ; Декремент младшего байта
        DCR     A           ;    счетчика бит
        STA     COUNT
        JNZ     LOOP        ; Повторять умножение
        PUSH    PSW         ; Сохранить флажок переноса
        LDA     COUNT+1     ; Проверить на нуль старший байт
        ANA     A           ;    счетчика бит
        JZ      EXIT        ; Он равен нулю, возврат
        DCR     A           ; Декремент старшего байта
        STA     COUNT+1     ;    счетчика бит
        POP     PSW         ; Восстановить флажок переноса
        JMP     LOOP        ; Повторять умножение
EXIT:   POP     PSW         ; Очистить стек
        RET                 ; Возврат
```

2. Цикл сдвига вправо на один бит старшей половины произведения с числом повторений, равным длине сомножителей в байтах. Этот цикл начинается с метки SHIFTP.

3. Цикл сдвига вправо на один бит множителя. Он также повторяется столько раз, чему равна длина сомножителей в байтах. Начало цикла идентифицирует метка SHIFTM.

4. Цикл суммирования множимого и старшей половины произведения. Он начинается с метки ADDL, а число повторений равно длине сомножителей в байтах.

Умножение целых знаковых чисел, представленных в дополнительном коде, с получением произведения также в дополнительном коде не вызывает серьезных трудностей. Традиционно умножение знаковых операндов выполняется путем определения знака произведения (он равен сумме по модулю 2 знаковых битов сомножителей) умножения абсолютных значений (для чего привлекается любая из рассмотренных выше подпрограмм умножения) и образования дополнительного кода произведения. Можно также воспользоваться следующим приемом. Анализируется знаковый бит множителя, и сомножители преобразуются так, чтобы множитель стал положительным: когда множитель отрицательный, сомножители умножаются на —1. После этого происходит умножение операндов как беззнаковых целых чисел. В результате получается произведение, сразу представленное в дополнительном коде, если множимое берется двойной длины с расширением знака в старшую половину. Известен также алгоритм Бута умножения чисел, представленных в дополнительном коде, с автоматическим получением произведения в дополнительном коде. Однако программная реализация алгоритма Бута, в котором необходимо анализировать по два соседних бита множителя, оказывается для МП К580 громоздкой.

Для умножения целых знаковых чисел X и Y, имеющих длину n бит и представленных в дополнительном коде, можно использовать следующий прием. Вычисляется $2n$-битное произведение P кодов X и Y, которые интерпретируются как беззнаковые целые числа; для такого умножения применима любая из рассмотренных выше подпрограмм. В зависимости от знаков сомножителей получение произведения в дополнительном коде может потребовать корректирующих действий.

Если оба сомножителя положительны ($X \geqslant 0$, $Y \geqslant 0$), то полученное значение P представляет собой истинное произведение и коррекция не нужна. В том случае, когда сомножители имеют разные знаки ($X \geqslant 0$, $Y < 0$ или $X < 0$, $Y \geqslant 0$), значение P равно

$$P = X(2^n - Y) = X2^n - XY,$$

или

$$P = (2^n - X)Y = Y2^n - XY.$$

Здесь коррекция заключается в том, чтобы вычесть из P положительный сомножитель, сдвинутый на n бит влево. Это эквивалентно прибавлению к старшей части произведения P дополнительного кода положительного сомножителя. Если оба сомножителя отрицательны, значение P равно

$$P=(2^n-X)(2^n-Y)=2^{2n}-X2^n-Y2^n+XY.$$

Для коррекции результата необходимо прибавить к P абсолютные значения сомножителей, сдвинутые на n бит влево. Это эквивалентно прибавлению к старшей части произведения дополнительных кодов исходных операндов.

Деление. Общая циклическая схема операции деления очень похожа на операцию умножения. Различие заключается в том, что умножение выполняется в цикле «сложение — сдвиг», а для деления потребуется цикл «вычитание — сдвиг». Так как частное можно получать только со старших разрядов, имеются два варианта деления: со сдвигом остатка влево и со сдвигом делителя вправо. Второй вариант на практике не применяется из-за необходимости иметь регистры остатка и делителя двойной длины. Операция деления характерна тем, что в ней всегда возможно переполнение, когда делитель равен нулю.

При делении целых чисел приходится получать частное и остаток также в виде целых чисел, причем обычно знак остатка совпадает со знаком делимого. После вычитания делителя из сдвинутого положительного остатка цифра частного равна инвертированному значению флажка переноса. Действительно, флажок переноса сбрасывается в 0, если текущий остаток больше делителя (бит частного равен 1), и устанавливается в 1, показывая заем, если текущий остаток меньше делителя (бит частного равен 0).

Если при вычитании получен очередной отрицательный остаток, приходится восстанавливать предыдущий положительный остаток путем прибавления делителя. Вариант деления без восстановления остатка, когда при получении отрицательного остатка он сдвигается влево, а затем суммируется с делителем, не применяется из-за громоздкости программы. Более распространен прием сохранения текущего положительного остатка в дополнительном буфере; при получении отрицательной разности восстановление остатка заключается в том, чтобы возвратить его из буфера.

Имеются две разновидности операции деления. В первой из них (более простой для программирования) делимое и делитель имеют одинаковую длину. Переполнение здесь может возникнуть только в том случае, когда делитель равен нулю. Во второй разновидности операции деления делимое имеет двойную длину и

переполнение возникает, если старшая половина делимого больше делителя.

Рассмотрим простую подпрограмму (2.12) деления 8-битных целых беззнаковых чисел со следующим распределением регистров МП: делимое находится в регистре E, делитель — в регистре D, частное образуется в регистре H и положительный остаток — в регистре C. Регистр L используется как счетчик битов. При нулевом делителе флажок нуля Z после возврата установлен в 1.

Программа 2.12. *Деление 8-битных целых беззнаковых чисел:*

```
        ; Делимое в регистре E, делитель в регистре D.
        ; Частное возвращается в регистре H, а положительный
        ; остаток в регистре C.
        ;
DIV88:  LXI   H,8       ; Образовать счетчик бит, сбросить частное
        MVI   C,0       ; Сбросить остаток
        MOV   A,D       ; Проверить нулевой делитель
        ORA   A
        RZ              ; Возврат,делитель равен нулю
LOOP:   MOV   A,E       ; Передать делимое в аккумулятор
        RAL             ; Сдвинуть его влево
        MOV   E,A       ; Вернуть делимое в регистр E
        MOV   A,C       ; Сдвинуть остаток влево
        RAL
        SUB   D         ; Вычесть делитель
        JNC   NOADD     ; Восстанавливать остаток не нужно
        ADD   D         ; Восстановить остаток
NOADD:  MOV   C,A       ; Вернуть остаток в регистр C
        CMC             ; Образовать бит частного
        MOV   A,H       ; Передать бит частного
        RAL             ;    в регистр H
        MOV   H,A
        DCR   L         ; Декремент счетчика бит
        JNZ   LOOP      ; Повторять деление
        INR   L         ; Сбросить флажок нуля
        RET             ; Возврат
```

В цикле деления, который начинается с метки LOOP, делимое сдвигается влево с передачей его старшего бита во флажок переноса. Затем остаток из регистра C передается в аккумулятор и бит делимого сдвигается из флажка переноса в его младший бит. Команда SUB D вычитает делитель из остатка. Если разность (т. е. очередной остаток) положительна, происходит переход на метку NOADD. Когда при вычитании возникает заем (очередной остаток отрицательный), команда ADD D восстанавливает пре-

дыдущий положительный остаток. После этого во флажке переноса образуется цифра частного и передается посредством сдвига влево в младший бит частного.

Подпрограмма (2.13) осуществляет деление 16-битных целых беззнаковых чисел. В ней сохранена такая же общая схема деления, как в предыдущей подпрограмме. Делимое находится в регистре HL, делитель — в регистре DE. Аккумулятор участвует в операциях сдвига влево и в вычитании делителя из остатка. Счетчик битов из-за нехватки регистров МП организован в ячейке памяти COUNT. Если делитель равен нулю, происходит выход из подпрограммы с установленным в 1 флажком нуля Z. В подпрограмме реализован алгоритм деления со сдвигом остатков влево.

Сначала в подпрограмме проверяется делитель и, если он равен нулю, происходит возврат с установленным в 1 флажком нуля Z. Затем делимое передается в регистр BC, в ячейке COUNT образуется счетчик битов и начинается цикл деления (метка LOOP). Делимое и остаток сдвигаются влево как 32-битные значения через аккумулятор, причем в освобождающиеся биты делимого передаются последовательно образуемые биты частного. Перед вычитанием делителя положительный остаток сохраняется в стеке и, если результат вычитания отрицательный, командой XTHL сохраненный остаток возвращается в регистр HL. Когда очередной остаток положительный, производится удаление предыдущего остатка из стека; оно осуществляется двумя командами INX SP без явного извлечения из стека. После декремента счетчика битов проверяется, необходимо ли продолжать цикл деления. При достижении счетчиком битов нуля остаток из регистра HL передается в регистр DE, а частное (со своей последней цифрой) передается в регистр HL.

Рассмотрим выполнение операции деления, когда операнды по-прежнему целые беззнаковые числа, а делимое имеет двойную длину. Используется следующее распределение регистров МП: делимое в регистрах HL—DE, делитель в регистре BC, частное образуется в регистре DE и остаток — в регистре HL. Признаком переполнения служит установленный в 1 флажок переноса.

Подпрограмма начинается с вычитания делителя из старшей части делимого для обнаружения переполнения. Если разность положительная (флажок переноса сброшен в 0), осуществляется возврат с установленным в 1 флажком переноса. При отсутствии переполнения командой XRA A аккумулятор сбрасывается, так как его старшие четыре бита используются как счетчик. Инкремент счетчика производится командой ADI 10H и при возникновении переноса цикл деления заканчивается: произведено 16 повторений цикла.

В программе 2.14 применяется способ деления со сдвигом остатков влево.

Программа 2.13. *Деление 16-битных целых беззнаковых чисел:*

```
        ; Делимое в регистре HL, делитель в регистре DE, частное
        ; возвращается в регистре HL, остаток в регистре DE.
        ; При переполнении флажок Z установлен в 1.
        ;
DIV16:  MOV     A,E         ; Проверить нулевой делитель
        ORA     A
        RZ                  ; Возврат, делитель равен нулю
        MOV     C,L         ; Передать делимое в регистр BC
        MOV     B,H
        LXI     H,0         ; Подготовить место для остатков
        MVI     A,16        ; Образовать счетчик бит
        ORA     A           ; Сбросить флажок переноса
        ; Подготовка к циклу деления закончена.
LOOP:   STA     COUNT       ; Сохранить счетчик бит в памяти
        MOV     A,C         ; Сдвинуть делимое влево
        RAL
        MOV     C,A
        MOV     A,B
        RAL
        MOV     B,A
        MOV     A,L         ; Сдвинуть остаток влево,
        RAL                 ;    связь с делимым через
        MOV     L,A         ;    флажок переноса
        MOV     A,H
        RAL
        MOV     H,A
        ; Вычитание делителя из остатка.
        PUSH    H           ; Сохранить текущий остаток
        MOV     A,L         ; Вычесть делитель
        SUB     E
        MOV     L,A
        MOV     A,H
        SBB     D
        MOV     H,A
        ; Инвертированная цифра частного во флажке переноса.
        CMC                 ; Образовать явную цифру частного
        JC      NOREC       ; Новый остаток положительный
        XTHL                ; Новый остаток отрицательный,
                            ;    вернуть предыдущий остаток из стека
NOREC:  INX     SP          ; Удалить остаток из стека
        INX     SP
        ; Проверить окончание деления.
        LDA     COUNT       ; Счетчик бит в аккумуляторе
        DCR     A           ; Декремент счетчика бит
        JNZ     LOOP        ; Повторять цикл деления
```

```
        ; Деление закончено, учесть последнюю цифру частного.
        XCHG            ; Остаток в регистре DE
        MOV     A,C     ; Передать частное в регистр HL,
        RAL             ;    учесть его последнюю цифру
        MOV     L,A
        MOV     A,B
        RAL
        MOV     H,A
        XRA     A       ; Сбросить
        INR     A       ;    флажок Z
        RET             ; Возврат
```

Программа 2.14. *Деление с 32-битным делимым и 16-битным делителем:*

```
        ; Делимое в регистрах HL - DE, делитель в регистре BC,
        ; частное возвращается в регистре DE, а остаток в регистре HL.
        ; При переполнении флажок переноса установлен в 1.
        ;
DIV32:  MOV     A,L     ; Пробное вычитание делителя
        SUB     C       ;    из старшей части делимого
        MOV     A,H     ;    для определения переполнения
        SBB     C
        CMC
        RC              ; Возврат, переполнение
        XRA     A       ; Подготовить счетчик бит
LOOP:   DAD     H       ; Сдвинуть остаток влево
        PUSH    PSW     ; Сохранить счетчик и перенос
        XCHG            ; Сдвинуть младшую часть делимого
        DAD     H
        XCHG
        JNC     L1      ; Из младшей части выдвинут 0
        INX     H       ; Из младшей части выдвинута 1
L1:     MOV     A,L     ; Вычесть делитель из остатка
        SUB     C
        MOV     L,A
        MOV     A,H
        SBB     B
        MOV     H,A
        JC      L2      ; Цифра частного равна 0
        POP     PSW     ; Восстановить счетчик и перенос
L3:     INX     D       ; Цифра частного равна 1
        JMP     L4      ; Перейти к проверке счетчика
L2:     POP     PSW     ; Восстановить счетчик и перенос
        JNC     L3      ; Цифра частного равна 1
        DAD     B       ; Восстановить остаток
L4:     ADI     10H     ; Инкремент счетчика бит
        JNC     LOOP    ; Повторить цикл деления
        ORA     A       ; Сбросить флажок переноса
```

Сначала выполняется сдвиг делимого влево, причем выдвигаемый старший бит, попадающий во флажок переноса, сохраняется в стеке. Затем делитель вычитается из старшей части делимого (остатка). Образующиеся единичные биты частно передаются в освобождающиеся младшие биты делимого командой INX D. При получении отрицательного остатка предыдущий положительный остаток восстанавливается командой DAD B, которая прибавляет делитель к остатку.

Деление чисел произвольной длины, как и умножение, вызывает определенные трудности, хотя общий принцип деления путем последовательных вычитаний сохраняется. Рассмотрим подпрограмму DIVRND деления целых беззнаковых чисел, в которой делимое и делитель имеют произвольную, но одинаковую, длину. Общая схема реализуемой ею операции показана на рис. 2.12, а граф-схема алгоритма — на рис. 2.13.

По сравнению с умножением в операции деления имеется особая ситуация, связанная с получением отрицательной разности при вычитании делителя. Чтобы вернуться к предыдущему поло-

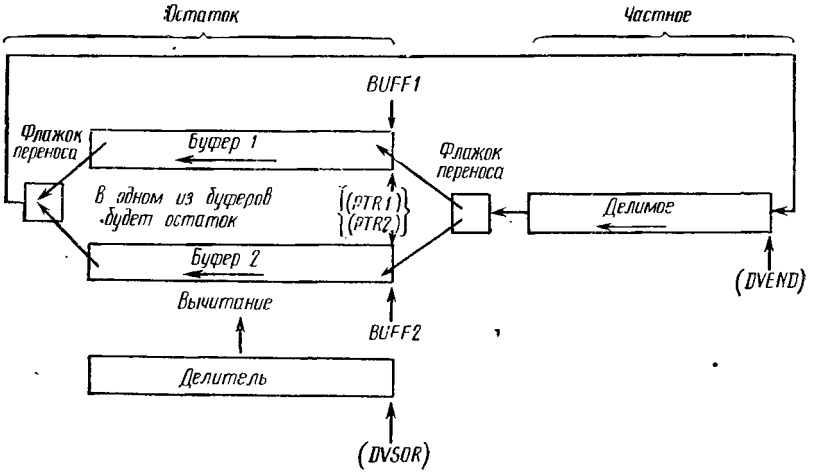


Рис. 2.12. Общая схема деления чисел произвольной длины

жительному остатку, приходится суммировать делитель и отрицательную разность. Можно избежать этой операции, если выделить для старшей части делимого две буферные области (или просто буфера) с начальными адресами BUFF1 и BUFF2; длина обоих буферов равна длине исходных операндов. Адреса буферов находятся в ячейках памяти PTR1 и PTR2. При вычитании делителя из старшей части делимого, находящейся в одном буфере (его называют текущим буфером), разность помещается в другой буфер. Если получена отрицательная разность (цифра частного рав-

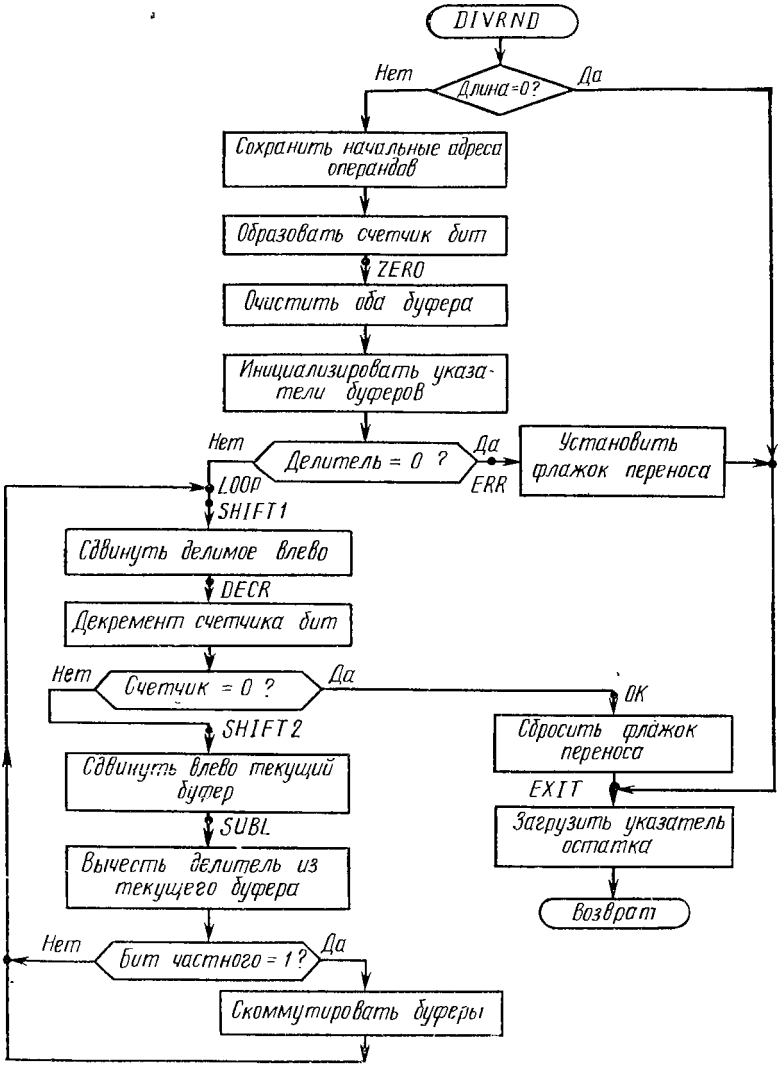


Рис. 2.13. Схема алгоритма деления чисел произвольной длины

на 0), в текущем буфере сохранилась предыдущая положительная старшая часть делимого и в следующем цикле деления необходимо использовать содержимое этого же буфера. Когда получена положительная разность (цифра частного равна 1), далее использовать именно эту разность, находящуюся в другом буфере.

Программа 2.15. *Деление беззнаковых целых чисел произвольной длины:*

```
        ; Начальный адрес делимого в HL, начальный адрес делителя в DE,
        ; длина операндов в регистре B. Частное возвращается на месте
        ; делимого, а HL адресует положительный остаток.
        ; Если делитель равен нулю, флажок переноса установлен в 1.
        ;
DIVRND: ; Проверить нулевую длину операндов и сохранить
        ; их начальные адреса.
        MOV     A,B         ; Передать длину в аккумулятор
        ORA     A           ; Возврат,
        JZ      OK          ;    если она равна нулю
        SHLD    DVEND       ; Сохранить адрес делимого
        XCHG                ; Сохранить
        SHLD    DVSOR       ;    адрес делителя
        MOV     C,B         ; Сохранить длину в регистре C
        ; Образовать счетчик бит и сохранить его в слове COUNT.
        MOV     L,C         ; В HL длина в байтах
        MVI     H,0
        DAD     H           ; Умножить ее на 8
        DAD     H           ;      для образования счетчика бит
        DAD     H
        INX     H
        SHLD    COUNT       ; Сохранить счетчик бит
        ; Очистить буферные области.
        LXI     H,BUFF1     ; Начальные адреса буферов
        LXI     D,BUFF2     ;    в HL и DE
        MOV     B,C         ; Длина в регистре B
        SUB     A           ; Сбросить аккумулятор
ZERO:   MOV     M,A         ; Передать нуль
        STAX    D           ;    в текущие байты буферов
        INX     H           ; Продвинуть указатели
        INX     D
        DCR     B           ; Декремент счетчика байт
        JNZ     ZERO        ; Повторять до завершения
        ; Инициализировать указатели буферов.
        LXI     H,BUFF1     ; Адрес первого текущего буфера
        SHLD    PTR1        ;    в слове PTR1
        LXI     H,BUFF2     ; Адрес второго буфера
        SHLD    PTR2        ;    в слове PTR2
        ; Проверить делитель на нуль.
        LHLD    DVSOR       ; В HL адрес делителя
        MOV     B,C         ; Счетчик бит в регистре B
        SUB     A           ; Сбросить аккумулятор
CHECK:  ORA     M           ; Объединить по ИЛИ очередной байт
        INX     H           ; Продвинуть указатель
        DCR     B           ; Декремент счетчика байт
        JNZ     CHECK       ; Повторять до завершения
```

```
        ORA     A           ; Проверить на нуль
        JZ      ERR         ; Ошибка, делитель равен нулю
        ; Подготовка к циклу деления закончена.
        ORA     A           ; Вначале  флажок переноса сброшен
LOOP:   LHLD    DVSOR       ; Начальный адрес
        XCHG                ;    делителя в DE
        LHLD    DVEND       ; Начальный адрес делимого в HL
        MOV     B,C         ; Счетчик байт в регистре B
        ; Сдвинуть делимое влево с учетом флажка переноса.
SHIFT1: MOV     A,M         ; Сдвинуть очередной байт
        RAL                 ;    делимого
        MOV     M,A         ;    влево
        INX     H           ; Продвинуть указатель
        DCR     B           ; Декремент счетчика байт
        JNZ     SHIFT1      ; Повторять до завершения
        ; Проверить достижение конца цикла деления.
DECR:   LDA     COUNT       ; Произвести декремент
        DCR     A           ;    счетчика бит (слова)
        STA     COUNT       ;    и перейти на метку OK,
        JNZ     CONT        ;    если деление закончено.
        LDA     COUNT+1     ;    Эти команды не изменяют
        DCR     A           ;    состояние флажка переноса.
        STA     COUNT+1
        JM      OK          ; Деление закончено
CONT:   ; Сдвинуть содержимое текущего буфера влево,
        ; передать флажок переноса в младший бит.
        LHLD    PTR1        ; В HL  начальный адрес текущего буфера
        MOV     B,C         ; Счетчик байт в регистре B
SHIFT2: MOV     A,M         ; Сдвинуть очередной байт
        RAL                 ;    текущего буфера влево
        MOV     M,A
        INX     H           ; Продвинуть указатель
        DCR     B           ; Декремент счетчика байт
        JNZ     SHIFT2      ; Повторять до завершения
        ; Произвести вычитание, помещая разность в другой буфер.
        PUSH    B           ; Сохранить длину, освободить BC
        MOV     A,C         ; Образовать счетчик байт
        STA     LENGTH      ;    в ячейке LENGTH
        LHLD    PTR2        ; В HL указатель другого буфера,
        MOV     C,L         ;     передать его в BC
        MOV     B,H
        LHLD    PTR1        ; В HL указатель текущего буфера,
        XCHG                ;    передать его в DE
        LHLD    DVSOR       ; В HL адрес делителя
        ORA     A           ; Сбросить флажок заема
```

```
SUBL:   LDAX    D           ; Произвести вычитание
        SBB     M           ;     и запомнить разность
        STAX    B
        INX     H           ; Продвинуть указатели
        INX     D
        INX     B
        LDA     LENGTH      ; Декремент счетчика байт
        DCR     A
        STA     LENGTH
        JNZ     SUBL        ; Повторять до завершения
        POP     B           ; Восстановить длину операндов
        ; Во флажке переноса инверсия бита частного.
        CMC                 ; Образовать явный бит частного
        JNC     LOOP        ; Разность отрицательна
        LHLD    PTR1        ; Разность положительна,
        XCHG                ;     необходимо скоммутировать
        LHLD    PTR2        ;     буферы (обменять
        SHLD    PTR1        ;     содержимое слов PTR1 и PTR2)
        XCHG
        SHLD    PTR2
        JMP     LOOP        ; Повторять цикл деления
ERR:    ; Ошибка - деление на нуль.
        STC                 ; Установить флажок переноса
        JMP     EXIT
OK:     ; Нормальный выход.
        ORA     A           ; Сбросить флажок переноса
EXIT:   LHLD    PTR1        ; В HL начальный адрес остатка
        RET                 ; Возврат
```

Следовательно, при получении положительной разности необходимо сделать текущим другой буфер, т. е. скоммутировать (переключить) буферы. Для этого достаточно обменять содержимое ячеек памяти PTR1 и PTR2, хранящих начальные адреса буферов.

В каждом цикле деления (он начинается с метки LOOP), число которых определяется счетчиком битов, производится сдвиг влево делимого, а также содержимого текущего буфера. При этом очередной бит делимого через флажок переноса попадает в младший бит текущего буфера. После этого выполняется вычитание делителя из содержимого текущего буфера и образованная во флажке переноса цифра частного (фактически она равна инверсии флажка переноса) передается в младший бит делимого. Таким образом, после выхода из цикла частное оказывается на месте делимого, а положительный остаток — в текущем буфере.

Предполагается, что при вызове подпрограммы регистр HL адресует делимое, регистр DE — делитель, а в регистре B находится число байтов операндов. Подпрограмма начинается с проверки длины операндов и сразу осуществляет возврат, если она равна нулю. Затем формируется и запоминается в слове COUNT счетчик битов, а обе буферные области очищаются. С помощью команд SHLD производится инициализация слов PTR1 и PTR2, служащих указателями буферов. Далее осуществляется проверка делителя на нуль и, если делитель равен нулю, подпрограмма возвращается с установленным в 1 флажком переноса (метка ERR).

Начиная с метки LOOP, реализован «глобальный» цикл деления с образованием последовательных битов частного. Первое действие в цикле заключается в сдвиге делимого на один бит влево (метка SHIFT1), причем выдвигаемый бит попадает во флажок переноса. После этого производится декремент счетчика битов и при достижении им нуля подпрограмма заканчивается. Если цикл деления не закончен, осуществляется сдвиг влево содержимого текущего буфера (метка SHIFT2), а затем из него вычитается делитель (метка SUBL). Как уже говорилось, формируемая разность помещается в другой буфер. В зависимости от полученной цифры частного (она определяется состоянием флажка переноса) производится или нет коммутация буферов и цикл деления повторяется.

Деление знаковых целых чисел обычно выполняется следующими действиями:

по знакам делимого и делителя определяется знак частного;

образуются абсолютные значения операндов, т. е. операнды превращаются в беззнаковые числа, для деления которых привлекается любая из рассмотренных выше подпрограмм;

с учетом знака частное представляется в дополнительном коде.

2.4.2. ОПЕРАЦИИ С ДЕСЯТИЧНЫМИ ЧИСЛАМИ

При решении некоторых задач приходится оперировать многоразрядными десятичными числами, которые, как правило, представлены в формате упакованных целых беззнаковых чисел. В МП К580 для десятичной арифметики предусмотрена единственная команда DAA десятичной коррекции аккумулятора. Она воздействует на находящуюся в аккумуляторе двоичную сумму двух байтов, содержащих упакованные десятичные числа, таким образом, что в аккумуляторе получается упакованное десятичное представление суммы и флажок переноса показывает правильный десятичный перенос. Благодаря этой команде легко реализуются операции сложения и вычитания, но в операциях умножения и деления появляются некоторые трудности.

Сложение и вычитание. Рассмотренную выше программу 2.1 сложения двоичных целых беззнаковых чисел произвольной длины очень легко превратить в программу сложения десятичных чисел: для этого достаточно после команды двоичного сложения ADC ввести команду DAA.

Программа 2.16. *Сложение упакованных десятичных целых беззнаковых чисел:*

```
        ; Начальные адреса операндов находятся в регистрах HL и DE,
        ; длина (в байтах) в регистре B.
        ; Сумма замещает операнд, адресуемый регистром DE.
        ;
ADDPCK: XRA     A           ; Сбросить флажки переноса
LOOP:   LDAX    D           ; Текущий байт первого операнда
        ADC     M           ; Прибавить байт второго операнда
        DAA                 ; Скорректировать сумму
        STAX    D           ; Сохранить текущий байт суммы
        INX     H           ; Продвинуть указатели
        INX     D
        DCR     B           ; Декремент счетчика байт
        JNZ     LOOP        ; Повторять до завершения
        RET                 ; Возврат
```

Установленный в 1 флажок переноса при возврате из подпрограммы сигнализирует о переполнении.

Вычитание упакованных десятичных чисел несколько усложняется тем обстоятельством, что команда DAA не корректирует результат двоичного вычитания. Поэтому операцию приходится выполнять в два этапа: сначала образуется дополнение вычитаемого до 10^{n+1} (т. е. десятичный дополнительный код), а затем полученное число суммируется с уменьшаемым. Результаты сложения можно корректировать командой DAA. В программе 2.17 приняты такие же начальные условия, как и в программе 2.1.

В этой подпрограмме оригинально используется команда обмена XCHG (обменивается содержимое регистров HL и DE), благодаря которой можно обращаться к памяти короткими командами ADD M и MOV M, A. Если не использовать такого приема, в подпрограмме появляются дополнительные команды межрегистровых передач.

Образование десятичного дополнительного кода вычитаемого и сложение его с уменьшаемым реализовано в одном цикле. Десятичный дополнительный код получается путем вычитания из девяток и инкремента результата, поэтому до входа в цикл флажок переноса устанавливается в 1 командой STC. Следовательно, только младший байт вычитаемого вычитается не из 99, а из 100, что

эквивалентно инкременту результата. Признаком получения отрицательной разности является сброшенный в 0 флажок переноса.

Умножение. При умножении упакованных десятичных чисел сохраняется общий принцип выполнения операции с двоичными числами: накапливающее суммирование множимого в зависимости от значений цифр множителя (см. рис. 2.10). Однако по сравнению с умножением двоичных чисел появляются некоторые особенности. Во-первых, основной единицей обработки операндов становится десятичная цифра, т. е. четыре бита. Например, сдвиги множимого и суммы частичных произведений необходимо осуществлять в цикле на четыре бита. Во-вторых, при умножении на каждую цифру множителя множимое прибавляется такое количество раз, равное значению цифры множителя (нулевые цифры, естественно, пропускаются). Каждое суммирование сопровождается командой десятичной коррекции DAA. Программа десятичного умножения содержит около 120 команд (в тексте не приводится).

Программа 2.17. *Вычитание упакованных десятичных целых беззнаковых чисел:*

```
        ; Адрес уменьшаемого в регистре DE, адрес вычитаемого
        ; в регистре HL, длина операндов (в байтах) в регистре B.
        ; Разность замещает уменьшаемое.
        ;
SUBPCK: STC             ; Для младшего байта флажок = 1
LOOP:   MVI   A,99H     ; Загрузить девятки
        ACI   0         ; Учесть флажок переноса
        SUB   M         ; Дополнение вычитаемого
        XCHG            ; Обменять указатели операндов
        ADD   M         ; Сложить с уменьшаемым
        DAA             ; Скорректировать как сумму
        MOV   M,A       ; Разность на месте уменьшаемого
        XCHG            ; Восстановить указатели
        INX   H         ; Продвинуть указатели
        INX   D
        DCR   B         ; Декремент счетчика байт
        JNZ   LOOP      ; Повторять до завершения
        RET             ; Возврат
```

Деление. В операции деления упакованных десятичных чисел также сохраняется общий принцип выполнения операции с двоичными числами: последовательные вычитания делителя из остатков (см. рис. 2.12). Но здесь, как и в операции умножения, появляются некоторые отличия. Основной единицей обработки операндов остается десятичная цифра, поэтому сдвиги остатка и частного производятся на четыре бита. При вычитании делителя из остатка фиксируется число вычитаний до получения

отрицательной разности; это число вычитаний и является очередной цифрой частного. Явное вычитание заменяется операцией сложения остатка с десятичным дополнительным кодом делителя. Длина программы деления десятичных чисел произвольной длины, как и программы умножения, составляет около 120 команд.

Знаковые упакованные десятичные числа представляются в десятичном дополнительном коде. В этом формате сложение и вычитание выполняются аналогично соответствующим операциям над беззнаковыми упакованными десятичными числами (см. программы 2.16 и 2.17). Результат представляется в десятичном дополнительном коде. Умножение и деление обычно выполняются над абсолютными значениями операндов, а знак результата определяется отдельным действием.

Программа 2.18. *Сложение неупакованных десятичных целых беззнаковых чисел:*

```
        ; Начальные адреса слагаемых находятся в регистрах HL и DE,
        ; длина в регистре B. Сумма замещает операнд, адресуемый
        ; регистром DE.
        ;
ADUNPK: XRA   A         ; Сбросить флажок переноса
        PUSH  PSW       ; Сохранить его в стеке
LOOP:   POP   PSW       ; Возвратить состояние переноса
        LDAX  D         ; Текущий байт первого операнда
        ADC   M         ; Прибавить байт второго операнда
        DAA             ; Скорректировать сумму
        ANI   1FH       ; Выделить нужные биты
        CPI   16        ; Передать межразрядный перенос
        CMC             ;    во флажок переноса
        PUSH  PSW       ; Сохранить перенос в стеке
        ORI   30H       ; Образовать код цифры
        LDAX  D         ; Сохранить текущий байт суммы
        INX   H         ; Продвинуть указатели
        INX   D
        DCR   B         ; Декремент счетчика байт
        JNZ   LOOP      ; Повторять до завершения
        POP   PSW       ; Вернуть значение переноса
        RET             ; Возврат
```

Операции с неупакованными десятичными числами на практике почти не встречаются. Рассмотрим все же выполнение сложения и вычитания беззнаковых целых чисел в этом формате. Напомним, что байт содержит 16-ричные коды от 30H до 39H, соответствующие десятичным цифрам от 0 до 9. При сложении таких чисел с произвольной длиной операция начинается с младших байтов и циклически продолжается в сторону старших

байтов с учетом межразрядных переносов. Команда ADC позволяет учесть межразрядный перенос, сформированный во флажке переноса. Однако фактический межразрядный перенос должен быть десятичным переносом из младшей тетрады суммируемых байтов. Поэтому в программе потребуются специальные команды, передающие межтетрадный перенос во флажок переноса. Если произвести двоичное сложение двух байтов, а затем скорректировать сумму командой DAA, то интересующий нас перенос будет зафиксирован в четвертом бите аккумулятора. Его состояние необходимо передать во флажок переноса; эту функцию выполняют команды ANI, CPI и CMC.

Программа 2.19. *Вычитание неупакованных десятичных целых беззнаковых чисел:*

```
          ; Адрес уменьшаемого находится в регистре DE, адрес
          ; вычитаемого в регистре HL, длина операндов - в регистре B.
          ; Разность замещает уменьшаемое.
          ;
SBUNPK:   STC             ; Для младшей цифры флажок = 1
          PUSH    PSW     ; Сохранить его в стеке
LOOP:     POP     PSW     ; Возвратить состояние переноса
          MVI     A,99H   ; Загрузить девятки
          ACI     0       ; Учесть флажок переноса
          SUB     M       ; Дополнение вычитаемого
          XCHG            ; Обменять указатели операндов
          ADD     M       ; Сложить с уменьшаемым
          DAA             ; Скорректировать как сумму
          ANI     1FH     ; Выделить нужные биты
          CPI     16      ; Определить состояние переноса
          PUSH    PSW     ; Сохранить перенос в стеке
          ORI     30H     ; Образовать код цифры
          MOV     M,A     ; Разность на месте уменьшаемого
          XCHG            ; Восстановить указатели
          INX     H       ; Продвинуть указатели
          INX     D
          DCR     B       ; Декремент счетчика байт
          JNZ     LOOP    ; Повторять до завершения
          POP     PSW     ; Вернуть значение переноса
          RET             ; Возврат
```

После команды десятичной коррекции DAA выделяются пять младших битов (команда ANI) и находящееся в них число сравнивается с 16 (команда CPI). Если оно больше или равно 16, флажок переноса сбрасывается в 0 (но десятичный перенос должен быть равен 1), а если число в аккумуляторе меньше 16, флажок переноса устанавливается в 1 (десятичный же перенос должен быть равным 0). Поэтому команда CMC инвертирования флажка переноса образует в нем правильное значение десятичного пере-

носа. Командой PUSH PSW оно сохраняется в стеке, так как следующая команда ORI 30H, образующая в аккумуляторе код цифры суммы, сбрасывает флажок переноса в 0. Следующие команды стандартным образом завершают цикл суммирования. При выходе из подпрограммы установленный в 1 флажок переноса означает переполнение.

Вычитание неупакованных десятичных чисел приводится к сложению путем образования десятичного дополнительного кода вычитаемого. Отметим, что здесь команда CMC не требуется.

2.4.3. ОПЕРАЦИИ НАД ЧИСЛАМИ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Для арифметических операций над числами с плавающей точкой в МП К580 разрабатываются довольно сложные подпрограммы. Если принять за исходный любой из форматов чисел с плавающей точкой, рассмотренных в гл. 1, то даже для 32-битного формата подпрограммы будут слишком громоздкими. Объясняется это катастрофической нехваткой внутренних регистров МП и «концентрацией» арифметических и логических операций, а также сдвигов в аккумуляторе А. Например, даже разместить два 32-битных числа в регистрах МП невозможно; операнды придется хранить в памяти и частями пересылать в МП для обработки. При этом длина подпрограмм (и время их выполнения) становится чрезмерно большой. Чтобы показать принципы выполнения операций над числами с плавающей точкой, принимается «урезанный» формат представления чисел. В этом формате сохранено кодирование порядка и мантиссы в соответствии со стандартом на одинарные числа с плавающей точкой, но мантисса имеет 15 значащих битов, а не 23. Напомним, что в старшем бите находится знак S числа, затем следует 8-битный смещенный порядок (смещение равно 127), а после него находится дробная часть мантиссы. Скрытый бит целой части мантиссы в нормализованных числах содержит 1. Истинный нуль кодируется нулевым набором, а разнообразные специальные числовые значения (бесконечность, неопределенность и не-числа) учитывать не будем. Несколько примеров кодирования чисел в этом формате:

$1.00111\times2^{3}=9\ \ 3/4$	01000001	00011100	00000000
$-1.0\times2^{9}=-512$	11000100	00000000	00000000
$1.0101\times2^{0}=1\ \ 5/16$	00111111	10101000	00000000
$-1.11\times2^{-3}=-7/32$	10111110	01100000	00000000

Примем стандартное размещение операндов во всех операциях: первый операнд X находится в регистрах EHL (старший байт в регистре E), а второй операнд Y — в регистрах DBC (старший байт в регистре D). Кроме того, результат операции возвращается на место операнда X, т. е. в регистр EHL.

Для того чтобы сократить длину листингов, воспользуемся несколькими короткими и простыми подпрограммами, которые выполняют функции, требующиеся в нескольких операциях.

1. Подпрограмма COMP образует дополнительный код числа, находящегося в регистре HL.

Программа 2.20. *Образование дополнительного кода числа в регистре* HL:

```
COMP:   MOV     A,H         ; Инвертировать код
        CMA                 ;    в регистре H
        MOV     H,A
        MOV     A,L         ; Инвертировать код
        CMA                 ;    в регистре L
        MOV     L,A
        INX     H           ; Образовать дополнительный код
        RET                 ; Возврат
```

Эта подпрограмма осуществляет нахождение дополнительного кода по способу «инвертирование и инкремент». Она не сигнализирует об особом случае, когда в HL содержится максимальное по модулю отрицательное число 8000H.

2. Подпрограмма NEG проверяет знак числа с плавающей точкой (он находится в старшем бите регистра E) и, если знак отрицательный, производится образование дополнительного кода мантиссы в регистре HL.

Программа 2.21. *Проверка знака и образование дополнительного кода:*

```
NEG:    MOV     A,E         ; Проверить знак
        ORA     E           ;    в старшем бите регистра E
        JP      NOTDK       ; Знак положительный
        CALL    COMP        ; Знак отрицательный, преобразовать (HL)
NOTDK:  RET
```

3. Подпрограмма SHIFT предназначена для сдвига вправо на один бит числа, находящегося в регистре HL. В освобождающийся левый бит помещается состояние флажка переноса.

Программа 2.22. *Сдвиг содержимого* HL *вправо на один бит:*

```
SHIFT:  MOV     A,H         ; Сдвинуть вправо (H),
        RAR                 ;    младший бит во флажке C
        MOV     H,A
        MOV     A,L         ; Сдвинуть вправо (L),
        RAR                 ;    связь через перенос
        MOV     L,A
        RET                 ; Возврат
```

МП К580 может сдвигать в аккумуляторе A только 8-битные коды. Поэтому сдвиг содержимого HL как единого 16-битного чис-

ла производится в два приема. Вначале команда RAR сдвигает вправо содержимое регистра H, причем в его старший бит помещается состояние флажка переноса, а выдвигаемый младший бит передается во флажок переноса. После этого второй командой RAR сдвигается содержимое регистра L и передаваемый через флажок переноса из регистра H бит попадает в старший бит регистра L.

4. Подпрограмма SWAP осуществляет обмен содержимого регистров EHL и DBC. Такая операция требуется в программах для того, чтобы разместить в регистрах EHL большее или меньшее из двух чисел. В подпрограмме SWAP оригинально используются стековые операции: сначала содержимое BC включается в стек (команда PUSH B), затем оно обменивается с содержимым HL (команда XTHL) и, наконец, содержимое вершины стека, т. е. находящееся там число из EHL, извлекается в регистр BC (команда POP B).

Программа 2.23. *Обмен содержимого регистров* EHL и DBC:

```
SWAP:   PUSH    B       ; Включить содержимое BC в стек
        XTHL            ; Обменять вершину стека с HL
        POP     B       ; Передать содержимое HL в BC
        MOV     A,D     ; Обменять содержимое
        MOV     D,E     ;    регистров D и E
        MOV     E,A
        RET             ; Возврат
```

Программа 2.24. *Восстановление числа с плавающей точкой:*

```
REC:    MOV     A,H     ; Старший байт мантиссы в регистре A
        ADD     A       ; Младший бит порядка во флажке переноса
        MOV     A,E     ; Образовать в регистре E
        RAL             ;    8-битный смещенный порядок
        MOV     E,A
        MOV     A,H     ; Восстановить скрытую единицу
        ORI     80H
        MOV     H,A
        RET             ; Возврат
```

5. Подпрограмма REC воспринимает в регистрах EHL число со скрытым битом мантиссы и возвращает в регистре E полный 8-битный смещенный порядок, а в регистре HL — мантиссу с явной старшей единицей.

6. Заключительная подпрограмма PACK выполняет действия, противоположные действиям предыдущей подпрограммы. Она воспринимает в регистрах EHL истинные порядок и мантиссу числа и в ячейке SIGN знак числа (в старшем бите). Как результат она возвращает в регистры EHL число в стандартном представлении. При этом флажок переноса сбрасывается в 0.

Программа 2.25. *Преобразование в стандартный формат:*

```
PACK:   LDA     SIGN        ; Передать знак в регистр A
        ADD     A           ; Знак во флажке переноса
        MOV     A,E         ; Полный порядок в регистре A
        MOV     D,A         ; Сохранить полный порядок в регистре D
        RAR                 ; Встроить бит знака
        MOV     E,A
        MOV     A,H         ; Подготовить бит
        ANI     7FH         ;    для младшего  бита порядка
        MOV     H,A
        MOV     A,D         ; Образовать в старшем бите A
        RRC                 ;    младший бит порядка
        ANI     80H         ; Выделить бит порядка
        ORA     H           ; Сформировать
        MOV     H,A         ;    второй байт числа
        RET                 ; Возврат
```

Имея в распоряжении такие удобные подпрограммы, можно разработать приемлемые по размеру программы всех арифметических операций над числами с плавающей точкой.

Сложение. Как было показано в гл. 1, сложение чисел с плавающей точкой состоит из нескольких этапов. Прежде всего выравниваются порядки, чтобы точка в обоих операндах находилась в одном и том же месте. При этом мантисса меньшего числа сдвигается вправо до тех пор, пока меньший порядок не будет равен большему. После этого производится сложение мантисс как чисел с фиксированной точкой, а за порядок результата принимается общий порядок (т. е. первоначально больших из двух порядков). Затем анализируется нарушение нормализации. При сложении чисел с одинаковыми знаками оно может быть только влево на один бит. Для устранения нарушения нормализации мантисса сдвигается вправо на один бит, а порядок увеличивается на единицу. Эта операция может вызвать переполнение, если порядок ненормализованного результата был максимальным.

В подпрограмме 2.26 число, находящееся в регистрах DBC, суммируется с числом в регистрах EHL и сумма сохраняется в этих же регистрах: EHL←(EHL)+(DBC). Исходное число в регистрах DBC разрушается.

Граф-схема алгоритма сложения, представленная на рис. 2.14, начинается с ряда проверок. Прежде всего проверяются знаки чисел и, если они различны, знак числа в регистрах DBC изменяется на противоположный с последующим переходом к подпрограмме вычитания SUBF. Далее проверяется, не является ли какой-либо операнд нулем. Если один из них нулевой, результат принимается равным второму операнду. Он при необходимости переда-

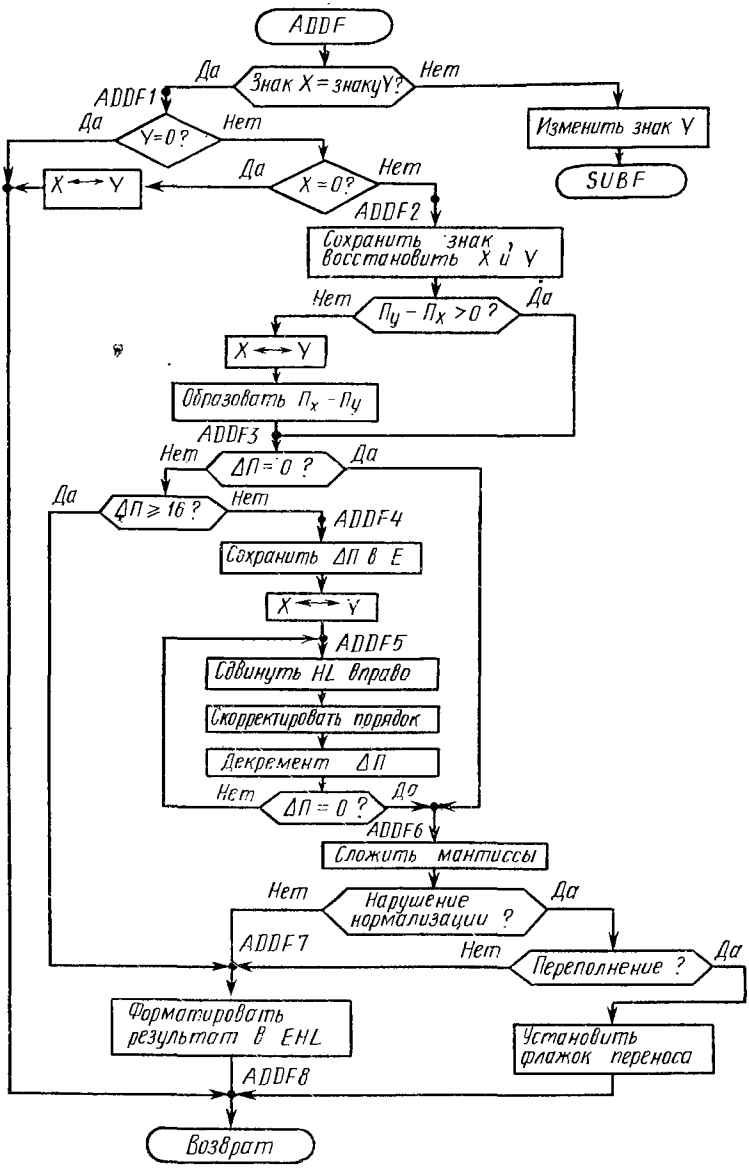


Рис. 2.14. Схема алгоритма сложения чисел с плавающей точкой

ется в регистры EHL и осуществляется возврат из подпрограммы. Следующая проверка выясняет равенство порядков. Если порядки операндов равны, происходит сложение мантисс и нормализация результата. Если же порядки не одинаковы, производится их выравнивание, а уже после этого сложение мантисс и нормализация результата. Переполнение фиксируется при возврате из подпрограммы установленным в 1 флажком переноса. Для удобства изучения подпрограммы использованные в ней метки показаны на рис. 2.14.

В подпрограмме ADDF отметим интересный прием. На метку ADDF4 она выходит со следующим содержимым регистров: в регистрах EHL находится бо́льшее число, в регистрах DBC — меньшее, в аккумуляторе A — положительная разность порядков $\Delta П < 16$. Для выравнивания порядков необходимо сдвигать вправо мантиссу меньшего числа. На первый взгляд, регистров МП не хватает, так как осуществить сдвиг вправо можно только в аккумуляторе, а он занят разностью порядков. Чтобы преодолеть возникшую трудность, разность порядков передается в регистр E, замещая больший порядок, и производится обмен чисел. В результате, меньшее число оказывается в регистрах EHL (напомним, что подпрограмма сдвига SHIFT осуществляет сдвиг мантиссы в регистре HL), а разность порядков — в регистре D. Начиная с метки ADDF5 реализован цикл сдвига мантиссы меньшего числа. Собственно сдвиг производится подпрограммой SHIFT, а счетчиком сдвигов служит регистр D, содержащий разность порядков. Одновременно с каждым сдвигом выполняется инкремент меньшего порядка в регистре E. Следовательно, при выходе из цикла по нулевому содержимому регистра D в регистре E оказывается меньший порядок, увеличенный на разность порядков, т. е. восстановленный больший порядок.

После этого суммируются мантиссы и по флажку переноса проверяется нарушение нормализации влево. Если флажок сброшен в 0, нарушения нормализации нет и подпрограмма переходит на метку ADDF7, где форматируется окончательный результат. При наличии нарушения нормализации производится инкремент порядка в регистре E и, если при этом в нем образуется нуль, т. е. порядок изменился с максимального 11111111 на нулевой, фиксируется переполнение. Подпрограмма переходит на метку ADDF8 с установленным в 1 флажком переноса. При переполнении результат в регистрах EHL не определен и не форматируется. Когда переполнения нет, мантисса сдвигается вправо, для чего вызывается подпрограмма SHIFT. При этом в старший бит регистра H не нужно записывать 1, так как она в процессе форматирования была бы скрыта, т. е. заменена на младший бит порядка.

Подпрограмма ADDF рассчитана только на сложение чисел с одинаковыми знаками. В случае операндов с разными знаками,

Программа 2.26. *Сложение чисел с плавающей точкой:*

```
        ; Первый операнд X находится в регистре EHL,
        ; второй Y - в регистрах DBC, сумма возвращается
        ; в регистрах EHL. При переполнении флажок переноса
        ; установлен в 1.
        ;
ADDF:   MOV     A,D         ; Сравнить знаки операндов
        XRA     E
        JP      ADDF1       ; Знаки операндов одинаковы
        MOV     A,D         ; Знаки различны,
        XRI     80H         ;    необходимо вычитать
        MOV     D,A
        JMP     SUBF
        ;
ADDF1:  MOV     A,D         ; Проверить на нуль
        ORA     B           ;    второй операнд
        ORA     C
        JZ      ADDF8       ; Результат находится в EHL
        MOV     A,E         ; Проверить на нуль
        ORA     H           ;    первый операнд
        ORA     L
        JNZ     ADDF2       ; Оба операнда ненулевые
        CALL    SWAP        ; Результат находится в EHL
        JMP     ADDF8
        ;
ADDF2:  MOV     A,D         ; Сохранить общий знак
        STA     SIGN        ;    операндов
        CALL    REC         ; Восстановить полные порядки
        CALL    SWAP        ;    и скрытые биты мантисс
        CALL    REC         ;    обоих операндов
        ;
        MOV     A,E         ; Сравнить порядки
        SUB     D           ;    с образованием разности порядков
        JNC     ADDF3       ; Число в EHL больше
        CALL    SWAP        ; Обменять числа
        MOV     A,E         ; Сравнить порядки
        SUB     D           ;    с образованием разности порядков
        ;
        ; В EHL большее число, в аккумуляторе разность порядков.
ADDF3:  JZ      ADDF6       ; Порядки одинаковы
        CPI     16          ; Сравнить разность поорядков с 16
        JC      ADDF4       ; Разность порядков меньше 16
        JMP     ADDF7       ; Результат равен большему числу
        ;
        ; Можно сдвигать мантиссу меньшего числа.
ADDF4:  MOV     E,A         ; Разность порядков в регистре E
```

```
        CALL    SWAP        ; Необходимо сдвигать меньшее число
ADDF5:  ORA     A           ; Сбросить флажок переноса
        CALL    SHIFT       ; Сдвинуть мантиссу меньшего числа
        INR     E           ; Увеличить меньший порядок
        DCR     D           ; Уменьшить разность порядков
        JNZ     ADDF5       ; Повторять сдвиг
        ;
        ; В регистре E общий порядок. Можно складывать мантиссы.
ADDF6:  DAD     B           ; Сложить мантиссы
        JNC     ADDF7       ; Нарушения нормализации нет
        INR     E           ; Скорректировать порядок
        JZ      ADDF8       ; Переполнение
        ORA     A           ; Сбросить флажок переноса
        CALL    SHIFT       ; Сдвинуть мантиссу вправо
ADDF7:  CALL    PACK        ; Форматировать результат
ADDF8:  RET                 ; Возврат
```

т. е. когда фактической операцией является вычитание, осуществляется переход на подпрограмму вычитания SUBF. Подпрограмма алгебраического сложения операндов оказалась бы несколько длиннее, в нее пришлось бы включить значительный фрагмент подпрограммы SUBF. Но при этом она практически без усложнений выполняла бы и вычитание чисел: нужно только изменить знак второго числа и произвести алгебраическое сложение.

Вычитание. Операция вычитания чисел похожа на операцию сложения, но мантисса меньшего числа после выравнивания порядков вычитается из мантиссы большего числа и соответственно корректируется знак результата. В подпрограмме 2.27 вычитания чисел число Y, находящееся в регистрах DBC, вычитается из числа X в регистрах EHL и разность возвращается в эти же регистры: EHL←(EHL)—(DBC).

Схема алгоритма вычитания, показанная на рис. 2.15, начинается с нескольких проверок. Сначала сравниваются знаки операндов и в случае разных знаков знак второго операнда Y изменяется на противоположный и происходит переход на подпрограмму сложения ADDF, так как фактической операцией здесь будет сложение. При одинаковых знаках операнды проверяются на нуль. Если один из операндов равен нулю, подпрограмма заканчивается с ненулевым операндом в регистрах EHL и при необходимости корректируется знак разности. Следующая проверка определяет бóльший (по модулю) операнд. При этом в случае одинаковых порядков приходится сравнивать старшие байты мантисс, а при их равенстве — еще и младшие байты мантисс. При абсолютном равенстве операндов получается нулевой результат.

После выявления бóльшего (по модулю) числа происходит вы-

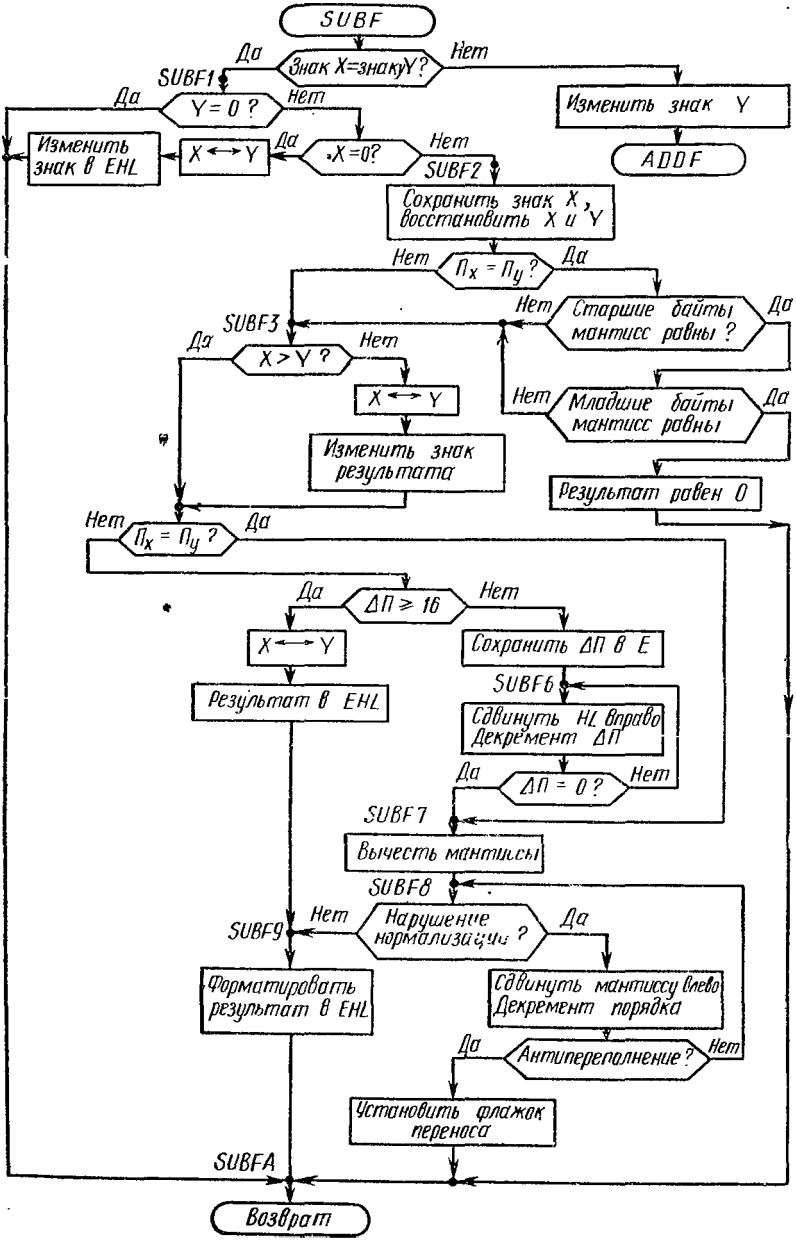


Рис. 2.15. Схема алгоритма вычитания чисел с плавающей точкой

равнивание порядков и вычитание мантисс. Результирующая разность размещается в регистрах EHL. При вычитании мантисс может возникнуть нарушение нормализации вправо, т. е. получение нулей в одном или нескольких старших битах мантиссы разности. Нормализация требует организации цикла, в котором мантисса сдвигается влево с декрементом порядка при каждом сдвиге. В ходе нормализации может возникнуть антипереполнение, о чем сигнализирует получение после декремента максимального порядка. Другими словами, антипереполнение возникает, когда порядок в процессе нормализации изменяется с минимального 0000000 на максимальный 1111111. В случае антипереполнения осуществляется установка в 1 флажка переноса и возврат из подпрограммы с неопределенным результатом.

Умножение. При умножении чисел с плавающей точкой необходимо сложить порядки и перемножить мантиссы. В этой операции могут возникнуть как переполнение, так и антипереполнение. В рассматриваемом формате чисел о возникновении любого особого случая можно судить по результату сложения порядков, так как при умножении мантисс нарушения нормализации вправо быть не может. Вместе с тем после умножения мантисс возможно нарушение нормализации влево максимум на один бит. Действительно, максимальное произведение мантисс равно

$$(2-2^{-n})\times(2-2^{-n})=4-2^{-n+2}+2^{-2n},$$

т. е. оказывается меньше 4.

В программе умножения 2.28 сомножители размещаются в регистрах EHL и DBC, а произведение возвращается в регистры EHL: EHL←(EHL)×(DBC).

При возникновении переполнения или антипереполнения флажок переноса устанавливается в 1 и произведение не определено.

Граф-схема алгоритма умножения приведена на рис. 2.16. Сначала оба операнда (по-прежнему обозначаем X число в регистрах EHL и Y число в регистрах DBC) проверяются на нуль. Если любой из них равен нулю, осуществляется возврат из подпрограммы с нулевым результатом. Затем путем сложения по модулю 2 знаковых битов операндов находится знак произведения и сохраняется в ячейке SIGN.

После этого анализируются особые случаи переполнения и антипереполнения. Для этого находится сумма смещенных порядков сомножителей. Если при сложении порядков возникает перенос, т. е. сумма больше 255, происходит переход на метку MULF2. В противном случае из суммы вычитается смещение 127, так как в сумме оно учтено дважды (в каждом из исходных порядков) и при отсутствии заема подпрограмма переходит на метку MULF3. Наличие заема при вычитании 127 свидетельствует о возникновении антипереполнения.

Программа 2.27. *Вычитание чисел с плавающей точкой:*

```
        ; Уменьшаемое X находится в регистрах EHL, вычитаемое Y
        ; в регистрах DBC, разность возвращается в регистрах EHL.
        ; При антипереполнении флажок переноса установлен в 1.
        ;
SUBF:   MOV     A,D         ; Сравнить знаки операндов
        XRA     E
        JP      SUBF1       ; Знаки операндов одинаковы
        MOV     A,D         ; Знаки различны
        XRI     80H         ;    необходимо складывать
        MOV     D,A
        JMP     ADDF
        ;
SUBF1:  MOV     A,D         ; Проверить на нуль
        ORA     B           ;    вычитаемое Y
        ORA     C
        JZ      SUBFA       ; Результат находится в EHL
        MOV     A,E         ; Проверить на нуль
        ORA     H           ;    уменьшаемое X
        ORA     L
        JNZ     SUBF2       ; Оба операнды ненулевые
        CALL    SWAP        ; Обменять числа
        MOV     A,E         ; Изменить знак результата
        XRI     80H
        MOV     E,A
        JMP     SUBFA       ; Результат находится  в EHL
        ;
SUBF2:  MOV     A,E         ; Сохранить знак
        STA     SIGN        ;    первого операнда X
        CALL    REC         ; Восстановить полные порядки
        CALL    SWAP        ;    и скрытые биты мантисс
        CALL    REC         ;    обоих операндов ( X в DBC, Y в EHL)
        ;
        MOV     A,D         ; Образовать разность
        SUB     E           ;    порядков ПХ - ПУ
        JNZ     SUBF3       ; Порядки не равны
        MOV     A,B         ; Сравнить
        CMP     H           ;    старшие байты мантисс
        JNZ     SUBF3       ; Они не равны
        MOV     A,C         ; Сравнить
        CMP     L           ;    младшие байты мантисс
        JNZ     SUBF3       ; Они не равны
        MVI     E,0         ; Числа равны,
        LXI     H,0         ;    результат равен нулю
        JMP     SUBFA
        ;
```

```
        ; Операнды не равны, необходимо вычитать.
SUBF3:  JNC     SUBF4       ; Первое число X больше, оно в DBC
        CALL    SWAP        ; Обменять числа, Y больше X, оно в DBC
        LDA     SIGN        ; Изменить знак результата
        XRI     80H
        STA     SIGN
        ;
        ; Необходимо вычитать EHL из DBC, результат в EHL.
SUBF4:  MOV     A,D         ; Образовать разность порядков,
        SUB     E           ;    она не отрицательна
        JZ      SUBF7       ; Порядки одинаковы
        CPI     16          ; Проверить диапазоны разности порядков
        JC      SUBF5       ; Необходимо вычитать операнды
        CALL    SWAP        ; Передать результат в EHL
        JMP     SUBF9       ; Форматировать результат
        ;
        ; В регистре A разность порядков, в DBC больший операнд.
SUBF5:  MOV     E,A         ; Сохранить разность порядков в регистре E
SUBF6:  ORA     A           ; Сбросить флажки переноса
        CALL    SHIFT       ; Выравнять порядки,
        DCR     E           ;    сдвигая мантиссу
        JNZ     SUBF6       ;    меньшего числа вправо
        ;
        ; Вычесть мантиссы, результат в регистрах EHL.
SUBF7:  MOV     A,C         ; Вычесть младшие байты мантисс
        SUB     L
        MOV     L,A
        MOV     A,B         ; Вычесть старшие байты мантисс
        BBB     H           ;    с учетом заема
        MOV     H,A
        MOV     E,D         ; Порядок результата в регистре E
        ;
        ; Нормализовать и проверить антипереполнение.
SUBF8:  MOV     A,H         ; Проверить старший
        ORA     H           ;    бит мантиссы
        JM      SUBF9       ; Результат нормализован
        DCR     E           ; Декремент порядка
        MOV     A,E         ; Проверить антипереполнение
        CPI     0FFH
        STC
        JZ      SUBFA       ; Возникло антипереполнение
        DAD     H           ; Сдвинуть мантиссу влево
        JMP     SUBF8       ; Повторять до завершения
        ;
        ; Результат в регистрах EHL.
SUBF9:  CALL    PACK        ; Форматировать разность
SUBFA:  RET                 ; Возврат
```

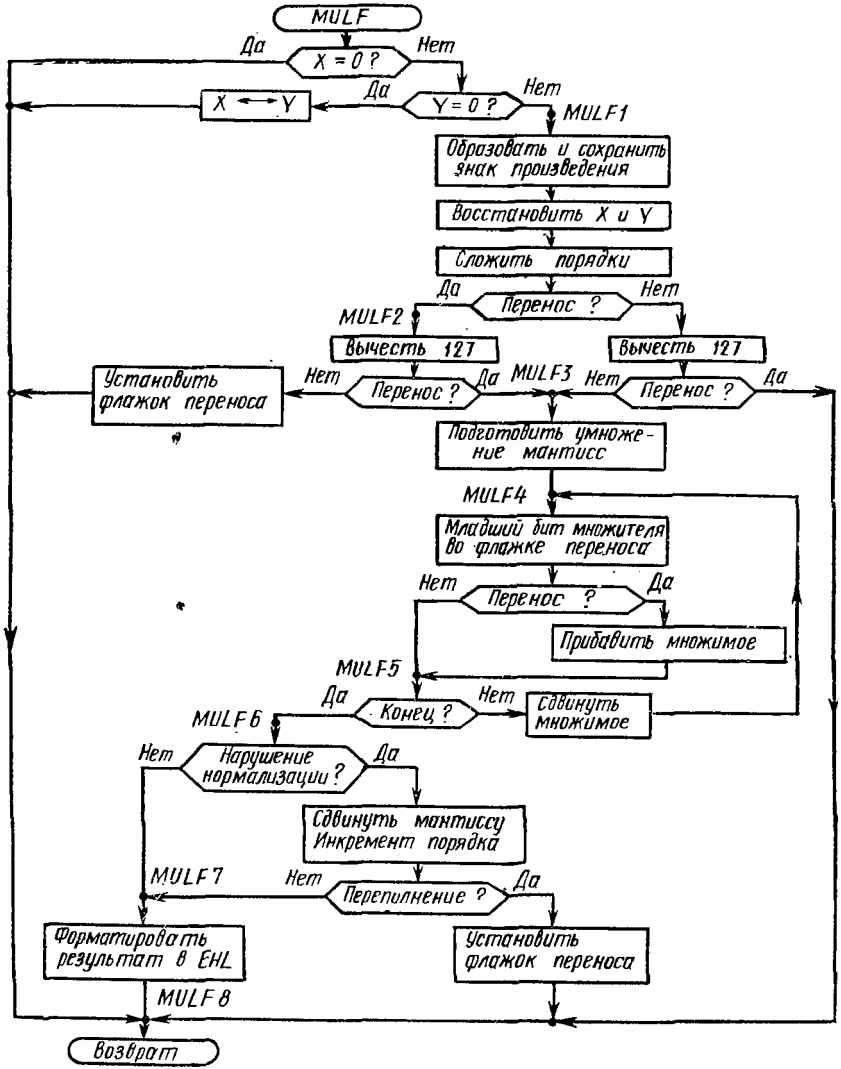


Рис. 2.16. Схема умножения чисел с плавающей точкой (первый вариант)

Когда сумма смещенных порядков больше 255 (метка MULF2), из нее также вычитается смещение 127. В этом случае отсутствие заема сигнализирует о переполнении, а наличие его — о возможности продолжать операцию умножения. Таким образом, на метку MULF3 подпрограмма выходит с образованным в аккумуляторе правильным смещенным порядком произведения.

Умножение 16-битных мантисс ставит определенные трудности, так как полное произведение имеет длину 32 бит и регистров МП не хватает. Рассмотрим два варианта операции умножения. В первом (упрощенном) варианте (программа 2.28) умножаются только старшие байты мантисс сомножителей с образованием в регистре HL 16-битного произведения. Здесь принят алгоритм умножения младшими разрядами вперед со сдвигом множимого влево. Чтобы учесть целые части сомножителей, множимое до собственно умножения сдвигается влево на один бит. После умножения мантисс проверяется нарушение нормализации влево и при его возникновении результат нормализуется. При этом возможно появление переполнения.

Во втором варианте умножения (программа 2.29), заключительная часть схемы которого представлена на рис. 2.17, реализовано умножение 16-битных операндов младшими разрядами вперед со сдвигом суммы частичных произведений вправо. При этом выдвигающиеся младшие биты произведения мантисс отбрасываются. Собственно умножение мантисс начинается с метки MULF3, на которую подпрограмма выходит со следующим состоянием регистров: в аккумуляторе A находится сумма порядков, а в регистрах HL и BC — мантиссы сомножителей. Здесь потребуется освободить регистр HL для накопления произведения, выделить регистр для счетчика цикла и производить сдвиг содержимого HL вправо. Сдвиг производится вызовом подпрограммы SHIFT, которая для своей операции привлекает аккумулятор. Следовательно, придется временно сохранить в памяти порядок произведения и старший байт мантиссы множителя, так как умножение начинается с младшего байта.

Деление. В операции деления чисел с плавающей точкой необходимо вычесть порядки и разделить мантиссы. Здесь могут возникнуть оба случая переполнения и антипереполнения. Об их появлении можно судить по результату вычитания порядков, так как при делении мантисс возможно только нарушение нормализации вправо максимум на один бит.

В программе деления 2.30 делимое X находится в регистрах EHL, делитель Y — в регистрах DBC и частное возвращается в регистрах EHL: EHL←(EHL)/(DBC).

При возникновении любого особого случая флажок переноса устанавливается в 1 и частное не определено.

Программа 2.28. *Умножение чисел с плавающей точкой (первый вариант):*

```
        ; Первый операнд X (множитель) находится в регистрах EHL,
        ; второй Y (множимое) в регистрах DBC, произведение
        ; возвращается в регистрах EHL.
        ; При возникновении особого случая флажок переноса = 1.
        ;
MULF:   MOV     A,E         ; Проверить множитель на нуль
        ORA     H
        ORA     L
        JZ      MULF8       ; Произведение равно нулю
        MOV     A,D         ; Проверить множимое на нуль
        ORA     B
        ORA     C
        JNZ     MULF1       ; Оба операнда ненулевые
        CALL    SWAP        ; Произведение
        JMP     MULF8       ;    равно нулю
        ;
        ; Операнды ненулевые, можно умножать.
MULF1:  MOV     A,D         ; Образовать и сохранить
        XRA     E           ;    знак произведения
        STA     SIGN
        CALL    REC         ; Восстановить полные порядки
        CALL    SWAP        ;    и скрытые биты мантисс
        CALL    REC         ;    X в DBC, Y в EHL
        MOV     A,D         ; Сложить
        ADD     E           ;    смещенные порядки
        JC      MULF2
        SUI     127         ; Вычесть смещение
        JNC     MULF3       ; Можно умножать мантиссы
        JMP     MULF8       ; Антипереполнение
MULF2:  ADI     129         ; Учесть потерю 256 из-за переноса
        JNC     MULF3       ; Можно умножать мантиссы
        JMP     MULF8       ; Переполнение
        ;
        ; В аккумуляторе A смещенный порядок произведения.
MULF3:  MOV     C,A         ; Сохранить порядок произведения
        MOV     E,B         ; Подготовить для умножения
        MVI     D,0         ;    множимое
        MOV     A,H         ; Старший байт множителя в A
        LXI     H,0         ; Место для накопления произведения
        XCHG                ; Учесть наличие целых частей
        DAD     H           ;    сдвигом множимого влево
        XCHG
        ;
        ; Здесь начинается цикл умножения.
MULF4:  ORA     A           ; Сбросить флажок переноса
        RAR                 ; Младший бит множителя
```

```
        JNC     MULF5       ;    во флажке переноса
        DAD     D           ; Прибавить множимое
MULF5:  JZ      MULF6       ; Умножение закончено
        XCHG                ; Сдвинуть множимое
        DAD     H           ;    влево на один бит
        XCHG
        JMP     MULF4       ; Повторять умножение
        ;
        ; Проверить нарушение нормализации.
MULF6:  JNC     MULF7       ; Нарушения нормализации нет
        CALL    SHIFT       ; Сдвинуть мантиссу вправо
        INR     C           ; Скорректировать порядок
        STC
        JZ      MULF8       ; Возникло переполнение
MULF7:  MOV     E,C         ; Передать порядок в регистр E
        CALL    PACK        ; Форматировать результат
MULF8:  RET                 ; Возврат
```

Граф-схема алгоритма деления показана на рис. 2.18. Как и прежде, вначале операнды анализируются на нуль. Если нулю равно делимое, образуется нулевое частное, а когда нулю равен делитель, фиксируется переполнение. Затем образуется и сохраняется в ячейке памяти SIGN знак частного и осуществляется восстановление операндов.

По результату вычитания исходных порядков и образования смещенного порядка частного определяется наличие особых случаев и, если их нет, подпрограмма переходит на метку DIVF2. Подготовка деления мантисс включает в себя следующие действия: порядок частного сохраняется в ячейке памяти EXP, мантисса делимого передается в регистр DE, в регистр HL загружается нуль, так как в HL будут последовательно формироваться биты частного, инициализируется на 16 и сохраняется в стеке счетчик цикла. В этой подпрограмме задействованы все регистры МП, поэтому для временного хранения счетчика, а также последнего положительного остатка привлекается стек.

Далее реализуются циклические действия по вычислению цифр частного, которые не требуют подробных пояснений. Последний положительный остаток всегда находится в стеке, поэтому когда при вычитании делителя из остатка получается очередной отрицательный остаток, восстановление остатка сводится к извлечению из стека положительного остатка. Цикл заканчивается при достижении счетчиком нуля. После этого реализуются стандартные действия по проверке нарушения нормализации и при необходимости ее устранения, а также формирование результата.

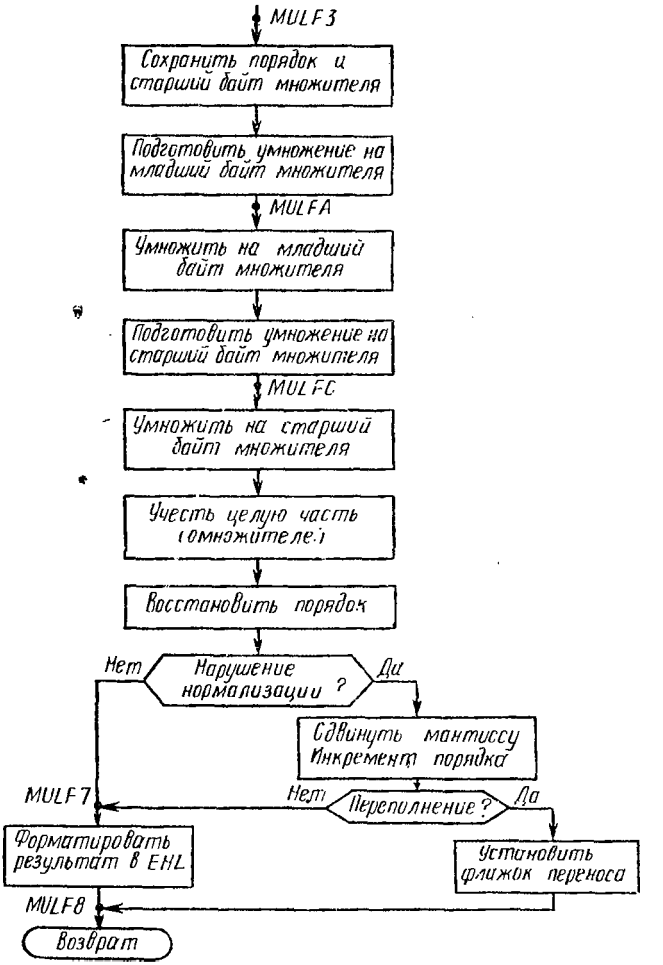


Рис. 2.17. Умножение чисел с плавающей точкой (второй вариант)

Программа 2.29. *Умножение чисел с плавающей точкой (второй вариант)*:

```
MULF3:  STA     EXP         ; Сохранить порядок произведения
        MOV     A,H         ; Сохранить старший байт
        STA     MPL         ;    мантиссы множителя
        MVI     E,8         ; Образовать счетчик цикла
        MOV     D,L         ; Младший байт мантиссы множителя
        LXI     H,0         ; Место для произведения
MULFA:  MOV     A,D         ; Проверить очередной
        RAR                 ;    младший бит множителя
        MOV     D,A         ; Возвратить множитель
        JNC     MULFB       ; Суммировать не нужно
        DAD     B           ; Прибавить множимое
MULFB:  CALL    SHIFT       ; Сдвинуть сумму частичных произведений
        DCR     E           ; Декремент счетчика
        JNZ     MULFA       ; Повторять умножение
        ;
        ; Умножить на старший байт мантиссы множителя.
        MVI     E,7         ; Счетчик цикла
        LDA     MPL         ; Передать старший байт
        MOV     D,A         ;    из памяти в регистр D
MULFC:  MOV     A,D         ; Повторение предыдущего
        RAR                 ;    фрагмента умножения
        MOV     D,A
        JNC     MULFD
        DAD     B
MULFD:  CALL    SHIFT
        DCR     E
        JNZ     MULFC
        DAD     B           ; Умножить на единицу целой части
        LDA     EXP         ; Передать порядок произведения
        MOV     E,A         ;    из памяти в регистр E
        JNC     MULF7       ; Нарушения нормализации нет
        CALL    SHIFT       ; Сдвинуть мантиссу вправо
        MOV     A,H         ; Учесть единицу
        ORI     80H         ;    из флажка переноса
        MOV     H,A
        INR     E           ; Скорректировать порядок
        STC                 ; Установить флажок переполнения
        JZ      MULF8       ; Возникло переполнение
MULF7:  CALL    PACK        ; Форматировать результат
MULF8   RET                 ; Возврат
```

Программа 2.30. *Деление чисел с плавающей точкой:*

```
        ; Первый операнд X (делимое) находится в регистрах EHL,
        ; второй Y (делитель) в регистрах DEC, частное возвращается
        ; в регистрах EHL.
        ; При возникновении особого случая флажок переноса = 1.
        ;
DIVF:   MOV     A,E         ; Проверить на нуль делимое
        ORA     H
        ORA     L
        JZ      DIVF7       ; Нулевой результат в EHL
        MOV     A,D         ; Проверить на нуль делитель
        ORA     B
        ORA     C
        STC                 ; Установить флажок переноса
        JZ      DIVF7       ; Возникло переполнение
        ;
        ; Операнды не равны нулю.
        MOV     A,D         ; Образовать и сохранить
        XRA     E           ;    знак частного
        STA     SIGN
        CALL    REC         ; Восстановить делимое и делитель,
        CALL    SWAP        ;    оставить их на места
        CALL    REC
        CALL    SWAP
        MOV     A,E         ; Образовать разность порядков
        SUB     D
        JNC     DIVF1       ; Порядок делимого больше
        ADI     127         ; Прибавить смещение
        CMC                 ; Если нет переноса
        JC      DIVF7       ;    возникло антипереполнение
        JMP     DIVF2       ; Перейти на деление мантисс
DIVF1:  ADI     127         ; Прибавить смещение
        JC      DIVF7       ; Возникло переполнение
        ;
        ; Можно начать деление мантисс.
DIVF2:  STA     EXP         ; Сохранить порядок
        XCHG                ; Мантисса делимого в регистре DE
        LXI     H,0         ; Подготовить место для частного
        MVI     A,16        ; Инициализировать счетчик
        PUSH    PSW         ; Сохранить счетчик в стеке
        JMP     DIVF4       ; Войти в цикл деления
DIVF3:  PUSH    PSW         ; Сохранить счетчик
        DAD     H           ; Сдвинуть влево
        XCHG                ;    частное и остаток
        DAD     H
        XCHG
DIVF4:  PUSH    D           ; Сохранить остаток в стеке
```

```
        MOV     A,E         ; Вычесть делитель
        SUB     C           ;     из остатка
        MOV     E,A
        MOV     A,D
        SBB     B
        MOV     D,A
        JC      DIVF5
        POP     PSW         ; Удалить остаток из стека
        INR     L           ; Цифра частного равна 1
        PUSH    D           ; Включить новый положительный остаток
DIVF5:  POP     D           ; Извлечь предыдущий остаток
        POP     PSW         ; Извлечь счетчик
        DCR     A           ; Декремент счетчика
        JNZ     DIVF3       ; Повторить цикл деления
        ;
        ; Деление мантисс закончено.
        LDA     EXP         ; Возвратить порядок частного
        MOV     E,A         ;     в регистр E
        ;
        ; Нормализовать частное.
        MOV     A,H         ; Проверить
        ORA     A           ;     старший бит мантиссы
        JM      DIVF6       ; Нарушения нормализации нет
        DAD     H           ; Сдвинуть мантиссу влево
        DCR     E           ; Декремент порядка
        CPI     0FFH        ; Проверить антипереполнение
        STC
        JZ      DIVF7       ; Возникло антипереполнение
DIVF6:  CALL    PACK        ; Форматировать результат
DIVF7:  RET                 ; Возврат
```

2.4.4. ВСПОМОГАТЕЛЬНЫЕ ПРОГРАММЫ

В этом параграфе рассматривается несколько простых программ, предназначенных для преобразований форматов числовых данных.

Преобразование 8-битного двоичного целого беззнакового числа в упакованное десятичное (программа 2.31). Предположим, что в аккумуляторе A находится байт, интерпретируемый как двоичное целое беззнаковое число (диапазон от 0 до 255), и необходимо образовать в регистре HL его представление как упакованного десятичного числа. Простой способ преобразования заключается в том, чтобы сначала определить цифру сотен, вычитая 100 из исходного числа. Затем находится цифра десятков последовательным вычитанием 10. После этого в аккумуляторе останется цифра единиц. Вычитание оба раза производится до по-

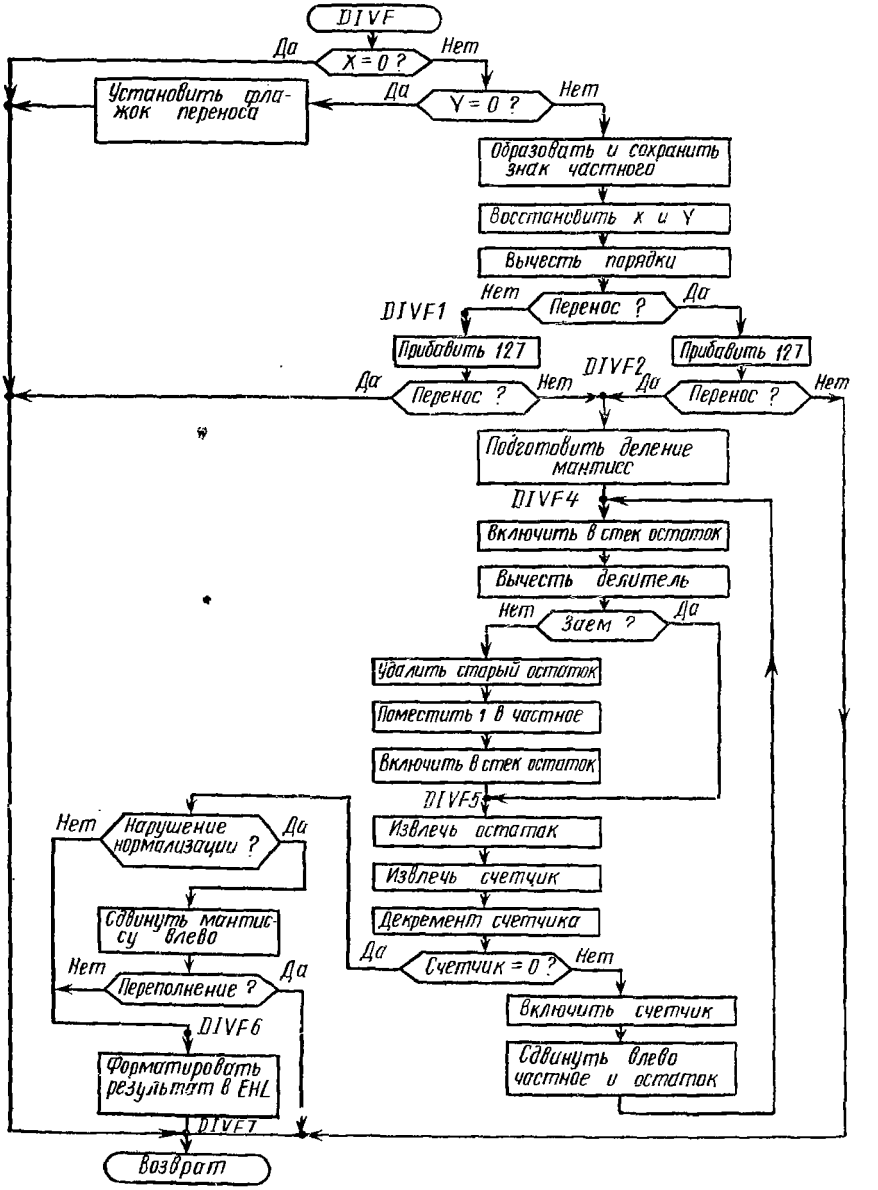


Рис. 2.18. Схема алгоритма деления чисел с плавающей точкой

лучения отрицательной разности с подсчетом числа вычитаний, а затем к отрицательной разности прибавляется 100 (или 10) для восстановления последней положительной разности, меньшей 100 (или 10).

Программа 2.31. *Преобразование 8-битного двоичного целого беззнакового числа в упакованное десятичное:*

```
        ; Исходное двоичное число в аккумуляторе A,
        ; результат возвращается в регистре HL.
        ;
BBCD:   MVI     H,-1        ; Начальное значение равно -1
L100:   INR     H           ; Прибавить 1 к цифре сотен
        SUI     100         ; Вычесть 100
        JNC     L100        ; Повторять цикл для цифры сотен
        ADI     100         ; Восстановить положительную разность
        MVI     L,-1        ; Начальное значение равно -1
L10:    INR     L           ; Прибавить 1 к цифре десятков
        SUI     10          ; Вычесть 10
        JNC     L10         ; Повторять цикл для цифры десятков
        ADI     10          ; Восстановить положительную разность
        ;
        ; Объединить цифры десятков и единиц в одном байте.
        MOV     C,A         ; Сохранить цифру единиц
        MOV     A,L         ; Передать цифру десятков
        RRC                 ;    в старшую тетраду
        RRC
        RRC
        RRC
        ORA     C           ; Объединить в регистре L -
        MOV     L,A         ;    цифры десятков и единиц
        RET                 ; Возврат
```

Преобразование упакованного десятичного целого беззнакового числа в двоичное (программа 2.32). Пусть в аккумуляторе A находится байт, представляющий собой упакованное десятичное целое беззнаковое число (диапазон от 0 до 99), и необходимо образовать в аккумуляторе эквивалентное двоичное число. Преобразование заключается в том, чтобы старшую тетраду аккумулятора (т. е. цифру десятков) умножить на 10 и прибавить к полученному произведению младшую тетраду (цифру единиц). Наиболее просто умножение на 10 (в двоичном коде 1010B) выполняется путем умножения цифры на 8 (сдвиг на три бита влево) и прибавления цифры, умноженной на 2 (сдвиг на один бит влево). Поскольку цифра десятков находится в старшей тетраде, вместо сдвига ее из младшей тетрады влево осуществляется сдвиг вправо.

Программа 2.32. *Преобразование упакованного десятичного целого беззнакового числа в двоичное:*

```
        ; Исходное десятичное число находится в аккумуляторе A,
        ; результат возвращается также в аккумуляторе.
        ;
BCDB:   MOV     B,A         ; Сохранить исходное число
        ANI     0F0H        ; Выделить цифру десятков
        RRC                 ; Она умножена на 8
        MOV     C,A         ; Сохранить промежуточный результат
        RRC                 ; Цифра десятков,
        RRC                 ;    умноженная на 2
        ADD     C           ; Цифра десятков умножена на 10
        MOV     C,A         ; Сохранить промежуточный результат
        MOV     A,B         ; В исходном числе
        ANI     0FH         ;    выделить цифру единиц
        ADD     C           ; Образовать двоичное число
        RET                 ; Возврат
```

Преобразование 8-битного двоичного целого беззнакового числа в неупакованное шестнадцатеричное (программа 2.33). Шестнадцатеричные числа, применяемые для индикации адресов при выводе на экран или принтер, представляются в неупакованном формате: байт содержит одну шестнадцатеричную цифру. В преобразованиях таких чисел необходимо учитывать, что коды «буквенных» цифр A—F (41H—46H) не следуют по порядку за кодами десятичных цифр 0—9 (30H—39H). Поэтому в подпрограммах преобразования с шестнадцатеричными неупакованными числами необходимо принимать во внимание «разрыв» или смещение между цифрами, которое равно 7 (41H—39H=7).

Простая программа 2.33 преобразует двоичное число, находящееся в аккумуляторе A, в двухразрядное неупакованное шестнадцатеричное число, которое возвращается в регистр HL.

Здесь для преобразования каждой тетрады в неупакованную шестнадцатеричную цифру вызывается подпрограмма CONV.

Преобразование неупакованного шестнадцатеричного числа в двоичное (программа 2.34). Следующая подпрограмма выполняет функцию, обратную по отношению к предыдущей подпрограмме: она воспринимает в регистре HL двухразрядное неупакованное шестнадцатеричное число и возвращает в аккумулятор A его двоичный эквивалент.

Преобразование неупакованного шестнадцатеричного числа в двоичное с контролем. В программе 2.34 не контролируется правильность кодов шестнадцатеричных цифр, поэтому при наличии в регистре HL запрещенных комбинаций она возвращает бессмысленный результат. При обработке ввода с клавиатуры приходится учитывать возможность того, что оператор нажимает неверную

клавишу. Программа 2.35 вводит с клавиатуры 4-разрядное шестнадцатеричное число в неупакованном формате, преобразует его в двоичное и размещает в регистре HL. Предполагается, что ввод с клавиатуры осуществляет подпрограмма INPUT, которая возвращает код символа нажатой клавиши в аккумулятор A. Для своей работы она использует регистр HL, поэтому его содержимое перед вызовом подпрограммы INPUT приходится сохранять в стеке, а затем восстанавливать.

Программа 2.33. *Преобразование двоичного целого беззнакового числа в неупакованное шестнадцатеричное:*

```
        ; Двоичное число находится в аккумуляторе A.
        ; Результат возвращается в регистре HL.
        ;
BHEX:   MOV     B,A         ; Сохранить двоичное число
        ANI     0F0H        ; Выделить старшую тетраду
        RRC                 ; Передать ее
        RRC                 ;    в младшую тетраду
        RRC
        RRC
        CALL    CONV        ; Преобразовать в 16-ную цифру
        MOV     H,A         ;    и поместить Y в регистр H
        MOV     A,B         ; Вернуть число в аккумулятор
        ANI     0FH         ; Выделить младшую тетраду
        CALL    CONV        ; Преобразовать в 16-ную цифру
        MOV     L,A         ;    и поместить ее в регистр L
        RET                 ; Возврат
        ;
        ; Подпрограмма CONV преобразует младшую тетраду
        ; аккумулятора в неупакованную 16-ричную цифру.
CONV:   CPI     10          ; Цифра больше 9?
        JC      NOT1        ; Нет, смещение не требуется
        ADI     7           ; Да, прибавить смещение
NOT1:   ADI     30H         ; Образовать код цифры
        RET                 ; Возврат
```

Преобразование неупакованного десятичного числа в двоичное с контролем (программа 2.36). Следующая подпрограмма аналогична предыдущей, но здесь с клавиатуры вводится десятичное число, поэтому содержимое регистра HL приходится умножать на десять. Для этого используется соотношение $10 \times X = 8 \times X + 2 \times X$.

Преобразование 16-битного двоичного числа в неупакованное шестнадцатеричное. Выше рассмотрена программа 2.33, преобразующая двоичное число, находящееся в аккумуляторе A, в двухразрядное неупакованное шестнадцатеричное число. Несколько ус-

ложним ситуацию и будем считать, что подлежащее выводу на экран, или принтер, двоичное число размещается в регистре HL. Собственно вывод одной цифры осуществляет подпрограмма PCHAR; выводимая цифра передается ей в аккумуляторе A.

Преобразование 16-битного двоичного числа в неупакованное десятичное (программа 2.38). Функция подпрограммы BINDEC аналогична функции предыдущей подпрограммы, но двоичное число, находящееся в регистре HL, преобразуется в неупакованный десятичный формат. Вывод одной цифры осуществляет подпрограмма PCHAR.

Программа 2.34. *Преобразование неупакованного шестнадцатеричного числа в двоичное:*

```
        ; Исходное число в регистре HL.
        ; Двоичное число возвращается в аккумуляторе.
        ;
HEXB:   MOV     A,L         ; Младшая цифра числа
        CALL    TRANS       ; Преобразовать ее в двоичную тетраду
        MOV     B,A         ; Сохранить в регистре B
        MOV     A,H         ; Старшая цифра числа
        CALL    TRANS       ; Преобразовать ее в двоичную тетраду
        RLC                 ; Передать в старшую тетраду
        RLC                 ;     аккумулятора
        RLC
        RLC
        ORA     B           ; Объединить две тетрады
        RET                 ; Возврат
        ;
        ; Подпрограмма TRANS преобразует неупакованную
        ; 16-ричную цифру в двоичную тетраду
TRANS:  SUI     30H         ; Убрать кодовое смещение для цифр
        CPI     10H         ; Цифра больше 9?
        JC      NOT1        ; Нет, результат готов
        SUI     7           ; Убрать смещение для буквенных цифр
NOT1:   RET                 ; Возврат
```

Программа 2.35. *Преобразование неупакованного шестнадцатеричного числа в двоичное с контролем:*

```
        ; С клавиатуры вводится 16-ричное число, которое преобразуется
        ; в двоичное и размещается в регистре HL. Контроль
        ; переполнения отсутствует. Ввод заканчивается нажатием
        ; нецифровой клавиши.
        ;
HEXBIN: LXI     H,0         ; Очистить регистр HL
NEWCH:  PUSH    H           ; Сохранить содержимое HL в стеке
        CALL    INPUT       ; Введенная цифра в аккумуляторе
        POP     H           ; Восстановить содержимое HL
        SUI     30H         ; Убрать кодовое смещение для цифр
        RM                  ; Конец - недействительная цифра
        CPI     10H         ; Цифра больше 9?
        JC      ADDTO       ; Нет, ввести ее в регистр HL
        SUI     7           ; Убрать смещение для буквенных цифр
        CPI     0AH         ; Цифра меньше A?
        RM                  ; Конец - недействительная цифра
        CPI     10H         ; Цифра больше F?
        RP                  ; Конец - недействительная цифра
ADDTO:  MOV     D,A         ; Сохранить цифру в регистре D
        DAD     H           ; Освободить место
        DAD     H           ;     для новой цифры
        DAD     H
        DAD     H
        MOV     A,L         ; Передать введенную цифру
        ORA     D           ;     в младшую тетраду регистра L
        MOV     L,A
        JMP     NEWCH       ; Вводить следующую цифру
```

Программа 2.36. *Преобразование неупакованного десятичного числа в двоичное:*

```
        ; С клавиатуры вводится десятичное число, которое
        ; преобразуется в двоичное и размещается в регистре HL.
        ; Контроль переполнения отсутствует. Ввод заканчивается
        ; нажатием нецифровой клавиши.
        ;
DECBIN: LXI     H,0         ; Очистить регистр HL
NEWDIG: PUSH    H           ; Сохранить содержимое HL
        CALL    INPUT       ; Введенная цифра в аккумуляторе
        POP     H           ; Восстановить содержимое HL
        SUI     30H         ; Убрать кодовое смещение для цифр
        RM                  ; Конец - недействительная цифра
        CPI     10H         ; Цифра больше 9?
        RP                  ; Конец - недействительная цифра
        DAD     H           ; Удвоить содержимое HL
        MOV     D,H         ; Передать его
        MOV     E,L         ;     в регистры DE
        DAD     H           ; Содержимое HL умножено на 4
        DAD     H           ; Содержимое HL умножено на 8
        DAD     D           ; Содержимое HL умножено на 10
        MVI     D,0         ; Прибавить новую цифру
        MOV     E,A
        DAD     D
        JMP     NEWDIG      ; Вводить следующую цифру
```

Программа 2.37. *Преобразование двоичного числа в неупакованное шестнадцатеричное:*

```
        ; Двоичное число находится в регистре HL. Вывод,
        ; начиная со старшей цифры, производится подпрограммой PCHAR.
        ;
BINHEX: MOV     A,H         ; Начать вывод
        CALL    PRINT1      ;    со старшей цифры
        MOV     A,H         ; Очередная цифра
        CALL    PRINT2
        MOV     A,L         ; Очередная цифра
        CALL    PRINT1
        MOV     A,L         ; Вывести младшую цифру
        CALL    PRINT2
        RET                 ; Возврат
        ;
        ; Подпрограмма преобразования тетрады в неупакованную
        ; 16-ричную цифру и вывода ее.
PRINT1: RLC                 ; Произвести обмен тетрад
        RLC                 ;    (старшая на месте младшей)
        RLC
        RLC
PRINT2: ANI     0FH         ; Выделить младшую тетраду
        ADI     30H         ; Преобразовать в код
        CPI     3AH         ; Если цифра больше 9,
        JC      NOT1        ;    прибавить смещение
        ADI     7
        CALL    PCHAR       ; Вывести цифру
        RET                 ; Возврат
```

Программа 2.38. *Преобразование двоичного числа в неупакованное десятичное:*

```
        ; Двоичное число находится в регистре HL. Вывод начинается
        ; со старшей цифры, производится подпрограммой PCHAR.
        ;
BINDEC: LXI     D,-10000  ; Печать цифры десятков тысяч
        CALL    PRINT
        LXI     D,-1000   ; Печать цифры тысяч
        CALL    PRINT
        LXI     D,-100    ; Печать цифры сотен
        CALL    PRINT
        LXI     D,-10     ; Печать цифры десятков
        CALL    PRINT
        MOV     A,L       ; Печать цифры единиц
        ORI     30H
        CALL    PCHAR
        RET               ; Возврат
        ;
        ; Подпрограмма определения и печати цифры.
PRINT:  MVI     C,2FH     ; Образовать счетчик
LOOP:   INR     C         ; Инкремент счетчика
        SHLD    TEMP      ; Сохранить положительную разность
        DAD     D         ; Вычесть степень 10
        JC      LOOP      ; Продолжать вычитание
        LHLD    TEMP      ; Восстановить положительную разность
        MOV     A,C       ; Вывести цифру
        CALL    PCHAR
        RET               ; Возврат
```

2.4.5. ПРЕОБРАЗОВАНИЕ ЧИСЕЛ ПО МЕТОДУ СДВИГА И КОРРЕКЦИИ

Рассмотренные выше алгоритмы и программы рассчитаны на преобразования чисел небольшой разрядности и при увеличении ее становятся довольно громоздкими. В то же время известны способы преобразования, в которых длина программ практически не зависит от разрядности чисел. Эти способы преобразования между двоичными и десятичными числами опираются на метод сдвига и коррекции. Он допускает относительно простую аппаратную реализацию и предполагает наличие двух регистров (регистр *1* и регистр *2*), связанных цепями сдвига (рис. 2.19). В регистре *1* находится исходное число, а в регистре *2* образуется результат преобразования. Одним из регистров является обычный двоичный сдвигающий регистр, а вторым — регистр, рассчитанный на хранение десятичных чисел, но имеющий цепи обычного двоичного

сдвига. Очевидно, сдвиг на один бит влево двоичного регистра эквивалентен умножению его содержимого на 2 (конечно, с учетом выдвигаемого слева бита). Если произвести n сдвигов, где n — длина регистра, слева будет последовательно «выдвинуто» все хранимое в регистре число, начиная со старших битов. Сдвиг на один бит вправо эквивалентен делению содержимого двоичного регистра на 2 и после n сдвигов справа будет выдвинуто хранимое число, начиная с младших битов. Если выдвигаемые из регистра биты подавать в другой регистр, выполняющий аналогичные операции, но в другом формате, в нем будет образован эквивалент исходного числа, представленный в требуемом формате.

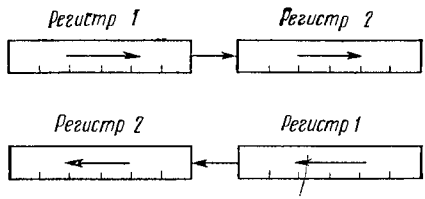


Рис. 2.19. Принцип преобразования по методу сдвига и коррекции

Однако обычный (двоичный) сдвиг десятичного регистра в общем случае не дает правильного результата. Поэтому для выполнения операций умножения и деления содержимого десятичного регистра на 2 требуются специальные корректирующие действия. Рассмотрим их для одного байта упакованного и неупакованного десятичного числа.

Деление на 2 десятичного однобайтного числа. Двоичный сдвиг упакованного десятичного числа вправо на один бит дает в i-й тетраде правильный результат, если из младшего бита соседней слева тетрады сдвигается нуль. Когда из старшей тетрады сдвигается 1, она приобретает в i-й тетраде вес 8, а правильный десятичный вес равен 5. Следовательно, в тех тетрадах десятичного регистра, в которых после двоичного сдвига старший бит содержит 1, необходима коррекция, заключающаяся в вычитании из тетрады числа 3. Приведем пример коррекции:

упакованное десятичное число	1001	0010	0111	0111
двоичный сдвиг вправо	0100	1001	0011	1011
коррекция		—0011		—0011
результат	0100	0110	0011	1000

Здесь исходное число равно 9277, а после сдвига и коррекции получается 4638 и справа выдвигается остаток 1.

Рассмотрим подпрограмму 2.39 деления на 2 упакованного однобайтного десятичного числа при следующих предположениях. Байт в аккумуляторе А содержит две цифры упакованного десятичного числа. Флажок переноса содержит значение бита, который передан из соседнего старшего десятичного разряда. После возврата из подпрограммы в аккумуляторе должен находиться результат деления исходного числа на 2 (с учетом состояния флажка переноса, который считается как бы разрядом сотен), а остаток, т. е. исходный младший бит аккумулятора, должен ока-

заться во флажке переноса. Для промежуточных результатов подпрограмма может использовать регистры D и E.

Программа 2.39. *Деление на два байта, содержащего две десятичные цифры:*

```
        ; Преобразуемый байт находится в аккумуляторе A.
        ; Младший бит соседней старшей тетрады во флажке переноса.
        ; Частное возвращается в аккумуляторе, а остаток во флажке
        ; переноса.
RPACK:  RAR             ; Двоичный сдвиг вправо через перенос
        PUSH    PSW     ; Сохранить флажок переноса
        MOV     D,A     ; Сохранить результат сдвига
        ANI     0FH     ; Выделить младшую тетраду
        CPI     8       ; Есть 1 из старшей тетрады?
        JC      NOC1    ; Нет, коррекция не нужна
        SUI     3       ; Скорректировать младшую тетраду
NOC1:   MOV     E,A     ; Сохранить младшую тетраду
        MOV     A,D     ; Вернуть сдвинутый байт
        ANI     0F0H    ; Выделить старшую тетраду
        CPI     80H     ; Была 1 из старшей тетрады?
        JC      NOC2    ; Нет, коррекция не нужна
        SUI     30H     ; Скорректировать старшую тетраду
NOC2:   ORA     E       ; Объединить тетрады
        MOV     E,A     ; Восстановить
        POP     PSW     ;    флажок переноса
        MOV     A,E     ;    (выдвинутый бит)
        RET             ; Возврат
```

Рассмотрим подпрограмму 2.40 деления на 2 однобайтного неупакованного десятичного числа. Считаем, что байт в аккумуляторе A содержит десятичную цифру (30H—39H), а флажок переноса — бит, переданный из соседнего старшего десятичного разряда. После возврата из подпрограммы в аккумуляторе должно быть частное от деления исходной цифры на 2 (с учетом состояния флажка переноса), а остаток, т. е. первоначальный младший бит аккумулятора, должен оказаться во флажке переноса.

Умножение на 2 десятичного однобайтного числа. Обычный (двоичный) сдвиг упакованного десятичного числа влево дает правильный результат, если в каждой тетраде находится цифра, меньшая 5. Когда цифра больше или равна 5, необходимо передать 1 в соседний старший десятичный разряд. В данном случае удобно произвести коррекцию до сдвига, обнаруживая цифры, большие или равные, 5 и прибавляя в соответствующие тетрады 3. Результирующая тетрада будет содержать 1

в старшем бите, которая передается в старший десятичный разряд при обычном двоичном сдвиге. Приведем пример коррекции:

упакованное десятичное число	1001	0010	0111	0111
коррекция	+0011		+0011	+0011
промежуточный результат	1100	0010	1010	1010
двоичный сдвиг влево	1000	0101	0101	0100

Здесь исходное число равно 9277, а после коррекции и сдвига получается 8554 (или с учетом единицы десятков тысяч — 18554).

Программа 2.40. *Деление на два байта, содержащего неупакованную десятичную цифру:*

```
        ; Преобразуемый байт находится в аккумуляторе A. Младший бит
        ; соседней старшей десятичной цифры во флажке переноса.
        ; Частное возвращается в аккумуляторе, а остаток во флажке
        ; переноса.
RUNPCK: RAR             ; Двоичный сдвиг вправо через перенос
        PUSH    PSW     ; Сохранить флажок переноса
        JM      COR     ; Требуется коррекция
        ANI     07FH    ; Выделить нужные биты
        JMP     FRES    ;    и образовать результат
COR:    ANI     07FH    ; Выделить нужные биты
        ADI     5       ; Скорректировать результат
FRES:   ORI     30H     ; Образовать код цифры
        MOV     E,A     ; Восстановить
        POP     PSW     ;    флажок переноса
        MOV     A,E     ;    (выдвинутый бит)
        RET             ; Возврат
```

В подпрограмме 2.41 умножения на 2 однобайтного упакованного десятичного числа предполагается, что байт в аккумуляторе содержит две десятичные цифры, а флажок переноса — бит, который передан из соседнего младшего десятичного разряда. После возврата из подпрограммы в аккумуляторе должен находиться результат умножения исходного числа на 2, а выдвигаемый бит должен попасть во флажок переноса.

Программа 2.41. *Умножение на два байта, содержащего две упакованные десятичные цифры:*

```
        ; Преобразуемый байт находится в аккумуляторе. Выдвинутый
        ; из соседней младшей десятичной тетрады бит содержится
        ; во флажке переноса. Результат возвращается в аккумуляторе
        ; и флажке переноса.
        ;
LPACK:  PUSH    PSW         ; Сохранить флажок переноса
        MOV     D,A         ; Сохранить исходный байт
        ANI     0FH         ; Выделить младшую тетраду
        CPI     5           ; Она больше 5?
        JC      NOC1        ; Нет, коррекция не нужна
        ADI     3           ; Скорректировать младшую тетраду
NOC1:   MOV     E,A         ; Сохранить младшую тетраду
        MOV     A,D         ; Передать исходный байт в аккумулятор
        ANI     0F0H        ; Выделить старшую тетраду
        CPI     50H         ; Она больше 5?
        JC      NOC2        ; Нет, коррекция не нужна
        ADI     30H         ; Скорректировать старшую тетраду
NOC2:   ORA     E           ; Объединить тетрады
        MOV     E,A         ; Скорректировать байт в регистре E
        POP     PSW         ; Вернуть флажок переноса
        MOV     A,E         ; Скорректировать байт в аккумуляторе
        RAL                 ; Сдвинуть влево через перенос
        RET                 ; Возврат
```

Подпрограмма LPACK разработана без привлечения команды DAA десятичной коррекции аккумулятора. С помощью этой команды функцию подпрограммы LPACK можно реализовать всего двумя командами:

```
        ADC     A           ; Удвоить содержимое аккумулятора
        DAA                 ; Скорректировать результат
```

Программа 2.42. *Умножение на два байта, содержащего неупакованную десятичную цифру:*

```
                ; Преобразуемый байт находится в аккумуляторе A.
                ; Выдвинутый из соседнего младшего десятичного разряда
                ; бит содержится во флажке переноса. Результат
                ; возвращается в аккумуляторе и флажке переноса.
                ;
LUNPCK: MVI     D,0         ; Отметить для будущего переноса
        RAL                 ; Сдвинуть влево через перенос
        ANI     1FH         ; Выделить 5 младших бит
        CPI     10          ; Была цифра больше 4?
        JC      NOC         ; Нет, коррекция не нужна
        ADI     6           ; Скорректировать цифру
        MVI     D,0FFH      ; Отметить для будущего переноса
NOC:    ORI     30H         ; Образовать код цифры
        MOV     E,A         ; Сохранить цифру
        MOV     A,D         ; Образовать правильный бит
        RAR                 ;    во флажке переноса
        MOV     A,E
        RET                 ; Возврат
```

При разработке подпрограммы 2.42 удвоения неупакованного десятичного числа считается, что байт в аккумуляторе A содержит десятичную цифру (30H—39H), а флажок переноса — бит, выдвинутый из соседнего младшего десятичного разряда. После возврата из подпрограммы в аккумуляторе должен быть результат умножения исходной цифры на 2, а выдвинутый слева бит должен находиться во флажке переноса.

Поскольку в этом преобразовании байт содержит одну десятичную цифру, коррекцию можно осуществить после двоичного сдвига. Такой вариант по сравнению с коррекцией до сдвига приводит к более короткой подпрограмме.

Имея набор из четырех подпрограмм, нетрудно запрограммировать преобразования между двоичными числами (целыми и дробями) и десятичными числами (целыми и дробями), представленными в упакованном и неупакованном форматах. Вместо аппаратных регистров и схем коррекции двоичные и десятичные числа размещаются в буферных областях памяти (часто их называют просто буферами), а необходимые сдвиги и коррекции осуществляются программно. В приводимых далее программах предполагается, что двоичное число находится в области памяти с начальным адресом BBUF и длиной N байт. Для десятичного числа выделена область с начальным адресом DBUF и длиной K байт. Значения N и K либо являются исходными данными, либо определяются в соответствии с точностью преобразования, так как, например, десятичная дробь может не иметь абсолютно точного

двоичного представления. Как всегда, адреса BBUF и DBUF относятся к младшим байтам чисел.

Все программы преобразований имеют циклическую структуру с тремя циклами. Глобальный цикл определяется длиной двоичного числа в битах, поэтому начальное значение счетчика этого цикла равно 8×N. Начало цикла идентифицирует метка LOOP, а счетчиком цикла служит регистр B. Следовательно, ограничиваем длину двоичных чисел 31 байт, что удовлетворяет всем практическим требованиям.

Два внутренних цикла осуществляют умножение или деление двоичного числа (этот цикл начинается с метки BLOOP) и десятичного числа (начало цикла показывает метка DLOOP). Начальными значениями счетчиков этих циклов являются N и K, а в качестве счетчика используется один и тот же регистр C. Сдвиг вправо начинается со старших байтов, а сдвиг влево — с младших байтов. В процессе сдвигов связь между байтами регистров, а также между обоими регистрами осуществляется через флажок переноса. Когда двоичное число преобразуется в десятичное, буферная область памяти для хранения десятичного числа должна быть очищена, т. е. должна содержать нули. Когда десятичное число преобразуется в двоичное, начальное содержимое двоичного буфера безразлично, так как оно будет вытеснено результатом преобразования. Во всех программах преобразования исходные числа ради простоты считаются беззнаковыми. Учет знака нетрудно осуществить отдельным действием.

Преобразование правильной двоичной дроби в десятичную. Пусть дана правильная двоичная дробь X длиной n бит ($n=8\times N$):

$$X=0.x_1x_2\ldots x_n=x_1 2^{-1}+x_2 2^{-2}+\ldots+x_n 2^{-n}.$$

Если произвести n сдвигов этой дроби вправо и подавать выдвигаемые справа биты в десятичный регистр, осуществляющий деление на 2, то в нем будет образован десятичный эквивалент.

Общая схема преобразования поясняется на рис. 2.20 и следующим примером для упакованного десятичного формата.

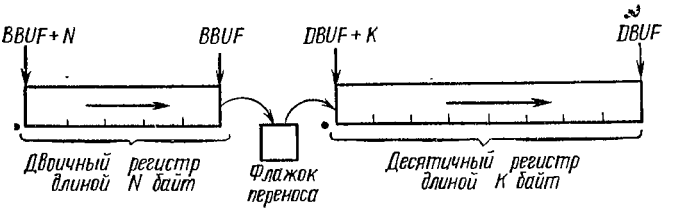


Рис. 2.20. Преобразование двоичной дроби в десятичную

Операция	Двоичная дробь	Десятичная дробь			
		10^{-1}	10^{-2}	10^{-3}	10^{-4}
Начальное состояние	0.1011	0000	0000	0000	0000
1. Сдвиг	0.0101	1000	0000	0000	0000
Коррекция		—0011			
Результат		0101	0000	0000	0000
2. Сдвиг	0.0010	1010	1000	0000	0000
Коррекция		—0011	—0011		
Результат		0111	0101	0000	0000
3. Сдвиг	0.0001	0011	1010	1000	0000
Коррекция			—0011	—0011	
Результат		0011	0111	0101	0000
4. Сдвиг	0.0000	1001	1011	1010	1000
Коррекция		—0011	—0011	—0011	—0011
Результат		0110	1000	0111	0101
		.6	8	7	5

Здесь исходная дробь равна $^{11}/_{16}$ и после преобразования получен правильный результат 0.6875. Нетрудно убедиться, что абсолютно точное преобразование n-битной двоичной дроби требует для результата n десятичных разрядов. На практике значение K выбирается в зависимости от требуемой точности преобразования.

Подпрограмма 2.43 преобразует двоичную дробь в упакованный десятичный формат. Для получения подпрограммы преобразования двоичной дроби в неупакованный десятичный формат необходимо заменить команду CALL RPACK на команду CALL RUNPCK.

Преобразование десятичного целого числа в двоичное. Данное преобразование очень похоже на преобразование двоичной дроби в десятичную. За исходное принимается десятичное число

$$X = x_{k-1}x_{k-1}\ldots x_1x_0 = x_{k-1}10^{k-1} + x_{k-2}10^{k-2} + \ldots + x_0,$$

в котором каждая цифра x_i представлена тетрадой (упакованный формат) или байтом (неупакованный формат).

Если последовательно делить X и получающиеся частные на 2, то выдвигаемые справа биты будут давать цифры двоичного представления исходного числа. Общая схема преобразования показана на рис. 2.21 и иллюстрируется следующим примером для упакованного десятичного формата.

Программа 2.43. *Преобразование двоичной дроби в десятичную:*

```
        ; Двоичная дробь в буфере BBUF длиной N байт,
        ; десятичная дробь в буфере DBUF длиной K байт.
        ; До вызова необходимо очистить буфер DBUF.
        ; После возврата буфер BBUF содержит нули.
        ;
FBD:    MVI     B,8*N       ; Образовать счетчик бит
LOOP:   LXI     H,BBUF+N-1  ; Сдвиг начинается со старшего байта
        XRA     A           ; Сбросить флажок переноса
        MVI     C,N         ; Образовать счетчик байт
        ;
        ; Сдвиг двоичной дроби вправо.
BLOOP:  MOV     A,M         ; Очередной байт двоичной дроби
        RAR                 ; Сдвинуть его вправо
        MOV     M,A         ; Вернуть в двоичный буфер
        DCX     H           ; Продвинуть указатель
        DCR     C           ; Декремент счетчика байт
        JNZ     BLOOP       ; Повторять для всей дроби
        ;
        ; Сдвиг десятичной дроби вправо.
        LXI     H,DBUF+K-1  ; Сдвиг со старшего байта
        MVI     C,K         ; Образовать счетчик байт
DLOOP:  MOV     A,M         ; Очередной байт десятичной дроби
        CALL    RPACK       ; Разделить его на два
        MOV     M,A         ; Вернуть в десятичный буфер
        DCX     H           ; Продвинуть указатель
        DCR     C           ; Декремент счетчика байт
        JNZ     DLOOP       ; Повторять для всей дроби
        ;
        ; Преобразование одного бита закончено.
        DCR     B           ; Декремент счетчика бит
        JNZ     LOOP        ; Повторять для всех бит
        RET                 ; Возврат
```

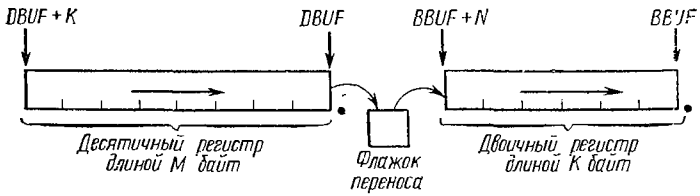


Рис. 2.21. Преобразование целого десятичного числа в двоичное

Операция	Десятичное число			Двоичное число
	10^2	10^1	10^0	
Начальное состояние	0010	0011	0111	XXXXXXXX
1. Сдвиг	0001	0001	1011	1XXXXXXX
Коррекция			—0011	
Результат	0001	0001	1000	
2. Сдвиг	0000	1000	1100	01XXXXXX
Коррекция		—0011	—0011	
Результат	0000	0101	1001	
3. Сдвиг	0000	0010	1100	101XXXXX
Коррекция			—0011	
Результат	0000	0010	1001	
4. Сдвиг	0000	0001	0100	1101XXXX
Коррекция (нет)				
Результат	0000	0001	0100	
5. Сдвиг	0000	0000	1010	01101XXX
Коррекция			—0011	
Результат	0000	0000	0111	
(Больше коррекции не будет)				
6. Сдвиг	0000	0000	0011	101101XX
7. Сдвиг	0000	0000	0001	1101101X
8. Сдвиг	0000	0000	0000	11101101

Полученное двоичное число равно исходному десятичному числу 237.

Приведенная подпрограмма 2.44 преобразует десятичное число, представленное в упакованном формате. Чтобы перейти к неупакованному формату, команду CALL RPACK следует заменить на команду CALL RUNPCK.

Преобразование целого двоичного числа в десятичное. Пусть целое двоичное число

$$X = x_{n-1}x_{n-2}\ldots x_1x_0 = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \ldots + x_12^1 + x^0$$

необходимо преобразовать в десятичное. Представим X в виде полинома Горнера:

$$X = ((x_{n-1}2 + x_{n-2})2 + \ldots + x_1)2 + x_0.$$

Десятичный эквивалент X можно получить, сдвигая двоичное число влево и подавая выдвигаемые двоичные цифры в младший разряд десятичного регистра. Одновременно со сдвигом двоичного регистра необходимо удваивать содержимое десятичного регистра. Это действие можно осуществить, вызывая для удвоения подпрограмму LPACK (упакованный формат) или LUNPCK (не-

упакованный формат). Общая схема преобразования показана на рис. 2.22 и иллюстрируется следующим примером.

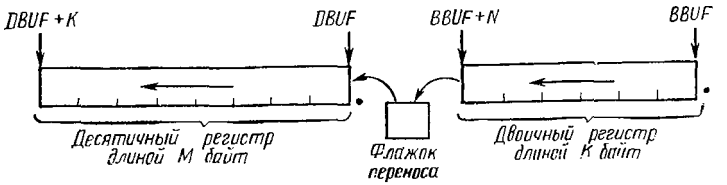


Рис. 2.22. Преобразование целого двоичного числа в десятичное

Программа 2.44. *Преобразование целого десятичного числа в двоичное:*

```
        ; Десятичное число в буфере DBUF длиной K байт,
        ; двоичное число в буфере BBUF длиной N байт.
        ; Начальное состояние буфера BBUF безразлично.
        ; После возврата буфер DBUF содержит нули.
        ;
IDB:    MVI     B,8*N       ; Образовать счетчик бит
LOOP:   LXI     H,DBUF+K-1  ; Сдвиг со старшего байта
        XRA     A           ; Сбросить флажок переноса
        MVI     C,K         ; Образовать счетчик байт
        ;
        ; Сдвиг десятичного числа.
DLOOP:  MOV     A,M         ; Очередной байт десятичного числа
        CALL    RPACK       ; Разделить его на 2
        MOV     M,A         ; Вернуть в буфер
        DCX     H           ; Продвинуть указатель
        DCR     C           ; Декремент счетчика байт
        JNZ     DLOOP       ; Повторять для всего числа
        ;
        ; Сдвиг двоичного числа.
        LXI     H,BBUF+N-1  ; Сдвиг со старшего байта
        MVI     C,N         ; Образовать счетчик байт
BLOOP:  MOV     A,M         ; Очередной байт двоичного числа
        RAR                 ; Сдвинуть его вправо через перенос
        MOV     M,A         ; Вернуть в память
        DCX     H           ; Продвинуть указатель
        DCR     C           ; Декремент счетчика байт
        JNZ     BLOOP       ; Повторять для всех байт
        ;
        ; Преобразование одного бита закончено.
        DCR     B           ; Декремент счетчика бит
        JNZ     LOOP        ; Повторять для всех бит
        RET                 ; Возврат
```

Операция	Десятичное число			Двоичное число
	10^2	10^1	10^0	
Начальное состояние	0000	0000	0000	11110101
1. Коррекция (нет)	0000	0000	0000	11110101
Сдвиг	0000	0000	0001	11101010
2. Коррекция (нет)	0000	0000	0001	
Сдвиг	0000	0000	0011	11010100
3. Коррекция (нет)	0000	0000	0011	
Сдвиг	0000	0000	0111	10101000
4. Коррекция			+0011	
Результат	0000	0000	1010	
Сдвиг	0000	0001	0101	01010000
5. Коррекция			+0011	
Результат	0000	0001	1000	
Сдвиг	0000	0011	0000	10100000
6. Коррекция (нет)	0000	0011	0000	01000000
Сдвиг	0000	0110	0001	
7. Коррекция		+0011		
Результат	0000	1001	0001	10000000
Сдвиг	0001	0010	0010	
8. Коррекция (нет)	0001	0010	0010	00000000
Сдвиг	0010	0100	0101	

Результатом преобразования является десятичное число, равное исходному двоичному числу.

Подпрограмма 2.45 преобразует двоичное число в упакованный десятичный формат. Для перехода к неупакованному десятичному формату команду CALL LPACK нужно заменить на команду CALL LUNPCK.

Преобразование десятичной дроби в двоичную. Для преобразования десятичной дроби

$$X=0.x_1x_2\ldots x_k=x_1 10^{-1}+x_2 10^{-2}+\ldots+x_k 10^{-k},$$

где x_i — десятичная цифра, можно применить традиционный способ умножения на 2 исходной дроби и дробных частей получающихся произведений. Цифрами двоичной дроби будут последовательно получаемые целые части произведений, т. е. выдвигаемые слева биты. Общая схема преобразования представлена на рис. 2.23. Следующий пример показывает абсолютно точное преобразование.

Операция	Двоичная дробь	Десятичная дробь		
		10^{-1}	10^{-2}	10^{-3}
Начальное состояние	0.XXX	0110	0010	0101
1. Коррекция		+0011		+0011
Результат		1001	0010	1000
Сдвиг	0.XX1	0010	0101	0000
2. Коррекция			+0011	
Результат		0010	1000	0000
Сдвиг	0.X10	0101	0000	0000
3. Коррекция		+0011		
Результат		1000	0000	0000
Сдвиг	0.101	0000	0000	0000

Программа 2.45. *Преобразование целого двоичного числа в десятичное:*

```
                ; Двоичное число находится в буфере BBUF длиной N байт,
                ; десятичное число в буфере DBUF длиной K байт.
                ; Первоначально буфер  DBUF должен быть очищен.
                ; После возврата буфер BBUF содержит нули.
                ;
IBD:    MVI     B,8*N       ; Образовать счетчик бит
LOOP:   LXI     H,BBUF      ; Начальный адрес двоичного числа
        ORA     A           ; Сбросить флажок переноса
        MVI     C,N         ; Образовать счетчик байт
        ;
        ; Сдвиг влево двоичного числа.
BLOOP:  MOV     A,M         ; Очередной байт двоичного числа
        RAL                 ; Сдвинуть его влево через перенос
        MOV     M,A         ; Вернуть в двоичный буфер
        INX     H           ; Продвинуть указатель
        DCR     C           ; Декремент счетчика байт
        JNZ     BLOOP       ; Повторять для всех байт
        ;
        ; Теперь удвоение десятичного числа.
        LXI     H,DBUF      ; Начальный адрес десятичного числа
        MVI     C,K         ; Образовать счетчик байт
DLOOP:  MOV     A,M         ; Очередной байт десятичного числа
        CALL    LPACK       ; Удвоить его
        MOV     M,A         ; Вернуть в буфер
        INX     H           ; Продвинуть указатель
        DCR     C           ; Декремент счетчика байт
        JNZ     DLOOP       ; Повторять для всех байт
        ;
        ; Один бит,обработан, повторить для других бит.
        DCR     B           ; Декремент счетчика бит
        JNZ     LOOP        ; Повторять до завершения
        RET                 ; Возврат
```

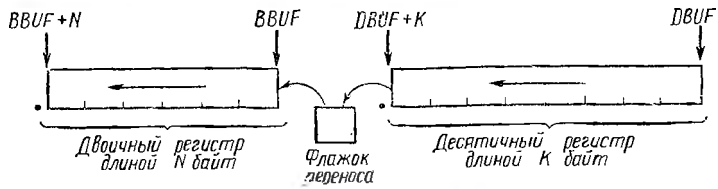


Рис. 2.23. Преобразование десятичной дроби в двоичную

Программа 2.46. *Преобразование десятичной дроби в двоичную:*

```
        ; Десятичная дробь находится в буфере DBUF длиной K байт,
        ; двоичная дробь в буфере BBUF длиной N байт.
        ; Начальное содержимое буфера BBUF произвольно.
        ; После возврата буфер DBUF содержит нули.
        ;
FDB:    MVI     B,8*N       ; Образовать счетчик бит
LOOP:   LXI     H,DBUF      ; Начальный адрес десятичной дроби
        ORA     A           ; Сбросить флажок переноса
        MVI     C,K         ; Образовать счетчик байт
        ;
        ; Сдвиг десятичной дроби влево.
DLOOP:  MOV     A,M         ; Очередной байт десятичной дроби
        CALL    LPACK       ; Удвоить его
        MOV     M,A         ; Вернуть в десятичный буфер
        INX     H           ; Продвинуть указатель
        DCR     C           ; Декремент счетчика байт
        JNZ     DLOOP       ; Повторять для всех байт
        ;
        ; Теперь удвоение двоичной дроби.
        LXI     H,BBUF      ; Начальный адрес двоичной дроби
        MVI     C,N         ; Образовать счетчик байт
BLOOP:  MOV     A,M         ; Очередной байт двоичной дроби
        RAL                 ; Сдвинуть его влево
        MOV     M,A         ; Вернуть в двоичный буфер
        INX     H           ; Продвинуть указатель
        DCR     C           ; Декремент счетчика байт
        JNZ     BLOOP       ; Повторять для всех байт
        ;
        ; Один бит обработан, повторить для других бит.
        DCR     B           ; Декремент счетчика бит
        JNZ     LOOP        ; Повторять до завершения
        RET                 ; Возврат
```

Полученная двоичная дробь $^5/_8$ эквивалентна исходной десятичной дроби 0.625.

В подпрограмме 2.46 предполагается представление десятичной дроби в упакованном формате. Для перехода к неупакованному формату команда CALL LPACK заменяется на команду CALL LUNPCK.

Преобразование форматов чисел с плавающей точкой. В заключение рассмотрим две программы, в первой из которых целое знаковое число преобразуется в формат с плавающей точкой, а во второй осуществляется обратное преобразование.

Пусть содержимое регистра HL интерпретируется как целое знаковое число в дополнительном коде. Необходимо образовать в регистрах EHL представление этого числа в 24-битном формате с плавающей точкой, который был принят для программ арифметических операций (знак, байт смещенного порядка и два байта мантиссы). Первое действие преобразования (2.47) — проверка исходного числа на нуль; если оно равно нулю, происходит возврат с тремя нулевыми байтами в регистрах EHL (истинный нуль). После этого в стеке сохраняется знак исходного числа (во флажке переноса), а в регистре HL образуется абсолютное значение числа. Далее организуется цикл сдвига числа влево (начало цикла показывает метка LOOP) до тех пор, пока мантисса не будет нормализованной (в старшем бите регистра HL находится 1). С каждым сдвигом происходит декремент смещенного порядка на 1; за исходный смещенный порядок принимается +16 (или 8FH). Заключительные действия подпрограммы, начинающиеся с метки FORM, связаны с форматированием результата.

Обратное преобразование числа с плавающей точкой требует отдельного представления его целой и дробной частей. Целая часть числа преобразуется в 16-битное знаковое число в дополнительном коде, причем точка фиксирована после младшего значащего разряда. Для дробной части примем также представление в дополнительном коде, но зафиксируем точку после знакового бита. Предполагается, что исходное число находится в регистрах EHL, а после преобразования целая часть будет в регистре BC, а дробная — в регистре HL.

В подпрограмме 2.48 исходное число вначале проверяется на нуль и, если оно равно нулю, в регистр результата BC загружается нуль. Флажок переноса устанавливается в 1, показывая успешное преобразование. Затем проверяется нахождение исходного числа в диапазоне представимых чисел выходного формата. Для этого в регистре B образуется байт смещенного порядка, который сравнивается с максимальным значением 8EH (истинный порядок равен 15) и минимальным значением 70H (истинный порядок равен —15). Если исходный смещенный порядок выходит за эти границы, подпрограмма заканчивается со сброшенным в 0 флажком переноса.

Программа 2.47. *Преобразование целого числа в формат с плавающей точкой:*

```
        ; Исходное число находится в регистре HL,
        ; результат возвращается в регистрах EHL.
        ;
ITOF:   MOV     A,H         ; Проверить исходное число
        ORA     L           ;    на нуль
        MOV     E,A
        JZ      EXIT        ; Число равно нулю, возврат
        MOV     A,H         ; Старший байт в аккумуляторе
        ADD     A           ; Знак во флажке переноса
        PUSH    PSW         ; Сохранить знак в стеке
        JNC     NOC         ; Число положительное
        CALL    COMP        ; Изменить знак числа
NOC:    MVI     E,8EH       ; Инициализировать порядок
LOOP:   DAD     H           ; Сдвинуть число влево
        DCR     E           ; Декремент порядка
        MOV     A,H         ; Проверить на окончание
        ORA     H           ;    нормализации
        JP      LOOP        ; Нормализация не закончена
FORM:   POP     PSW         ; Вернуть знак во флажке переноса
        MOV     A,E         ; Передать знак в старший бит
        RAR
        MOV     E,A
        MOV     A,H         ; Форматировать
        RAL                 ;    старший байт мантиссы
        RRC
        MOV     H,A
EXIT:   RET                 ; Возврат, конец преобразования
```

Когда число находится в допустимых границах, определяется, имеет ли оно целую часть. Для этого смещенный порядок числа вычитается из кода 8EH. Если разность превышает 16, в регистр целой части загружается нуль и происходит переход к преобразованию дробной части (метка FRAC). В случае ненулевой целой части мантисса сдвигается вправо так, чтобы разряд единиц оказался в младшем разряде регистра HL. Собственно сдвиг производится подпрограммой SHIRD. Она отличается от подпрограммы сдвига SHIFT (см. программу 2.22) тем, что в освобождающийся при сдвиге старший бит регистра HL помещается нуль, а не значение флажка переноса, и счетчиком сдвигов служит регистр D. Подпрограмма COMP при необходимости образует в регистре HL дополнительный код целой части числа. Затем сформированная целая часть включается в стек, а в регистр HL из стека передается сохраненная в нем исходная мантисса.

Программа 2.48. *Выделение целой и дробной частей числа с плавающей точкой:*

```
        ; Исходное число с плавающей точкой в регистрах EHL,
        ; целая часть результата в регистре BC, дробная часть
        ; в регистре HL. О выходе за диапазон сигнализирует
        ; сброшенный в 0 флажок переноса.
        ;
FTOI:   MOV     A,E         ; Проверить исходное число
        ORA     H           ;     на нуль
        ORA     L
        MOV     B,A
        MOV     C,A
        JZ      QUIT        ; Результат равен нулю
        MOV     A,H         ; Образовать в регистре B
        RAL                 ;     байт смещенного порядка
        MOV     A,E
        RAL
        MOV     B,A
        ;
        ; Проверить нахождение числа в допустимом диапазоне.
        CPI     8EH         ; Сравнить с максимальным порядком
        JNC     EXIT        ; Выход за верхнюю границу
        CPI     70H         ; Сравнить с максимальным порядком
        CMC
        JNZ     EXIT        ; Выход за нижнюю границу
        ;
        ; Преобразование возможно.
        MOV     A,H         ; Восстановить
        ORI     80H         ;     скрытую единицу мантиссы
        MOV     H,A
        PUSH    H           ; Сохранить мантиссу
        MVI     A,8EH       ; Образовать в регистре D
        SUB     B           ;     счетчик сдвигов влево
        MOV     D,A
        CPI     16          ; Имеется ли целая часть?
        JC      INTEG       ; Да, преобразовать ее
        LXI     H,0         ; Целая часть равна нулю
        JMP     FRAC        ; Перейти к получению дробной части
        ;
        ; Преобразование целой части.
        CALL    SHIRD       ; Сдвинуть мантиссу вправо
        MOV     A,E         ; Проверить знак числа
        ORA     A           ;     и при необходимости образовать
        JP      FRAC        ;     дополнительный код
        CALL    COMP
        ;
        ; Преобразование дробной части.
FRAC:   XTHL                ; Целая часть в стеке, мантисса в HL
```

```
        MOV     A,B         ; Образовать истинный
        SUI     7FH         ;    порядок числа
        JZ      NOSH        ; Истинный порядок равен нулю
        JNC     SHILA       ; Мантиссу необходимо сдвигать влево
        CMA                 ; Образовать счетчик для сдвига
        INR     A           ;    мантиссы вправо
        MOV     D,A
        CALL    SHIRD       ; Сдвинуть мантиссу вправо
        JMP     SIGN        ; Учесть знак числа
SHILA:  DAD     H           ; Сдвинуть мантиссу влево
        DCR     A
        JNZ     SHILA
        JMP     SIGN        ; Учесть знак числа
NOSH:   MOV     A,H         ; Подавить восстановленную
        ANI     7FH         ;    скрытую единицу
        MOV     H,A
        ;
        ; Учесть знак числа и разместить результат.
SIGN:   MOV     A,E         ; Проверить знак числа
        ORA     A           ;    и при необходимости образовать
        CALL    COMP        ;    дополнительный код
NOC1:   POP     B           ; Разместить целую часть
QUIT:   STC                 ; Отметить успешное преобразование
EXIT:   RET                 ; Возврат
        ;
        ; Подпрограмма сдвига содержимого регистра HL вправо.
        ; Счетчик сдвигов в регистре D.
SHIRD:  ORA     A           ; Сбросить флажок переноса
        MOV     A,H         ; Сдвинуть регистр H,
        RAR                 ;    связь через флажок переноса
        MOV     H,A
        MOV     A,L         ; Сдвинуть регистр L
        RAR
        MOV     L,A
        DCR     D           ; Декремент счетчика сдвигов
        JNZ     SHIRD       ; Повторять сдвиги
        RET                 ; Возврат
```

Путем вычитания из смещенного порядка значения 7FH определяются действия, необходимые для образования в регистре HL дробной части. Если исходный порядок равен нулю, требуется просто подавить восстановленную скрытую 1 разряда целой части. При положительном истинном порядке мантиссу приходится сдвигать влево, а когда он отрицательный — вправо. Для сдвига влево применяется команда DAD H. Дробная часть числа также преобразуется в дополнительный код.

Контрольные вопросы и упражнения

1. Покажите, как выглядят адресные пространства памяти и ввода-вывода в МП К580.

2. Перечислите достоинства и недостатки изолированного ввода-вывода и ввода-вывода, отображенного на память.

3. Назовите сходства и различия между регистром М и другими регистрами микропроцессора.

4. Почему в любой программе число стековых операций PUSH должно быть равно числу стековых операций POP? Поясните, к каким последствиям приведет нарушение этого правила.

5. Пусть максимальный адрес области стека равен TOP, а минимальный — BOTTOM. Разработайте граф-схему контроля нахождения текущего стека в указанных границах.

6. Предположим, что выполняется команда POP извлечения данных из вершины стека. Каким образом можно извлечь из стека эти же данные еще раз?

7. Каждая ли строка исходной ассемблерной программы содержит машинную команду?

8. Требуется ли комментарий в каждой строке ассемблерной программы?

9. В каком случае поле операции ассемблерной строки может отсутствовать?

10. Пусть аккумулятор А содержит 7АН, а регистр В — 0В5Н. Определите содержимое аккумулятора и состояния флажков после выполнения каждой из следующих команд: ADD B; SUB B; ADI 0FCH; INR B; DCR A.

11. Назовите все различия команд INR A и ADI 1; DCR A и SUI 1.

12. Постройте граф-схему действий, выполняемых командой DAA.

13. Пусть в аккумуляторе находится число 8АН, а в регистре С — число 2СН. Определите состояния флажков после выполнения команд CMP B и CPI 0E9H.

14. Приведите начальное состояние необходимых регистров и напишите команду, после выполнения которой будет установлено в 1 максимальное число флажков. Существует ли команда, после выполнения которой все флажки будут находиться в состоянии 1?

15. Напишите программный фрагмент, который осуществляет кольцевую пересылку содержимого всех регистров общего назначения.

16. При выполнении команд микропроцессор формирует машинный цикл M_i при каждом обращении к памяти или ввод-выводу. Определите число машинных циклов, необходимых для выполнения команд ADD B; ADD M; MVI A, 37H; STAX D; OUT PORT; PUSH B; SHLD ADDR; XTHL.

17. Напишите все команды, которыми можно сбросить аккумулятор А, и укажите различия между ними.

18. Приведите последовательность команд, которая увеличивает содержимое аккумулятора А в 5 раз; в 7 раз.

19. Приведите команды, с помощью которых:
    бит 6 аккумулятора А устанавливается в 1,
    биты 3 и 5 аккумулятора А сбрасываются в 0,
    биты 1, 3, 5 и 7 аккумулятора А инвертируются,
    бит 7 регистра В устанавливается в 1.
    Содержимое всех остальных регистров не изменяется.

20. Напишите команды, передающие управление метке LABEL, если:
    аккумулятор А содержит нуль.
    аккумулятор А не содержит 0FFH,
    аккумулятор А содержит положительное число.

21. Пусть в аккумуляторе А находится знаковое число X, представленное в дополнительном коде. Запишите программный фрагмент, вызывающий переход к метке LESS, если X<—25, и к метке GREATER, если X>50.

22. Предположим, что в регистрах В и С находятся целые знаковые числа, представленные в дополнительном коде. Разработайте граф-схему алгорит-

ма и соответствующую программу их сложения. Результат должен находиться в аккумуляторе A, а флажок C должен показывать переполнение.

23. Измените программу 2.1 так, чтобы сумма помещалась на место операнда, адресуемого регистром HL.

24. Как изменилась бы программа 2.1, если команда DCR B модифицировала бы флажок переноса?

25. Укажите максимальную длину операндов, допустимую в программе 2.1. При каком начальном состоянии она получается?

26. Можно ли в программе 2.4 заменить команды RAL и CC OVER на команду CM OVER?

27. От чего зависит время выполнения программы 2.6? Приведите условия, при которых оно будет максимальным (минимальным).

28. Что окажется в регистрах A-HL (программа 2.7), если команда LXI H, 0 отсутствует?

29. Можно ли в программе 2.7 поставить метку NOADD у команды ACI 0?

30. Постройте граф-схему умножения для программы 2.9.

31. Предположим, что в программе 2.9 команда LXI H, 0 отсутствует и при вызове MUL16 в регистре HL находится число X. Какой результат будет возвращен?

32. Каким образом время умножения в программе 2.11 зависит от длины сомножителей?

33. Необходимо ли сбрасывать регистр частного в программе 2.12? Другими словами, можно ли заменить трехбайтную команду LXI H, 8 двухбайтной командой MVI L, 8?

34. Команда DAA будет «корректировать» результат двоичного сложения и в том случае, когда операнды не являются десятичными числами. Определите, какой результат будет получен в аккумуляторе A после выполнения команд ADD B и DAA, если (A)=0D6AH, (B)=72H; (A)=0FFH, (B)=0C8H?

35. Как изменится общее описание программы 2.17, если поменять местами соседние команды MOV M, A и XGHG?

36. Определите диапазон и точность представления чисел для формата с плавающей точкой, принятого в п. 2.4.3.

37. При каких условиях получается максимальное (минимальное) время выполнения программ 2.26 и 2.27?

38. Какое исходное число обеспечивает минимальное (максимальное) время выполнения программы 2.31?

39. Можно ли в программе 2.31 заменить четыре команды RRC на четыре команды RLC?

40. Зависит ли время выполнения программы 2.32 от исходного десятичного числа в аккумуляторе A?

41. Возможно ли переполнение в программе 2.47?

42. Почему в программе 2.47 за исходное значение порядка принято +16?

ГЛАВА 3

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ В МИКРОПРОЦЕССОРЕ К1810ВМ86

Настоящая глава построена так же, как и гл. 2, и посвящена гораздо более мощному 16-битному однокристальному микропроцессору К1810ВМ86, который широко применяется в отечественных профессиональных персональных компьютерах. После обсуждения общей характеристики микропроцессора, его программной модели, разнообразных режимов адресации и обширной системы команд рассмотрен вопрос о программной совместимости микропроцессоров К580ИК80 и К1810ВМ86.

По-прежнему основной материал главы представлен алгоритмами и программами арифметических операций над числами в различных форматах. Кроме того, приведены программы преобразований форматов и извлечения корня квадратного из целого 32-битного числа.

3.1. ОБЩАЯ ХАРАКТЕРИСТИКА МИКРОПРОЦЕССОРА

Микросхема К1810ВМ86 (далее сокращенно К1810) представляет собой однокристальный ЦП с фиксированными длиной слова (16 бит) и системой команд. Как и в случае МП К580, для организации функционально законченного изделия к МП К1810 необходимо подключать память и средства ввода-вывода. Сфера применения этого МП благодаря его возросшим возможностям гораздо шире, чем у МП К580. Достаточно сказать, что производительность МП К1810 примерно на порядок выше, а по функциональным возможностям он приближается к процессорам миникомпьютеров. К достоинствам МП К1810 относятся встроенные средства работы с операционными системами, мультипрограммирования, организации мультипроцессорных систем, обработки сложных структур данных и эффективной реализации языков высокого уровня. Обладая мощной системой команд, МП оперирует такими типами данных, как биты, байты, слова (16 бит), длинные слова (32 бит), упакованные и неупакованные десятичные числа, цепочки байтов и слов.

МП выполнен по высококачественной NMOП-технологии и вы-

пускается в 40-контактном корпусе типа DIP. На кристалле размещено около 30 000 транзисторов. Напряжение питания составляет +5 В ± 5%, потребляемая мощность не превышает 1,75 Вт, а рабочий температурный диапазон от 10 до 70°С. Синхронизация осуществляется однофазными сигналами с частотой 2—5 МГц, которые формирует микросхема генератора синхронизации К1810ГФ84.

Особенностью МП является возможность работы в одном из двух режимов (минимальном и максимальном), который определяется уровнем напряжения на одном из входов. В зависимости от режима изменяется интерпретация восьми управляющих сигналов (так называемое «аппаратное программирование»). Минимальный режим ориентирован на относительно простые однопроцессорные микросистемы. В этом режиме МП сам генерирует все управляющие сигналы системной шины. В максимальном режиме, рассчитанном на сложные мультипроцессорные системы, для генерирования управляющих сигналов системной шины требуется микросхема контроллера шины К1810ВГ88. Кроме того, у МП по-

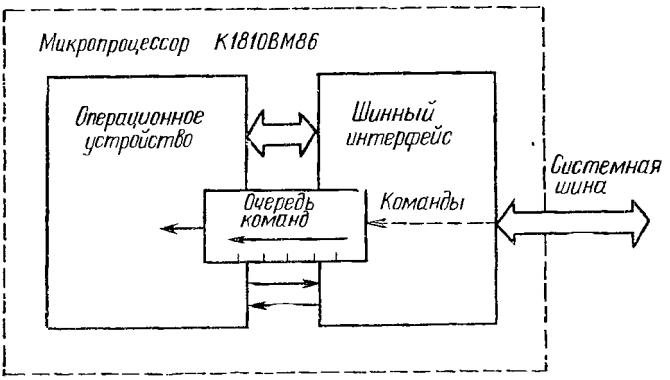


Рис. 3.1. Структурная схема микропроцессора К1810ВМ86

является несколько сигналов, упрощающих построение мультипроцессорных систем и предназначенных, в частности, для управляемого доступа к разделенным ресурсам системы и синхронизации работы процессоров. Для программиста максимальный режим характерен тем, что в нем можно пользоваться командой (префиксом) блокировки шины LOCK, который применяется для управления семафорами, и командами арифметического сопроцессора (см. гл. 4).

Структурно МП состоит из двух практически автономных устройств, показанных на рис. 3.1. Операционное устройство выполняет команды, уже выбранные из программной памяти шинным интерфейсом и находящиеся во внутренней 6-байтной очереди ко-

манд. Шинный интерфейс выбирает команды из памяти, считывает операнды и записывает результаты. Оба устройства могут работать параллельно и в большинстве случаев обеспечивают совмещение выборки и выполнения команд, что повышает эффективную производительность МП.

В процессе выполнения программы между устройствами осуществляется динамическое взаимодействие. Операционное устройство получает командные байты из очереди команд. Если команда связана с обращением к памяти или вводу-выводу, оно запрашивает об этом шинный интерфейс. Когда последний свободен, он сразу удовлетворяет запрос и производит требуемое обращение. Но бывают редкие ситуации, когда шинный интерфейс занят своим обращением к памяти и операционному устройству приходится некоторое время ожидать. По мере считывания команд из очереди в ней образуются свободные («пустые») байты. Как только шинный интерфейс фиксирует два пустых байта, он самостоятельно инициирует выборку командного слова из программной памяти. Конечно, очередь эффективно действует при естественном порядке выполнения команд. Когда операционное устройство исполняет команду передачи управления, шинный интерфейс сбрасывает очередь, выбирает команду по новому адресу, сразу же передает ее операционному устройству, а затем начинает заполнение (реинициализацию) очереди из следующих ячеек памяти. Механизм опережающей выборки команд полностью скрыт от программиста и никак не влияет на программирование.

Система команд МП К1810 намного богаче системы команд МП К580. Хотя число базовых команд увеличилось ненамного (113 по сравнению с 79), фактически мощность системы команд возросла в несколько раз. Например, одна базовая команда MOV заменяет такие команды МП К580, как MOV, MVI, LXI, LHLD, SHLD, LDA, STA, LDAX, STAX и SPHL. Если команды всех бинарных операций в МП К580 явно адресуют только один операнд, а второй подразумевается командой (в подавляющем большинстве им является содержимое аккумулятора А), то в МП К1810 бопьшинство команд бинарных операций явно адресуют оба операнда. Это означает, в частности, что почти все функции аккумулятора А может выполнять любой общий регистр.

МП К1810 имеет общую организацию регистр — память. Такое определение подразумевает, что двухоперандные команды имеют типы регистр — регистр, регистр — память и память — регистр, а команды типа память — память отсутствуют (за исключением команд передачи и сравнения цепочек). В МП имеются все основные режимы адресации, характерные для современных процессоров и ориентированные на эффективную реализацию языков высокого уровня: прямая (абсолютная), регистровая, регистровая косвенная, непосредственная, базовая, индексная и некоторые их модификации.

В обоих режимах работы адресное пространство памяти состоит из 2^{20} (1 048 576) байт — см. рис. 3.2. Физический адрес, выдаваемый МП при любом обращении к памяти, имеет длину 20 бит. По-прежнему два любых соседних байта в памяти образуют слово по принципу «младшее — по меньшему адресу» (по этому же принципу размещаются адреса и непосредственные операнды в командах). МП К1810 передает слово, находящееся по четному адресу, за один цикл шины, а по нечетному — за два цикла. Поэтому целесообразно размещать слова данных по четным адресам; особенно это относится к стеку, который оперирует только словами. Особенности формирования 20-битных физических адресов памяти внутри МП рассмотрены в § 3.2.

В системах на базе МП К1810 можно организовать как изолированный ВВ, так и ВВ, отображенный на память. Для изолированного ВВ применяются команды ввода IN и вывода OUT, при-

Рис. 3.2. Адресное пространство памяти микропроцессора К1810ВМ86

чем в них допускается прямая и косвенная адресация входных и выходных портов, а передаваться могут байты и слова. Адресное пространство ВВ для обоих режимов адресации показано на рис. 3.3. Прямой адрес порта содержится в команде как константа, допуская адресацию 256 входных и 256 выходных портов. Иногда такой способ обращения к портам называется *статическим* ВВ. Косвенный адрес порта находится в одном из регистров МП и имеет длину 16 бит. Поскольку изменение содержимого этого регистра заставляет одну и ту же команду обращаться к различным портам, этот способ ВВ называется *динамическим*. В самом

МП для вводимых и выводимых данных выделены два регистра — 8-битный аккумулятор AL и 16-битный аккумулятор AX. Обычным образом в системе реализуется ВВ, отображенный на память; в этом способе адреса входных и выходных портов, как и ячеек памяти, имеют длину 20 бит.

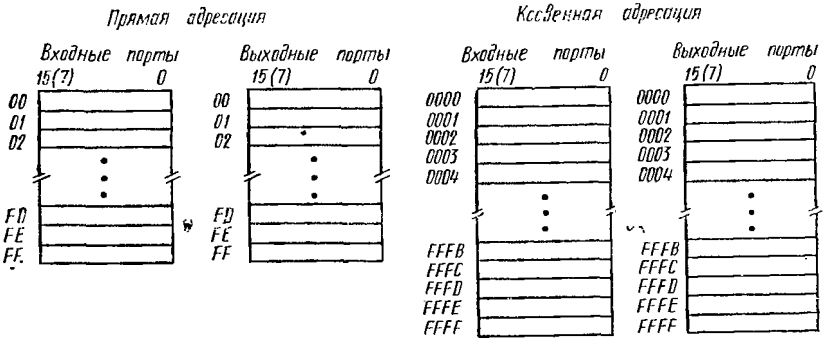


Рис. 3.3. Адресное пространство ввода-вывода микропроцессора К1810ВМ86

Взаимодействие МП с другими компонентами системы осуществляется по системной шине, состоящей из шин адреса, данных и управления. Ограниченное число контактов корпуса МП заставило использовать в нем мультиплексированные (многофункциональные) линии, т. е. передавать/принимать по одним и тем же линиям различные сигналы с разделением во времени. Главный входной и выходной тракт МП образован шестнадцатью двунаправленными тристабильными линиями адреса/данных AD15—AD0. На них в начале каждого цикла шины МП выдает младшие 16 бит физического адреса памяти или полный адрес порта ВВ. Старшие 4 бит адреса памяти выдаются на однонаправленные линии A19/ST6—A16/ST3. Через некоторое время линии AD превращаются в шину данных, по которой либо вводятся данные из памяти или ВВ, либо выводятся данные, подлежащие записи в память или ВВ. В это же время на линиях A19/ST6—A16/ST3 действуют сигналы состояния («статуса»). Таким образом, временна́я диаграмма функционирования МП К1810 сложнее временно́й диаграммы работы МП К580 и требует специальных аппаратных средств и управляющих сигналов для демультиплексирования шины адреса/данных, т. е. превращения ее в отдельные шины адреса и данных. Об этих тонкостях работы МП программисту знать не обязательно.

В программной модели МП, показанной на рис. 3.4, имеется несколько групп регистров с различным функциональным назначением.

Группа регистров общего назначения представлена регистрами AX, BX, CX и DX. Их особенность заключается в том, что в командах допускается указывать их старшую (High) и младшую (Low) половины. Такой двойственный характер регистров AX—DX позволяет многим командам оперировать байтами и словами. Остальные регистры можно использовать только «целиком». Регистры AX—DX находятся в полном распоряжении программиста и единообразно участвуют в арифметических и логических операциях. Но в некоторых командах они специализированы, что отражено в их названиях.

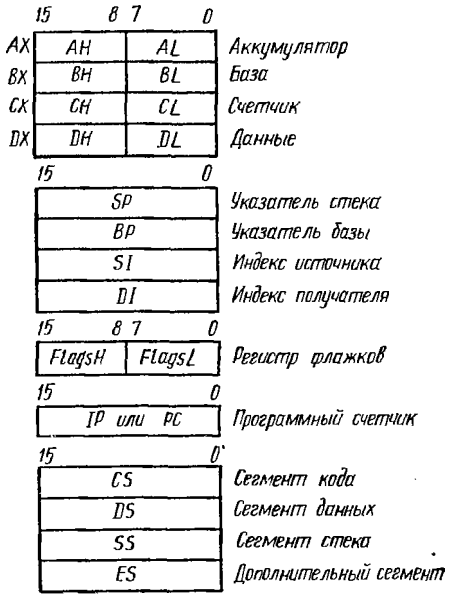


Рис. 3.4. Программная модель микропроцессора К1810ВМ86

Регистр AX используется в операциях умножения, деления, ввода-вывода слов и в некоторых операциях с цепочками. Регистр AL участвует в аналогичных операциях с байтами, в операциях преобразования и десятичной арифметики. Регистр AH используется в умножении и делении байтов. Программы получаются компактнее, если регистры AX и AL максимально привлекаются для арифметических и логических операций и пересылок данных.

Регистр BX интенсивно применяется для адресации данных в памяти и участвует в операции преобразования.

Регистр CX выполняет функции счетчика повторений в программных циклах и в операциях с цепочками. Регистр CL служит счетчиком сдвигов.

Регистр DX участвует в операциях умножения и деления слов; кроме того, он содержит адрес порта в командах ВВ с косвенной адресацией.

Регистры BP, SP, SI и DI образуют группу указательных и индексных регистров. Такое название подчеркивает, что они предназначены для хранения адресов, обеспечивая косвенную адреса-

цию памяти и участвуя в вычислениях эффективного адреса (об адресации памяти см. далее). Но эти регистры могут привлекаться для арифметических и логических операций так же, как и регистры первой группы. Поэтому регистры AX—DX и BP—DI называются *общими регистрами*. Всего получается восемь общих регистров и для выбора любого из них в командах достаточно трех битов. Указатель стека SP адресует вершину TOS аппаратного стека в памяти, и обе стековые операции (push и pop) автоматически модифицируют SP. С помощью указателя базы BP обеспечивается простой доступ к данным, находящимся в стеке (не только в вершине стека TOS). Наиболее часто регистр BP привлекается для адресации параметров, передаваемых через стек подпрограммам. Индексные регистры SI и DI применяются для адресации данных, а также цепочек в специальных командах.

Довольно нерегулярная структура общих регистров требует, чтобы программист (или транслятор с языка высокого уровня) тщательно распределял регистры и следил за их использованием. В то же время неявное указание некоторых регистров в операциях и режимах адресации позволяет компактнее закодировать команды.

Формат регистра флажков МП К1810 приведен на рис. 3.5.

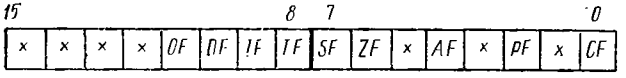


Рис. 3.5. Формат регистра флажков микропроцессора К1810ВМ86

Шесть арифметических флажков CF, PF, AF, ZF, SF и OF фиксируют определенные признаки результата арифметической или логической операции. Команды МП воздействуют на них по-разному, но, в общем, эти флажки показывают следующие особенности результата (первые 5 флажков аналогичны флажкам МП К580):

Флажок переноса CF фиксирует значение бита переноса (заема), возникающего при сложении (вычитании) байтов или слов, а также значение выдвигаемого бита во всех операциях сдвигов. Он же показывает особенность результата операций умножения и деления. Как уже было видно в программах для МП К580, флажок переноса играет исключительно важную роль.

Флажок паритета (четности) PF регистрирует установкой в 1 наличие четного числа единиц в 8 младших битах результата операции. Он применяется для контроля правильности передач данных.

Флажок AF вспомогательного переноса аналогичен флажку CF, но фиксирует перенос или заем из младшей тетрады резуль-

тата. Этот флажок необходим только в операциях десятичной арифметики.

Флажок нуля ZF сигнализирует о получении нулевого результата операции.

Флажок знака SF повторяет значение старшего бита результата, который в дополнительном коде соответствует знаку числа.

Флажок переполнения OF отмечает потерю старшего бита результата операции сложения или вычитания над знаковыми числами. Переполнение возникает, когда значения переносов в старший бит и из старшего бита не совпадают. Флажок OF показывает также изменение старшего (знакового) бита в арифметических сдвигах влево.

Три оставшихся флажка управляют некоторыми действиями МП. Программист может одной или несколькими командами задать состояние любого из них.

1. Флажок направления DF определяет сканирование (просмотр) цепочек от меньших адресов к большим (DF=0) или наоборот (DF=1).

2. Флажок прерывания IF задает реакцию МП на запрос прерывания по входу INT. Если IF=0, запрос прерывания игнорируется, а если IF=1, МП распознает и соответственно реагирует на него. Флажок IF не влияет на восприятие немаскируемых прерываний по входу NMI и внутренних прерываний.

3. Флажок трассировки TF при установке в 1 переводит МП в покомандный режим работы, в котором МП автоматически генерирует внутреннее прерывание после выполнения каждой команды.

Указатель команды IP выполняет функции программного счетчика PC (далее используется последнее название). В процессе выборки команд из программной памяти производится соответствующая модификация PC для того, чтобы он адресовал следующую команду.

Наличие в программной модели четырех сегментных регистров кода CS, данных DS, стека SS и дополнительных данных ES объясняется способом адресации памяти. МП имеет 20-битную шину внешнего физического адреса памяти, но в программной модели нет ни одного регистра длиной 20 бит. Внутри МП адрес памяти представлен двумя 16-битными словами, одно из которых называется *базовым (начальным) адресом сегмента* (эквивалентные названия — сегментный адрес, сегмент, база), а второе — *внутрисегментным смещением* (синонимы — относительный адрес, смещение). Эти два слова представляют собой логический адрес памяти. Устройство преобразования адресов в составе шинного интерфейса при каждом обращении к памяти превращает логический адрес в физический.

Для программы адресное пространство памяти (см. рис. 3.2) состоит из четырех сегментов, являющихся логическими единица-

ми памяти с максимальным размером 64К байт. Конечно, реальный размер сегмента не обязательно должен быть максимальным; минимальный размер сегмента, как будет показано далее, равен 16 байт. На размещение сегментов в пространстве 1 Мбайт накладывается только одно ограничение: базовый 20-битный адрес сегмента должен быть кратен 16, т. е. его 4 младших бита должны быть нулевыми. Другими словами, базовый адрес сегмента имеет вид XXXX0H. Нулевые биты можно не хранить, а подразумевать, и тогда для базового адреса сегмента достаточно 16 бит. Именно такие «урезанные» базовые адреса сегментов находятся в регистрах CS, DS, SS и ES. Следовательно, при фиксированном содержимом сегментных регистров максимальное рабочее пространство, к которому МП имеет доступ в любой момент времени, состоит из 64К байт для кода (собственно программы), 64К байт для стека и 128К байт для данных. Если программе требуется бо́льшее рабочее пространство, она должна модифицировать содержимое сегментных регистров.

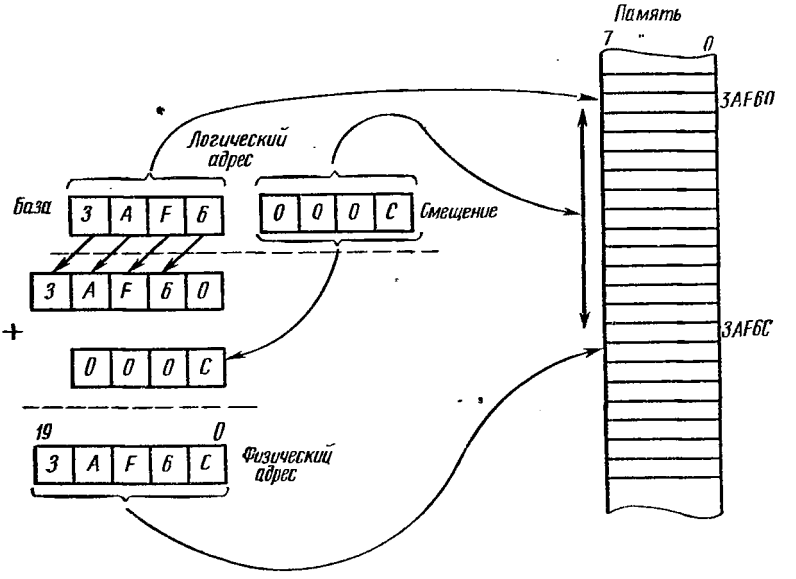


Рис. 3.6. Преобразование логического адреса в физический

Для локализации конкретного байта в сегменте служит вторая компонента логического адреса — смещение. Оно, являясь 16-битным целым беззнаковым числом, показывает расстояние этого байта от начала сегмента. Следовательно, для образования из пары сегмент : смещение физического адреса необходимо сдвинуть базовый адрес сегмента влево на 4 бит и прибавить смещение (рис. 3.6).

Шинный интерфейс получает логический адрес из различных источников в зависимости от типа обращения к памяти. Возможные способы формирования физического адреса приведены в табл. 3.1.

Таблица 3.1. **Источники логического адреса**

Тип обращения к памяти	База (по умолчанию)	Варианты	Смещение
Выборка команды	CS	Нет	PC
Стековая операция	SS	»	SP
Обращение к переменной	DS	CS, SS, ES	EA
Цепочка-источник	DS	CS, SS, ES	SI
Цепочка-получатель	ES	Нет	DI
BP как базовый регистр	SS	CS, DS, ES	EA

Команды всегда выбираются из текущего сегмента кода: базовый адрес сегмента находится в регистре CS, а смещение берется из регистра PC. Стековые команды всегда обращаются к текущему сегменту стека — логический адрес есть SS:SP. Считается, что большинство переменных находится в текущем сегменте данных с базовым адресом в регистре DS, но программист может заставить МП в конкретной команде обратиться к переменной в другом сегменте. Эта возможность показана в столбце «Варианты» табл. 3.1 и реализуется так называемым оператором замены сегмента. Например, команда MOV AX, [DI] обращается к сегменту данных, а команда MOV AX, ES:[DI] — к дополнительному сегменту. В обеих командах смещение берется из регистра DI, что показано посредством заключения названия регистра в квадратные скобки.

Смещение переменной вычисляет операционное устройство в соответствии с заданным в команде режимом адресации. Результат этого вычисления называется *эффективным адресом* EA. Эффективный адрес может быть константой в команде, содержимым одного из адресных регистров (BX, BP, SI, DI), суммой содержимого двух адресных регистров (допускаются пары BX-SI, BX-DI, BP-SI, BP-DI) и, наконец, суммой содержимого двух адресных регистров и константы в команде. В командах с адресацией памяти через регистр BP обращение производится к сегменту стека, а с оператором замены сегмента допускаются обращения к данным в любом из текущих сегментов.

Сегментная организация памяти имеет достоинства и недостатки. К достоинствам ее относят модульность программ (в них четко выделены области данных, стека и собственно кода), простое перемещение программ в пространстве памяти, что важно в мультипрограммной среде, простое переключение с одной программы на другую. Один из недостатков заключается в трудности мани-

пуляций физическими адресами, например в сравнении двух физических адресов (ясно, что неравенство логических адресов не свидетельствует о неравенстве физических адресов). Кроме того, требуются команды передачи управления как внутри текущего сегмента кода (эти команды модифицируют только PC и имеют тип NEAR), так и вне текущего сегмента кода (модифицируются CS и PC, тип FAR). Наконец, несколько усложняется синтаксис языка Ассемблера из-за появления нескольких директивов управления сегментами.

Режимы адресации. Если МП К580 обладает простейшими режимами адресации, то в МП К1810 их реализовано гораздо больше и для разработки прикладных программ режимы адресации нужно знать. Назначение режима адресации заключается в идентификации операнда команды, т. е. в указании способа формирования эффективного адреса EA. Он является либо адресом данных, либо адресом перехода (в командах передачи управления). Как уже говорилось, EA представляет собой целое беззнаковое число, являющееся смещением относительно базы некоторого сегмента.

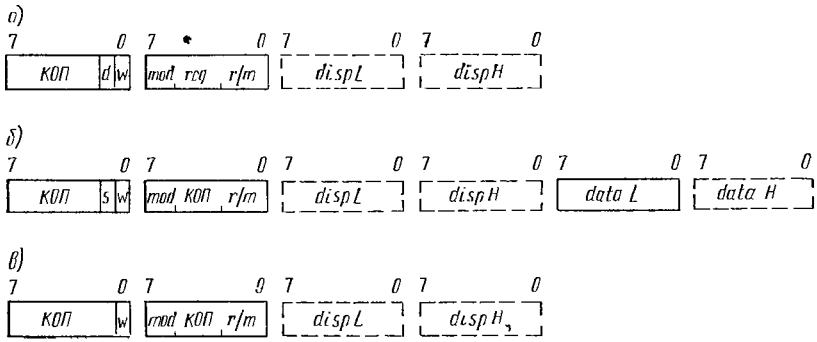


Рис. 3.7. Типичные форматы команд

Команды МП К1810 адресуют максимум два операнда, но не могут адресовать две ячейки памяти. Первым операндом в двухоперандной команде обычно является содержимое регистра или ячейки памяти, а вторым — содержимое регистра или непосредственный операнд. Нумерация «первый» и «второй» довольно условна. Общий формат двухоперандной команды приведен на рис. 3.7, *а* (пунктир показывает необязательные байты). Первый байт содержит код операции (КОП) и два однобитных поля. Поле *d* определяет направление передачи: если $d=1$, то направление «в», если $d=0$, направление «из». Само направление относится ко второму операнду — регистру, указанному в поле *reg* второго байта команды, который называется *постбайтом* (или просто байтом)

режима адресации. Поле w идентифицирует тип операнда: слово ($w=1$) или байт ($w=0$).

Участвующие в операции регистры или регистр и ячейку памяти указывает постбайт, состоящий из трех полей. Поле *reg* (регистр), как уже отмечалось, определяет второй операнд, находящийся в регистре. Кодирование регистров МП в поле *reg* показано в последних двух столбцах табл. 3.2.

Таблица 3.2. **Постбайтные режимы адресации**

r/m \ mod	00	01	10	11	
				w=0	w=1
000	(BX)+(SI)	(BX)+(SI)+D8	(BX)+(SI)+D16	AL	AX
001	(BX)+(DI)	(BX)+(DI)+D8	(BX)+(DI)+D16	CL	CX
010	(BP)+(SI)	(BP)+(SI)+D8	(BP)+(SI)+D16	DL	DX
011	(BP)+(DI)	(BP)+(DI)+D8	(BP)+(DI)+D16	BL	BX
100	(SI)	(SI)+D8	(SI)+D16	AH	SP
101	(DI)	(DI)+D8	(DI)+D16	CH	BP
110	D16	(BP)+D8	(BP)+D16	DH	SI
111	BX	(BX)+D8	(BX)+D16	BH	DI

Поле *mod* (режим) специфицирует режим адресации, показывая, как интерпретируется поле r/m (регистр/память) при нахождении первого операнда. Если $mod=11$, операнд находится в регистре и поле r/m определяет регистр с таким же кодированием, как и в поле *reg*. Три остальных комбинации в поле *mod* (00, 01, 10) относятся к адресации памяти и показывают, чему равно смещение *disp*, содержащееся в команде как константа:

$$mod=\begin{cases} 00,\ disp=0 \text{ — смещение отсутствует,} \\ 01,\ disp=dispL \text{ — один байт смещения (он расширяется со знаком до двух байт),} \\ 10,\ disp=dispH,\ dispL \text{ — два байт смещения.} \end{cases}$$

Смещение *disp* длиной в два байта можно считать и абсолютным адресом (см. режим, в котором $mod=00$ и $r/m=110$).

В случае адресации памяти ($mod\neq 11$) поле r/m показывает, как формируется EA, и его кодирование представлено в табл. 3.2. Как видно из этой таблицы, операнд в памяти допускается адресовать прямо (два байта *disp*) или косвенно (возможно, с одно- или двухбайтным смещением *disp*). Во втором случае память можно адресовать через базовый регистр BX или BP, через индексный регистр SI или DI, а также через комбинацию базового и индексного регистров. Всего получается 24 режима адресации

памяти: три комбинации в поле *mod* и восемь комбинаций в поле *r/m*.

В командах с непосредственным операндом (см. их формат на рис. 3.7, *б*) второй операнд адресовать не нужно, так как он находится в команде, поэтому поле *reg* используется как расширение кода операции. Кроме того, здесь не нужен бит *d*, так как результат можно поместить только на место первого операнда. Но в этом формате необходимо определить тип непосредственного операнда *data*. Для этого служат поля *s* и *w*, интерпретируемые следующим образом:

$$s:w = \begin{cases} \text{X0},\ data = dataL & \text{— один байт данных,} \\ \text{01},\ data = dataH,\ dataL & \text{— два байт (слово) данных,} \\ \text{11},\ data = dataL & \text{— один байт данных, который расширяется со знаком до двух байт.} \end{cases}$$

Поля *mod* и *r/m* имеют такой же смысл, как и в предыдущем формате.

Наконец, на рис. 3.7, *в* показан формат одноопернадной команды. В этом формате нет ничего нового по сравнению с рассмотренными выше.

В МП К1810 есть избыточные форматы, которые позволяют сократить на один байт длину часто используемых команд. В основном они относятся к операциям с регистрами и особенно с аккумуляторами AL и AX. Программа-ассемблер всегда выбирает более короткую команду.

Рассмотрим кратко стандартные режимы адресации с учетом приведенных выше способов формирования EA.

Регистровая адресация. Операнд или операнды находятся в общих регистрах МП, а в некоторых командах — в сегментных регистрах. Команды, оперирующие содержимым регистров, оказываются наиболее короткими и выполняются за минимальное время.

Примеры ассемблерных команд с регистровой адресацией.

```
MOV     AX,SI       ; Передать содержимое SI в AX
MOV     ES,AX       ; Передать содержимое AX в ES
ADD     BX,DI       ; Прибавить к BX содержимое DI
SUB     CL,AH       ; Вычесть из CL содержимое AH
RCL     DX,1        ; Сдвинуть содержимое DX влево
OR      CX,DX       ; Объединить по ИЛИ содержимое CX и DX
```

Как показывают эти примеры, регистры МП К1810 стали гораздо универсальнее регистров МП К580. По существу, каждый 8- или 16-битный общий регистр функционально можно считать аккумулятором.

Непосредственная адресация. Непосредственные операнды *data* представляют собой константы длиной в байт или слово, находящиеся в командах. Благодаря постбайту режима адресации можно оперировать константами и содержимым регистров или ячеек памяти (нет только команд загрузки констант в сегментные регистры и включения констант в стек). Константы применяются для инициализации регистров и ячеек памяти, в качестве масок в поразрядных операциях, для сравнения с граничными значениями и т. д.

Примеры записи команд с непосредственными операндами на языке Ассемблера:

```
MOV     CL,24          ; Загрузить в CL десятичное число 24
XOR     SI,1           ; Инвертировать младший бит SI
AND     AL,80H         ; Выделить старший бит AL
OR      DI,8000H       ; Установить в 1 старший бит DI
CMP     [BX],40H       ; Сравнить содержимое памяти с 64
TEST    AL,30H         ; Проверить нахождение в AL числа 48
```

Абсолютная адресация. В абсолютной (прямой) адресации EA берется из поля *disp* команды. Этот режим применяется для обращения к простым переменным (скалярам), например:

```
MOV     AX,BETA        ; Загрузить в AX переменную BETA
INC     COUNT          ; Инкремент счетчика
MUL     MPL            ; Умножить на значение MPL
ROR     TEMP           ; Сдвинуть значение TEMP вправо
```

Символические имена переменных BETA, COUNT, MPL и TEMP должны быть определены в программе.

Отметим интересную особенность. В командах INC, MUL и ROR невозможно узнать размер операнда — байт или слово. Это примеры так называемых «анонимных» обращений к памяти. Чтобы устранить неоднозначность, ассемблеру требуется дополнительная информация, которую сообщает оператор атрибута типа. Если, например, команда INC COUNT должна производить инкремент слова, необходимо записать INC WORD PTR COUNT; чтобы команда ROR TEMP сдвигала байт, ее нужно записать как ROR BYTE PTR TEMP.

Косвенная регистровая адресация. В этом режиме EA находится в одном из регистров BP, BX, SI или DI. Следовательно, адреса памяти можно вычислять во время выполнения программы, что необходимо для обращения к элементам регулярных структур данных. При изменении содержимого регистра одна и та же команда обращается к различным ячейкам памяти. Ис-

пользование регистра в качестве источника ЕА указывается заключением его имени в квадратные скобки, например:

```
ADD     AX,[DI]       ; Прибавить к AX содержимое ячейки
                      ;    памяти, адресуемой DI
INC     BITE PTR [BX] ; Инкремент байта в памяти
DIV     WORD PTR [SI] ; Разделить на слово из памяти
XOR     [BP],DL       ; Операция с байтом в памяти
```

Базовая адресация. В базовой адресации ЕА равен сумме значения *disp,* находящегося в команде, и содержимого регистра BX или BP. Напомним, что при указании BP шинный интерфейс обращается к операнду в текущем сегменте стека; это упрощает доступ к параметрам подпрограмм, передаваемым в стеке.

Основное применение базовой адресации связано с обработкой структур данных, когда смещение (номер) элемента структуры известен при ассемблировании программы, а базовый (начальный) адрес структуры определяется при выполнении программы. Другими словами, структура данных может находиться в различных областях памяти, а модификация базового регистра обеспечивает доступ к этим областям. В языке Ассемблера базовая адресация обозначается в виде BREG [DISP] или [BREG±DISP], где BREG — один из базовых регистров. Примеры команд с базовой адресацией:

```
MOV     AX,[BP+10]  ; Передать в AX шестое слово массива,
                    ;    базовый адрес которого в BP
ADD     [BX]TEMP,CL ; Прибавить содержимое CL к байту
                    ;    TEMP массива, адресуемого BX
```

Индексная адресация. В режиме индексной адресации ЕА равен сумме смещения *disp*, находящегося в команде, и содержимого регистра SI или DI. Обычно смещение определяет известный при ассемблировании начальный (базовый) адрес массива, а значение в индексном регистре адресует нужный элемент. Простые манипуляции содержимым индексного регистра позволяют обращаться к любому элементу массива.

Режимы базовой и индексной адресации аналогичны, так как длина базовых адресов и индексов равна 16 бит. Однако при разработке МП предполагалось, что регистры BX и BP будут использоваться как базовые, а регистры SI и DI как индексные. В соответствии с этим подразумевается их использование в некоторых командах.

В языке Ассемблера индексный режим обозначается в виде DISP [IREG] или [IREG±DISP], где IREG — один из индексных регистров. Примеры команд с индексной адресацией:

```
MOV     ARRAY[SI],AL  ; Передать AL в элемент массива
ADD     DI,MATRIX[SI] ; Операция с элементом массива
MOV     DI,[DI+4]     ; Загрузить в DI слово из памяти
```

Базовая индексная адресация. В этом режиме ЕА равен сумме содержимого базового регистра BP или BX, индексного регистра SI или DI и необязательного смещения *disp*, находящегося в команде. Базовая индексная адресация является наиболее гибкой, так как две компоненты адреса можно определять и модифицировать при выполнении программы. В этом режиме обеспечивается удобный способ адресации элементов массива, находящегося в стеке, а также доступ к двумерным массивам. В ассемблерных программах применяется обозначение [BREG]DISP [IREG], т. е. как комбинация базовой и индексной адресации. Допускается также обозначение в виде [BREG ± число][IREG ± ± число]. Примеры команд с базовой индексной адресацией:

```
ADD     [BP]ALPHA[SI],AX        ; Прибавить AX к слову
INC     BYTE PTR [BP+3][SI-10] ; Инкремент байта
CMP     [BP]BETA[DI],4000H      ; Сравнить со словом
```

Относительная адресация. В режиме относительной адресации, который не показан в табл. 3.2, ЕА вычисляется как сумма смещения, находящегося в команде, и текущего содержимого программного счетчика PC. При этом значение в PC равно адресу байта после команды с относительной адресацией. В МП К1810 этот режим применяется только в командах условных и безусловных переходов и управления циклами. Смещение *disp* длиной 8 или 16 бит представлено в дополнительном коде и имеет диапазоны —128 — +127 и —32768 — +32767 соответственно. В ассемблерных командах указывается не значение смещения, а метка той команды, которой передается управление. Требуемое значение смещения *disp* автоматически вычисляет программа-ассемблер.

Адресация цепочек. Под цепочкой (строкой) понимается любая последовательность байтов или слов, находящихся в смежных ячейках памяти. При обработке цепочек аппаратно предполагается, что регистр SI адресует байт или слово цепочки-источника (отсюда происходит его название «индекс источника»), а регистр DI — байт или слово цепочки-получателя. В повторяющихся цепочечных операциях МП автоматически корректирует регистры SI и DI (инкремент или декремент в зависимости от состояния флажка DF) по мере перехода к следующим элементам цепо-

чек. Цепочка-источник может находиться в любом сегменте (по умолчанию принимается сегмент данных), а цепочка-получатель — только в дополнительном сегменте данных, базовый адрес которого находится в регистре ES.

Адресация портов ввода-вывода. Для обращения к входным и выходным портам в пространстве ВВ предусмотрены два режима адресации (прямая и косвенная). В прямой адресации номер порта длиной в байт находится в команде, что обеспечивает доступ к портам 0—255. В косвенной адресации номер порта содержится в регистре DX и имеет диапазон от 0 до 65535. В самом МП для ВВ предназначены аккумуляторы AX и AL. Примеры команд ВВ:

```
IN      AL,80H      ; Ввод байта из фиксированного порта
OUT     DX,AX       ; Вывод слова по адресу из DX
IN      AX,DX       ; Ввод слова по адресу из DX
OUT     20H,AL      ; Вывод байта в фиксированный порт
```

Режим адресации и время выполнения команд. Эффективный адрес памяти получается в результате некоторых вычислений, на которые расходуется время. Следовательно, время выполнения команды оказывается зависящим от режима адресации, что показано в табл. 3.3. Задание в команде оператора замены сегмента увеличивает находящиеся в таблице величины еще на два такта синхронизации.

Таблица 3.3. **Время вычисления эффективного адреса**

Адресация	Обозначение	Число тактов
Прямая	*disp*	6
Косвенная регистровая	[BX], [BP], [SI], [DI]	5
Базовая или индексная	[BX, BP, SI, DI]+*disp*	9
Базовая индексная (без смещения)	[BP] [DI], [BX] [SI]	7
	[BP] [SI], [BX] [DI]	8
Базовая индексная (со смещением)	[BP] [DI]+*disp*	11
	[BX] [SI]+*disp*	
	[BP] [SI]+*disp*	
	[BX] [DI]+*disp*	12

3.3. СИСТЕМА КОМАНД МИКРОПРОЦЕССОРА

Система команд МП К1810 состоит из 113 базовых команд, многие из которых допускают различные режимы адресации и, следовательно, порождают множество машинных команд. Будем придерживаться такого же группирования команд, как и для МП К580, и пользоваться следующими условными обозначениями:

reg — общий регистр МП,
ac — аккумулятор AL или AX,
sreg — сегментный регистр,
mem — байт или слово в памяти с любым режимом адресации,
mem/reg — ячейка памяти с любым режимом адресации или общий регистр,
data — непосредственные данные в команде (8 или 16 бит),
disp — смещение в команде (8 или 16 бит),
port — адрес входного или выходного порта,
src — операнд, который не изменяется при выполнении команды (источник),
dst — операнд, который изменяется при выполнении команды (получатель).

Основные форматы команд МП К1810 были приведены на рис. 3.7.

3.3.1. КОМАНДЫ ПЕРЕДАЧ ДАННЫХ

Команды передач данных приведены в табл. 3.4. Они осуществляют передачи регистр — регистр, регистр — память, память —

Таблица 3.4. **Команды передач данных**

Название	Мнемоника	Функция
Пересылка данных	MOV mem/reg_1, mem/reg_2	$mem/reg_1 \leftarrow (mem/reg_2)$
	MOV *mem/reg, data*	*mem/reg*←*data*
	MOV *reg, data*	*reg*←*data*
	MOV *ac, mem*	*ac*←(*mem*)
	MOV *mem, ac*	*mem*←(*ac*)
	MOV *sreg, mem/reg*	*sreg*←(*mem/reg*)
	MOV *mem/reg, sreg*	*mem/reg*←(*sreg*)
Обмен данных	XCHG mem/reg_1, mem/reg_2	$(mem/reg_1) \leftrightarrow (mem/reg_2)$
	XCHG *reg*	(AX) ↔ (*reg*)
Преобразование	XLAT	AL←((BX)+(AL))
Загрузка адреса	LEA *reg, mem*	*reg*←EA
	LDS *reg, mem*	*reg*←(*mem*), DS←(*mem*+2)
	LES *reg, mem*	*reg*←(*mem*), ES←(*mem*+2)
Передача флажков в AH	LAHF	AH←(младший байт флажков)
Передача AH во флажки	SAHF	Младший байт флажков←←(AH)
Включение в стек	PUSH *mem/reg*	SP←(SP)—2, TOS←(*mem/reg*)
	PUSH *reg*	SP←(SP)—2, TOS←(*reg*)
	PUSH *sreg*	SP←(SP)—2, TOS←(*sreg*)
	PUSHF	SP←(SP)—2, TOS←флажки
Извлечение из стека	POP *mem/reg*	*mem/reg*←(TOS), SP←(SP)+2
	POP *reg*	*reg*←(TOS), SP←(SP)+2

Название	Мнемоника	Функция
	POP *sreg*	*sreg*←(TOS), SP←(SP)+2
	POPF	Флажки←(TOS), SP←(SP)+2
Ввод	IN *ac, port*	*ac*←((*port*))
	IN *ac*, DX	*ac*←(*port*(DX))
Вывод	OUT *port, ac*	*port*←(*ac*)
	OUT DX, *ac*	*port*(DX)←(*ac*)

регистр. Наиболее мощной среди них является команда MOV со следующим обобщенным представлением:

```
#    MOV    dst,src          dst <-- (src)
```

Эта команда передает содержимое источника *src* в получатель *dst*, не воздействуя при этом на флажки. Форматы команды MOV показаны на рис. 3.8 (далее машинные форматы команд не приводятся).

В мнемонике MOV mem/reg_1, mem/reg_2 подразумевается, что источником и получателем может быть регистр или ячейка памяти с любым режимом адресации, но адресовать две ячейки памяти в командах МП К1810 нельзя.

Как видно из рис. 3.8, с помощью команды MOV можно осуществить следующие передачи байта или слова: из регистра в ре-

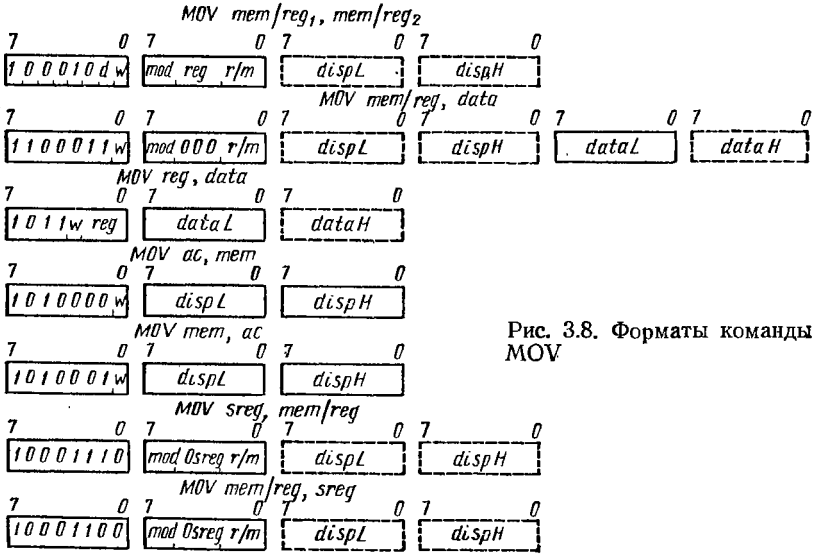


Рис. 3.8. Форматы команды MOV

гистр, из регистра в память и, наоборот, непосредственного операнда в регистр (за исключением сегментных регистров) или память. Имеются более короткие форматы команды MOV, в которых фигурирует аккумулятор *ac*. По существу, одна команда MOV заменяет собой подавляющее большинство команд передач данных в МП К580 (см. табл. 2.2).

Команда MOV *sreg, mem/reg* предназначена для инициализации сегментных регистров. Если, например, в сегментный регистр DS необходимо загрузить A000H, то потребуются команды:

```
MOV     AX,0A000H     ; Инициализировать
MOV     DS,AX         ;      регистр DS на A000H
```

Для инициализации сегментных регистров всегда используется аккумулятор AX, так как команда MOV *ac, data* короче более общей команды MOV *mem/reg, data.*

Команда обмена XCHG со следующим общим описанием

```
XCHG    dst,src          (dst) <--> (src)
```

позволяет обменять содержимое (байты или слова) двух общих регистров, а также любого общего регистра и ячейки памяти. Однако в ней нельзя указывать сегментные регистры.

Безоперандная команда преобразования XLAT заменяет содержимое аккумулятора AL на байт из 256-байтной таблицы, начальный адрес которой находится в регистре BX (в сегменте данных). Исходное содержимое AL служит индексом таблицы и выбираемый из нее байт передается в AL. Эта команда обычно применяется для быстрого преобразования символов из одного кода в другой.

Команды загрузки адреса LEA, LDS и LES предназначены для передачи в регистр(ы) адресов, поэтому их основное применение связано с инициализацией регистров-указателей. При выполнении команды загрузки эффективного адреса LEA *reg, mem* вычисляется EA и его значение передается в указанный общий регистр, например в регистр BX для его инициализации перед командой XLAT. Команды LDS и LES реализуют следующие действия: вычисленный EA преобразуется в физический адрес и адресуемое им слово из памяти загружается в указанный общий регистр *reg,* а следующее слово из памяти передается в регистр DS или ES. Обычно в команде LDS указывается регистр SI, а в команде LES — регистр DI, что согласуется с логикой цепочечных команд.

Команды LAHF и SAHF введены для упрощения совместимости микропроцессоров К580 и К1810. Команда LAHF передает младший байт регистра флажков в регистр AH, а команда SAHF выполняет противоположную передачу. Команда SAHF участвует

также в передаче кода условия из арифметического сопроцессора К1810ВМ87 в регистр флажков МП К1810.

Стековые команды PUSH и POP осуществляют соответствующие операции со стеком. Вершина TOS стека в сегменте стека адресуется регистрами SS и SP. Все стековые команды оперируют только словами и сопровождаются автоматической модификацией SP: при включении в стек производится декремент, а при извлечении из стека — инкремент SP.

Команды ввода IN и вывода OUT передают байт или слово между аккумуляторами AL или AX и адресуемыми входными/выходными портами. Допускаются режимы прямой и косвенной адресации портов.

3.3.2. АРИФМЕТИЧЕСКИЕ КОМАНДЫ

Микропроцессор К1810 имеет достаточно широкий набор арифметических команд, что позволяет применять его в сложных системах обработки данных. Арифметические операции выполняются над целыми числами четырех форматов: двоичные беззнаковые (байты и слова), двоичные знаковые (байты и слова), десятичные упакованные (байты) и десятичные неупакованные (байты). В табл. 3.5 приведены почти все арифметические команды, а подробное рассмотрение команд десятичной арифметики дано в п. 3.4.2. Сложение и вычитание двоичных чисел обоих типов осуществляется одними и теми же командами, а для умножения и деления предусмотрены отдельные команды. В арифметических операциях особенности получающихся результатов фиксируются в шести арифметических флажках, состояния которых (за исключением флажка AF) можно проверить командами условных переходов.

МП К1810 имеет команду ADD собственно сложения и команду ADC сложения с переносом:

```
ADD     dst,src         dst <-- (dst) + (src)
ADC     dst,src         dst <-- (dst) + (src) + CF
```

В качестве *dst* и *src* можно указывать общие регистры и ячейки памяти в любом режиме адресации, а в качестве *src* еще и непосредственные операнды. Следовательно, любой общий регистр (длиной 8 или 16 бит) и любая ячейка памяти могут выполнять функции аккумулятора. Это обстоятельство значительно упрощает программирование вычислительных алгоритмов.

К командам сложения обычно относят одноопернадную команду инкремента INC:

```
INC     dst             dst <-- (dst) + 1
```

Таблица 3.5. **Арифметические команды**

Название	Мнемоника	Функция
Сложение	ADD *mem/reg*$_1$, *mem/reg*$_2$	*mem/reg*$_1$←(*mem/reg*$_1$)+(*mem/reg*$_2$)
	ADD *mem/reg, data*	*mem/reg*←(*mem/reg*)+*data*
	ADD *ac, data*	*ac*←(*ac*)+*data*
Сложение с переносом	ADC *mem/reg*$_1$, *mem/reg*$_2$	*mem/reg*$_1$←(*mem/reg*$_1$)+(*mem/reg*$_2$)+CF
	ADC *mem/reg, data*	*mem/reg*←(*mem/reg*)+*data*+CF
	ADC *ac, data*	*ac*←(*ac*)+*data*+CF
Инкремент	INC *mem/reg*	*mem/reg*←(*mem/reg*)+1
	INC *reg*	*reg*←(*reg*)+1
Вычитание	SUB *mem/reg*$_1$, *mem/reg*$_2$	*mem/reg*$_1$←(*mem/reg*$_1$)—(*mem/reg*$_2$)
	SUB *mem/reg, data*	*mem/reg*←(*mem/reg*)—*data*
	SUB *ac, data*	*ac*←(*ac*)—*data*
Вычитание с заемом	SBB *mem/reg*$_1$, *mem/reg*$_2$	*mem/reg*$_1$←(*mem/reg*$_1$)—(*mem/reg*$_2$)—CF
	SBB *mem/reg, data*	*mem/reg*←(*mem/reg*)—*data*—CF
	SBB *ac, data*	*ac*←(*ac*)—*data*—CF
Декремент	DEC *mem/reg*	*mem/reg*←(*mem/reg*)—1
	DEC *reg*	*reg*←(*reg*)—1
Изменение знака	NEG *mem/reg*	*mem/reg*←0—(*mem/reg*)
Умножение (беззнаковое)	MUL *mem/reg*	8 бит : AX←(AL)×(*mem/reg*) 16 бит : DX : AX←(AX)×(*mem/reg*)
Умножение (знаковое)	IMUL *mem/reg*	
Деление (беззнаковое)	DIV *mem/reg* — 8 бит :	(AH остаток, AL частное)←(AX)/(*mem/reg*)
Деление (знаковое)	IDIV *mem/reg* — 16 бит :	(DX остаток, AX частное)←(DX : AX)/(*mem/reg*)
Преобразование байта в слово	CBW	AH←AL7
Преобразование слова в двойное слово	CWD	DX←AX15
Сравнение	CMP *mem/reg*$_1$, *mem/reg*$_2$	(*mem/reg*$_1$)—(*mem/reg*$_2$)
	CMP *mem/reg, data*	(*mem/reg*)—*data*
	CMP *ac, data*	(*ac*)—*data*

В этой команде операнд *dst*, которым может быть общий регистр или ячейка памяти, считается целым беззнаковым числом и при ее выполнении состояние флажка переноса CF не изменяется.

Команды вычитания:

```
SUB     dst,src         dst <-- (dst) - (src)
SBB     dst,src         dst <-- (dst) - (src) - CF
DEC     dst             dst <-- (dst) - 1
```

отличаются от соответствующих команд сложения только выполняемой операцией. По-прежнему флажки CF и AF становятся флажками заема и устанавливаются в 1, когда уменьшаемое меньше вычитаемого. Команда декремента DEC не модифицирует флажок CF.

К командам вычитания относится также команда NEG изменения знака:

```
NEG     dst             dst <-- 0 - (dst)
```

Если операнд равен нулю, его значение не изменяется. Попытка изменить знак максимального по модулю отрицательного числа (80H или 8000H) не модифицирует операнд, но устанавливает в 1 флажок переполнения OF. При выполнении команды NEG флажок CF всегда устанавливается в 1, кроме случая, когда операнд равен нулю (тогда CF=0).

В МП К1810 имеются две команды умножения:

```
MUL     src             ext:ac <-- (ac) * (src)
IMUL    src
```

предназначенных для умножения двоичных беззнаковых (MUL) и знаковых (IMUL) целых чисел. Обе они выполняют умножение содержимого аккумулятора *ac* на адресуемый операнд *src*, которым может быть регистр или ячейка памяти с любым режимом адресации (но операнд не может быть непосредственным значением). В операции над байтами аккумулятором служит AL, а произведение образуется в регистрах AH и AL, причем AH называется расширением *ext* аккумулятора AL. В операции со словами аккумулятором является AX, а произведение образуется в регистрах DX и AX. Здесь регистр DX выступает расширением *ext* аккумулятора AX. Если в команде MUL старшая половина произведения отличается от нулевой, а в команде IMUL является расширением знака младшей половины, флажки CF и OF устанавливаются в 1; в противном случае CF, OF=0. Состояния остальных арифметических флажков после выполнения команд умножения не определены.

В двух командах деления операндами являются двоичные беззнаковые (DIV) и знаковые (IDIV) целые числа:

```
DIV     src         ac <-- quot ((ext:ac) / (src))
IDIV    src         ext <-- rem ((ext:ac) / (src))
```

Здесь делимым служит содержимое аккумулятора *ac* и его расширения *ext*, делителем — адресуемый операнд (содержимое регистра или ячейки памяти), частное *quot* образуется в аккумуляторе *ac*, а остаток — в его расширении *ext*. Дробное частное усе-

кается до целого. Состояния всех арифметических флажков не определены. Если размер частного превышает длину аккумулятора или если делитель равен нулю, МП генерирует внутреннее прерывание типа 0 (см. далее) и автоматически переходит к соответствующей процедуре прерывания. При этом результат деления не определен.

Особенность команд умножения и деления заключается в том, что время их выполнения зависит от значений операндов. Например, умножение 8-битных беззнаковых чисел (источником *src* служит регистр) может потребовать от 70 до 77 тактов синхронизации, а деление 32-битного делимого на 16-битный делитель (источником *src* является слово в памяти) требует (171—190)+EA тактов синхронизации. Здесь EA — это время на вычисление эффективного адреса (см. табл. 3.3).

Для расширения делимого предназначены две команды преобразования: команда CBW преобразования байта в слово копирует в регистр AH знак числа, находящегося в AL, а команда CWD преобразования слова в двойное слово копирует в регистр DX знак числа, находящегося в регистре AX. Обе команды не изменяют текущих состояний арифметических флажков.

Команда CMP сравнения:

```
CMP     dst,src          (dst) - (src)
```

похожа на команду вычитания SUB, но результат вычитания нигде не сохраняется — команда производит неразрушающее сравнение операндов. Состояния всех арифметических флажков определяются значением получающейся разности.

3.3.3. ЛОГИЧЕСКИЕ КОМАНДЫ И КОМАНДЫ СДВИГОВ

Поразрядные логические операции в МП К1810 представлены конъюнкцией AND, дизъюнкцией OR, сложением по модулю два XOR, проверкой TEST и инверсией NOT. Соответствующие команды имеют следующее общее описание:

```
AND     dst,src          dst <-- (dst) /\ (src)
OR      dst,src          dst <-- (dst) \/ (src)
XOR     dst,src          dst <-- (dst) + (src)
TEST    dst,src                  (dst) /\ (src)

                                  ---
NOT     src              src <-- (src)
```

Возможные способы адресации операндов показаны в табл. 3.6. В качестве *dst* и *src* можно указывать общие регистры и

Таблица 3.6. **Команды логических операций**

Название	Мнемоника	Функция
Конъюнкция	AND mem/reg_1, mem/reg_2 AND *mem/reg, data* AND *ac, data*	$mem/reg_1 \leftarrow (mem/reg_1) \wedge (mem/reg_2)$ $mem/reg \leftarrow (mem/reg) \wedge data$ $ac \leftarrow (ac) \wedge data$
Дизъюнкция	OR mem/reg_1, mem/reg_2 OR *mem/reg, data* OR *ac, data*	$mem/reg_1 \leftarrow (mem/reg_1) \vee (mem/reg_2)$ $mem/reg \leftarrow (mem/reg) \vee data$ $ac \leftarrow (ac) \vee data$
Сложение по модулю 2	XOR mem/reg_1, mem/reg_2 XOR *mem/reg, data* XOR *ac, data*	$mem/reg_1 \leftarrow (mem/reg_1) \oplus (mem/reg_2)$ $mem/reg \leftarrow (mem/reg) \oplus data$ $ac \leftarrow (ac) \oplus data$
Проверка	TEST mem/reg_1, mem/reg_2 TEST *mem/reg, data* TEST *ac, data*	$(mem/reg_1) \wedge (mem/reg_2)$ $(mem/reg) \wedge data$ $(ac) \wedge data$
Инверсия	NOT *mem/reg*	$mem/reg \leftarrow \overline{(mem/reg)}$

ячейки памяти с любым режимом адресации, а в качестве *src* еще и непосредственные данные (за очевидным исключением команды NOT). Операндами могут быть байты и слова.

Команды бинарных операций воздействуют на арифметические флажки следующим образом:

флажки CF и OF переводятся в нулевое состояние;
состояние флажка AF не определено;
состояния флажков SF, ZF и PF зависят от результата;
команда NOT не влияет на состояния флажков.

Команды AND и OR применяются в основном для установки в 0 или 1 тех битов операнда *dst*, которые определяются другим операндом *src*, называемым маской. С помощью команды XOR можно инвертировать отдельные биты операнда *dst* (благодаря тождеству $1 \oplus x = \bar{x}$), сравнивать операнды на абсолютное равенство и перевести регистр в нулевое состояние (пользуясь тождеством $x \oplus x = 0$).

Действия команд сдвигов показаны на рис. 3.9. Поле операнда всех команд имеет вид *mem/reg, count*. Здесь *mem/reg* обычным образом адресует общий регистр или ячейку памяти, содержащие байт или слово, а счетчик *count* определяет число сдвигов. Счетчик может быть указан как константа 1 (статический сдвиг на один бит) или как регистр CL (динамический сдвиг, в котором число сдвигов определяется содержимым CL). С помощью команд сдвигов осуществляются циклические (кольцевые) и «обычные» сдвиги. В циклических сдвигах выдвигаемый бит помещается на место освобождающегося бита (см. первые 4 команды на рис. 3.9). Команды RCL и RCR называются циклическими сдвигами через перенос, так как в кольцо сдвига включен флажок CF. «Обычные» сдвиги, в которых выдвигаемый бит те-

ряется, подразделяются на логические (команды SHL и SHR) и арифметические (команды SAL и SAR) сдвиги. В арифметическом сдвиге вправо знаковый бит не сдвигается, а копируется в соседний справа бит.

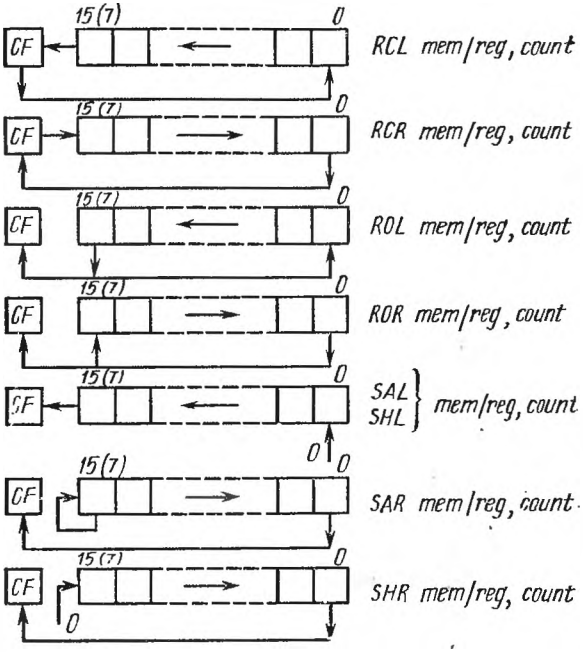


Рис. 3.9. Команды сдвигов микропроцессора К1810ВМ86

При выполнении команд сдвигов флажки модифицируются следующим образом:

флажок CF всегда содержит значение последнего выдвинутого бита;

состояние флажка AF всегда не определено;

в статическом сдвиге OF=1, если знаковый бит операнда изменился, а в динамическом сдвиге состояние OF не определено;

циклические сдвиги воздействуют только на флажки CF и OF;

в «обычных» сдвигах флажки SF, ZF и PF модифицируются в соответствии с полученным результатом.

Время выполнения команд динамических сдвигов зависит от значения счетчика сдвигов, т. е. содержимого регистра CL. Если, например, команда RCL AX, 1 выполняется за два такта синхронизации, то выполнение команды RCL AX, CL требует $8+4\times\times(CL)$ тактов синхронизации.

3.3.4. КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ

В разветвляющихся и циклических программах, а также при организации подпрограмм необходимо нарушать естественный порядок следования команд и передавать управление по адресу перехода, т. е. модифицировать указатели программной памяти — регистры CS и PC. Команды передачи управления, осуществляющие это действие, не изменяют состояний флажков (за исключением команды IRET возврата из прерывания, которая возвращает из стека сохраненные в нем состояния всех флажков).

Сегментная организация памяти определяет две разновидности команд передачи управления. Передача управления в пределах текущего сегмента кода называется *внутрисегментной* (тип NEAR); при этом модифицируется только PC и адрес перехода представлен одним словом. Передача управления за пределы текущего сегмента кода называется *межсегментной* (тип FAR); здесь необходимо модифицировать CS и PC и адрес перехода представлен двумя словами сегмент: смещение (или *seg : off*). Такие команды позволяют передать управление в любую точку всего адресного пространства 1М байт. Отметим, что в ассемблерных программах операндом команд передачи управления служит метка той команды, которой передается управление.

Команды безусловных переходов. МП К1810 имеет пять форматов команд безусловного перехода с одной и той же мнемоникой JMP и одинаковым общим представлением JMP *target*. Тип команды программа-ассемблер выбирают в соответствии с атрибутами операнда *target*.

Двухбайтная команда JMP содержит в первом байте код операции, а во втором — знаковое смещение с диапазоном значений от —128 до +127. При выполнении команды смещение прибавляется к содержимому PC, которое соответствует адресу команды, находящейся после команды JMP. Переход с таким форматом удобен для организации коротких программных циклов. Трехбайтная команда JMP производит такое же действие, как и предыдущая команда, но она имеет 16-битное смещение. Оно попрежнему считается целым знаковым числом, поэтому область перехода расширяется до —32768 — +32767 байт.

Команда JMP типа NEAR осуществляет косвенный безусловный переход. Здесь адресом перехода, загружаемым в PC, служит содержимое общего регистра или слова в памяти, определяемых постбайтом режима адресации. Например, команда JMP BX загружает в PC содержимое регистра BX, а команда JMP NEAR PTR [SI] загружает в PC слово из памяти, которое находится в текущем сегменте данных и имеет смещение в регистре SI.

Последние две команды JMP реализуют прямой и косвенный межсегментные переходы, т. е. имеют тип FAR. Одна из них содержит два слова прямого адреса перехода: первое из них за-

гружается в PC, а второе — в CS. В команде косвенного межсегментного перехода, например JMP FAR PTR [SI], допускается адресовать только память. При ее выполнении слово из адресуемой ячейки загружается в PC, а следующее слово — в CS.

Команды условных переходов. При выполнении команды условного перехода проверяется некоторое условие, представленное текущим состоянием флажков (а в команде JCXZ — содержимым регистра CX) и в зависимости от удовлетворения условия переход осуществляется или нет. Эти команды позволяют проверить состояния всех арифметических флажков, кроме AF, а также ряд комбинаций состояний нескольких флажков. Если условие истинно, управление передается по адресу перехода путем прибавления к PC однобайтного знакового смещения, находящегося во втором байте команды, а если условие ложно, выполняется следующая по порядку команда. Следовательно, все условные переходы являются короткими и их диапазон перехода составляет —128 — +127 байт. Многие команды условных переходов имеют две мнемоники, подчеркивающих содержательный смысл проверяемого условия. Обычно условные переходы применяются после команды сравнения и позволяют проверить все отношения между знаковыми и беззнаковыми числами (см. табл. 3.7). Термины

Таблица 3.7. **Условные переходы после сравнения**

Оператор	Знаковые		Беззнаковые	
$=$	JE, JZ	Равно или нуль	JE, JZ	Равно или нуль
$\neq$	JNE, JNZ	Не равно, не нуль	JNE, JNZ	Не равно, не нуль
$>$	JG, JNLE	Больше (не меньше или равно)	JA, JNBE	Выше (ниже или равно)
$\geqslant$	JGE, JNLE	Больше или равно (не меньше)	JAE, JNB	Выше или равно (не ниже)
$<$	JL, JNGE	Меньше (не больше или равно)	JB, JNAE	Ниже (не выше или равно)
$\leqslant$	JLE, JNG	Меньше или равно	JBE, JNA	Ниже или равно (не выше)

«больше» и «меньше» относятся к знаковым числам, а «выше» и «ниже» — к беззнаковым. Например, число 0BEH «меньше» и «выше» числа 37H.

Кроме команд, указанных в табл. 3.7, имеются еще команды JC и JNC (проверяют флажок CF) JO и JNO (проверяют флажок OF), JP/JPE и JNP/JPO (проверяют флажок PF), JS и JNS (проверяют флажок SF).

Время выполнения команд условных переходов составляет 4

(перехода нет) или 8 (переход происходит) тактов синхронизации.

Команды вызовов подпрограмм. В поле операнда команды CALL вызова подпрограммы находится метка первой команды вызываемой подпрограммы и заключительное действие этой команды заключается в загрузке значения метки (т. е. адреса перехода) в регистр PC (тип NEAR) или в регистры CS и PC (тип FAR). Однако при переходе к подпрограмме необходимо временно запомнить точку, откуда она вызывается, т. е. адрес возврата — адрес команды, находящейся после команды CALL. Завершающая подпрограмму команда RET должна передать управление по запомненному адресу возврата. Удобным «хранилищем» адресов возвратов является стек. Поэтому первое действие команды CALL заключается в том, чтобы включить в стек содержимое PC (тип NEAR) или содержимое регистров CS и PC (тип FAR) с соответствующей модификацией SP.

Команда CALL допускает такие же режимы адресации, что и команда JMP, но короткий вызов (с байтным смещением) отсутствует. Действие наиболее сложной команды косвенного межсегментного вызова показано на рис. 3.10.

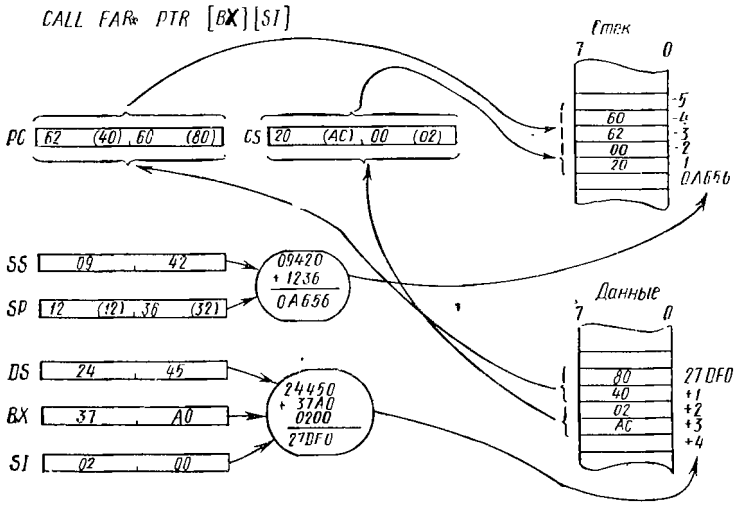


Рис. 3.10. Команда косвенного межсегментного вызова

Команды возвратов из подпрограмм. Каждая подпрограмма должна иметь минимум одну команду RET возврата, передающую управление вызывающей программе. Для этого из стека извлекается адрес возврата, включенный в него командой вызова подпрограммы. В соответствии с типами команды CALL предусмотрены однобайтные команды возвратов двух типов — внутрисег-

ментный и межсегментный. Выполнение команды межсегментного возврата заключается в следующих действиях: слово из вершины стека TOS передается в PC; производится инкремент SP на 2; слово из новой TOS передается в CS; осуществляется заключительный инкремент SP на 2.

Две трехбайтные команды возвратов содержат код операции и два байта данных, интерпретируемых как беззнаковое целое число. Дополнительное действие этих команд заключается в прибавлении к SP находящегося в команде числа после извлечения из стека адреса возврата. Они упрощают возврат из тех подпрограмм, параметры которым передаются в стеке, так как увеличение SP эквивалентно «удалению» параметров из стека.

Команды управления циклами. Три двухбайтных команды управления циклами упрощают организацию программных циклов. В них предполагается, что счетчиком цикла служит регистр CX. Второй байт представляет собой целое знаковое число, которое для перехода к началу цикла прибавляется к содержимому PC. Следовательно, диапазон перехода этих команд составляет —128 — +127 байт. В ассемблерных программах поле операнда команд управления циклами содержит метку первой команды цикла. При выполнении команды LOOP производится декремент регистра CX и, если $(CX) \neq 0$, второй байт команды прибавляется к PC, т. е. происходит переход к началу цикла; в противном случае выполняется следующая по порядку команда. Таким образом, команда цикла LOOP MORE эффективно заменяет две команды: DEC CX и JNZ MORE.

Мнемоники LOOPE и LOOPZ определяют одну и ту же машинную команду, которая производит декремент CX, а затем передает управление в начало цикла, если $(CX) \neq 0$ и $ZF = 1$. В командах LOOPNE и LOOPNZ дополнительным условием перехода к началу цикла (помимо $(CX) \neq 0$) является нулевое состояние флажка ZF.

Команды прерываний. В МП К1810 имеется три команды программных прерываний. Их выполнение напоминает реакцию МП на запросы внешних прерываний по входам INT и NMI, но без циклов шины подтверждения прерывания: в стек последовательно включается содержимое регистров флажков, CS и PC, а затем в соответствии с типом прерывания осуществляется косвенный межсегментный переход через память к нужной процедуре прерывания. Двухбайтная команда INT *type* во втором байте содержит тип прерывания, который программист задает в поле операнда как беззнаковое целое число со значением от 0 до 255. Если поле операнда пустое, формируется однобайтная команда прерывания с типом 3 — это прерывание контрольной точки или контрольного останова. Наконец, команда INTO прерывания при переполнении вызывает прерывание с типом 4, если флажок переполнения OF установлен в 1.

Возврат из процедуры прерывания осуществляется однобайтной командой IRET, извлекающей из стека запомненное при прерывании содержимое регистров PC, CS и флажков.

3.3.5. ЦЕПОЧЕЧНЫЕ КОМАНДЫ

Цепочка — это последовательность байтов или слов, находящихся в смежных ячейках памяти. МП К1810 имеет 5 однобайтных команд, оперирующих одним элементом цепочки (эти команды иногда называются *примитивами*). Однако команде может предшествовать префикс повторения REP, вызывающий повторение действия команды над следующим элементом. Благодаря префиксу повторения цепочки обрабатываются значительно быстрее, чем при организации цикла. Повторение рассчитано на максимальную длину цепочки 64К байт и заканчивается по одному или двум условиям. Аппаратно подразумевается, что цепочка-источник по умолчанию находится в сегменте данных (но возможна замена сегмента) и смещение ее текущего элемента содержится в регистре SI. Цепочка-получатель всегда находится в дополнительном сегменте данных, а смещение ее текущего элемента содержится в регистре DI. Ассемблер игнорирует поле операнда цепочечных команд. При выполнении команды индексные регистры автоматически модифицируются, чтобы адресовать следующие элементы цепочек. Флажок направления DF определяет их автоинкремент (DF=0) или автодекремент (DF=1).

При задании префикса повторения в каждом выполнении команды производится декремент счетчика CX. Когда CX достигает нуля, управление передается следующей по порядку команде.

Префикс повторения. Префикс повторения имеет 5 мнемонических обозначений, которые определяют всего два кода байта префикса. Префикс REP в командах MOVS и STOS означает «повторить до достижения конца цепочки», т. е. до получения (CX)=0. Префиксы REPE (REPZ) в командах CMPS и SCAS для инициирования следующего повторения дополнительно требуют ZF=1. Префиксы REPNE (REPNZ) для повторения команды требуют ZF=0.

Команда MOVS. Команда MOVS передачи цепочки передает

```
MOVS    dst,src          dst <-- (src)
```

байт (слово) из цепочки *src* в цепочку *dst* и соответственно модифицирует регистры SI и DI. Эта команда с префиксом REP осуществляет блоковую передачу память — память. Тип элементов цепочек обычно указывается мнемониками MOVSB (B — байт) и MOVSW (W — слово).

Команда CMPS. Команда сравнения цепочек CMPS имеет следующее общее описание:

```
CMPS    dst,src          (src) - (dst)
```

По результату вычитания устанавливаются все арифметические флажки, сами операнды не изменяются, а регистры SI и DI продвигаются на следующие элементы цепочек. Префикс REPE (REPZ) придает команде смысл «сравнивать до тех пор, пока не достигнут конец цепочек или элементы цепочек будут не равны», а префикс REPNE (REPNZ) — «сравнивать до тех пор, пока не достигнут конец цепочек или элементы цепочек будут равны». Для явного указания типа элементов цепочек допускаются мнемоники CMPSB и CMPSW.

Команда SCAS. Команда сканирования (просмотра) цепочки

```
SCAS    dst              (ac) - (dst)
```

вычитает элемент цепочки *dst*, адресуемый DI, из содержимого аккумулятора AL/AX. Разность определяет состояния арифметических флажков, но сами операнды не изменяются. С префиксом REPE (REPZ) команду SCAS можно использовать для поиска элемента цепочки, отличающегося от заданного значения, а с префиксом REPNE (REPNZ) — равного заданному значению. Тип элементов обычно указывается мнемониками SCASB и SCASW.

Команда LODS. Команда загрузки элемента цепочки в аккумулятор

```
LODS    src              ac <-- (src)
```

передает элемент цепочки, адресуемый SI, в аккумулятор AL/AX и продвигает SI на следующий элемент. Обычно команда LODS с префиксом повторения не применяется, но ее удобно использовать в программных циклах вместо команд MOV *ac*, *src* и INC SI (или DES SI). Допускаются мнемоники LODSB или LODSW, явно указывающие тип элемента.

Команда STOS. Команда запоминания аккумулятора в цепочке

```
STOS    dst              dst <-- (ac)
```

передает байт (слово) из аккумулятора AL/AX в элемент цепочки, адресуемый DI, и продвигает DI на следующий элемент. С префиксом REP можно инициализировать всю цепочку на фиксированное значение, например пробел или нуль. Мнемоники STOSB и STOSW явно определяют тип элемента цепочки.

3.3.6. КОМАНДЫ УПРАВЛЕНИЯ МИКРОПРОЦЕССОРОМ

Часть команд этой группы предназначена для управления состояниями флажков. С их помощью можно установить (STC), сбросить (CLC) и инвертировать (CMC) флажок переноса CF; установить (STD) и сбросить (CLD) флажок направления DF; установить (STI) и сбросить (CLI) флажок прерывания IF.

Команда HLT останова прекращает действия МП и переводит его в состояние останова. Из этого состояния МП выводится сигналом CLR сброса, а также прерываниями на входах NMI и INT.

Команда WAIT ожидания заставляет МП периодически проверять сигнал на входе $\overline{\text{TEST}}$. При появлении активного (низкого) уровня сигнала $\overline{\text{TEST}}$ выполняется команда, находящаяся за командой WAIT.

Команда LOCK, называемая префиксом блокировки, заставляет МП сформировать активный сигнал $\overline{\text{LOCK}}$ на время выполнения команды, находящейся за префиксом LOCK. В мультипроцессорных системах сигнал $\overline{\text{LOCK}}$ подается в арбитр шины, который блокирует запросы доступа к шине других процессоров.

Холостая команда NOP (нет операции) не производит никаких действий и обычно применяется в программных циклах задержки.

Команда ESC переключения на сопроцессор позволяет сопроцессору получать предназначенные для него команды и операнды в процессе работы МП К1810. Сам МП по этой команде не делает ничего, кроме обращения к памяти за операндом и выдачи его на шину данных (если, конечно, обращение к памяти определено в коде операции команды ESC).

3.3.7. ПРОГРАММНАЯ СОВМЕСТИМОСТЬ МИКРОПРОЦЕССОРОВ К580 И К1810

В тех редких случаях, когда необходимо переместить разработанное для МП К580 программное обеспечение на МП К1810, возникает проблема их программной совместимости. Прямая программная совместимость (и даже совместимость вверх) между ними отсутствует. Однако подавляющее большинство команд МП К580 можно однозначно заменить соответствующими командами МП К1810. Но у МП К580 имеются команды, которые для выполнения эквивалентных действий требуют две, а иногда три команды МП К1810, например команды условных вызовов и возвратов, команды DAD, XTHL и др. Это связано в основном с особенностями воздействия этих команд на флажки. Обычно принимаемое соответствие регистров обоих микропроцессоров приведено в табл. 3.8.

При указанном соответствии регистров для любой команды МП К580 нетрудно построить эквивалентную команду или последовательность команд МП К1810. Следовательно, принципиально

Таблица 3.8. **Соответствие регистров**

Регистры МП К580	Регистры МП К1810	Регистры МП К580	Регистры МП К1810
A	AL	D	DH
H	BH	E	DL
L	BL	SP	SP
B	CH	PC	PC
C	CL	PSW	Младший байт регистра флажков и AL

возможно осуществить покомандное преобразование ассемблерных программ для МП К580 в ассемблерные программы для МП К1810. Однако при таком преобразовании не используются более широкие возможности системы команд МП К1810, например команды умножения, деления и др., и оно оказывается малоэффективным. Кроме того, МП К1810 имет больше регистров, которые можно применять в качестве указателей памяти. Чтобы показать хотя бы часть новых возможностей МП К1810, рассмотрим программы 3.1 и 3.2 умножения и деления чисел произвольной длины, функционально эквивалентные программам 2.11 и 2.15 для МП К580. Для наглядности сохраним общие схемы умножения и деления и воспользуемся одинаковыми метками.

Программы для МП К580 имеют длину 70 и 99 команд, а число команд в программах для МП К1810 составляет 41 и 59 соответственно. Таким образом, приближенная оценка показывает, что программы для МП К1810 оказываются в 1,5—2 раза короче (по числу команд) аналогичных программ для МП К580.

3.4. АЛГОРИТМЫ И ПРОГРАММЫ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

Микропроцессор К1810 поддерживает следующие типы численных данных: двоичные целые беззнаковые и знаковые числа длиной 8 и 16 бит, упакованные десятичные числа (байт содержит две десятичные цифры в коде 8421) и неупакованные десятичные числа (байт содержит в младшей тетраде одну десятичную цифру). Система команд обеспечивает выполнение арифметических операций над числами всех типов, за исключением умножения и деления упакованных десятичных чисел. При переходе к другим типам чисел, например многобайтным числам или к числам с плавающей точкой, для арифметических операций приходится разрабатывать

Программа 3.1. *Умножение беззнаковых целых чисел произвольной длины:*

```
        ; Начальный адрес множимого в регистре BX, начальный
        ; адрес множителя в регистре BP, длина сомножителей в AH.
        ; Старшая половина произведения начинается с адреса HIGH,
        ; а младшая находится на месте множителя.
        ; Предполагается, что в регистрах DS и ES находятся одинаковые
        ; значения.
        ;
MPRND:  ; Проверить нулевую длину сомножителей.
        OR      AH,AH           ; Установить флажки
        JZ      EXIT            ; Возврат, если длина равна нулю
        ;
        ; Образовать конечные адреса старшей половины
        ; произведения и множителя.
        MOV     CL,AH           ; Длина в регистре CX
        MOV     CH,0
        ADD     BP,CX           ; В BP конечный адрес множителя
        MOV     SI,HIGH         ; В SI конечный адрес
        ADD     SI,CX           ;    старшей половины произведения
        ;
        ; Образовать счетчик бит.
        MOV     DL,AH           ; Длина в регистре DX
        MOV     DH,0
        MOV     CL,3            ; Умножить ее на 8,
        SHL     DX,CL           ;    счетчик бит в регистре DX
        ;
        ; Очистить старшую половину произведения.
        MOV     CL,AH           ; Счетчик байт в регистре CX
        MOV     DI,HIGH         ; Начальный адрес в регистре DI
        CLD                     ; Продвижение вперед
        MOV     AL,0            ; Значение для инициализации
REP     STOSB                   ; Заполнить нулями
        ;
        ; Подготовка к умножению закончена.
        CLC                     ; Флажок переноса должен быть сброшен
        ; Сдвинуть вправо старшую половину произведения.
LOOPM:  MOV     CL,AH           ; Счетчик байт в регистре CX
SHIFTP: DEC     SI              ; Продвинуть указатель
        RCR     BYTE PTR [SI],1 ; Сдвинуть байт
        LOOP    SHIFTP          ; Продолжать сдвиг
        ;
        ; Сдвинуть вправо множитель.
        MOV     CL,AH           ; Длина в регистре CX
        ADD     SI,CX           ; Восстановить конечный адрес
SHIFTM: DEC     BP              ; Продвинуть указатель
        RCR     BYTE PTR DS:[BP],1 ; Сдвинуть байт
```

```
        LOOP    SHIFTM          ; Продолжать сдвиг
        PUSHF                   ; Сохранить флажки
        ;
        ; Анализ очередного бита множителя во флажке переноса.
        MOV     CL,AH           ; Длина в регистре CX
        ADD     BP,CX           ; Восстановить конечный адрес
        POPF                    ; Восстановить флажки
        JNC     DECR            ; Бит множителя равен нулю
        ;
        ; Прибавить множимое к произведению.
        MOV     DI,HIGH         ; Начальный адрес произведения
        CLC                     ; Сбросить флажок переноса
ADDL:   MOV     AL,[BX]         ; Очередной байт множимого
        ADC     [DI],AC         ; Прибавить к произведению
        INC     BX              ; Продвинуть указатели
        INC     DI
        LOOP    ADDL            ; Продолжать сложение
        MOV     CL,AH           ; Длина в регистре CX
        SUB     BX,CX           ; Восстановить адрес множимого
        ;
        ; Декремент и анализ счетчика бит.
DECR:   DEC     DX              ; Декремент счетчика бит
        JNZ     LOOPM           ; Повторять умножение
EXIT:   RET                     ; Возврат
```

специальные подпрограммы. Как правило, в них сохраняется общая циклическая структура, которая была показана в подпрограммах для МП К580. Благодаря мощной системе команд, большей универсальности регистров и гибким режимам адресации подпрограммы для МП К1810 оказываются компактнее соответствующих подпрограмм для МП К580. На длину подпрограмм влияет выбор рабочих регистров, используемые режимы адресации и применение специальных команд, например, команд управления циклами и операций с цепочками. В операциях со знаковыми числами помогает наличие флажка переполнения OF.

При программировании операций с многобайтными числами необходимо помнить о способе хранения их в памяти: адресом числа является меньший адрес, по которому хранится младший байт. Для обращения к отдельным байтам чисел удобно пользоваться косвенной регистровой адресацией, в которой указателями памяти выступают регистры BP, BX, SI и DI. В качестве счетчика цикла почти всегда привлекается регистр CX, так как именно он участвует в команде управления циклом LOOP. Целесообразно по возможности использовать для обработки многобайтных чисел цепочечные команды LODS и STOS, в которых передача байта (слова) сопровождается автоматической модификацией регистров SI и DI.

Программа 3.2. *Деление беззнаковых чисел произвольной длины:*

```
        ; Начальный адрес делимого в регистре BP, начальный адрес
        ; делителя в регистре BX, длина операндов в регистре AH.
        ; Частное возвращается на месте делимого, начальный адрес
        ; остатка в регистре SI.
        ; Предполагается, что в регистрах DS и ES находятся
        ; одинаковые значения.
DIVRND: ;
        ; Проверить нулевую длину операндов.
        OR      AH,AH           ; Установить флажки
        JZ      OK              ; Возврат, если длина равна нулю
        ;
        ; Образовать счетчик бит в регистре DX.
        MOV     DL,AH           ; Длина в регистре DX
        MOV     DH,0
        MOV     CL,3            ; Умножить ее на 8
        SHL     DX,CL
        INC     DX              ; Счетчик бит в регистре DX
        ;
        ; Очистить буферные области.
        MOV     AL,0            ; Начальное значение
        MOV     CH,0            ; Длина в регистре CX
        MOV     CL,AH
        MOV     DI,BUFF1        ; Начальный адрес текущего буфера
REP     STOSB                   ; Очистить буфер
        MOV     DI,BUFF2        ; Начальный адрес другого буфера
        MOV     CL,AH           ; Длина в регистре CX
REP     STOSB                   ; Очистить буфер
        ;
        ; Инициализировать указатели буферов.
        MOV     SI,BUFF1
        MOV     DI,BUFF2
        ;
        ; Проверить делитель на нуль.
        MOV     CL,AH           ; Счетчик байт в регистре CX
CHECK:  OR      AL,[BX]         ; Объединить по ИЛИ очередной байт
        INC     BX              ; Продвинуть указатель
        LOOP    CHECK           ; Повторять до завершения
        OR      AL,AL           ; Установить флажки
        JZ      ERR             ; Ошибка, делитель равен нулю
        MOV     CL,AH           ; Длина в регистре CX
        SUB     BX,CX           ; Восстановить адрес делителя
        ;
        ; Подготовка к циклу деления закончена.
        ; Флажок переноса сброшен.
LOOPD:  MOV     CL,AH           ; Счетчик байт в регистре CX
        ; Сдвинуть делимое влево с учетом флажка переноса.
```

```
SHIFT1: RCL     BYTE PTR DS:[BP],1  ; Сдвинуть текущий байт
        INC     BP                  ; Продвинуть указатель
        LOOP    SHIFT1              ; Повторять до завершения
        MOV     CL,AH               ; Длина в регистре CX
        SUB     BP,CX               ; Восстановить адрес делимого
         ; Проверить достижение конца цикла деления.
        DEC     DX                  ; Декремент счетчика бит
        JS      OK                  ; Деление закончено
        ;
CONT:   ; Сдвинуть содержимое текущего буфера влево,
        ; передать флажок переноса в младший бит.
SHIFT2: RCL     BYTE PTR [SI],1     ; Сдвинуть байт
        INC     SI                  ; Продвинуть указатель
        LOOP    SHIFT2              ; Повторять до завершения
        MOV     CL,AH               ; Длина в регистре CX
        SUB     SI,CX               ; Восстановить адрес текущего буфера
        ; Произвести вычитание, помещая разность в другой буфер.
        ; Счетчик байт в CX, флажок заема сброшен.
SUBL:   MOV     AL,[SI]             ; Произвести вычитание
        SBB     AL,[BX]             ;     и запоминание разности
        MOV     [DI],AL
        INC     SI                  ; Продвинуть указатели
        INC     DI
        INC     BX
        LOOP    SUBL                ; Повторять до завершения
        ; Во флажке переноса инверсия бита частного.
        PUSHF                       ; Сохранить флажок в стеке
        MOV     CL,AH               ; Длина в регистре CX
        SUB     SI,CX               ; Восстановить начальные адреса
        SUB     DI,CX
        SUB     BX,CX
        POPF                        ; Восстановить флажки
        CMC                         ; Образовать бит частного
        JNC     LOOPD               ; Разность отрицательна
        XCHG    SI,DI               ; Скоммутировать буферы
        JMP     LOOPD               ; Повторять цикл деления
        ;
        ; Ошибка - деление на нуль.
ERR:    STC                         ; Установить флажок переноса
        JMP     EXIT
OK:     ; Нормальный выход.
        CLC                         ; Сбросить флажок переноса
EXIT:   RET                         ; Возврат
```

В настоящем параграфе, построенном аналогично § 2.4, рассмотрены алгоритмы и программы операций над числами в различных форматах.

3.4.1. ОПЕРАЦИИ НАД ДВОИЧНЫМИ ЦЕЛЫМИ ЧИСЛАМИ

Сложение и вычитание. При программировании операций сложения и вычитания многобайтных чисел, как обычно, потребуются два указателя памяти для адресации слагаемых, если, конечно, сумма замещает одно из слагаемых. В качестве указателей рекомендуется максимально использовать регистры SI и DI, а рабочим регистром выбирать аккумулятор AL (для байтов) или AX (для слов).

В программе сложения 3.3 предполагается, что начальные адреса операндов находятся в регистрах SI и DI, длина их передается в регистре CX, а сумма замещает слагаемое, адресуемое регистром DI.

Программа 3.3. *Сложение двоичных целых чисел (байтами):*

```
        ;
        ; Начальные адреса операндов в регистрах SI и DI,
        ; длина (в байтах) в регистре CX.
        ; Сумма замещает операнд, адресуемый регистром DI.
        ;
ADDB:   CLC                 ; Сбросить флажок переноса
ALOOP:  MOV     AL,[SI]     ; Текущий байт первого операнда
        ADC     [DI],AL     ; Прибавить к байту второго операнда
        INC     SI          ; Продвинуть указатели
        INC     DI          ;     операндов
        LOOP    ALOOP       ; Повторять до завершения
        RET                 ; Возврат
```

Аналогично выглядит программа 3.4 сложения многобайтных чисел, если воспользоваться командами сложения для слов. Конечно, в регистре CX должна находиться длина слагаемых в словах. Для ускорения операции следует обеспечить четные начальные адреса операндов (напомним, что слова с четными адресами передаются в одном цикле шины, а слова с нечетными адресами требуют двух циклов шины).

Отметим, что использовать вместо двух команд инкремента INC одну команду ADD SI,2 нельзя, так как она будет сбрасывать флажок переноса CF, состояние которого должно учитываться при сложении следующих слов.

Программа 3.4. *Сложение двоичных целых чисел (словами):*

```
        ; Начальные адреса операндов в регистрах SI и DI,
        ; длина ( в словах) в регистре CX.
        ; Сумма замещает операнд, адресуемый регистром DI.
        ;
ADDW:   CLC                     ; Сбросить флажок переноса
ALOOP:  MOV     AX,[SI]         ; Текущее слово первого операнда
        ADC     [DI],AX         ; Прибавить к слову второго операнда
        INC     SI              ; Продвинуть указатели операндов
        INC     SI              ;     на следующее слово
        INC     DI
        INC     DI
        LOOP    ALOOP           ; Повторять до завершения
        RET                     ; Возврат
```

Применение регистров SI и DI как указателей памяти, а аккумулятора AL/AX как рабочего регистра позволяет реализовать компактные программы с цепочечными командами LODS и STOS. Помимо своих основных функций загрузки и запоминания содержимого аккумулятора эти команды производят модификацию регистров SI и DI на длину элемента цепочки (1 — для байтов и 2 — для слов). Выполнение инкремента или декремента зависит от состояния флажка направления DF. Следует помнить, что команда LODS обращается к текущему сегменту данных (через регистр DS), а команда STOS — к текущему дополнительному сегменту (через регистр ES). Поэтому, если оба операнда находятся в сегменте данных, необходимо обеспечить одинаковое содержимое регистров DS и ES. Для этого потребуются команды MOV AX, DS и MOV ES, AX. Кроме сокращения длины программы команды LODS и STOS уменьшают и время ее выполнения. Вот как трансформируются программы 3.3 и 3.4, когда в них применяются цепочечные команды.

Программа 3.5. *Сложение двоичных целых чисел (байтами):*

```
        ; Начальные адреса операндов в регистрах SI и DI,
        ; длина (в байтах) в регистре CX.
        ; Сумма замещает операнд, адресуемый регистром DI.
        ;
ADDB:   CLC                     ; Сбросить флажок переноса
        CLD                     ; Задать инкремент указателя
ALOOP:  LODSB                   ; Текущий байт первого операнда
        ADC     AL,[DI]         ; Прибавить байт второго операнда
        STOSB                   ; Запомнить байт суммы
        LOOP    ALOOP           ; Повторять до завершения
        RET                     ; Возврат
```

Программа 3.6. *Сложение двоичных целых чисел (словами):*

```
        ; Начальные адреса операндов в регистрах SI и DI,
        ; длина (в словах) в регистре CX.
        ; Сумма замещает операнд, адресуемый регистром DI.
        ;
ADDW:   CLC                 ; Сбросить флажок переноса
        CLD                 ; Задать инкремент указателей
ALOOP:  LODSW               ; Текущее слово первого операнда
        ADC     AX,[DI]     ; Прибавить слово второго операнда
        STOSW               ; Запомнить слово суммы
        LOOP    ALOOP       ; Повторять до завершения
        RET                 ; Возврат
```

Цикл из 7 команд в программе 3.4 имеет длину 10 байт и выполняется за 60 тактов синхронизации, а в программе 3.6 такой цикл состоит из 4 команд, имеет длину 6 байт и выполняется на 54 такта синхронизации.

Способ обнаружения переполнения в программах 3.3—3.6 зависит от форматов операндов. Если слагаемые являются беззнаковыми числами, то о переполнении сигнализирует установленный в 1 флажок переноса CF; при сложении знаковых чисел, представленных в дополнительном коде, переполнение фиксирует установленный в 1 флажок переполнения OF.

Для превращения программ 3.3—3.6 в программы вычитания целых беззнаковых или знаковых чисел необходимо только заменить команду ADC на команду SBB. При вычитании беззнаковых чисел состояние флажка CF=1 означает получение отрицательной разности (она представлена в дополнительном коде), а при вычитании знаковых чисел установленный в 1 флажок OF по-прежнему свидетельствует о переполнении.

Наличие в МП К1810 четырех указателей памяти, находящихся в распоряжении программиста, позволяет очень просто организовать сложение (вычитание) без разрушения одного из операндов. В программе 3.7 слагаемые адресуют регистры SI и BX, а сумма (разность) помещается в область памяти, адресуемую регистром DI.

В этой программе цикл сложения (вычитания — при замене команды ADC на команду SBB) состоит всего из 5 команд. В аналогичной же программе для МП К580 (программа 2.2) цикл состоит из 10 команд.

Программа 3.7. *Сложение двоичных целых чисел:*

```
        ; Адрес первого операнда находится в SI, адрес второго
        ; операнда в BX, адрес суммы в DI. Число байт операндов
        ; в регистре CX. Содержимое регистров DS и ES одно и то же.
        ;
ADDM:   CLC                     ; Сбросить флажок переноса
        CLD                     ; Задать инкремент указателей
ALOOP:  LODSB                   ; Текущий байт первого операнда
        ADC     AL,[BX]         ; Прибавить байт второго операнда
        STOSB                   ; Запомнить байт суммы
        INC     BX              ; Продвинуть указатель
        LOOP    ALOOP           ; Повторять до завершения
        RET                     ; Возврат
```

В заключение отметим, что каждая из программ 2.3 и 2.4 для МП К580, которые осуществляют сложение и вычитание знаковых 16-битных чисел в дополнительном коде, реализуется всего одной командой МП К1810 — ADD и SUB, операнды которых могут находиться в любых общих регистрах. О переполнении сигнализирует установленный в 1 флажок OF.

Умножение. Для умножения двоичных 8- и 16-битных беззнаковых и знаковых чисел в МП К1810 имеются команды MUL и IMUL. Напомним, что они предполагают размещение множимого в аккумуляторе AL при умножении байт или AX при умножении слов, а множитель указывается в команде в любом режиме адресации, за исключением непосредственного. Произведение двойной длины образуется в регистрах AH : AL при умножении байтов и в регистрах DX : AX при умножении слов. Следовательно, программы умножения для МП К580 с соответствующими операндами (программы 2.7—2.10) заменяются всего одной командой MUL или IMUL. Только переход к более длинным сомножителям требует разработки специальных программ. Рассмотрим две таких программы: первая из них предназначена для умножения 32-битных сомножителей с получением 64-битного произведения, а вторая — для умножения двоичных чисел произвольной длины.

Предположим, что 32-битные сомножители в формате беззнаковых целых чисел размещаются в области памяти, показанной на рис. 3.11, а ее начальный адрес содержится в регистре BX. Операция осуществляется четырьмя умножениями 16-битных сомножителей и суммированием частичных произведений с учетом их положения в полном произведении. Полагаем, что первоначально байты области для произведения содержат нули.

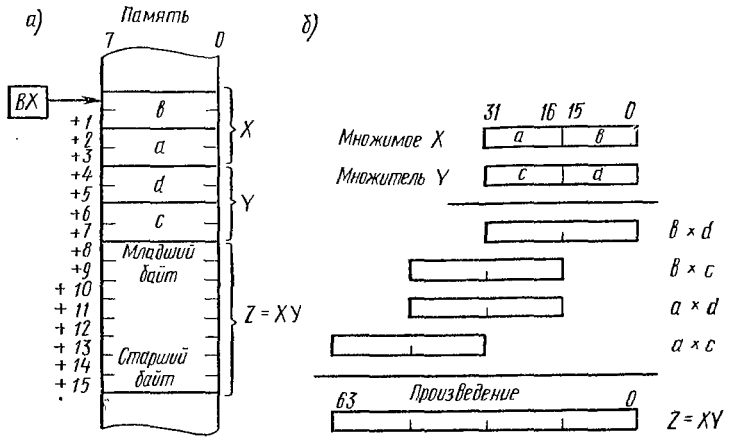


Рис. 3.11. Умножение 32-битных сомножителей

Программа 3.8. *Умножение 32-битных сомножителей:*

```
        ; Регистр BX адресует область памяти, в которой
        ; размещаются сомножители и произведение.
        ;
MULT32: MOV     AX,[BX]      ; Умножить b*d
        MUL     WORD PTR [BX+4]
        MOV     [BX+8],AX    ; Сохранить
        MOV     [BX+10],DX   ;     произведение b*d
        MOV     AX,[BX]      ; Умножить b*c
        MUL     WORD PTR [BX+6]
        ADD     [BX+10],AX   ; Сложить
        ADC     [BX+12],DX   ;     b*d и b*c
        ADC     [BX+14],0    ; Учесть перенос
        MOV     AX,[BX+2]    ; Умножить a*d
        MUL     WORD PTR [BX+4]
        ADD     [BX+10],AX   ; Прибавить
        ADC     [BX+12],DX   ;     к полному произведению
        ADC     [BX+14],0    ; Учесть перенос
        MOV     AX,[BX+2]    ; Умножить a*c
        MUL     WORD PTR [BX+6]
        ADD     [BX+12],AX   ; Прибавить
        ADC     [BX+14],DX   ;     к полному произведению
        RET                  ; Возврат
```

Для умножения многобайтных двоичных чисел в МП К1810 можно воспользоваться общим способом умножения множимого на отдельные биты множителя и суммирования частичных произведений, как это было сделано для МП К580 (программа 2.11). Однако в этом способе не используются возможности команды MUL. Поэтому для умножения многобайтных чисел целесообразно применить способ умножения байтов множимого на отдельные байты множителя с накоплением получающихся произведений, учитывая их правильное положение в полном произведении. Программа 3.9 несколько усложняется необходимостью учета межбайтных переносов в процессе накопления частичных произведений. Вместе с тем благодаря гибкой адресации МП К1810 программа оказывается довольно короткой.

Программа 3.9. *Умножение многобайтных целых беззнаковых чисел:*

```
        ; Абсолютные начальные адреса множимого, множителя
        ; и произведения есть MPCND, MPL и PROD.
        ; Длина сомножителей N байт, произведения 2N байт.
        ; Область для произведения должна содержать нули.
        ;
BMULTN: MOV     CX,N            ; Счетчик байт множителя
        XOR     DI,DI           ; Сбросить индекс множителя
MLOOP:  MOV     DH,N            ; Счетчик байт множимого
        XOR     SI,SI           ; Сбросить индекс множимого
MCBYTE: MOV     BX,DI           ; Индекс частичных произведений
        MOV     AL,MPCND[SI]        ; Текущий байт множимого
        MUL     BYTE PTR MPL[DI]    ; Умножить на байт множителя
        ADD     [BX]PROD[DI],AX     ; Прибавить в произведение
        INC     BX              ; Продвинуть индекс
        INC     BX              ;    частичных произведений
        MOV     DL,N-1          ; Счетчик оставшихся байт
CARRY:  ADC     [BX]PROD[SI],0  ; Учесть возникающие
        DEC     DL              ;    переносы
        JNC     CARRY
        INC     SI              ; Умножить остальные байты
        DEC     DH              ;    множимого
        JNZ     MCBYTE
        INC     DI              ; Умножить на все
        LOOP    MLOOP           ; байты множителя
        RET
```

В программе 3.9 имеются три цикла (см. поясняющий рис. 3.12). Внешний цикл (его начало идентифирует метка MLOOP) связан с выбором для умножения очередного байта множителя. Счетчиком цикла служит регистр CX, а индексирование байтов множите-

ля осуществляется с помощью регистра DI. Следующий вложенный цикл (метка MCBYTE) производит умножение отдельных байтов множимого на выбранный байт множителя. Получающееся при этом в регистре AX частичное произведение (точнее, двухбайтное произведение отдельных байтов множимого и множителя) прибавляется к сумме частичных произведений. Счетчик этого цикла

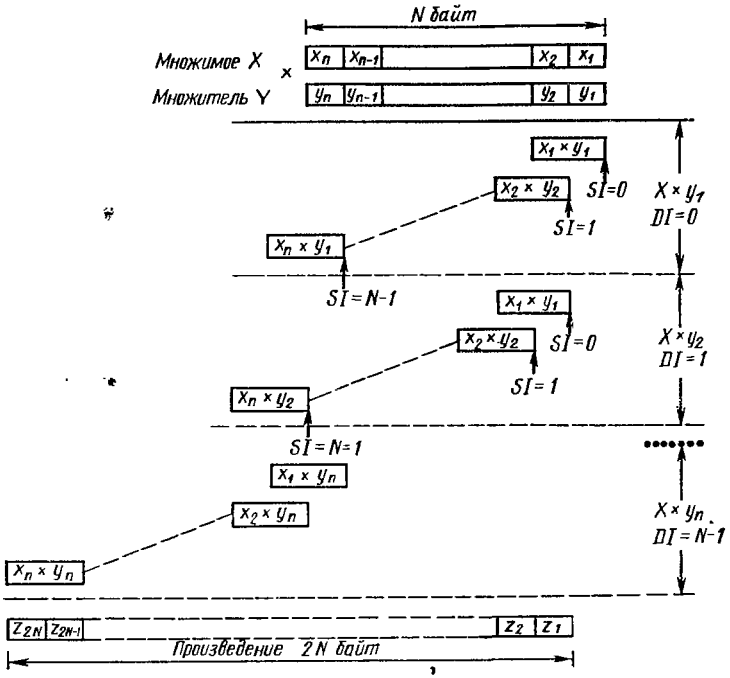


Рис. 3.12. Умножение многобайтных чисел с помощью команды

организован в регистре DH, а регистр SI используется для индексирования байтов множимого. Учет правильного положения произведений отдельных байтов в полном произведении осуществляется с помощью регистра BX. Самый внутренний цикл, начинающийся в строке с меткой CARRY, предназначен для учета межбайтных переносов, возникающих при сложении.

Программа умножения многобайтных чисел состоит всего из 20 команд; аналогичная программа для МП К580 потребовала 70 команд. Воспользуемся таким же приемом, как в программе 3.9, для умножения многоразрядных неупакованных десятичных чисел (программа 3.15).

Деление. Команды деления DIV и IDIV, операндами которых

являются целые знаковые и беззнаковые числа различной длины, закрывают почти все практические потребности в операции деления. Если делимое и делитель имеют одинаковую длину, то перед командой деления потребуется расширять делимое с помощью команд CBW и CWD. В тех случаях, когда длина операндов не удовлетворяет требованиям команд DIV и IDIV, приходится разрабатывать обычные циклические программы, которые были достаточно подробно рассмотрены в § 2.4. Кроме того, один из вариантов программы деления целых чисел произвольной длины был приведен в § 3.3 (программа 3.2).

3.4.2. ОПЕРАЦИИ С ДЕСЯТИЧНЫМИ ЧИСЛАМИ

Система команд МП К1810 ориентирована на оба формата десятичных чисел: упакованный и неупакованный. В обоих форматах многоразрядные десятичные числа представляются последовательностями байтов. Команды десятичной арифметики ориентированы только на обработку байтов, причем основным рабочим регистром во всех десятичных операциях является регистр AL, эквивалентный аккумулятору А в МП К580. Для поддержки десятичной арифметики предусмотрено несколько команд коррекции, благодаря которым относительно легко реализуются сложение и вычитание упакованных десятичных чисел и все арифметические операции над неупакованными десятичными числами. Далее ради простоты предполагаются беззнаковые операнды.

Операции с упакованными десятичными числами. Рассмотрим вначале, как обрабатываются упакованные десятичные числа, учитывая, что здесь много общего с операциями в МП К580.

Сложение. Сложение упакованных десятичных чисел производится в два этапа:

байты операндов суммируются в регистре AL как двоичные числа одной из команд двоичного сложения,

полученная в регистре AL сумма корректируется командой DAA (теперь эта мнемоника означает «десятичная коррекция для сложения»); после нее в регистре AL образуется правильное упакованное представление суммы, а флажок переноса CF показывает десятичный перенос.

Действия команд DAA в МП К580 и К1810 одни и те же, поэтому программа 3.10 сложения многоразрядных десятичных чисел для МП К1810 практически не отличается от соответствующей программы для МП К580 (см. программу 2.16). Здесь имеется больше режимов адресации памяти, а для управления циклом обычно применяется команда LOOP. Предположим, что начальные адреса слагаемых находятся в регистрах SI и DI, а регистр CX содержит длину операндов в байтах. Пусть сумма замещает операнд, адресуемый регистром DI.

Программа 3.10. *Сложение упакованных десятичных чисел произвольной длины:*

```
        ; Начальные адреса операндов в регистрах SI и DI,
        ; длина (в байтах) в регистре CX.
        ; Сумма замещает операнд, адресуемый регистром DI.
        ;
ADDPCK: CLC                 ; Сбросить флажок переноса
ALOOP:  MOV     AL,[SI]     ; Текущий байт первого операнда
        ADC     AL,[DI]     ; Прибавить байт второго операнда
        DAA                 ; Скорректировать сумму
        MOV     [DI],AL     ; Запомнить байт суммы
        INC     SI          ; Продвинуть указатели
        INC     DI
        LOOP    ALOOP       ; Повторять до завершения
        RET                 ; Возврат
```

После возврата из подпрограммы о переполнении сигнализирует установленный в 1 флажок CF.

Адресация операндов с помощью регистров SI и DI в циклических программах с привлечением регистра AL позволяет применить цепочечные команды LODS и STOS. Конечно, при этом в сегментных регистрах DS и ES должны находиться одни и те же значения. Кроме того, состояние флажка направления DF должно учитывать направление обработки отдельных байтов операндов. Обычно оно соответствует направлению от меньших адресов к большим, но в программе деления требуется обратное направление. Программы с использованием цепочечных команд становятся короче и выполняются быстрее. Например, программа 3.10 приобретает следующий вид.

Программа 3.11. *Сложение упакованных десятичных чисел произвольной длины:*

```
        ; Начальные адреса операндов в регистрах SI и DI,
        ; длина операндов (в байтах) в регистре CX.
        ; Сумма замещает операнд, адресуемый регистром DI.
        ; Содержимое регистров DS и ES должно быть одним и тем же.
        ;
ADDPCK: CLC                 ; Сбросить флажок переноса
        CLD                 ; Движение от младшего байта
ALOOP:  LODSB               ; Текущий байт первого операнда
        ADC     AL,[DI]     ; Прибавить байт второго операнда
        DAA                 ; Скорректировать сумму
        STOSB               ; Запомнить байт суммы
        LOOP    ALOOP       ; Повторять до завершения
        RET                 ; Возврат
```

Вычитание. Операция вычитания упакованных десятичных чисел реализуется аналогично сложению благодаря наличию специальной команды DAS десятичной коррекции для вычитания. Эта

команда помещается после одной из команд двоичного вычитания и образует в аккумуляторе AL правильную десятичную разность, причем флажок CF показывает заем из соседнего старшего десятичного разряда.

Действия команды DAS заключаются в следующем:

если AF=1 или младшая тетрада регистра AL содержит запрещенную комбинацию, из содержимого AL вычитается 06H и флажок AF устанавливается в 1;

если CF=1 или старшая тетрада регистра AL содержит запрещенную комбинацию, из содержимого AL вычитается 60H и флажок CF устанавливается в 1.

Необходимо отчетливо представлять несколько непривычные действия команды DAS. Когда МП выполняет команду вычитания, он фактически прибавляет к уменьшаемому дополнительный код вычитаемого. При этом флажки AF и CF устанавливаются в 1 тогда, когда при фактическом сложении значения переносов равны нулю. Приведем несколько примеров.

Пример 3.1. Десятичное вычитание

Уменьшаемое	93		86		25		54	
	1001	0011	1000	0110	0010	0101	0101	0100
Вычитаемое	21		07		84		96	
	0010	0001	0000	0111	1000	0100	1001	0110
Двоичный дополнительный код вычитаемого	1101	1111	1111	1001	0111	1100	0110	1010
Сложение	1001	0011	1000	0110	0010	0101	0101	0100
	+ 1101	+ 1111	+ 1111	+ 1001	+ 0111	+ 1100	+ 0110	+ 1010
	0111	0010	1110	1111	1010	0001	1011	1110
Состояния флажков	CF=0, AF=0		CF=0, AF=1		CF=1, AF=0		CF=1, AF=1	
Коррекция командой DAS	Нет		0111	1111	1010	0001	1011	1110
			− 0000	− 0110	− 0110	− 0000	− 0110	− 0110
	0111	0010	0111	1001	0100	0001	0101	1000
Состояние CF	0		0		1		1	
	Заема нет		Заема нет		Есть заем		Есть заем	

В десятичной системе счисления примеры имеют вид:

39	86	25	54
− 21	− 07	− 84	− 96
72	79	41	58
		Заем=1	Заем=1

Здесь значение заема 1 необходимо учитывать как —100, поэтому фактический результат, например, вычитания (25—84) равен (—100+41) =—59. Можно также полагать, что отрицательная разность представлена в дополнительном коде.

Программа 3.12. *Вычитание упакованных десятичных чисел произвольной длины:*

```
        ; Начальные адреса операндов в регистрах SI и DI,
        ; длина (в байтах) в регистре CX.
        ; Разность замещает уменьшаемое, адресуемое регистром SI.
        ;
SUBPCK: CLC                 ; Сбросить флажок переноса
SLOOP:  MOV     AL,[SI]     ; Текущий байт уменьшаемого
        SBB     AL,[DI]     ; Вычесть байт вычитаемого
        DAS                 ; Скорректировать разность
        MOV     [SI],AL     ; Запомнить байт разности
        INC     SI          ; Продвинуть указатели
        INC     DI
        LOOP    SLOOP       ; Повторять до завершения
        RET                 ; Возврат
```

Если после возврата из подпрограммы флажок CF установлен в 1, то разность является отрицательной и представлена в десятичном дополнительном коде. Целесообразно сравнить программу 3.12 с аналогичной программой для МП К580 (программа 2.17) и убедиться в удобстве и простоте использования команды DAS.

Умножение и деление. Команды коррекции для умножения и деления упакованных десятичных чисел в МП К1810 отсутствуют. Следовательно, все трудности программирования этих операций для МП К580, о которых говорилось в гл. 2, распространяются и на программирование для МП К1810.

Операции с неупакованными десятичными числами. Напомним, что в этом формате байт содержит одну десятичную цифру (в младшей тетраде), а старшая тетрада содержит либо 0011 (символьный код), либо нули. Первая комбинация 0011 допустима в операциях сложения и вычитания, но в операциях умножения и деления старшие тетрады операндов должны содержать нули.

Сложение. Неупакованные десятичные числа, как и упакованные, складываются в два приема: сначала байты операндов суммируются как двоичные числа командами ADD или ADC с получением в регистре AL промежуточного результата, а затем команда коррекции для сложения AAA преобразует промежуточный результат в неупакованный формат. Действия команды AAA, как и других команд коррекции для неупакованного формата, построены с учетом простоты выполнения операций над многоразрядными числами. Коррекция для сложения включает в себя следующие шаги:

1) если младшая тетрада регистра AL содержит допустимую комбинацию и AF=0, перейти к шагу 3;

2) если младшая тетрада регистра AL содержит запрещенную комбинацию или AF=1, то необходимо прибавить 06H к содержимому регистра AL, прибавить единицу к содержимому регистра AH и установить AF=1;

3) сбросить старшую тетраду регистра AL;

4) установить флажок CF в то же состояние, в каком находится флажок AF.

Рассмотрим пример выполнения команды AAA. Пусть в регистре AX находится число 0535H, а в регистре BL — число 39H. Тогда после команды ADD AL, BL в регистре AL будет получено число 6EH и флажок AF=0. Следовательно, команда AAA прибавит 06H к содержимому регистра AL с получением в нем 74H, произведет инкремент регистра AH (в нем будет число 06H), установит флажок AF в 1, затем сбросит старшую тетраду регистра AL (в нем образуется 04H) и передаст состояние флажка AF во флажок CF (CF=1). Окончательный результат: (AX)=0604H и CF=1, что соответствует сложению 5+9=14.

Отметим, что после выполнения команды AAA содержательный смысл имеют только состояния флажков AF и CF, а состояния остальных арифметических флажков не определены.

Остановимся на сложении многоразрядных чисел. Обозначим $a_{n-1}a_{n-2}\ldots a_1a_0$ первое слагаемое, $b_{n-1}b_{n-2}\ldots b_1b_0$ второе слагаемое и $c_{n-1}c_{n-2}\ldots c_1c_0$ их сумму. Тогда алгоритм сложения $a+b$ состоит из следующих этапов:

1) сбросить флажок переноса CF;

2) повторить следующий цикл n раз ($i=0, 1, \ldots, n-1$ — переменная цикла):

— загрузить a_i в регистр AL;

— прибавить к AL цифру b_i, пользуясь командой ADC;

— скорректировать результат в регистре AL командой AAA;

— передать содержимое регистра AL в c_i.

Программа 3.13. *Сложение неупакованных десятичных чисел произвольной длины:*

```
        ; Начальные адреса операндов в регистрах SI и DI,
        ; длина (в байтах) в регистре CX. Коды цифр 30H - 39H.
        ; Сумма вмещает операнд, адресуемый регистром SI.
        ;
AUNPCK: CLC                 ; Сбросить флажок переноса
ALOOP:  MOV     AL,[SI]     ; Текущий байт первого операнда
        ADC     AL,[DI]     ; Прибавить байт второго операнда
        AAA                 ; Скорректировать сумму
        LAHF                ; Сохранить флажок переноса
        OR      AL,30H      ; Образовать код цифры
        SAHF                ; Восстановить флажок переноса
        MOV     [SI],AL     ; Запомнить байт суммы
        INC     SI          ; Продвинуть указатели
        INC     DI
        LOOP    ALOOP       ; Повторять до завершения
        RET                 ; Возврат
```

В приведенной выше программе 3.13 предполагается, что начальные адреса слагаемых находятся в регистрах SI и DI, длина операндов в регистре CX, а сумма замещает первое слагаемое. Операнды представлены в символьном коде (цифры 30H—39H), сумма имеет такой же формат. Отметим наличие в программе команд LAHF и SAHF. Введение их объясняется тем, что для образования символьного кода цифры суммы требуется команда OR, сбрасывающая в 0 флажок CF. Если в байтах результата допускаются нулевые старшие тетрады, команды LAHF и SAHF можно убрать. После возврата из подпрограммы о переполнении сигнализирует установленный в 1 флажок CF.

Вычитание. Команда коррекции для вычитания AAS формирует в регистре AL правильное представление двоичной разности байтов операндов, полученной командами SUB или SBB. Команда AAS действует следующим образом:

1) если младшая тетрада регистра AL содержит допустимую комбинацию и AF=0, перейти к шагу 3;

2) если младшая тетрада регистра AL содержит запрещенную комбинацию или AF=1, необходимо вычесть 06H из содержимого регистра AL, вычесть 1 из содержимого регистра AH и установить AF=1;

3) сбросить старшую тетраду регистра AL;

4) установить флажок CF в такое же состояние, в каком находится флажок AF.

Поясним действия команды AAS на примере. Пусть (AX)= =0432H и выполняется команда SUB AL, 35H. В регистре AL образуется разность OFDH и AF=1, а команда AAS образует в регистре AL правильный результат 07H и флажок CF=1 показывает заем. Кроме того, из содержимого регистра вычитается 1 и он будет содержать 03H. Команда AAS, как и команда AAA, воздействует только на флажки AF и CF.

Для операции вычитания многоразрядных чисел предположим, что $a_{n-1}a_{n-2}\ldots a_1a_0$ — уменьшаемое, $b_{n-1}b_{n-2}\ldots b_1b_0$ — вычитаемое, а $c_{n-1}c_{n-2}\ldots c_1c_0$ — разность. Алгоритм вычитания включает в себя такие шаги:

1) сбросить флажок CF;

2) выполнить цикл n раз (i=0, 1, ..., n—1 — переменная цикла);
   — загрузить a_i в регистр AL;
   — вычесть из AL цифру b_i, пользуясь командой SBB;
   — скорректировать результат в AL командой AAS;
   — передать содержимое регистра AL в c_i.

Этот алгоритм реализует программа 3.14 (ради простоты предполагается, что старшие тетрады байтов операндов содержат нули; в таком же формате будет представлена и разность).

Программа 3.14. *Вычитание неупакованных десятичных чисел произвольной длины:*

```
        ; Начальный адрес уменьшаемого в регистре SI,
        ; начальный адрес вычитаемого в регистре DI,
        ; длина операндов (в байтах) в регистре CX.
        ; Разность замещает уменьшаемое. Цифры операндов 00 - 09.
        ;
SUNPCK: CLC                     ; Сбросить флажок переноса
SLOOP:  MOV     AL,[SI]         ; Текущий байт уменьшаемого
        SBB     AL,[DI]         ; Вычесть байт уменьшаемого
        AAS                     ; Скорректировать разность
        MOV     [SI],AL         ; Запомнить байт разности
        INC     SI              ; Продвинуть указатели
        INC     DI
        LOOP    SLOOP           ; Повторять до завершения
        RET                     ; Возврат
```

Если после возврата из подпрограммы флажок CF установлен в 1, то получена отрицательная разность.

Умножение. Умножение неупакованных десятичных чисел, как сложение и вычитание, выполняется в два этапа:

умножение одноразрядных сомножителей (обязательно с нулевыми старшими тетрадами) командой MUL, которая формирует в регистре AL двоичное произведение;

команда коррекции для умножения AAM преобразует полученный результат в двухбайтное произведение, находящееся в регистрах AH (старший разряд) и AL (младший разряд).

Команда AAM осуществляет деление содержимого регистра AL на десять (0H) и помещает частное в регистр AH, а остаток в регистр AL. Состояния флажков SF, ZF и PF зависят от содержимого регистра AL, а состояния флажков OF, AZ и CF не определены.

Пусть, например, регистр AL содержит 07H, а регистр BL—09H. Тогда после выполнения команд MUL BL и AAM в регистре AH будет образовано число 06H, а в регистре AL—03H. Они соответствуют правильному результату $7\times9=63$.

С помощью команды AAM реализуется умножение многоразрядного множимого $a_{n-1}a_{n-2}\ldots a_1a_0$ на одноразрядный множитель с получением произведения $c_nc_{n-1}\ldots c_1c_0$. Алгоритм умножения имеет следующий вид (в предположении, что старшие тетрады байт множимого и множителя ненулевые):

1) сбросить старшую тетраду множителя b;
2) сбросить цифру c_0;
3) повторить следующий цикл n раз ($i=0, 1, \ldots, n-1$ — переменная цикла):

— сбросить старшую тетраду a_i;

— передать a_i в регистр AL;
— умножить командой MUL содержимое AL на b;
— скорректировать произведение командой AAM;
— прибавить к содержимому AL значение c_i;
— скорректировать сумму командой AAA;
— передать содержимое регистра AL в c_i;
— передать содержимое регистра AH в c_{i+1}.

Действия этого алгоритма реализует программа 3.15.

Программа 3.15. *Умножение многоразрядного множимого на одноразрядный множитель:*

```
        ; Начальный адрес множимого в регистре SI,
        ; множитель в регистре DL, длина множимого в регистре CX,
        ; Произведение помещается в область памяти,
        ; начальный адрес которой находится в регистре DI.
        ;
MUNPCK: AND     DL,0FH          ; Подготовить множитель
        MOV     [DI],0          ; Очистить младшую цифру произведения
MLOOP:  MOV     AL,[SI]         ; Текущий байт множимого
        INC     SI              ; Продвинуть указатель множимого
        AND     AL,0FH          ; Сбросить старшую тетраду
        MUL     DL              ; Осуществить умножение
        AAM                     ; Скорректировать произведение
        ADD     AL,[DI]         ; Прибавить к произведению из памяти
        AAA                     ; Скорректировать сумму
        MOV     [DI],AL         ; Запомнить байт произведения
        INC     DI              ; Продвинуть указатель произведения
        MOV     [DI],AH         ; Учесть старшую цифру произведения
        LOOP    MLOOP           ; Повторять до завершения
        RET                     ; Возврат
```

Умножение многобайтных сомножителей усложняется необходимостью учета межразрядных переносов в процессе суммирования произведения цифр множимого и множителя с суммой частичных произведений. Суммирование должно производиться с учетом положения двухразрядного частичного произведения в накапливаемом полном произведении. Программирование операции умножения несколько упрощает гибкая адресация памяти в МП К1810. Принцип работы приводимой ниже программы 3.16 для трехбайтных сомножителей иллюстрируется на рис. 3.13.

Деление. Деление неупакованных десятичных чисел отличается от предыдущих операций тем, что необходимая коррекция делимого производится до собственно деления. В команде коррекции для деления AAD предполагается, что в регистрах AH и AL находится двухразрядное делимое (AH — цифра десятков, AL — цифра единиц), причем старшие тетрады обоих регистров нулевые. Команда ADD выполняет следующие действия:

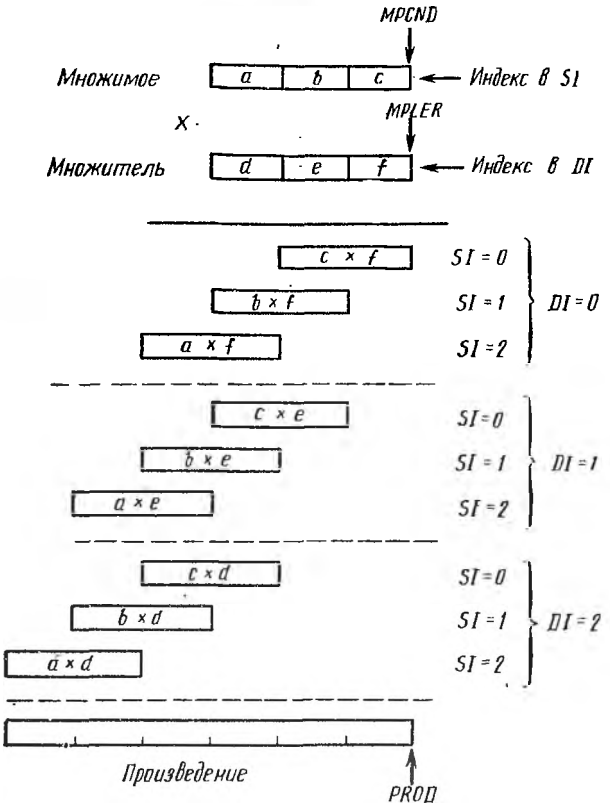


Рис. 3.13. Умножение многоразрядных неупакованных десятичных чисел

1) содержимое регистра AH умножается на десять (0AH);

2) полученный результат прибавляется к содержимому регистра AL (т. е. в AL образуется двоичный код делимого);

3) регистр AH сбрасывается.

Состояния флажков SF, ZF и PF будут зависеть от результата в регистре AL, а состояния флажков OF, AF и CF не определены. Если, например, (AX) = 0604H, то после выполнения команды AAD в регистре AX будет образовано число 0040H, т. е. десятичное число 64 в двоичном коде.

После коррекции полученный в регистре AX двоичный код делимого делится на одноразрядный делитель командой двоичного деления DIV.

Программа 3.16. *Умножение многоразрядных неупакованных десятичных чисел:*

```
        ; Начальные адреса множимого, множителя
        ; и произведения есть MPCND, MPLER и PROD
        ; Длина в байтах сомножителей N, произведения 2N.
        ; Область для произведения предполагается очищенной.
        ; Старшие тетрады байт сомножителей и произведения нулевые.
        ;
MNUNPK: MOV   CX,N          ; Образовать счетчик цифр множителя
        XOR   DI,DI         ; Сбросить индекс цифр множителя
MLOOP:  MOV   DH,N          ; Счетчик цифр множимого
        XOR   SI,SI         ; Сбросить индекс множимого
MCNDIG: MOV   BX,DI         ; Еще один индекс множимого
        MOV   AL,MPCND[SI]      ; Текущая цифра множимого
        MUL   MPLER[DI]         ; Умножить на цифру множителя
        AAM                 ; Скорректировать произведение
        ADD   AL,[BX]PROD[DI]   ; Прибавить двухразрядное
        AAA                     ;     произведение к сумме
        MOV   [BX],PROD[DI],AL  ;     частичных произведений
        INC   BX
        MOV   AL,AH
        ADC   AL,[BX]PROD[SI]
        AAA
        MOV   [BX]PROD[DI],AL
        INC   BX
        MOV   DL,N-1        ; Образовать счетчик оставшихся байтов
CARRY:  MOV   AL,[BX]PROD[SI]   ; Учесть возникшие переносы
        ADC   0                 ;     в сумме частичных
        AAA                     ;     произведений
        MOV   [BX]PROD[SI],AL
        INC   BX
        DEC   DL
        JNZ   CARRY
        INC   SI            ; Умножить остальные
        DEC   DH            ;     цифры множимого
        JNZ   MCNDIG
        INC   DI            ; Продвинуть указатель множителя
        LOOP  MLOOP         ; Повторять до завершения
        RET                 ; Возврат
```

С помощью команды AAD легко осуществляется деление многоразрядного делимого на одноразрядный делитель. Обозначим через $a_{n-1}a_{n-2}\ldots a_1a_0$ делимое, b — одноразрядный делитель и $c_{n-1}c_{n-2}\ldots c_1c_0$ — частное. Деление производится по следующему алгоритму (в предположении, ради общности, что делимое и делитель представлены в символьном неупакованном формате, т. е. старшие тетрады их байтов ненулевые):

1) сбросить старшую тетраду b;
2) сбросить регистр AH;
3) выполнить следующий цикл n раз ($i=n-1, n-2, ..., 1, 0$ — переменная цикла):
— сбросить старшую тетраду a_i;
— передать a_i в регистр AL;
— скорректировать содержимое AX командой AAD;
— разделить содержимое регистра AL на b с помощью команды DIV;
— передать содержимое регистра AL в c_i.

Действия алгоритма реализуются программой 3.17.

Программа 3.17. *Деление многоразрядного делимого на однозразрядный делитель:*

```
        ; Начальный адрес делимого в регистре SI,
        ; делитель находится в регистре DL, длина
        ; делимого в регистре CX. Частное помещается
        ; в область памяти, начальный адрес которой в DI.
        ; Регистры SI и DI содержат адреса младших байтов
        ;
DUNPCK: AND     DL,0FH      ; Подготовить делитель
        XOR     AH,AH       ; Сбросить регистр AH
        ADD     SI,CX       ; Образовать в регистре SI адрес
        DEC     SI          ;    старшего разряда делимого
        ADD     DI,CX       ; Образовать в регистре DI адрес
        DEC     DI          ;    старшего разряда частного
DLOOP:  MOV     AL,[SI]     ; Текущий байт делимого
        DEC     SI          ; Продвинуть указатель делимого
        AND     AL,0FH      ; Подготовить делимое
        AAD                 ; Скорректировть для деления
        DIV     DL          ; Разделить на делитель
        OR      AL,30H      ; Образовать код цифры частного
        MOV     [DI],AL     ; Запомнить цифру частного
        DEC     DI          ; Продвинуть указатель частного
        LOOP    DLOOP       ; Повторять до завершения
        RET                 ; Возврат
```

В этой программе приходится учитывать, что частное получается, начиная со старших разрядов. Поэтому сначала в регистрах SI и DI образуются адреса старших разрядов делимого и частного.

Алгоритм и программа деления с многоразрядными делимым и делителем оказываются довольно громоздкими, так как в них невозможно применить команду AAD.

3.4.3. ОПЕРАЦИИ НАД ЧИСЛАМИ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Для операций над числами с плавающей точкой приходится разрабатывать специальные подпрограммы. Расширенные возмож-

Программа 3.18. *Сложение чисел с плавающей точкой:*

```
        ; Первый операнд X в регистрах BX:SI, второй Y в регистрах DX:DI,
        ; сумма возвращается в регистрах BX:SI.
        ; При переполнении флажок CF устанавливается в 1.
        ;
ADDF:   ;
        ; Сравнить знаки операндов и определить операцию.
        MOV     AL,BH           ; Сравнить
        XOR     AL,DH           ;    знаки операндов
        JNS     ADDF1           ; Знаки операндов одинаковы
        XOR     DH,80H          ; Знаки различны,
        JMP     SUBF            ;    перейти к вычитанию
        ;
        ; Проверить операнды на нуль.
ADDF1:  MOV     AX,DX           ; Проверить на нуль
        OR      AX,DI           ;    второй операнд
        JZ      ADDF8           ; Результат в BX:SI
        MOV     AX,BX           ; Проверить на нуль
        OR      AX,SI           ;    первый операнд
        JNZ     ADDF2           ; Оба операнда ненулевые
        XCHG    BX,DX           ; Первый операнд равен нулю,
        XCHG    SI,DI           ;    сумма равна
        JMP     ADDF8           ;    второму операнду
        ;
        ; Оба операнда ненулевые, можно складывать.
ADDF2:  MOV     AH,BH           ; Сохранить общий знак в AH
        SHL     BX,1            ; Восстановить скрытый бит
        STC                     ;    мантиссы первого операнда
        RCR     BL,1
        SHL     DX,1            ; Восстановить скрытый бит
        STC                     ;    мантиссы второго операнда
        RCR     DL,1
        ;
        ; Сравнить порядки, образовать разность порядков.
        CMP     BH,DH           ; Сравнить порядки
        JNC     ADDF3           ; Порядок числа в BX:SI больше
        XCHG    BX,DX           ; Передать большее число
        XCHG    SI,DI           ;    в BX:SI
ADDF3:  SUB     BH,DH           ; Образовать разность порядков
        JZ      ADDF6           ; Порядки одинаковы
        CPI     BH,24           ; Сравнить  разность порядков с 24
        JC      ADDF5           ; Разность порядков меньше 24
        JMP     ADDF7           ; Результат в BX:SI
        ;
        ; Необходимо сдвигать вправо мантиссу меньшего числа в DL:DI.
ADDF5:  SHR     DL,1            ; Сдвинуть мантиссу
        RCR     DI,1
```

```
        INC     DH              ; Увеличить меньший порядок
        DEC     BH              ; Декремент разности порядков
        JNZ     ADDF5           ; Повторять сдвиг
        ;
        ; Можно складывать мантиссы. В регистре DH общий порядок.
ADDF6:  ADD     SI,DI           ; Сложить младшие части мантисс
        ADC     BL,DL           ; Сложить старшие части мантисс
        JNC     ADDF7           ; Нарушения нормализации влево нет
        INC     DH              ; Скорректировать порядок
        CPI     DH,255          ; Проверить переполнение
        STC
        JZ      ADDF8           ; Возникло переполнение
        RCR     BL,1            ; Сдвинуть
        RCR     SI,1            ;    мантиссу вправо
        MOV     BH,DH           ; Передать порядок в DH
        ; Форматировать результат.
ADDF7:  ADD     AH,AH           ; Знак во флажке переноса
        RCR     BH,1            ; Знак числа на месте
        RCR     AH,1            ; Младший бит порядка в AH7
        OR      AH,7FH          ; Образовать маску
        AND     BL,AH           ; Образовать второй байт числа
ADDF8:  RET                     ; Возврат
```

ности МП 1810 позволяют запрограммировать эти операции намного короче, чем для МП К580. Тем не менее учет всех требований стандарта на числа и арифметику с плавающей точкой (см. гл. 1) и операции над числами двойной точности приводят к довольно громоздким подпрограммам. Поэтому рассмотрим подпрограммы арифметических операций для формата одинарной точности, так как в этом случае операнды можно разместить в регистрах МП. Напомним, что в этом 32-битном формате старший бит является знаковым, затем следует 8-битный смещенный порядок (смещение равно 127) и после него находятся 23 бит дробной части мантиссы. Разряд целой части мантиссы, содержащий 1, явно в исходных числах не фигурирует (скрытый бит). Для удобства сравнения программ мы воспользуемся алгоритмами и схемами из § 2.4 и сохраним те же самые метки.

Сложение. В программе 3.18 сложения числа с плавающей точкой предполагается следующее распределение регистров МП: первое слагаемое в регистрах BX:SI, второе в регистрах DX:DI и сумма образуется на месте первого слагаемого. В подпрограмме ADDF складываются только числа с одинаковыми знаками; если же знаки операндов различны, происходит переход к подпрограмме вычитания SUBF (с изменением знака второго числа).

Сравнение программы 3.18 с соответствующей программой 2.26 для МП К580 может вызвать некоторое удивление. Действитель-

но, программа для МП К1810 состоит из 50 команд, а программа для МП К580 — «всего» из 46 команд. Объясняется эта парадоксальная ситуация двумя причинами. Во-первых, в программах 2.26 и 3.18 используются различные форматы чисел. Если для МП К580 принять такой же 32-битный формат, какой принят для МП К1810, его программа заметно усложнится. Во-вторых, программа 2.26 написана с привлечением подпрограмм SWAP (длина 7 команд, вызывается четыре раза), SHIFT (длина 7 команд, вызывается два раза), REC (длина 9 команд, вызывается два раза) и PACK (длина 15 команд, вызывается один раз). Без этих подпрограмм длина программы 2.26 увеличилась бы более чем в два раза. В то же время в программе 3.18 нет ни одной подпрограммы, так как действия указанных выше подпрограмм кодируются небольшим числом команд МП К1810. Например, подпрограмма SWAP обмена операндов состоит всего из двух команд:

```
XCHG  BX,DX          Обменять операнды
XCHG  SI,DI
```

Вычитание. Операция вычитания, как уже говорилось, похожа на операцию сложения, но теперь мантисса меньшего числа после выравнивания порядков вычитается из мантиссы большего числа и соответственно корректируется знак результата. При вычитании чисел с одинаковыми знаками, на что рассчитана программа SUBF, необходимо проверять, не возникло ли антипереполнение.

В программе 3.19 вычитания чисел с плавающей точкой предполагается следующее распределение регистров МП: уменьшаемое находится в регистрах BX : SI, вычитаемое в регистрах DX : DI, разность образуется на месте уменьшаемого. Вычитаются только числа с одинаковыми знаками; если же знаки различны, происходит переход к программе ADDF (с изменением знака вычитаемого). Суть операции заключается в том, что из большего по модулю числа вычитается меньшее.

Умножение. Умножение чисел с плавающей точкой заключается в сложении порядков и умножении мантисс. Рассмотрим два варианта умножения: в первом из них реализован стандартный алгоритм с циклом умножения на отдельные биты множителя, а во втором применяется команда умножения MUL.

В программе 3.20 предполагается стандартное распределение регистров: первый сомножитель в регистрах BX : SI, второй — в регистрах DX : DI и произведение образуется на месте первого сомножителя.

Приведем теперь фрагмент умножения мантисс, в котором применяется команда умножения MUL (программа 3.21). Фрагмент начинается с метки MULF4 и продолжается до метки MULF6.

Программа 3.19. *Вычитание чисел с плавающей точкой:*

```
        ; Первый операнд X (уменьшаемое) находится в регистрах BX:SI,
        ; второй Y (вычитаемое) в регистрах DX:DI, разность
        ; возвращается в регистрах BX:SI.
        ; При антипереполнении флажок CF установлен в 1.
        ;
SUBF:   ; Сравнить знаки операндов и определить операцию.
        MOV     AL,BH           ; Сравнить знаки операндов
        XOR     AL,DH
        JNS     SUBF1           ; Знаки операндов одинаковы
        XOR     DH,80H          ; Знаки различны,
        JMP     ADDF            ;    перейти к сложению
        ;
        ; Проверить операнды на нуль.
SUBF1:  MOV     AX,DX           ; Проверить на нуль
        OR      AX,DI           ;    второй операнд
        JZ      SUBFA           ; Результат в регистрах BX:SI
        MOV     AX,BX           ; Проверить на нуль
        OR      AX,SI           ;    первый операнд
        JNZ     SUBF2           ; Оба операнда ненулевые
        XCHG    BX,DX           ; Первый операнд равен нулю,
        XCHG    SI,DI           ;    разность равна
        XOR     BH,80H          ;    второму операнду
        JMP     SUBFA           ;    с измененным знаком
        ;
        ; Оба операнда ненулевые, можно вычитать.
SUBF2:  MOV     AH,BH           ; Сохранить знак уменьшаемого
        SHL     BX,1            ; Восстановить скрытый бит
        STC                     ;    мантиссы уменьшаемого
        RCR     BL,1
        SHL     DX,1            ; Восстановить скрытый бит
        STC                     ;    мантиссы вычитаемого
        RCR     DL,1
        ;
        ; Проверить отношение между числами.
        CMP     BH,DH           ; Сравнить порядки чисел
        JNZ     SUBF3           ; Порядки не равны
        CMP     BL,DL           ; Сравнить старшие байты мантисс
        JNZ     SUBF3           ; Они не равны
        CMP     SI,DI           ; Сравнить младшие слова мантисс
        JNZ     SUBF3           ; Они не равны
        MOV     BX,0            ; Числа равны,
        MOV     SI,0            ;    разность равна нулю
        JMP     SUBFA           ; Результат в регистрах BX:SI
        ;
        ; Операнды не равны, необходимо вычитать.
SUBF3:  JNC     SUBF4           ; Уменьшаемое больше вычитаемого
```

```
        XCHG    BX,DX           ; Вычитаемое больше уменьшаемого,
        XCHG    SI,DI           ;    обменять числа
        XOR     AH,80H          ; Изменить знак результата
        ;
        ; Образовать разность порядков.
SUBF4:  SUB     BH,DH           ; Разность порядков в BH
        JZ      SUBF7           ; Порядки одинаковы
        CMP     BH,24           ; Проверить диапазон разности
        JC      SUBF6           ; Можно выравнивать порядки
        JMP     SUBF9           ; Результат в BX:SI
        ;
        ; Сдвигать мантиссу меньшего числа вправо.
SUBF6:  SHR     DL,1            ; Сдвигать мантиссу
        RCR     DI,1            ;    в регистрах DL:DI
        INC     DH              ; Увеличить меньший порядок
        DEC     BH              ; Декремент разности порядков
        JNZ     SUBF6           ; Повторять сдвиг
        MOV     BH,DH           ; Передать общий порядок в BH
        ;
        ; Вычитание мантисс и образование результата.
SUBF7:  SUB     SI,DI           ; Вычесть мантиссы
        SBB     BL,DL
SUBF8:  OR      BL,BL           ; Проверить старший бит мантиссы
        JS      SUBF9           ; Результат нормализован
        DEC     BH              ; Декремент порядка
        CMP     BH,255          ; Проверить антипереполнение
        STC                     ; Установить флажок CF
        JZ      SUBFA           ; Возникло антипереполнение
        SHL     SI,1            ; Сдвинуть мантиссу влево
        RCL     BL,1
        JMP     SUBF8           ; Повторять до нормализации
SUBF9:  ADD     AH,AH           ; Знак во флажке переноса
        RCR     BH,1            ; Знак результата на месте
        RCR     AH,1            ; Младший бит порядка в AH7
        OR      AH,7FH          ; Образовать маску
        AND     BL,AH           ; Образовать второй байт разности
SUBFA:  RET                     ; Возврат
```

Программа 3.20. *Умножение чисел с плавающей точкой:*

```
        ; Первый операнд X находится в регистрах BX:SI.
        ; второй Y в регистрах DX:DI, произведение
        ; возвращается в регистрах BX:SI.
        ; При возникновении особого случая флажок CF содержит 1.
        ;
MULF:   ; Проверить операнды на нуль.
        MOV     AX,BX           ; Проверить на нуль
        OR      AX,SI           ;    первый операнд
        JZ      MULF8           ; Произведение равно нулю
        MOV     AX,DX           ; Проверить на нуль
        OR      AX,DI           ;    второй операнд
        JNZ     MULF1           ; Операнды не равны нулю
        XCHG    BX,DX           ; Произведение равно нулю
        XCHG    SI,DI
        JMP     MULF8
        ;
        ; Оба операнда ненулевые, можно умножать.
        ; Образовать знак произведения, восстановить мантиссы.
MULF1:  MOV     CH,BH           ; Знак произведения
        XOR     CH,DH           ;   в регистре CH
        SHL     BX,1            ; Восстановить скрытый бит
        STC                     ;    мантиссы первого операнда
        RCR     BL,1
        SHL     DX,1            ; Восстановить скрытый бит
        STC                     ;    мантиссы второго операнда
        RCR     DL,1
        ;
        ; Сложить порядки.
        MOV     AL,BH           ; Сложить в AL
        ADD     AL,DH           ;    смещенные порядки
        JC      MULF2           ; Возник перенос
        SUB     AL,127          ; Вычесть смещение
        JNC     MULF3           ; Можно умножать
        JMP     MULF8           ; Возникло антипереполнение
MULF2:  ADD     AL,129          ; Учесть потерю 256 из-за переноса
        JNC     MULF3           ; Можно умножать
        JMP     MULF8           ; Возникло переполнение
        ;
        ; Можно умножать мантиссы.
MULF3:  MOV     BH,AL           ; Порядок произведения в BH
        XOR     DH,DH           ; Подготовить место
        XOR     AX,AX           ;    для произведения
        MOV     CL,24           ; Образовать счетчик бит
        ; Здесь начинается цикл умножения.
MULF4:  RCR     DL,1            ; Сдвинуть множитель
        RCR     DI,1            ;    влево на один бит
```

```
        JNC     MULF5           ; Бит множителя равен нулю
        ADD     AX,SI           ; Прибавить множимое
        ADC     DH,BL           ;    к произведению
MULF5:  RAR     DH,1            ; Сдвинуть сумму
        RAR     AX,1            ;    частичных произведений
        DCR     CL              ; Декремент счетчика бит
        JNZ     MULF4           ; Повторять до завершения
        MOV     BL,DH           ; Вернуть произведение
        MOV     SI,AX           ;    в регистры BL:SI
        ;
        ; Проверить нарушение нормализации влево.
MULF6:  OR      BL,BL           ; Проверить старший бит мантиссы
        JNS     MULF7           ; Нарушения нормализации нет
        INC     BH              ; Увеличить порядок на 1
        STC                     ; Установить флажок CF
        JZ      MULF8           ; Возникло переполнение
        JMP     MULFA           ; Переполнения нет
MULF7:  SHL     SI,1            ; Сдвинуть мантиссу
        RCL     BL,1            ;    влево на один бит
        ;
        ; Форматировать результат
MULFA:  ADD     CH,CH           ; Знак во флажке переноса
        RCR     BH,1            ; Знак числа на месте
        RCR     CH,1            ; Младший бит порядка в CH7
        OR      CH,7FH          ; Образовать маску
        AND     BL,CH           ; Образовать второй байт произведения
MULF8:  RET                     ; Возврат
```

Применение команды MUL ускоряет умножение мантисс примерно в 1,5 раза. Приведенный фрагмент выполняется в среднем за 500 тактов синхронизации, а соответствующий фрагмент в программе 3.20 — в среднем за 750 тактов синхронизации.

Деление. Чтобы разделить числа с плавающей точкой, необходимо из порядка делимого вычесть порядок делителя, а мантиссу делимого разделить на мантиссу делителя. В операции деления возможны оба особых случая — переполнение и антипереполнение. В программе 3.22 принято стандартное размещение делимого в регистрах BX : SI и делителя в регистрах ДХ : DI, частное возвращается на месте делимого.

Сначала в программе анализируются нулевые операнды, а затем восстанавливаются скрытые биты мантисс, вычитаются порядки (с проверкой особых случаев). Начиная с метки DIVF3 реализован цикл деления мантисс. Если получен ненормализованный результат, производится нахождение дополнительного бита мантиссы частного. Заключительные команды стандартным образом форматируют результат.

Программа 3.21. *Умножение мантисс с применением команды* MUL:

```
        ; Мантиссы находятся в BL:SI, знак в CH,
        ; порядок в BH.
        ;
        MOV     CL,DL           ; Освободить регистр DX
MULF4:  MOV     AX,SI           ; Умножить
        MUL     DI              ;    младшие слова мантисс
        MOV     TEMP,DX         ; Сохранить старшую часть произведения
        MOV     AL,BL           ; Умножить BL на DL
        MOV     AH,0
        MUL     DI
        ADD     TEMP,AX         ; Учесть результат
        JNC     NEXT            ;    в полном произведении
        INC     DX
NEXT:   MOV     TEMP+2,DX       ; Сохранить произведение
        MOV     AL,CL           ; Умножить
        MOV     AH,0            ;    следующие части мантисс
        MUL     SI
        ADD     TEMP,AX         ; Учесть результат
        JNC     NEXT1           ;    в полном произведении
        INC     DX
NEXT1:  ADD     DX,TEMP+2
        MOV     AL,BL           ; Умножить
        MUL     CL              ;    старшие байты мантисс
        ADD     AX,DX           ; Образовать полное произведение
        MOV     BL,AH           ;    и разместить его
        MOV     AH,AL           ;    в регистрах BL,SI
        MOV     AL,TEMP+1
        MOV     SI,AX
MULF6:                          ; Далее как прежде
```

К сожалению, использовать команду DIV для деления мантисс чисел с плавающей точкой не представляется возможным.

Программа 3.22. *Деление чисел с плавающей точкой:*

```
        ; Делимое X находится в регистрах BX:SI, делитель Y
        ; в регистрах DX:DI, частное возвращается на месте делимого.
        ; При возникновении особого случая флажок CF содержит 1.
        ;
        ;
DIVF:   ; Проверить операнды на нуль.
        MOV     AX,BX           ; Проверить на нуль
        OR      AX,SI           ;    делимое
        JZ      DIVF7           ; Ненулевой результат в BX:SI
        MOV     AX,DX           ; Проверить на нуль
        OR      AX,DI           ;    делитель
        STC                     ; Если деление на нуль,
        JZ      DIVF7           ;    установить флажок CF
        ;
        ; Оба операнда ненулевые.
        MOV     CH,BH           ; Образовать знак частного
        XOR     CH,DH           ;    в регистре CH
        SHL     BX,1            ; Восстановить скрытый бит
        STC                     ;    мантиссы делимого
        RCR     BL,1
        SHL     DX,1            ; Восстановить скрытый бит
        STC                     ;    мантиссы делителя
        RCR     DL,1
        ;
        ; Вычесть порядки.
        MOV     AL,BH           ; Образовать разность порядков
        SUB     AL,DH           ;    в регистре AL
        JNC     DIVF1           ; Порядок делимого больше
        ADD     AL,127          ; Прибавить смещение
        CMC                     ; Если нет переноса,
        JC      DIVF7           ;    возникло антипереполнение
        JMP     DIVF2           ; Перейти к делению мантисс
DIVF1:  ADD     AL,127          ; Прибавить смещение
        JC      DIVF7           ; Возникло переполнение
        ; Деление мантисс.
        MOV     BH,0            ; Очистить старшие байты
        MOV     DH,0            ;    перед мантиссами
        MOV     CL,24           ; Образовать счетчик бит
DIVF3:  SUB     SI,DI           ; Вычесть мантиссу делителя
        SBB     BX,DX           ;    из мантиссы делимого
        CMC                     ; Образовать в CF бит частного
        PUSHF                   ; Сохранить его в стеке
        JC      DIVF4           ; Остаток положительный
        ADD     SI,DI           ; Восстановить предыдущий
        ADC     BX,DX           ;    положительный остаток
```

```
DIVF4:  POPF                    ; Вернуть бит частного в CF
        RCL     BP,1            ; Передать бит частного
        RCL     AH,1            ;     в буферные регистры AH:BP
        SHL     SI,1            ; Сдвинуть остаток
        RCL     BX,1
        DEC     CL              ; Декремент счетчика бит
        JNZ     DIVF3           ; Повторять до завершения
        ;
        ; Проверить нарушение нормализации вправо.
        TEST    AH,80H          ; Проверить старший бит мантиссы
        JNZ     DIVF5           ; Нарушения нормализации нет
        DEC     AL              ; Декремент порядка
        STC                     ; Проверить возможность
        JZ      DIVF7           ;     антипереполнения
        SUB     SI,DI           ; Определить еще один бит
        SBB     BX,DX           ;     мантиссы частного
        CMC                     ; Поместить бит частного
        RCL     BP,1            ;     в нужное место
        RCL     AH,1
DIVF5:  MOV     BL,AH           ; Разместить частное
        MOV     SI,BP           ;     в регистрах BX:SI
        MOV     BH,AL
        ;
        ; Форматировать результат.
        ADD     CH,CH           ; Передать знак во флажке CF
        RCL     BH,1            ; Знак числа на месте
        RCL     CH,1            ; Младший бит порядка в CH7
        OR      CH,7FH          ; Образовать маску
        AND     BL,CH           ; Образовать второй байт частного
DIVF7:  RET                     ; Возврат
```

3.4.4. ВСПОМОГАТЕЛЬНЫЕ ПРОГРАММЫ

Преобразование форматов. Не будем приводить многочисленные программы преобразования форматов, аналогичные программам, рассмотренным в п. 2.4.4. При необходимости читатель может без труда разработать их самостоятельно. Остановимся только на преобразованиях, в которых участвуют числа с плавающей точкой.

Пусть в регистрах BL : SI находится целое знаковое число в дополнительном коде и требуется образовать в регистрах BX : SI его представление в 32-битном формате с плавающей точкой, который был принят в п. 3.4.3 для программ арифметических операций (знак, байт смещенного порядка и три байта мантиссы со скрытым битом).

В программе ITOF (3.23) преобразуемое число вначале проверяется на нуль; если оно равно нулю, осуществляется возврат с

истинным нулем в регистрах BX : SI. При ненулевом числе его знак сохраняется в регистре AL, в регистрах BL : SI образуется абсолютное значение числа. Отметим здесь использование команды ADD SI, 1 вместо традиционной команды инкремента INC SI; вызвано это тем, что последняя команда INC не воздействует на флажок переноса, состояние которого нужно учесть при преобразовании содержимого регистра BL.

Далее организуется цикл сдвига числа влево до тех пор, пока мантисса не будет нормализованной. С каждым сдвигом происходит декремент смещенного порядка на 1; за исходный смещенный порядок принято число 151 (127+24). Заключительные действия программы, начиная с метки FORM, связаны с форматированием результата.

Программа 3.23. *Преобразование целого числа в формат с плавающей точкой;*

```
        ; Исходное число в регистрах BL:SI, результат
        ; возвращается в регистрах BX:SI.
        ;
ITOF:   MOV     BH,0            ; Сбросить регистр BH
        MOV     AX,SI           ; Проверить исходное число
        OR      AX,BX           ;    на нуль
        JZ      EXIT            ; Число равно нулю
        MOV     AL,BL           ; Сохранить знак в регистре AL
        TEST    AL,80H          ; Проверить знак числа
        JC      NOC             ; Число положительное
        NOT     SI              ; Число отрицательное.
        NOT     BL              ;    образовать его
        ADD     SI,1            ;    абсолютное значение
        JNC     NOC
        INC     BL
NOC:    MOV     BH,151          ; Исходный порядок
SLOOP:  TEST    BL,80H          ; Проверить старший бит
        JNZ     FORM            ; Нормализация закончена
        SHL     SI,1            ; Сдвинуть число влево
        RCL     BL,1
        DEC     BH              ; Декремент порядка
        JMP     SLOOP           ; Повторять нормализацию
FORM:   ADD     AL,AL           ; Передать знак во флажок CF
        RCR     BH,1            ; Знак числа на месте
        RCR     AH,1            ; Младший бит порядка в AH7
        OR      AH,7FH          ; Образовать маску
        AND     BL,AH           ; Образовать второй байт числа
EXIT:   RET                     ; Возврат
```

Обратное преобразование числа с плавающей точкой требует отдельного представления его целой и дробной частей. Целая часть числа преобразуется в 24-битное знаковое число в дополнительном

коде, причем точка фиксирована после младшего значащего разряда. Для дробной части примем такое же представление в дополнительном коде, но зафиксируем точку после знакового бита. Предлагается, что исходное число находится в регистрах BX : SI, а после преобразования целая часть находится в регистрах BL : SI и дробная часть — в регистрах DL : DI.

В программе FTOI (3.24) исходное число вначале проверяется на нуль и, если оно равно нулю, в регистры DX : DI загружаются нули, флажок CF устанавливается в 1, показывая успешное преобразование, и осуществляется возврат. Затем проверяется нахождение исходного числа в диапазоне представимых чисел выходного формата. Для этого в регистре BH образуется байт смещенного порядка, который сравнивается с максимальным значением 127+ +23=150 и минимальным значением 127—23=104. Если исходный смещенный порядок находится вне этих границ, подпрограмма заканчивается со сброшенным в 0 флажком CF.

Когда число находится в допустимом диапазоне, определяется, имеет ли оно целую часть. Для этого смещенный порядок вычитается из 150. Если разность превышает 23, т. е. целая часть отсутствует, в регистр целой части загружаются нули и осуществляется переход к преобразованию дробной части (метка FRAC). В случае ненулевой целой части мантисса сдвигается вправо так, чтобы разряд единиц оказался в младшем разряде регистра SI. В зависимости от знака числа происходит образование дополнительного кода.

Преобразование дробной части начинается с получения в регистре BH истинного порядка. Затем анализируется его знак и определяется необходимость сдвига мантиссы влево или вправо. С учетом знака числа в регистрах DL : DI образуется дополнительный код дробной части.

Извлечение квадратного корня. В инженерных и научных расчетах довольно часто возникает необходимость извлечения квадратного корня из целого числа. Традиционно эта операция реализуется алгоритмом, очень похожим на алгоритм деления, но программирование его в терминах системы команд МП встречает определенные трудности. Вместе с тем известен итерационный метод последовательных приближений или метод Ньютона, который дает результат за приемлемое число итераций. В этом методе доказано, что если x_i является приближенным значением квадратного корня из числа N, то число

$$x_{i+1}=(N/x_i+x_i)/2$$

является лучшим следующим приближением. Обычно за первое приближение принимается значение $x=N/200+2$. Покажем применение рассмотренного метода на следующем примере.

Программа 3.24. *Выделение целой и дробной частей числа с плавающей точкой:*

```
        ; Исходное число с плавающей точкой в регистрах BX:SI.
        ; Целая часть образуется в регистрах BL:SI, дробная часть
        ; в регистрах DL:DI ( обе в дополнительном коде).
        ; О выходе за диапазон сигнализирует флажок CF=0.
        ;
FTOI:   MOV     AX,SI           ; Проверить исходное число
        OR      AX,BX           ;    на нуль
        MOV     DX,AX           ; Передать возможный нуль
        MOV     DI,AX           ;    в регистры DX:DI
        JZ      QUIT            ; Результат равен нулю
        MOV     AH,BH           ; Сохранить знак в регистре AH
        SHL     BX,1            ; Восстановить
        STC                     ;    скрытый бит
        RCR     BL,1            ;    мантиссы
        ;
        ; Проверить нахождение числа в допустимом диапазоне.
        CMP     BH,150          ; Сравнить с максимальным порядком
        JNC     EXIT            ; Выход за верхнюю границу
        CMP     BH,104          ; Сравнить с минимальным порядком
        CMC
        JNC     EXIT            ; Выход за нижнюю границу
        ;
        ; Преобразование возможно.
        MOV     DI,SI           ; Сохранить число для получения
        MOV     DX,BX           ;    дробной части
        MOV     AL,150          ; Образовать в регистре AL
        SUB     AL,BH           ;    счетчик сдвигов
        CMP     AL,23           ; Имеется ли целая часть?
        JC      INTEG           ; Да, преобразовать ее
        MOV     BL,0            ; Целая часть равна нулю
        MOV     SI,0
        JMP     FRAC            ; Преобразовать дробную часть
        ;
        ; Преобразование целой части.
INTEG:  SHR     BL,1            ; Сдвигать мантиссу вправо
        RCR     SI,1            ;    до правильной позиции
        DCR     AL              ;    целой части
        JNZ     INTEG
        TEST    AH,80H          ; Проверить знак числа
        JZ      FRAC
        NOT     SI              ; Учесть знак числа
        NOT     BL
        ADD     SI,1
        JNC     FRAC
        INC     BL
        ;
```

```
        ; Преобразование дробной части.
FRAC:   SUB     BH,127          ; Образовать истинный порядок числа
        JZ      NOSH            ; Он равен нулю
        JNC     SHILA           ; Мантиссу сдвигать влево
        NEG     BH              ; Образовать счетчик для сдвига вправо
RLOOP:  SHR     DL,1            ; Сдвигать мантиссу вправо
        RCR     DI,1            ;    в правильную позицию
        DCR     BH
        JNZ     RLOOP
        JMP     SIGN            ; Учесть знак числа
SHILA:  SHL     DI,1            ; Сдвигать мантиссу влево
        RCL     DL,1            ;    в правильную позицию
        DEC     BH
        JNZ     SHILA
        JMP     SIGN            ; Учесть знак числа
NOSH:   AND     DL,7FH          ; Подавить единицу целой части
SIGN:   TEST    AH,80H          ; Проверить знак числа
        JZ      NOCI            ;    и при необходимости образовать
        NOT     DI              ;    дополнительный код
        NOT     DL
        ADD     SI,1
        JNC     QUIT
        INC     DL
QUIT:   STC                     ; Отметить успешное преобразование
EXIT:   RET                     ; Возврат
```

Пусть N = 14 400 и точное значение квадратного корня равно 120. Тогда $x_1 = 14\,400/200 + 2 = 74$. Затем последовательно получаем:

$$x_2 = (14\,400/74 + 74)/2 = 134,$$

$$x_3 = (14\,400/134 + 134)/2 = 120,$$

$$x_4 = (14\,400/120 + 120)/2 = 120.$$

Здесь всего за две итерации (не считая вычисления первого приближения) получено точное значение квадратного корня, которое в дальнейшем изменяться не будет.

Для того чтобы окончить процедуру последовательных приближений, целесообразно сравнить два соседних значения x_i и x_{i-1}. Итерации прекращаются, если они равны или отличаются на единицу. Рассмотренный метод извлечения квадратного корня реализует программа 3.25.

Программа 3.25. *Извлечение квадратного корня из 32-битного числа:*

```
                ; Исходное число находится в регистрах DX:AX,
                ; результат возвращается в регистре BX.
                ;
SQRT    PROC                    ; Начало процедуры
        PUSH    BP              ; Освободить рабочий регистр BP
        PUSH    DX.             ; Сохранить исходное число
        PUSH    AX              ;    в стеке
        MOV     BP,SP           ; BP адресует число в стеке
        MOV     BX,200          ; Вычислить в регистре AX
        DIV     BX              ;    первое приближение
        ADD     AX,2
        ;
        ; Итерационный цикл.
NEXT:   MOV     BX,AX           ; Передать XI в регистр BX
        MOV     AX,[BP]         ; Вернуть число из стека
        MOV     DX,[BP+2]       ;    в регистры DX:AX
        DIV     BX              ; Вычислить
        ADD     AX,BX           ;    следующее
        SHR     AX,1            ;    приближение X(I+1)
        CMP     AX,BX           ; Сравнить XI и X(I+1)
        JE      DONE            ;    и либо закончить,
        SUB     BX,AX           ;    либо повторять итерации
        CMP     BX,1
        JE      DONE
        CMP     BX,-1
        JNE     NEXT
DONE:   MOV     BX,AX           ; Передать результат в регистр BX
        POP     AX              ; Восстановить
        POP     BX              ;    исходное число
        POP     BP              ;    и рабочий регистр BP
        RET                     ; Возврат
SQRT    ENDP                    ; Конец процедуры
```

Контрольные вопросы и упражнения

1. Покажите адресные пространства памяти и ввода-вывода МП К1810ВМ86.
2. В чем заключается принцип опережающей выборки команд? Назовите его достоинства. Есть ли у него недостатки?
3. Что такое физический и логический адреса в МП К1810ВМ86? На какие сигнальные линии он выдает физический адрес памяти?
4. Каким образом адресное пространство памяти МП К1810ВМ86 превратить в такое же адресное пространство, какое имеет МП К580?
5. Какие регистры МП могут участвовать в формировании физического адреса памяти при выборке команд? при обращении к переменным?
6. Предположим, что в регистрах МП содержатся следующие данные: (CS) = 0100H, (IP) = 2000H, (DS) = 1300H, (SI) = 0020H, (DI) = 0040H, (BX) = 3000H. Определите физические адреса памяти следующей команды и

данных, если в ассемблерных командах имеются такие спецификации операндов: [SI], [DI], [BX][SI], [BX—10H], [BX+5][DI—20H].

7. Какова минимальная и максимальная длина команд в МП К1810ВМ86?

8. Какие три атрибута имеет переменная?

9. Какой флажок показывает переполнение в арифметических операциях со знаковыми числами? с беззнаковыми числами?

10. Являются ли допустимыми следующие команды: MOV CH, 500H; MOV DX, AL; ADD BH, CH; PUSH AL?

11. Напишите команды для пересылки слова из ячейки X1 в ячейку X2.

12. Напишите команды для включения в стек константы 200H, включения в стек значений от 1 до 10.

13. Напишите бесконечный цикл, который выводит в выходной порт PORT последовательность чисел, начинающуюся с нуля.

14. Опишите последовательность действий, выполняемых командами CALL и RET (внутрисегментных и межсегментных).

15. Эквивалентны ли команды RET 10 и ADD SP, 10 и RET? Напишите команды, действия которых в точности соответствуют действиям команды RET 10.

16. Напишите команду(ы), которая(ые):

устанавливают в 1 три старших бита регистра BX;

сбрасывают в 0 четыре младших бита регистра BH;

устанавливают в 1 те биты регистра AX, состояния которых отличаются от состояний соответствующих битов регистра CX;

сдвигают содержимое регистра BX влево на 8 бит;

делят содержимое регистра AX на 16;

реализуют логические операции НЕ-ИЛИ, НЕ-И над содержимым регистров AX и BX;

обменивают старшую и младшую тетрады регистра AH.

17. Постройте машинные коды команд MOV BX, AX; ADD SI, DI; ADD AX, 8000H; XOR BH, CH; OR AX, 1; SUB BP SI; SBB AX, 20H; SBB AX, 2000H.

18. Каким образом задается направление сканирования цепочки?

19. При каких условиях можно сравнивать цепочки различной длины?

20. Напишите фрагмент инициализации области памяти с начальным адресом BUFFER на код пробела (20H), пользуясь и не пользуясь цепочечными командами.

21. Напишите программный фрагмент, который в цепочке из 200 символов с начальным адресом STRING заменяет десятичные цифры на код пробела (20H).

22. Разработайте программу преобразования беззнакового двоичного числа из регистра AX в цепочку из четырех 16-ричных цифр и размещения ее в области BUFF.

23. Приведите результаты команд ADD AX, BX и SUB AX, BX, если регистры AX и BX содержат следующие пары чисел: 07FFH и F150H, 0010H и 2000H. 8000H и 8000H, FFFFH и FFFFH, EEEEH и 3456H.

24. Найдите результаты команд MUL BH и IMUL BH, если в регистрах AL и BH содержатся следующие пары чисел: 01H и FFH, F3H и 04H, FFH и FFH, 00H и 2AH.

25. Найдите результаты команд DIV CH и IDIV CH, если в регистрах AX и CH находятся следующие пары чисел: FFFFH и FFH, 0375H и A0H, 8000H и 7FH, 0060H и 05H.

26. Можно ли использовать команды CBW и CWD для беззнаковых чисел?

27. Оцените время выполнения программ 3.3—3.6, считая параметром длину операндов N в байтах или словах.

28. Почему в программах 3.1 и 3.2 нельзя использовать метки LOOP? Почему в них требуется одинаковое содержимое регистров DS и ES?

29. Трансформируйте программу 3.7 для сложения слов.

30. Как зафиксировать переполнение в программе 3.8?

31. Постройте схему действий команды DAS.

32. Команды DAA и DAS будут «корректировать» результаты двоичного

сложения и вычитания, даже если операнды не являются десятичными числами. Определите, какой результат будет получен в аккумуляторе AL после выполнения пар команд ADD AL, BL и DAA, SUB AL, BL и DAS, если в регистрах AL и BL находятся числа 0F6H и 7AH, 0ECH и 8FH.

33. Постройте схемы действий команд AAA, AAS, AAM и AAD.

34. Почему в программе 3.16 необходимо очищать область памяти для произведения?

35. Как изменится программа 3.16, если не предполагать старшие нулевые тетрады байтов сомножителей?

36. Какой результат даст команда AAD, если в младшей тетраде регистра AH находится запрещенная комбинация?

37. Разработайте более простой фрагмент форматирования результата в программе 3.18 (начиная с метки ADDF7).

38. Разработайте схему округления результата в программах 3.18 и 3.19.

39. Извлеките квадратный корень из чисел 10201, 1048576 и 225 в соответствии с программой 3.15.

АРИФМЕТИЧЕСКИЙ СОПРОЦЕССОР К1810ВМ87

В данной главе рассмотрен арифметический сопроцессор К1810ВМ87, рассчитанный на использование только совместно с центральным процессором К1810ВМ86. Появление микросхемы К1810ВМ87 расширяет сферы применения микропроцессоров на область математических расчетов, в которых требуются очень широкий диапазон и высокая точность представления чисел, т. е. переход к аппаратному формату с плавающей точкой. Показана такая особенность сопроцессорной конфигурации, как необходимость синхронизации центрального процессора и сопроцессора по командам и данным.

Рассмотрены программная модель сопроцессора, внешние и внутренний форматы чисел и система команд, включающая в себя сложные математические операции.

Отдельный параграф посвящен специальным числам и автоматической регистрации особых случаев; допускается маскирование особых случаев и сопроцессор при этом образует математически наиболее приемлемый результат.

Приведены программы суммирования элементов массива, статистической обработки экспериментальных данных, возведения в произвольную степень, логарифмирования и др. В заключение показаны особенности представления чисел в персональных компьютерах.

4.1. ОСОБЕННОСТИ СОПРОЦЕССОРНЫХ КОНФИГУРАЦИЙ

Как было показано в гл. 2 и 3, однокристальные МП оперируют числами, представленными в простейших форматах — двоичные знаковые и беззнаковые числа, и ограниченно поддерживают десятичные целые числа. Вычислительные возможности их ограничены арифметическими операциями. Расширение форматов чисел в таких МП с увеличением количества команд ведет к их чрезмерному усложнению и трудностям программирования. Программная реализация сложных операций и операций над числами с плавающей точкой значительно снижает производительность. Стандарты на численные данные (см. гл. 1) усложнили форматы чисел и потребовали учета

различных особых случаев. Программы операций над числами в стандартных форматах становятся весьма громоздкими. Применение секционных МП с микропрограммным управлением требует значительного числа микросхем и связано с увеличением потребляемой мощности.

В этой ситуации плодотворным оказался принцип специализации, который давно применяется в процессорах средних и крупных компьютеров. Суть его заключается в разработке вспомогательных специализированных процессоров, ориентированных на конкретные прикладные области. Такие процессоры работают под управлением центрального (главного) процессора и разделяют с ним основную память. Специализация позволяет достичь высокого быстродействия вспомогательных процессоров и повысить эффективную производительность системы благодаря параллельной работе нескольких процессоров.

В сопроцессорной конфигурации вспомогательный процессор (сопроцессор) подключается к системной шине параллельно с центральным процессором (ЦП). Сопроцессор не имеет своей отдельной программы и не может считывать команды из памяти, но может обращаться к ней для записи и считывания данных, запрашивая для этого шину у ЦП. Кроме того, сопроцессор контролирует системную шину и может «перехватывать» адреса и данные, когда к памяти обращается ЦП. Часть кодов операций ЦП резервируется для команд сопроцессора и действия ЦП при их выполнении сводятся к вычислению физического адреса и к обращению в память. Сопроцессор не может выполнять команды ЦП, но свои команды выполняет очень быстро (по сравнению с их программной эмуляцией в командах ЦП). Программа оказывается смесью команд ЦП и сопроцессора, причем выборку команд из памяти осуществляет только ЦП, а затем команды подаются в оба процессора. Каждый из них выбирает из общего командного потока и выполняет свои команды. Такое своеобразное «разделение труда» позволяет достичь очень высокой производительности в тех задачах, на которые ориентирован сопроцессор.

Процессор численных данных или арифметический сопроцессор К1810ВМ87 предназначен для работы с ЦП К1810ВМ86. Он рассчитан на применение в таких системах, где числа изменяются в очень широком диапазоне, требуется высокая точность вычислений и необходима такая производительность, которая превышает возможности ЦП. Относительная простота программирования и поразительные возможности сопроцессора К1810ВМ87 делают его доступным большой группе пользователей, не знакомых с тонкостями программирования сложных вычислительных задач. Достаточно сказать, что он оперирует числами из диапазона $\pm 10^{\pm 5000}$ и обеспечивает точность 18 десятичных разрядов. Сопроцессор имеет команды таких сложных операций, как извлечение квадратного корня, возведение в степень, логарифмирование и др.

Сопроцессор К1810ВМ87 не может работать изолированно от ЦП К1810ВМ86. Вместе они образуют мощный тандем, производительность которого в задачах численной обработки в 10—50 раз и более выше производительности одного ЦП. В этом тандеме объединены системы команд, внутренние регистры и форматы чисел обоих процессоров. Его вычислительная мощность сравнима с вычислительной мощностью миникомпьютеров.

Сопроцессор представляет собой аппаратное расширение ЦП и не может работать автономно. Схема объединения их в систему показана на рис. 4.1. При этом ЦП К1810ВМ86 должен работать в максимальном режиме.

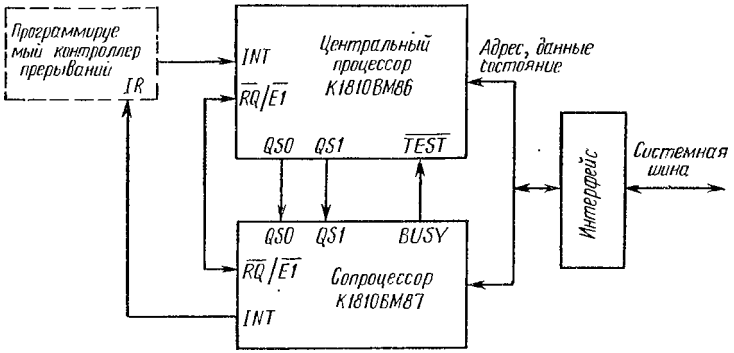


Рис. 4.1. Система с арифметическим сопроцессором

Когда выбранная из памяти команда оказывается командой ЦП (для простоты не учитываем опережающую выборку команд), он выполняет ее обычным образом, а сопроцессор не привлекается — он просто игнорирует такие команды. Когда же выбирается команда сопроцессора (в системе команд ЦП они обозначены мнемоникой ESC), действия ЦП зависят от специфики конкретной команды. Если она не связана с обращением к памяти, ЦП игнорирует ее («проскакивает») и переходит к следующей команде. Но если команда требует обращения к памяти, ЦП вычисляет физический адрес операнда в соответствии с указанным в команде режимом адресации и обращается к памяти. При этом сопроцессор перехватывает с общей шины физический адрес операнда, а в операции со считыванием из памяти еще и данные. После этого сопроцессор реализует конкретные действия по выполнению команды. Они могут производиться параллельно с дальнейшими действиями ЦП, что повышает эффективную производительность системы.

Такое объяснение взаимодействия ЦП и сопроцессора необходимо уточнить для тех ситуаций, когда их совместная работа требует синхронизации. ЦП «выполняет» команду ESC гораздо быстрее сопроцессора. Например, команду FSQRT извлечения квадрат-

ного корня ЦП «выполняет» всего за два такта синхронизации, а сопроцессору требуется 180 тактов. Поэтому в двух случаях необходимо синхронизировать действия процессоров.

Синхронизация по командам. Очевидно, команда сопроцессора не может начинаться до завершения им предыдущей команды, т. е. ЦП не должен пропускать в сопроцессор его команды быстрее, чем сопроцессор может выполнять их. Следовательно, перед каждой командой сопроцессора в программе должна быть специальная команда ЦП, которая только проверяет текущее состояние сопроцессора и, если он занят, переводит ЦП в состояние ожидания. Этой командой является команда WAIT проверки и ожидания. Когда сопроцессор занят выполнением команды, он формирует на своей выходной линии BUSY сигнал высокого уровня, подающийся на вход $\overline{\text{TEST}}$ ЦП. При выполнении команды WAIT ЦП опрашивает сигнал на входе $\overline{\text{TEST}}$ и, пока им не будет сигнал низкого уровня, не переходит к следующей команде. Дешифрирование команды, находящейся за командой WAIT, оба процессора осуществляют одновременно. Отметим, что введение этих команд WAIT в объективную программу может произвести программа-ассемблер автоматически, без специальных указаний программиста.

Принципиально команду WAIT можно помещать и после каждой команды сопроцессора. Но в этом случае ЦП не будет выполнять никаких (даже своих!) команд до тех пор, пока сопроцессор не освободится и степень параллелизма в работе процессоров ухудшается.

Синхронизация по данным. Если выполняемая сопроцессором команда записывает результат в ячейку памяти, то перед последующей командой ЦП, обращающейся к этой же ячейке, также необходима команда WAIT. Другими словами, ЦП не должен выполнять команду со считыванием из памяти операнда, адресуемого сопроцессором, до тех пор, пока сопроцессор не закончит запись в выбранную ячейку. Автоматически учесть такие ситуации довольно сложно, поэтому вводить команды, проверяющие состояние сопроцессора и при необходимости заставляющие ЦП ожидать, здесь должен программист. Благодаря этому сопроцессор обязательно закончит запись данного в память до того, как к нему обратится ЦП.

Мы пояснили синхронизацию работы двух процессоров без учета опережающей выборки команд и наличия в МП К1810ВМ86 очереди команд. Очевидно, для правильной совместной работы ЦП и сопроцессора в последнем также должна быть внутренняя очередь команд. Чтобы обеспечить синхронизм работы двух очередей, выходные сигналы состояния очереди QS микропроцессора К1810ВМ86 должны быть поданы на соответствующие входы сопроцессора.

Когда команда сопроцессора должна считывать из памяти более одного слова или записывать любые данные в память, сопроцессор запрашивает у ЦП шину сигналом на линии $\overline{\text{RQ}}/\overline{\text{E}}$. При

считывании из памяти сопроцессор запрашивает шину сразу после операции считывания ЦП, а в команде с записью в память — после того, как результат готов к записи, т. е. преобразован в формат получателя. В любом случае сопроцессор производит передачи данных в последовательных циклах шины, а затем освобождает шину. Обычно линия $\overline{RQ}/\overline{E0}$ сопроцессора подключается к линии $\overline{RQ}/\overline{E1}$ ЦП.

Сопроцессор запрашивает прерывание ЦП выходным сигналом INT, который обычно подается на один из входов программируемого контроллера прерываний. Прерывание генерируется при возникновении особого случая, т. е. некоторой аномалии в вычислительном процессе, например переполнении, при условии, что особый случай не замаскирован и прерывания от сопроцессора разрешены. Если особый случай замаскирован, сопроцессор реализует маскированную реакцию и не прерывает ЦП.

Остановимся на достоинствах и недостатках сопроцессорной конфигурации. Ее альтернативой является разработка ЦП со всеми функциональными возможностями сопроцессора. Такая задача оказывается довольно сложной — сопроцессор имеет на кристалле 75 000 транзисторов по сравнению с 30 000 транзисторов ЦП. Поэтому ЦП и сопроцессор в отдельности разработать проще, а при умелом программировании оба процессора могут работать параллельно. Наличие двух процессоров обеспечивает разработчикам микросистем дополнительную гибкость — сопроцессор применяется только там, где он нужен. Кроме того, никаких специальных приемов и ограничений на программирование систем с сопроцессором нет.

Конечно, сопроцессорный подход имеет и ряд недостатков. Отметим среди них вынужденное дублирование в сопроцессоре схем обращений к памяти, трудности построения систем с несколькими сопроцессорами (ЦП имеет только один вход $\overline{TEST}$) и, наконец, реализацию условных передач управления по результатам операций сопроцессора только через ЦП, что несколько снижает производительность системы.

4.2. ВНУТРЕННЯЯ ОРГАНИЗАЦИЯ И ПРОГРАММНАЯ МОДЕЛЬ СОПРОЦЕССОРА

Микросхема К1810ВМ87 производится по высококачественной NMOП — технологии, имеет 40-контактный корпус типа DIP, напряжение питания составляет $+5$ В$\pm 5\%$, потребляемая мощность около 2 Вт. Механизм взаимодействия ЦП и сопроцессора требует, чтобы они синхронизировались от одного и того же источника сихроимпульсов.

Временна́я диаграмма работы сопроцессора и его управляющие сигналы совместимы с МП К1810ВМ86, поэтому они подключаются

к системной шине и взаимодействуют друг с другом без согласующих схем. Сопроцессор имеет почти такую же разводку сигнальных линий, что и МП К1810ВМ86, но с учетом особенностей его работы имеются следующие различия в определении сигнальных линий: сигналы NMI, $\overline{\text{LOCK}}$ и MN/$\overline{\text{MX}}$ отсутствуют, вход $\overline{\text{TEST}}$ заменен на выход BUSY, сигнал INT стал выходным, а сигналы QS — входными.

Структурно сопроцессор состоит из двух практически автономных устройств: устройства управления (шинного интерфейса) и численного (операционного исполнительного) устройства. Устройство управления предназначено для восприятия команд, считывания и записи данных, выполнения команд управления сопроцессором, а также согласования действий сопроцессора и ЦП. Уже говорилось, что команды сопроцессора в потоке команд, выбираемых из программной памяти только ЦП, чередуются с командами самого ЦП. Сопроцессор определяет цикл выборки команды по сигналам состояния ST и при появлении байта или слова команды на шине адреса/данных устройство управления подключается к ней параллельно с ЦП и воспринимает команду. Она попадает в очередь команд сопроцессора, которая благодаря соединению линий QS работает параллельно с очередью команд ЦП. Первые пять битов кода операции всех команд сопроцессора одинаковы (код 11011) — они определяют команду ESC переключения на сопроцессор. Мнемоника ESC в системе команд ЦП как бы заменяет собой мнемоники всех команд сопроцессора и в программах не применяется — вместо нее указывается мнемоника одной из команд сопроцессора. «Свою» команду устройство управления либо выполняет само, либо передает в численное операционное устройство, а команды ЦП оно игнорирует.

ЦП различает несколько типов команды ESC. Если в команде требуется обращение к памяти, он вычисляет физический адрес, а затем инициирует цикл обращения к памяти. Считываемые данные ЦП игнорирует, т. е. для него действие оказывается «фиктивным» считыванием. Но отсюда следует, что в командах сопроцессора допустимы все режимы адресации памяти ЦП, хотя сам сопроцессор никаких адресных манипуляций делать не может. Когда же команда ESC не связана с обращением к памяти, ЦП просто переходит к следующей команде.

Команда ESC может потребовать загрузки операнда из памяти или записи результата в память, но может быть и не связана с обращением к памяти. В первых двух случаях устройство управления сопроцессора использует цикл «фиктивного» считывания, инициируемый ЦП. Оно воспринимает и сохраняет 20-битный физический адрес операнда, который ЦП выдает на шину адреса/данных/состояния. Если команда определяет загрузку, устройство управления воспринимает первое, и, возможно, единственное слово операнда в момент появления его на шине адреса/данных. Когда же длина

операнда превышает одно слово, устройство управления сразу запрашивает шину у ЦП и считывает остальную часть операнда. Команда с записью в память заставляет устройство управления только зафиксировать физический адрес памяти. Если сопроцессор готов к операции записи, устройство управления запрашивает шину у ЦП и записывает результат по ранее «перехваченному» адресу. В зависимости от длины результата инициируется необходимое число последовательных циклов шины.

В устройстве управления имеются два программно доступных 16-битных регистра, в которых хранятся слово управления и слово состояния. Еще четыре 16-битных регистра указателей содержат физические адреса команды и операнда, а также 11 бит кода операции (старшие 5 бит кода операции всех команд сопроцессора содержат 11011). Указатели предназначены для процедур обработки особых случаев, предоставляя возможность узнать, какая команда и с каким операндом вызвала особый случай.

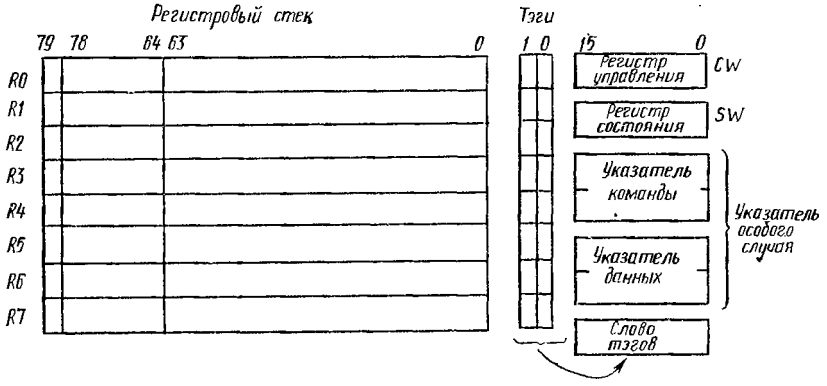


Рис. 4.2. Программная модель сопроцессора

Численное операционное устройство выполняет все команды обработки чисел, так или иначе связанные с внутренним регистровым стеком. В его составе есть несколько быстродействующих специализированных модулей, реализующих операции над мантиссами и порядками, а также параллельные сдвиги. Передачи чисел по внутренней шине данных осуществляются с очень высокой скоростью.

С точки зрения программиста сопроцессор можно считать просто расширением ЦП К1810ВМ86 в части регистров, допустимых форматов чисел и системы команд. Взаимодействие между ними на аппаратном уровне «невидимо» для программ.

Основу программной модели сопроцессора, показанной на рис. 4.2, образует регистровый стек из восьми 80-битных регистров R0—R7, иногда называемые арифметическими регистрами. В них хранятся числа, представленные в так называемом временном

вещественном формате (см. § 4.3). В любой момент времени трехбитное поле ST (Stack Top) в слове состояния определяет регистр, являющийся текущей вершиной стека и обозначаемый ST(0) или просто ST (в гл. 3 мы пользовались аббревиатурой TOS). Операция включения в стек (push) осуществляет декремент поля ST и загружает адресуемые данные в новую вершину стека. В операции извлечения из стека (pop) в получатель передается содержимое вершины стека, а затем производится инкремент поля ST. Таким образом, в стандартных стековых операциях поле ST выполняет функции традиционного указателя стека SP.

В организации регистрового стека сопроцессора имеется несколько отличий от «обычного» стека, который в большинстве современных процессоров аппаратно реализуется в оперативной памяти. Во-первых, стек имеет круговую (кольцевую) организацию: если поле ST содержит 000 и производится его декремент, новым содержимым ST будет 111, а если поле ST содержит 111 и осуществляется его инкремент, новым содержимым ST будет 000. Контроль за использованием стека должен вести программист. Во-вторых, в командах сопроцессора допускается явное или неявное обращение к регистрам стека с модификацией или без модификации поля ST. Явная адресация регистров осуществляется только относительно текущей вершины стека и обозначение ST (i) определяет i-й регистр в стеке, $0 \leqslant i \leqslant 7$, считая от ST (0). Примером может служить команда

```
ADD     ST,ST(5)     ST(0) <-- ST(0) + ST(5)
```

Наконец, во многих командах сопроцессора не выдерживаются обычные соглашения о стеке, т. е. о том, что любая стековая операция автоматически модифицирует указатель стека. Например, команда FST (сохранить в памяти) передает содержимое вершины стека в память, но не производит инкремента поля ST.

С каждым регистром стека ассоциирован двухбитный *тэг* (признак), а их совокупность для всех регистров образует слово тэгов. Тэг регистра R0 находится в младших битах слова, а тэг регистра R7 в старших битах. Тэг фиксирует наличие в регистре «обычного» числа — код 00, истинного нуля — код 01, специального числа — код 10 и отсутствие данных — код 11. В последнем случае регистр называется *пустым* и попытка извлечь число из пустого регистра фиксируется как особый случай недействительной операции. Аналогичный особый случай регистрируется и при попытке загрузить число в непустой регистр. Таким образом, наличие регистра тэгов позволяет сопроцессору обнаруживать особые случаи и эффективнее обрабатывать специальные числа.

Остальными регистрами в программной модели сопроцессора являются 16-битные регистр управления CW, регистр состояния SW, два регистра указателя команды и два регистра указателя

данных (операнда). Все эти регистры предназначены для системных программистов, разрабатывающих процедуры обработки особых случаев, а для большинства прикладных программистов наличие этих регистров можно безопасно игнорировать. Определенный интерес представляет регистр управления, содержимое которого задает режим работы сопроцессора, например, способ округления, точность вычислений и интерпретацию бесконечности. Однако для подавляющего большинства программ наиболее благоприятны режимы, задаваемые по умолчанию при аппаратной (сигналом сброса) или программной (командой FINIT) инициализации сопроцессора. В регистре состояния наибольший интерес представляют биты кода условия, в которых фиксируются особенности результата команд проверки, сравнения и анализа. С их помощью осуществляются передачи управления по результатам вычислений сопроцессора. Мы кратко рассмотрим функции этих регистров в § 4.5.

4.3. ФОРМАТЫ ЧИСЕЛ

Арифметический сопроцессор К1810ВМ87 оперирует числами трех классов: двоичные целые, упакованные десятичные целые и двоичные вещественные числа (числа с плавающей точкой). Может показаться неожиданным, что сопроцессор работает с целыми числами, так как их может обрабатывать ЦП. Но если исключить этот тип чисел, сопроцессор не смог бы вычислять выражения, в которых фигурируют целые и вещественные числа.

Форматы чисел показаны на рис. 4.3, а их основные характери-

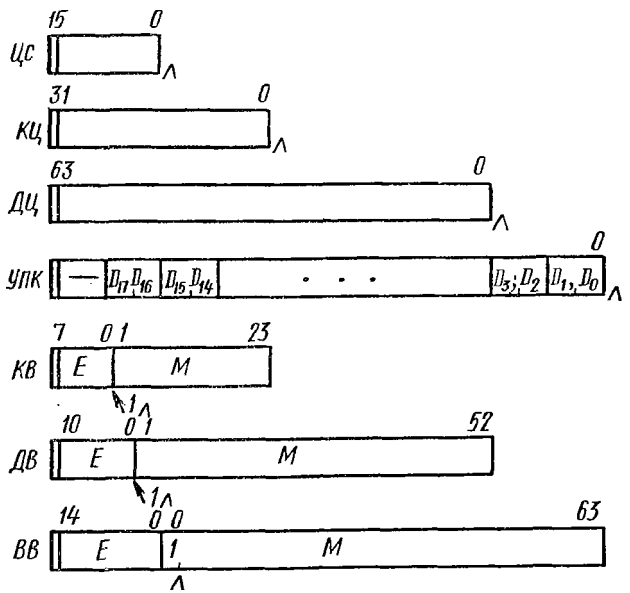


Рис. 4.3. Форматы чисел сопроцессора

стики (диапазон и точность) приведены в табл. 4.1. На рис. 4.3 знак вставки (∧) показывает позицию неявной точки, отделяющей целую часть числа от дробной. Во всех форматах старший бит отведен для знака числа со стандартным кодированием: «0» означает

Таблица 4.1. **Основные характеристики численных форматов**

Формат	Диапазон	Точность	Особенность
Целое слово	10^4	16 бит	Дополнительный код
Короткое целое	10^9	32 бит	То же
Длинное целое	10^{19}	64 бит	»
Упакованное десятичное	10^{18}	18 цифр	Прямой код
Короткое вещественное	$10^{\pm 38}$	24 бит	Неявный бит F_0
Длинное вещественное	$10^{\pm 308}$	53 бит	Неявный бит F_0
Временное вещественное	$10^{\pm 4932}$	64 бит	Явный бит F_0

плюс, а «1» — минус. На рис. 4.3 и в дальнейшем приняты следующие сокращения: ЦС — целое слово, КЦ — короткое целое, ДЦ — длинное целое, УПК — упакованное десятичное, КВ — короткое вещественное, ДВ — длинное вещественное и ВВ — временное вещественное.

Двоичные целые числа. Три формата целых двоичных чисел (ЦС, КЦ и ДЦ) отличаются только длиной, влияющей на диапазон представимых чисел. Только в этих форматах применяется стандартный дополнительный код. Число нуль имеет единственное представление 00...000 (положительный нуль). Формат целого слова соответствует основному формату чисел микропроцессора К1810ВМ86.

Упакованные десятичные целые числа. В этом формате каждый байт содержит две десятичные цифры в коде 8421. Старший бит левого байта отведен для знака числа, а остальные биты этого байта игнорируются, а при записи в память в них помещаются нули. Для десятичных чисел принят прямой код, который проще преобразовывать в последовательность символов для индикации по сравнению с дополнительным кодом. В прямом коде появляются два представления нуля — положительный и отрицательный нули, которые сопроцессор не различает. Младшая тетрада в старшем байте не используется, что объясняется стремлением к совместимости со стандартами некоторых языков программирования.

Отметим, что при наличии в тетрадах запрещенных комбинаций 1010—1111 результат операции с десятичным операндом не определен. Другими словами, сопроцессор не контролирует правильность десятичных цифр.

Вещественные числа. Для вещественных чисел применяется формат с плавающей точкой (КВ, ДВ и ВВ). Значащие цифры числа

находятся в поле мантиссы M, поле порядка E показывает фактическое положение двоичной точки в разрядах мантиссы, а бит знака S определяет знак числа. Мантисса, называемая также дробью F (Fraction), представлена в прямом коде.

Порядок E дается в смещенной форме:

$$E = \text{истинный порядок} + \text{смещение}.$$

Величина смещения для соответствующих форматов равна 127, 1023 и 16 383. Значение числа равно:

$$(-1)^S \times 2^{E-\text{смещение}} \times F_0.F_1F_2 \ldots F_n.$$

Представления чисел в коротком и длинном вещественных форматах удовлетворяют требованиям стандарта на арифметику с плавающей точкой (см. § 1.4). Отметим наличие в мантиссе бита единиц F_0. Сопроцессор обычно поддерживает представление мантиссы в нормализованной форме, т. е. ее старший бит F_0 равен 1. Следовательно, за исключением числа нуль мантисса состоит из целой части и дроби в виде 1. $F_1F_2 \ldots F_n$, где F_i равно 0 или 1. Благодаря нормализации устраняются старшие нули в числах, меньших единицы, что максимизирует количество значащих цифр мантиссы при ее фиксированной длине.

В коротком и длинном вещественных форматах бит F_0 при передачах чисел и хранении их в памяти отсутствует. Это так называемый скрытый или неявный бит, который в нормализованных числах содержит 1. Числа во временном вещественном формате имеют явный бит F_0. Такой формат позволяет повысить скорость выполнения операций. Числа во временном вещественном формате называются еще числами с расширенной точностью.

Покажем представление десятичного числа —247.375 в вещественных форматах сопроцессора. Двоичный код его равен —11110111.011 и истинный порядок +7. Смещенные порядки в трех вещественных форматах равны 134, 1030 и 16 390. С учетом бита F_0 имеем следующие представления:

	Знак	Порядок	Мантисса
Короткое вещественное	1	10000110	11101110110 ... 0
Длинное вещественное	1	10000000110	11101110110 ... 0
Временное вещественное	1	100000000000110	111101110110 ... 0

Независимо от исходного формата при загрузке числа из памяти в регистр сопроцессора оно автоматически преобразуется во временный вещественный формат, а при записи в память осуществляется обратное преобразование в формат получателя. Таким образом, временный вещественный формат является единственным внутренним форматом представления чисел, причем в нем абсолютно точно кодируются любые загружаемые из памяти числа. Числа во временном вещественном формате можно передавать в память;

это приходится делать для хранения промежуточных результатов из-за нехватки внутренних регистров.

Благодаря аппаратному преобразованию всех внешних форматов во временный вещественный формат программист может не заботиться с явных преобразованиях форматов. Умножение числа с плавающей точкой на упакованное десятичное число (после загрузки его в сопроцессор) осуществляется так же просто, как умножение целых чисел. Конечно, при возвращении результата в память программист должен обеспечить, чтобы формат получателя был достаточен для восприятия результата.

Разумеется, сопроцессор не может представить все вещественные числа из диапазонов своих форматов. Между любыми двумя соседними числами всегда существует промежуток, и результат операции может попасть именно в этот промежуток. В таких ситуациях сопроцессор скругляет истинный результат до числа, которое он может представить. Следовательно, вещественное число с большим количеством значащих цифр, чем допускает сопроцессор, будет представлено неточно. Как обычно, в формате с плавающей точкой, между любыми последовательными степенями двух находится одно и то же количество представимых чисел. Например, между 4 и 8 находится столько же представимых чисел, как и между 524 288 и 1 048 576. Другими словами, промежутки между представимыми числами расширяются по мере увеличения чисел. Однако целые числа из диапазона -2^{64}—$+2^{64}$ (примерно $\pm 10^{18}$) представляются во временном вещественном формате абсолютно точно.

Хранение чисел в памяти. В сопроцессоре принят такой же способ хранения чисел в памяти «младшее — по меньшему адресу», как и в МП К1810ВМ86. Логически во всех форматах левый бит является старшим, а правый младшим. В физической памяти первым, т. е. по меньшему адресу, хранится младший байт; адрес этого байта считается и адресом всего числа. Последним, т. е. по большему адресу, хранится старший байт. Передача данных обычно начинается с младшего байта.

Рекомендации по применению. В подавляющем большинстве применений для исходных данных и результатов рекомендуется использовать длинный вещественный формат. Он обеспечивает достаточные для получения правильных результатов диапазон и точность, требуя от программиста минимальных усилий. Короткий вещественный формат целесообразен в системах с ограничениями на память, но, конечно, он имеет меньшие диапазон и точность. Его удобно применять для отладки программ, так как ошибки округления проявляются в этом формате наиболее быстро. Временный вещественный формат не предназначен для представления входных и выходных данных, его следует применять для промежуточных результатов, в циклических фрагментах и для представления констант. Его фантастически огромный диапазон и высокая точность гарантируют защиту окончательных результатов от ошибок округ-

ления, а также уменьшают вероятность возникновения переполнений и антипереполнений. Как правило, такие ситуации свидетельствуют об ошибках в данных или программе.

4.4. СИСТЕМА КОМАНД СОПРОЦЕССОРА

Система команд сопроцессора содержит 69 базовых команд, которые удобно разделить на шесть групп: команды передач данных, арифметические команды, команды сравнения, команды трансцендентных операций, команды загрузки констант и команды управления сопроцессором.

Типичная команда воспринимает один или два операнда, выполняет свою операцию и формирует результат. Операндами наиболее часто служит содержимое регистров, но может привлекаться и содержимое ячеек памяти. Операнды некоторых команд определяются неявно, например «безоперандная» команда FSQRT извлечения квадратного корня имеет операндом содержимое вершины стека ST (0) и возвращает результат также в вершине стека, замещая операнд. Другие команды допускают или требуют явного задания операндов. Например, имеются команды с одним явным и одним неявным операндом, которым бывает содержимое вершины стека.

Команды бинарных операций допускают несколько альтернативных форм. В случае пустого поля операнда операция выполняется с двумя верхними элементами стека ST(0) и ST(1). После производства операции осуществляется инкремент указателя стека и результат помещается в новую вершину, т. е. замещает исходное содержимое ST (1). Другими словами, два операнда заменяются в стеке одним результатом; такое действие считается классической операцией в стековых машинах.

Когда в бинарной операции определен один операнд, она выполняется с привлечением указанного в команде регистра (или ячейки памяти) и содержимого вершины стека. Результат загружается в «старую» вершину стека и указатель стека не модифицируется.

Если же в бинарной операции указаны два операнда, ими является содержимое двух регистров стека ST(0) и ST(i). Всего получается три варианта интерпретации команды:

источником является ST(0), а получателем ST(i),

источником выступает ST(i), а получателем ST(0),

источником служит ST(0), получателем ST(i) и производится извлечение из стека, т. е. содержимое ST(0) удаляется из стека.

При программировании удобно наличие обратной формы команд вычитания и деления. Например, в обычной форме команды деления получатель делится на источник, а в обратной форме источник делится на получатель. Конечно, в обеих формах команд результат помещается в получатель. Благодаря обратной форме команд операции вычитания и деления становятся симметричными.

Таким образом, многие команды, в частности команды всех ариф-

метических операций, допускают несколько способов задания операндов. Например, команда FADD сложения вещественных чисел разрешает запись без операндов, только с источником или с источником и получателем. Альтернативные формы операндов условно показываются с помощью наклонной или косой черты (ее также называют чертой деления). Например, команда FADD имеет следующий общий вид:

```
FADD    //src /dst,src
```

Такое обозначение подразумевает три возможных формы команды: без операндов, с одним источником, с получателем и источником.

В мнемониках команд приняты следующие соглашения:

первая буква всегда F (Floating); она позволяет легко идентифицировать команды сопроцессора в программе;

вторая буква I (Integer) обозначает операцию с целым двоичным числом, буква B (Binary-coded decimal) — операцию с десятичным операндом, а «пустая» вторая буква определяет операцию с вещественными числами;

предпоследняя или последняя буква R (Reverse) указывает обратную операцию;

последняя буква P (Popping) показывает команду, заключительным действием которой является извлечение из стека.

Для программирования систем с сопроцессором следует расширить стандартный ассемблер МП К1810ВМ86 средствами поддержки сопроцессора. Расширенные форматы данных сопроцессора требуют введения в ассемблер специальных директив определения данных. Для резервирования и инициализации памяти для переменных и констант сопроцессора применяются директивы:

DW — определить слово (16 бит);

DD — определить двойное слово (32 бит);

DQ — определить счетверенное слово (64 бита);

DT — определить десять байтов (80 бит).

Иногда программисту необходимо воспользоваться командой, тип операнда которой не объявлен директивой. Но поскольку ассемблер все же должен знать тип операнда, его можно задать в команде с помощью атрибутного оператора PTR. Например, в команде

```
FLD     [BX][SI]
```

не ясно, какое вещественное число загружается из памяти. В команде же с указателем

```
FLD     QWOORD PTR [BX][SI]
```

определена загрузка длинного вещественного числа.

Ассемблер не контролирует тип операнда в командах управления сопроцессором, так как в них структура операнда подразумевается смыслом команды. Например, команда восстановления полного состояния сопроцессора FRSTOR [BP] предполагает, что регистр BP адресует 94-байтную область в сегменте стека, в которой ранее было запомнено полное состояние сопроцессора.

4.4.1. КОМАНДЫ ПЕРЕДАЧ ДАННЫХ

Команды этой группы производят передачи данных между регистрами стека, а также между вершиной стека и памятью. Одной командой число из памяти, представленное в любом формате сопроцессора, преобразуется во временный формат и загружается (включается) в стек; аналогичным образом, но в обратном порядке, осуществляется передача числа в память. При выполнении команд автоматически модифицируется тэг регистра, отражая его новое содержимое.

Команды загрузки. Три команды загрузки имеют следующий вид:

вещественное:	FLD	*src*	ST←(ST)—1, ST(0)←(*src*)
двоичное целое:	FILD	*src*	
десятичное целое:	FBLD	*src*	

Как видно из общего описания, эти команды осуществляют декремент указателя стека и передачу в новую вершину стека содержимого источника, т. е. производят включение в стек. Для предотвращения переполнения стека перед загрузкой проверяется тэг регистра. Если регистр не отмечен как пустой, генерируется особый случай недействительной операции и загрузка не производится.

В команде FLD источником может быть один из регистров стека или вещественное число с любым форматом в памяти, а в командах FILD и FBLD — только операнд в памяти. При выполнении всех команд (за естественным исключением команд вида FLD ST(*i*) и с операндом в памяти во временном вещественном формате) производится автоматическое преобразование операнда во временный вещественный формат с сохранением специальных чисел.

Команды запоминания. Две команды запоминания

вещественное:	FST	*dst*	*dst*←ST(0)
двоичное целое:	FIST	*dst*	

производят передачу содержимого вершины стека в память без модификации указателя стека ST и, разумеется, содержимого ST(0).

В команде FST получателем может быть регистр стека или вещественная переменная в памяти (только короткий или длинный формат). Мантисса округляется, а порядок корректируется с учетом длины и смещения порядка получателя.

В команде FIST получателем является переменная в памяти, имеющая формат короткого целого или целого слова. Команда округляет содержимое вершины стека до целого и передает результат в получатель. При наличии в ST(0) отрицательного нуля он будет запомнен как положительный нуль в виде 00...00.

Команды запоминания с извлечением из стека. Эти команды помимо передачи содержимого ST(0) в получатель осуществляют извлечение из стека: регистр, бывший вершиной стека, отмечается как пустой и производится инкремент указателя стека:

вещественное:	FSTP *dst*	*dst*←ST(0), ST←(ST)+1
двоичное целое:	FISTP *dst*	
десятичное целое:	FBSTP *dst*	

Чтобы предотвратить антипереполнение (опустошение) стека, перед запоминанием проверяется тэг регистра, являющегося вершиной стека. Если тэг показывает неинициализированное значение (пустой регистр), генерируется особый случай недействительной операции.

Действия команды FSTP очень похожи на действия команды FST с добавлением, конечно, извлечения из стека. Однако она позволяет передать в память число в любом вещественном формате, даже во временном вещественном формате, чего не может делать команда FST.

Команда FISTP, похожая на команду FIST, обеспечивает передачу в память числа в любом формате целого двоичного, включая и длинное целое. Последний формат недопустим в команде FIST. Некоторая асимметричность команд сопроцессора объясняется недостатком двоичных наборов для кодов операций.

Наконец, команда FBSTP преобразует операнд из вершины стека в упакованное десятичное число, передает его в память и производит извлечение из стека. Округление реализуется посредством прибавления к исходному числу 0.5 и последующего отбрасывания разрядов дробной части.

В командах запоминания обоих видов при преобразовании формата может возникнуть несколько особых случаев. Если в процессе округления число изменяется, генерируется особый случай точности. Когда округленное число слишком велико для формата получателя, фиксируется особый случай переполнения. Наконец, при преобразовании в короткий или длинный вещественный формат возникает особый случай антипереполнения, когда округленное число не является нулем, но меньше порога антипереполнения соответствующего формата.

Команда обмена. Команда обмена содержимого регистров

```
FXCH  //dst        ST(0) <---> (dst)
```

обменивает содержимое получателя ST(*i*) и вершины стека ST(0). В случае пустого поля операнда обменивается содержимое регист-

ров ST(1) и ST(0). Наличие команды FXCH объясняется тем, что многие команды сопроцессора оперируют содержимым вершины стека, а с помощью команды обмена их действия можно распространить на все регистры стека. Отметим, что команда FXCH ST(0) эквивалентна холостой команде.

4.4.2. АРИФМЕТИЧЕСКИЕ КОМАНДЫ

Набор арифметических команд сопроцессора включает в себя разнообразные варианты основных арифметических операций, а также удобные команды извлечения квадратного корня, масштабирования, выделения частей вещественного числа и др. В табл. 4.2 приведены допустимые комбинации операций и операндов для основных арифметических операций. В дополнение к четырем обычным операциям команды двух обратных операций делают вычитание и деление «симметричными», как сложение и умножение.

Таблица 4.2. **Формы основных арифметических команд**

Форма команды	Мнемоника	Операнды	Пример записи
Стековая	F*op*	{ST(1), ST}	FSUB
Регистровая	F*op*	ST(*i*), ST или ST, ST(*i*)	FADD ST, ST(2)
Регистровая с извлечением из стека	F*op*P	ST(*i*), ST	FMULP ST(3), ST
Вещественные операнды (с памятью)	F*op*	{ST,} короткое/длинное вещественное	FADD BETA
Целые операнды (с памятью)	FI*op*	{ST,} целое слово / короткое целое	FIDIV GAMMA

Примечания: 1. Фигурные скобки обозначают неявные операнды, которые в ассемблерных командах могут не указываться. 2. Обозначения

```
op = ADD   {dst,src}    dst <-- (dst) + (src)
     SUB   {dst,src}    dst <-- (dst) - (src)
     SUBR  {dst,src}    dst <-- (src) - (dst)
     MUL   {dst,src}    dst <-- (dst) * (src)
     DIV   {dst,src}    dst <-- (dst) / (src)
     DIVR  {dst,src}    dst <-- (src) / (dst)
```

Приведенные в табл. 4.2 пять основных форм команд могут быть использованы для всех операций.

Стековая форма превращает сопроцессор в классическую стековую машину. В этой форме поле операнда пустое: под источником подразумевается вершина стека ST(0), а получателя — следующий регистр стека ST(1). Выполнив операцию, сопроцессор производит

инкремент указателя стека и загружает результат в новую вершину стека.

Регистровая форма представляет собой обобщение предыдущей: одним из операндов является содержимое вершины стека, вторым — произвольный регистр стека, а результат можно загрузить на место любого из операндов. Указание получателем вершины стека обеспечивает удобный доступ к константам из других регистров. Когда вершина стека служит источником, регистр-получатель превращается в аккумулятор.

Часто операнд, находящийся в вершине стека ST(0), необходим только для одной операции, а в дальнейшем не требуется. Регистровая форма с извлечением из стека обеспечивает выбор вершины стека в качестве источника и последующее уничтожение этого операнда посредством инкремента указателя стека. Задание операндов в виде ST(1), ST(0) с мнемоникой извлечения из стека эквивалентно классической стековой операции: содержимое вершины стека удаляется, а результат помещается в новую вершину стека.

Две формы команд с обращением к памяти позволяют использовать как источник вещественное или целое число, находящееся в памяти. Это удобно в тех ситуациях, когда операнды привлекаются редко и их хранение в регистрах нецелесообразно. Для указания таких операндов применяются все режимы адресации памяти ЦП. Необходимо отметить, что вещественные числа в памяти не могут быть в формате временного вещественного, а целые числа — в формате длинного целого. Здесь вновь сказывается недостаточность наборов кодов операций.

Команды сложения. Операция сложения реализуется командами со следующими формами:

вещественные числа:	FADD //*src*/*dst*, *src*
вещественные числа (с извлечением из стека):	FADDP *dst*, *src*
целые числа:	FIADD *src*

Отметим, что команда FADD ST, ST(0) удваивает содержимое вершины стека.

Команды вычитания. Обычное вычитание *dst*←(*dst*)—(*src*) осуществляют команды:

вещественные числа:	FSUB//*src*/*dst*, *src*
вещественные числа (с извлечением из стека):	FSUBP *dst*, *src*
целые числа:	FISUB *src*

Для производства обратного вычитания *dst*←(*src*)—(*dst*) предназначены команды FSUBR, FSUBRP и FISUBR, имеющие аналогичные формы.

Команды умножения. Операция умножения реализуется следующими командами:

вещественные числа:	FMUL//*src*/*dst*, *src*
вещественные числа (с извлечением из стека):	FMULP *dst*, *src*
целые числа:	FIMUL *src*

Команды деления. Для выполнения операции обычного деления предусмотрены команды:

вещественные числа:	FDIV *//srs/dst, src*
вещественные числа (с извлечением из стека):	FDIVP *dst, src*
целые числа:	FIDIV *src*

Соответствующие команды обратного деления FDIVR, FDIVRP и FIDIVR загружают в получатель частное от деления источника на получатель.

Хотя сопроцессор имеет полный набор команд целочисленной двоичной арифметики, применять их в «массовом» порядке не рекомендуется, так как быстродействие сопроцессора в такой арифметике значительно ниже быстродействия ЦП. Например, МП К1810ВМ86 выполняет 16-битное сложение память — регистр примерно за 20 тактов синхронизации, а аналогичная команда сопроцессора занимает 120 тактов. Объясняется такая несколько парадоксальная ситуация тем, что в сопроцессоре для обработки целых чисел применяется временный вещественный формат, что требует соответствующего преобразования операнда (и обратного преобразования при передаче результата в память). Преимущества сопроцессора при обработке таких «малых» для него чисел проявляются в вычислениях с привлечением вещественных и целых чисел. Благодаря тому, что сопроцессор представляет любые числа в одном и том же внутреннем формате, в вычислениях могут участвовать числа различных форматов.

Сделаем замечание о явном и неявном указании регистровых операндов. В стековой форме ST(0) всегда подразумевается источником, а ST(1) получателем, и они в командах явно не фигурируют. Отсутствующие операнды могут вызвать некоторую путаницу у неопытного программиста. Дело в том, что команда с двумя неявными операндами имеет другой смысл, чем та же самая команда с явными операндами. По соглашению два неявных операнда сообщают ассемблеру о том, что после выполнения операции следует произвести извлечение из стека. Например, команда FADD подразумевает источником ST(0) и получателем ST(1). Ассемблер транслирует эту команду как FADDP ST (1), ST, которая отличается от команды FADD ST (1), ST. Поэтому целесообразно, хотя бы на первых порах, явно указывать в командах оба операнда.

Отметим также, что во всех арифметических операциях могут возникать особые случаи. Уникальна в этом отношении операция деления, в которой могут возникнуть все шесть особых случаев, фиксируемых сопроцессором.

Дополнительные команды. К арифметическим командам относятся также семь дополнительных команд, имеющих «безоперандную» форму.

Команда FSQRT извлечения квадратного корня заменяет число, находящееся в вершине стека, значением квадратного корня:

```
FSQRT          ST(0) <---  √ST(0)
```

В этой команде, по определению, полагается, что $\sqrt{-0}=-0$. Относительно команды FSQRT отметим следующее. Во-первых, она выполняется несколько быстрее команд деления, так как здесь не нужно контролировать переполнение и антипереполнение. Во-вторых, точность ее соответствует точности обычных арифметических операций (ошибка результата равна 1/2 младшего бита мантиссы). Наконец, в команде FSQRT доступны режимы округления.

Команда FSCALE масштабирования интерпретирует содержимое регистра ST(1) как целое двоичное число и прибавляет его к смещенному порядку числа, находящегося в вершине стека:

```
FSCALE         ST(0) <--- ST(0) * 2^ST(1)
```

Таким образом, команда FSCALE осуществляет быстрое умножение (когда $ST(1)>0$) или деление (когда $ST(1)<0$) содержимого вершины стека на целую степень двух.

В этой команде предполагается, что масштабный коэффициент в ST(1) является целым числом из диапазона $-2^{15}<ST(1)<2^{15}$. Если же он не удовлетворяет этому ограничению, но находится в указанном диапазоне и больше по абсолютной величине 1, в команде принимается ближайшее целое, меньшее по абсолютной величине исходного масштабного коэффициента. Другими словами, команда FSCALE производит усечение к нулю. Когда же число в ST(1) находится вне допустимого диапазона или является правильной дробью, команда формирует непредсказуемый результат и не сигнализирует об особом случае. Поэтому во избежание возможных ошибок рекомендуется всегда позаботиться о задании масштабного коэффициента в виде целого слова. Нахождение масштабного коэффициента не в вершине стека, а в регистре ST(1) обеспечивает удобное масштабирование последовательности чисел, например элементов массива. На каждый элемент массива требуют три операции (загрузка, масштабирование и запоминание) без дополнительных действий.

Команда FPREM вычисляет частичный остаток (смысл этого термина будет понятен из описания команды) от деления числа, находящегося в вершине стека ST(0), на следующий элемент стека ST(1) и загружает результат в ST(0):

```
FPREM          ST(0) <--- ST(0) - (q * ST(1))
```

где q — целое число. Другими словами, здесь содержимое ST(1) выступает модулем в операции деления. Знак остатка совпадает со знаком исходного делимого.

Команда FPREM предназначена в основном для приведения аргумента (операнда) периодических трансцендентных функций в диапазон, допустимый в соответствующих командах сопроцессора. Например, команда FPTAN вычисления частичного тангенса требует, чтобы аргумент находился в диапазоне от нуля до $\pi/4$. Для повышения точности вычисления функций необходимо, чтобы команда FPREM давала точный результат. Этого можно достичь только путем последовательных масштабированных вычитаний модуля из делимого до достижения момента, когда вычитание без получения отрицательной разности невозможно, т. е. когда очередная разность меньше модуля. Такой способ требует значительного времени в тех ситуациях, когда исходное делимое намного больше модуля с соответствующей задержкой ЦП. Чтобы предотвратить «зависание» ЦП в продолжительной операции без реакции на запросы прерываний, когда FPREM рассчитана на повторяющееся (итеративное) выполнение в программном цикле. Она производит максимум 64 вычитания и возвращает полученный при этом результат, даже если необходимы дальнейшие вычитания.

Когда команда FPREM дает остаток, меньший модуля, ее функция считается законченной и бит C2 кода условия в слове состояния сопроцессора будет содержать нуль. Если же приведение не закончено, бит C2 содержит 1 и результат в вершине стека ST(0) называется *частичным остатком*. Программа должна проверить состояние C2 после выполнения команды FPREM и при необходимости инициировать ее повторное выполнение с использованием в качестве делимого частичного остатка из ST(0).

Проверка состояния бита C2 осуществляется с привлечением ЦП: слово состояния передается через память в регистр AX, затем команда SAHF пересылает код условия в регистр флажков и состояние бита C2 показывает флажок PF. Другой способ определения завершения команды FPREM заключается в сравнении ST(0) и ST(1): остаток получен, если ST(0) $<$ ST(1).

Кроме остатка в ST(0) команда FPREM образует в битах C3, C1, C0 слова состояния три младших бита частного. Фактически по своим весам они упорядочены как C0, C3, C1. Наличие трех битов частного позволяет определить нахождение исходного угла в одном из октантов. Следует иметь в виду, что точные младшие биты частного получаются при производстве в команде FPREM не более 62 вычитаний.

Следующая команда FRNDINT осуществляет округление числа, находящегося в вершине стека, до целого. Режим округления задан соответствующим полем в слове управления сопроцессора.

Команда FXTRACT выделения компонент числа с плавающей точкой преобразует число, находящееся в вершине стека ST(0), в два числа, представляющие собой фактические значения его порядка и мантиссы. Выделенный порядок заменяет исходный операнд в вершине стека, а мантисса включается в стек. После выполнения

команды в ST(0) находится вещественное число, знак и мантисса которого равны знаку и мантиссе исходного операнда, а истинный порядок равен нулю (смещенный порядок равен 16383). В регистре ST(1) находится истинный порядок исходного операнда, также выраженный в формате вещественного числа. Когда операнд равен нулю, команда FXTRACT образует нули в ST(0) и ST(1) со знаком исходного числа.

Две последние команды выполняют элементарные операции нахождения абсолютного значения и изменения знака числа, которое содержится в вершине стека:

```
FABS        ST(0) <--- |ST(0)|
FCHS        ST(0) <--- -ST(0)
```

4.4.3. КОМАНДЫ СРАВНЕНИЯ

Команды этой группы предназначены для анализа числа в вершине стека (иногда по отношению к другому числу) и формирования кода условия в слове состояния сопроцессора. К основным операциям относятся сравнение, проверка (или сравнение с нулем) и анализ (получение подробной информации о числе). Имеются специальные формы команд, допускающие сравнение с целым и вещественным числами, находящимися в памяти, и извлечения из стека после сравнения. Проверить образованный код условия может только ЦП.

Команда сравнения вещественных чисел имеет форму FCOM // *src* и осуществляет сравнение содержимого вершины стека ST (0) и источника *src*. Источником может быть регистр стека или вещественное число в памяти (в формате короткого или длинного вещественного). Если поле операнда пустое, производится сравнение ST (0) и ST (1). Код условия отражает отношение между числами-операндами в соответствии с табл. 4.3. Операнды считаются несравнимыми (C3, C0=11), когда хотя бы один из них является специальным числом.

Команда FCOMP // *src* сравнения и извлечения из стека действует аналогично команде FCOM, но дополнительно осуществляет извлечение из стека. Следующая команда FCOMPP сравнения ST (0) и ST (1) и двойного извлечения из стека похожа на команду FCOMP ST (1), но дополнительно она производит еще одно извлечение из стека, так что оба операнда оказываются «уничтоженными».

При выполнении команды FICOM содержимое источника (память), интерпретируемое как целое слово или короткое целое, преобразуется во временной вещественный формат и сравнивается с ST (0). Команда FICOMP *src* производит такие же действия и дополнительно реализует извлечение из стека.

Таблица 4.3. **Интерпретация кода условия**

Команда	C3	C2	C1	C0	Смысл
Сравнение и проверка (FCOM, FCOMP, FCOMPP, FTST)	0	X	X	0	(ST) > источника (*src*) или 0
	0	X	X	1	(ST) < источника (*src*) или 0
	1	X	X	0	(ST) = источнику (*src*) или 0
	1	X	X	1	Не сравнимы
Анализ (FXAM)	0	0	0	0	Положительное, ненормализованное
	0	0	0	1	Положительное, не-число
	0	0	1	0	Отрицательное, ненормализованное
	0	0	1	1	Отрицательное, не-число
	0	1	0	0	Положительное, нормализованное
	0	1	0	1	Положительная бесконечность
	0	1	1	0	Отрицательное, нормализованное
	0	1	1	1	Отрицательная бесконечность
	1	0	0	0	Положительный нуль
	1	0	0	1	Пустой регистр
	1	0	1	0	Отрицательный нуль
	1	0	1	1	Пустой регистр
	1	1	0	0	Положительное, денормализованное
	1	1	0	1	Пустой регистр
	1	1	1	0	Отрицательное, денормализованное
	1	1	1	1	Пустой регистр

Команда FTST производит проверку числа, находящегося в вершине стека, посредством сравнения его с нулем. Результат проверки фиксируется в коде условия в соответствии с табл. 4.3, но с заменой источника (*src*) в команде сравнения на нуль.

Последняя в этой группе команда FXAM анализа содержимого вершины стека формирует в битах C3 — C0 кода условия подробное сообщение об особенностях операнда (см. табл. 4.3).

4.4.4. КОМАНДЫ ТРАНСЦЕНДЕНТНЫХ ФУНКЦИЙ

Команды настоящей группы выполняют базовые вычисления, относящиеся к тригонометрическим, обратным тригонометрическим, логарифмическим и показательным функциям. Операнды команд находятся в одном или двух верхних регистрах стека и результат также возвращается в стеке.

Предполагается, что операнды являются нормализованными числами и находятся в допустимом для каждой из команд диапазоне. Ответственность за удовлетворение этих требований возлагается на программиста. При нарушении их результат операции

непредсказуем; более того, сопроцессор не оповещает об этом никаким особым случаем.

Команда FPTAN вычисления частичного тангенса, как результат, формирует два числа X и Y, отношение которых дает тангенс угла α в виде $\operatorname{tg}\alpha = Y/X$. Значение угла α в радианах должно находиться в вершине стека ST (0) и быть в диапазоне от нуля до $\pi/4$ $(0<\alpha<\pi/4)$. После выполнения команды FPTAN значение Y замещает аргумент, а значение X включается в стек. Допустимый диапазон угла в команде FPTAN исключает нуль, поэтому $\operatorname{tg}0$ должен фиксироваться программой и вычисляться как специальный случай. Особых сложностей это не вызывает, так как для малых α справедливо приближенное равенство $\operatorname{tg}\alpha \approx \alpha$.

Несколько необычное представление результата команды FPTAN (термин «частичный» показывает необходимость дополнительной команды для получения истинного тангенса) предназначено для удобного вычисления остальных тригонометрических функций, которые получаются из значений X и Y на основе тригонометрических тождеств.

Команда FPATAN вычисления частичного арктангенса формирует результат $\alpha = \operatorname{arctg}(Y/X)$, причем значение X берется из вершины стека ST (0), а значение Y — из регистра ST (1). Значения исходных операндов должны удовлетворять требованию $0<Y<X<\infty$, которое совместимо с результатами команды FPTAN. При выполнении команды происходит извлечение из стека значений X и Y, а затем результат (значение α в радианах) помещается в новую вершину стека, замещая собой Y.

Обе тригонометрические команды FPTAN и FPATAN оказываются очень точными (ошибка результата составляет несколько единиц младшего разряда) и выполняются довольно быстро — всего в 3—4 раза медленнее деления. Эти команды, а также команда FSQRT образуют основу для вычисления всех остальных тригонометрических и обратных тригонометрических функций. Если, например, обозначить через z значение в ST (0) до выполнения команды FPTAN, а через X и Y — значения в ST (0) и ST (1) после деления z на 2 и последующего выполнения команды FPTAN, то

$$\sin z = \frac{2(Y/X)}{1+(Y/X)^2}, \quad \cos z = \frac{1-(Y/X)^2}{1+(Y/X)^2},$$

$$\operatorname{tg}(z/2) = Y/X, \quad \operatorname{ctg}(z/2) = X/Y,$$

$$\sec z = \frac{1+(Y/X)^2}{1-(Y/X)^2}, \quad \operatorname{cosec} z = \frac{1+(Y/X)^2}{2(Y/X)}.$$

Здесь тригонометрические функции выражены через $\operatorname{tg}(z/2)$, а не $\operatorname{tg} z$, так как при этом уменьшается ошибка округления.

Если обозначить через z аргумент обратной тригонометриче-

ской функции f, через X и Y значения в ST (0) и ST (1) до выполнения команды FPATAN, то получение в ST (0) значений $f(z)$ осуществляется по следующим формулам:

$$\arcsin z = \operatorname{arctg}\left(\frac{z}{\sqrt{(1-z)(1+z)}}\right) = \operatorname{arctg}(Y/X),$$

$$\arccos z = 2\operatorname{arctg}\left(\sqrt{\frac{1-z}{1+z}}\right) = 2\operatorname{arctg}(Y/X),$$

$$\operatorname{arctg} z = \operatorname{arctg}(z/1) = \operatorname{arctg}(Y/X),$$

$$\operatorname{arcctg} z = \operatorname{arctg}(1/z) = \operatorname{arctg}(Y/X),$$

$$\operatorname{arcsec} z = \operatorname{arctg}\left(\frac{1}{\sqrt{(z-1)(z+1)}}\right) = \operatorname{arctg}(Y/X),$$

$$\operatorname{arccosec} z = 2\operatorname{arctg}\left(\sqrt{\frac{z-1}{z+1}}\right) = 2\operatorname{arctg}(Y/X).$$

Отметим различие в вычислениях тригонометрических и обратных тригонометрических функций. Первые находятся с первоначального выполнения команды FPTAN с последующими операциями над операциями над двумя числами X и Y, полученными этой командой. Во вторых функциях вначале производятся операции над аргументом, а затем выполняется команда FPATAN над двумя результатами этих операций. Например, для вычисления $\arcsin z$ необходимы следующие действия:

вычислить $X = \sqrt{(1-z)(1+z)}$ и $Y = z$,
включить Y и X в стек,
выполнить команду FPATAN.

Команда с несколько необычной мнемоникой F2XM1 вычисляет значение функции $Y = 2^X - 1$. В мнемонике как раз и завуалирована функция команды: два в степени X минус 1. Значение X берется из вершины стека ST (0) и должно находиться в диапазоне $0 \leqslant X \leqslant 0.5$. Результат операции замещает значение X в вершине стека.

На первый взгляд, более естественной кажется команда, вычисляющая 2^X вместо $2^X - 1$. Однако команда F2XM1 позволяет получить очень точный результат, когда аргумент близок к нулю. Например, значение $2^{0.000001}$ приблизительно равно 1.000000693. Нетрудно убедиться в том, что вычитание единицы сохраняет в результате больше значащих цифр.

Можно осуществить возведение в степень X любых чисел, пользуясь формулами:

$$10^X = 2^{X\log_2 10},\quad e^X = 2^{X\log_2 e},\quad Y^X = 2^{X\log_2 Y}.$$

Необходимые для таких вычислений константы $\log_2 10$ и $\log_2 e$

встроены в сопроцессор, а рассматриваемая ниже команда вычисляет двоичный логарифм любого числа.

Команда FYL2X предназначена для вычисления значения функции $Z=Y\log_2 X$. Аргумент X находится в вершине стека ST (0), а аргумент Y — в регистре ST (1). Диапазоны аргументов составляют $0<X<\infty$, $-\infty<Y<\infty$. Команда производит извлечение из стека аргументов и загружает Z в новую вершину стека, заменяя Y. С помощью этой команды удобно вычисляются логарифмы чисел по любому основанию с применением тождества $\log_n X = \log_n 2 \times \log_2 X$.

Последняя из трансцендентных команд FYL2XP1 вычисляет значение функции $Z=Y\log_2(X+1)$. Аргумент X берется из вершины стека ST (0) и должен находиться в диапазоне $0<|X|<1-\sqrt{2}/2$, а аргумент Y из регистра ST (1) и должен быть в диапазоне $-\infty<Y<\infty$. Команда производит извлечение аргументов из стека и загружает Z в новую вершину стека, т. е. на место аргумента Y.

Команда FYL2XP1 обеспечивает большую точность по сравнению с предыдущей командой при вычислении логарифмов чисел, близких к единице. Задание как аргумента функции числа ε ($\varepsilon \ll 1$) вместо $(1+\varepsilon)$ позволяет сохранить в аргументе больше значащих цифр.

4.4.5. КОМАНДЫ ЗАГРУЗКИ КОНСТАНТ

Простые команды загрузки наиболее часто встречающихся в вычислениях констант приведены в табл. 4.4. Загрузка осуществляется путем включения константы в стек, т. е. декремента указателя стека, и передачи в новую вершину стека значения константы, представленного во временном вещественном формате. Так как в этом формате каждая из констант занимала бы десять байтов памяти, двухбайтные команды загрузки констант обеспечивают экономию памяти, повышение производительности и, кроме того, упрощают программирование.

Таблица 4.4. **Команды загрузки констант**

Мнемоника	Операция	Мнемоника	Операция
FLDZ	Загрузить +0.0	FLDL2E	Загрузить $\log_2 e$
FLD1	Загрузить +1.0	FLDLG2	Загрузить $\log_{10} 2$
FLDPI	Загрузить π	FLDLN2	Загрузить $\log_e 2$
FLDL2T	Загрузить $\log_2 10$		

4.4.6. КОМАНДЫ УПРАВЛЕНИЯ СОПРОЦЕССОРОМ

Не будем подробно рассматривать команды этой группы, ориентированные на операции системного уровня. Для прикладных программистов наибольший интерес представляют команды, оперирующие словами управления и состояния, а также команда инициализации процессора. С помощью этих команд можно задать режим работы сопроцессора, а также проанализировать результаты команд сравнения и проверки.

Команда FINIT инициализации сопроцессора функционально эквивалентна сигналу сброса CLR, но она не влияет на синхронизацию выборки команд ЦП и сопроцессора. Состояние сопроцессора после команды инициализации приведено в табл. 4.5.

Таблица 4.5. **Состояние сопроцессора после команды инициализации**

Поле	Состояние	Интерпретация
Слово управления		
Управление бесконечностью	0	Проектированная
Управление точностью	11	64 бита
Управление округлением	00	Округление к ближайшему
Маска разрешения прерываний	1	Прерывания запрещены
Маски особых случаев	111111	Замаскированы
Слово состояния		
Занятость	0	Не занят
Код условия	XXXX	Не определен
Указатель стека	000	Начало стека
Запрос прерывания	0	Отсутствует
Флажки особых случаев	000000	»
Слово тэгов		
Тэги	11	Пустые регистры
Регистры и указатели особого случая		Не изменяются

Основное назначение команды FLDCW *src* заключается в том, чтобы загрузить в регистр управления сопроцессора новое содержимое из источника *src* (им должно быть целое слово в памяти) с целью установки или изменения режима работы, например режима управления округлением.

Команды FSTCW *dst* и FSTSW *dst* осуществляют запоминание текущих слов управления и состояния соответственно в ячейке памяти, определяемой получателем *dst*. Программист довольно

часто пользуется командой FSTSW для реализации условных переходов по результатам команд сопроцессора, определения занятости сопроцессора, а также вызова процедур обработки особых случаев в таких системах, где не применяются прерывания. После этой команды следует указать команду ожидания WAIT, чтобы состояние сопроцессора, в частности код условия, было передано в память до продолжения программы. Получатель *dst* должен иметь формат целого слова.

4.5. СПЕЦИАЛЬНЫЕ ЧИСЛА И ОСОБЫЕ СЛУЧАИ

В большинстве компьютеров каждый двоичный набор в конкретном формате является действительным (допустимым) числом. Однако в сопроцессоре К1810ВМ87 большой класс двоичных наборов зарезервирован для представления специальных чисел и даже не-чисел, например значений неинициализированных переменных в программе. Прикладные программисты могут безопасно игнорировать наличие этих наборов. Для специальных чисел в вещественных форматах отведены максимальный (11 ... 11) и минимальный (00 ... 00) смещенные порядки.

Вещественные числа с нарушением нормализации. Ненулевые вещественные числа обычно хранятся в нормализованной форме, т. е. старший бит F_0 мантиссы содержит 1. Следовательно, требование нормализации предполагает наличие в каждом формате минимального представимого числа с ненулевым смещенным порядком (далее в силу симметрии положительных и отрицательных чисел будем обозначать знак символом X):

$$\delta: X\ 00\ldots01 \quad 1\_00\ldots00$$

В вычислениях могут возникать столь малые числа (меньшие δ), что их смещенный порядок должен быть отрицательным. Такая ситуация называется исчезновением порядка или антипереполнением. В большинстве компьютеров оно ведет к возвращению нуля как результата операции. Однако сопроцессор сдвигает мантиссу вправо с одновременным инкрементом порядка до тех пор, пока он не будет равным нулю. Такие числа с нулевым смещенным порядком (истинный порядок равен минимальному ненулевому значению —126, —1022 и —16382 для различных вещественных форматов) и ненулевой мантиссой называются *денормализованными*. Конечно, каждый старший нуль в мантиссе ведет к потере значимости, поэтому расширение диапазона в область малых чисел сопровождается потерей точности. Наличие в регистре денормализованного числа отмечается тэгом специального числа.

При загрузке из памяти или использовании как операнда денормализованное число превращается в эквивалентное ненормализованное число (с ненулевым смещенным порядком) путем сдвига мантиссы вправо и инкремента порядка. Следовательно,

денормализованные числа допустимы в арифметических операциях, хотя трансцендентные команды предполагают без проверки нормализованные операнды, а при нарушении нормализации возвращают непредсказуемый результат.

В сопроцессоре допускается еще один вид чисел с нарушением нормализации. Это *ненормализованные* числа, в которых имеется один или несколько старших нулей в мантиссе и произвольный, но не равный нулю и максимальному значению порядок. Такие числа существуют только во временном вещественном формате и распознаются по нулю в старшем бите F_0 мантиссы и ненулевому смещенному порядку. Ненормализованные числа допускаются во всех арифметических операциях, причем по возможности результат операции нормализуется.

Для программиста наличие ненормализованных и денормализованных чисел удобно, так как иногда в вычислительных алгоритмах возникают очень малые промежуточные результаты. Вместо сигнализации об антипереполнении сопроцессор разрешает продолжить выполнение программы с поддержанием максимально возможной точности. Расширение диапазона представимых чисел на ненормализованные и денормализованные числа называется постепенным или плавным антипереполнением.

Число нуль имеет нулевой смещенный порядок и нулевую мантиссу (так называемый *истинный* нуль) и обычно не учитывается как специальное число. Однако необходимо знать, как сопроцессор выполняет операции с нулевыми операндами. Вещественный нуль может иметь положительный или отрицательный знак, но в операциях (кроме деления) знак нуля игнорируется. Некоторые операции с нулевыми операндами сопроцессор выполняет необычно; например, результатом деления X/0 будет бесконечность (X — ненулевое конечное число), а результатом 0/0 будет неопределенность (см. далее). Отметим, что при необходимости с помощью команды FXAM можно узнать знак нуля.

Во временном вещественном формате существует класс специальных чисел, называемых псевдонулями. *Псевдонуль* — это ненормализованное число с нулевой мантиссой и ненулевым смещенным порядком. У псевдонулей не может быть максимального смещенного порядка 11...11, так как он отведен для других специальных чисел. Псевдонуль может возникнуть при умножении двух ненормализованных чисел, в которых суммарное число старших нулевых битов сомножителей больше 64. Как операнды псевдонули обрабатываются как ненормализованные числа, но в следующих ситуациях они аналогичны истинному нулю:

команды сравнения и проверки,

команда FRNDINT округления до целого,

операция деления, в которой делимое является истинным нулем или псевдонулем, а делитель — псевдонулем.

Специальные числа с максимальным смещенным порядком.

В вещественных форматах предусмотрено представление специальных чисел, называемых *бесконечностями.* Они кодируются с максимальным смещенным порядком 11...11 и мантиссой $1_{\wedge}00...00$. От других специальных чисел, имеющих максимальный смещенный порядок, бесконечности отличаются кодированием мантиссы. Бесконечности допускаются как операнды во всех арифметических операциях и обрабатываются по соответствующим правилам.

В вещественных форматах существует класс специальных чисел, называемых «не-числами» (NAN — Not A Number). Не-число имеет любой знак, максимальный смещенный порядок и любую мантиссу, кроме $1_{\wedge}00...00$. Наличие его в регистре сопроцессора отмечается тэгом специального числа. Когда не-число оказывается операндом, сопроцессор фиксирует особый случай недействительной операции. Если этот особый случай замаскирован, сопроцессор возвращает как результат не-число, а если оба операнда являются не-числами, результатом будет не-число с большей мантиссой. Таким образом, получившееся однажды не-число распространяется в вычислениях и выдается как окончательный результат. Однако в трансцендентных командах операнды не контролируются и в случае операнда не-числа результат непредсказуем.

Для каждого типа чисел в сопроцессоре особый код зарезервирован для представления специального числа, называемого *неопределенностью.* Оно относится к классу не-чисел и возвращается как результат в таких операциях, когда никакой осмысленный ответ невозможен (человек в таких ситуациях говорит «не знаю»). Примерами могут служить извлечение квадратного корня из отрицательного числа, деление 0/0, умножение $0\times\infty$ и т. п. Неопределенность имеет отрицательный знак, максимальный смещенный порядок 11...11 и мантиссу $1_{\wedge}100...00$.

Во всех трех форматах двоичных целых чисел наибольшее по модулю отрицательное число, т. е. -2^{15}, -2^{31} и -2^{63}, считается неопределенностью. Предусмотрена также десятичная неопределенность с кодированием 1111 1111 1111 1111 XXXX...XXXX. Использования «целых неопределенностей» следует избегать, так как сопроцессор считает двоичные неопределенности максимальными отрицательными числами, т. е. они не распространяются в вычислениях, а при загрузке десятичной неопределенности в регистр его содержимое оказывается непредсказуемым. Двойная интерпретация кода неопределенности (как специального числа и максимального по модулю отрицательного числа) является следствием отсутствия «особых» наборов в форматах целых чисел. Целая двоичная неопределенность возникает только в одной недействительной операции при записи в память не-числа в целом формате.

Режимы работы и состояние. Сопроцессор имеет два программно доступных 16-битных регистра, содержимое которых определяет его режим работы и текущее состояние. Форматы этих

регистров, хранящих слово управления CW (Control Word) и слово состояния SW (Status Word), приведены на рис. 4.4, *а*, *б*.

Слово управления. Слово управления определяет для сопроцессора один из нескольких вариантов обработки чисел. Хотя программист может загрузить из памяти любое содержимое CW

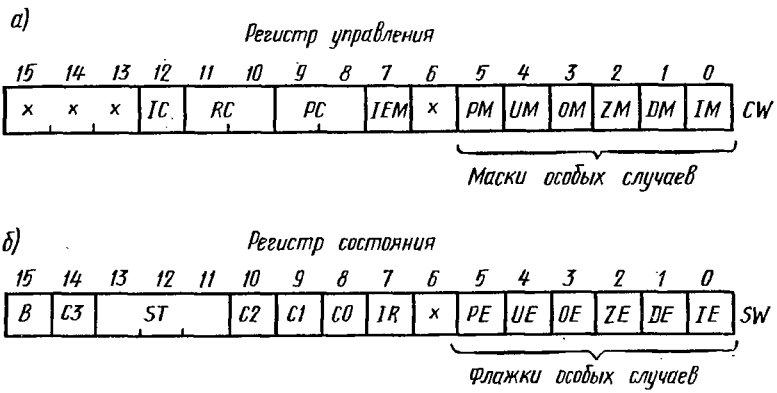


Рис. 4.4. Форматы регистров управления (*а*) и состояния (*б*):
IM, IE — недействительная операция; DM, DE — денормализованный операнд; ZM, ZE — деление на нуль; OM, OE — переполнение, UM, UE — антипереполнение; PM, PE — точность; IEM — маска разрешения прерываний; PC — управление точностью; RC — управление округлением; IC — управление бесконечностью; UE — антипереполнение; IR — запрос прерывания; C3 ... C0 — код условия; ST — указатель стека; B — занятость

(имеется команда FLDCW), для подавляющего большинства прикладных программ наиболее благоприятно содержимое полей CW, устанавливающееся при инициализации сопроцессора (см. табл. 4.5).

Таблица 4.6. **Режимы округления**

Поле RC	Режим	Принимаемый результат
00	Округление к ближайшему	Принимается то число из *a* и *c*, которое ближе к *b*; при равенстве расстояний берется число с нулевым младшим битом (четное)
01	Округление вниз (к $-\infty$)	*a*
10	Округление вверх (к $+\infty$)	*c*
11	Отбрасывание (усечение к нулю)	Берется меньшее по абсолютной величине из чисел *a* и *c*

Шесть младших битов CW представляют собой индивидуальные маски особых случаев. Если любой из них установлен в 1, то возникновение соответствующего особого случая не вызывает прерывания ЦП, а если бит содержит 0 — сопроцессор устанав-

ливает в 1 бит запроса прерывания, относящийся к конкретному особому случаю, в слове состояния и при общем разрешении прерываний генерирует сигнал INT прерывания ЦП.

Особый случай денормализованного операнда фиксируется, когда в операции встречается денормализованный операнд. При попытке деления ненулевого конечного числа на нуль сопроцессор регистрирует особый случай деления на нуль. Особый случай переполнения возникает, если порядок истинного результата больше максимального допустимого порядка получателя. Если значение порядка истинного результата слишком мало, фиксируется особый случай антипереполнения. В типичных алгоритмах появление очень больших или очень малых чисел характерно для промежуточных, а не окончательных результатов. При хранении промежуточных результатов во временном вещественном формате переполнение и антипереполнение в сопроцессоре возникают очень редко. Наконец, когда результат операции не может быть точно представлен в формате получателя, сопроцессор производит его округление и фиксирует особый случай (потери) точности. Он возникает довольно часто и указывает лишь на то, что произошла некоторая, обычно приемлемая, потеря точности.

Анализ рассмотренных особых случаев показывает, что для каждого из них нетрудно предусмотреть возвращение математически приемлемого результата операции, вызвавшей особый случай:

денормализованный операнд — преобразовать его в эквивалентное число (ненормализованное) и продолжить операцию;

деление на нуль и переполнение — возвратить как результат операции бесконечность с правильным знаком;

антипереполнение — возвратить как результат денормализованное число;

точность — не предпринимать никаких действий, т. е. возвратить округленный результат.

Наиболее тяжелым является особый случай недействительной операции, который, как правило, свидетельствует об ошибке в программе. Он включает в себя много ситуаций, которые не имеют таких очевидных решений, как предыдущие особые случаи. Например, без анализа вычислительных шагов, которые привели к делению нуля на нуль, невозможно образовать приемлемый результат. Это же относится к умножению и делению бесконечностей, умножению бесконечности на нуль, сложению бесконечностей с разными знаками, извлечению квадратного корня из отрицательного числа, любой операции с не-числом, загрузке в непустой регистр, считыванию из пустого регистра и т. п. Лучшее, что можно сделать при возникновении особого случая недействительной операции,— это возвратить как результат операции не-число (если одним или обоими операндами являются не-числа) и неопределенность («не знаю»!) в остальных ситуациях.

Возникновение особого случая отмечается установкой в 1 соответствующего флажка в слове состояния. Далее сопроцессор проверяет маску в CW и определяет, следует ли только зарегистрировать особый случай или сформировать запрос прерывания ЦП с вызовом процедуры обработки особого случая. В первом варианте сопроцессор выполняет встроенную процедуру маскированной реакции без прерывания программы, а во втором прерывает ЦП.

Маскированные реакции сопроцессора были тщательно разработаны так, чтобы образовать наиболее правильный результат для каждого условия. Выше были приведены наиболее естественные реакции для всех особых случаев (подробнее они изложены далее). В большинстве прикладных программ рекомендуется маскировать все особые случаи, за исключением недействительной операции.

Бит 7 слова управления содержит маску управления прерыванием IEM (Interrupt Enable Mask), которая разрешает (IEM=0) или запрещает (IEM=1) прерывание ЦП. Если IEM=1, прерывания ЦП не будет даже при возникновении индивидуально не замаскированного особого случая.

Двухбитное поле управления точностью PC (Precision Control) определяет точность вычислений в 24 бит (PC=00), 53 бит (PC=10) или 64 бит (PC=11). По умолчанию вводится режим с максимальной точностью в 64 бит. Остальные режимы предусмотрены только для совместимости с некоторыми языками программирования. При задании пониженной точности производительность сопроцессора не увеличивается.

Двухбитное поле управления округлением RC (Rounding Control) определяет один из четырех вариантов округления результатов операций. Интерпретация поля RC показана в табл. 4.6, где принято, что $a<b<c$, причем числа a и c представимы в формате сопроцессора, а результат операции b не представим. Округление заключается в замене b на a или c.

Округление происходит в арифметических операциях, а также при записи в память, если формат получателя не позволяет точно представить истинный результат. Оно вносит ошибку, величина которой не превышает единицы последнего младшего разряда, сохраняемого в результате. По умолчанию принимается режим округления к ближайшему, обеспечивающий наиболее точную и статистически несмещенную оценку результатов.

Округления вверх и вниз применяются в интервальной арифметике с целью получения достоверного результата независимо от ошибок округления. Верхняя и нижняя границы интервала значений результата находятся путем реализации алгоритма два раза — один раз с округлением вверх, а второй — с округлением вниз. Режим округления с отбрасыванием применяется в целочисленной арифметике.

Бит 12 управления режимом бесконечности IC (Infinity Control) определяет одну из двух моделей интерпретации бесконечности: проективную (IC=0) или аффинную (IC=1). По умолчанию вводится проективный режим, в котором сопроцессор обрабатывает два специальных числа «плюс бесконечность» и «минус бесконечность» как одно и то же число «бесконечность», не имеющее знака. В аффинном режиме сопроцессор допускает положительную и отрицательную бесконечности.

Слово состояния. В слове состояния SW младшие 6 бит отведены для регистрации рассмотренных выше особых случаев. Бит 7 запроса прерывания IR (Interrupt Request) устанавливается в 1 при возникновении любого незамаскированного особого случая, но генерирование прерывания ЦП зависит от состояния бита IEM в слове управления. Биты C3 — C0 фиксируют код условия в командах сравнения, проверки и анализа. Три бита ST представляют собой указатель стека и показывают регистр, являющийся текущей вершиной стека ST (0). Наконец, флажок занятости B (BUSY) устанавливается в 1, когда численное операционное устройство выполняет операцию. Состояние этого бита выведено как сигнал BUSY.

При программировании важную роль играют биты кода условия, которые аналогичны арифметическим флажкам ЦП и фиксируют особенности результата операции. Коды условия привлекаются для реализации условных переходов. Сопроцессор самостоятельно не может влиять на ход выполнения программы, поэтому для условного перехода по результату операции сопроцессора приходится вначале передавать код условия в память, а затем загружать в регистр AH центрального процессора. После этого код условия передается в регистр флажков ЦП и производится условный переход. Интерпретация кода условия была приведена в табл. 4.3.

Маскированные реакции на особые случаи. Сопроцессор регистрирует особый случай, устанавливая в 1 бит флажка конкретного особого случая в слове состояния. После этого он проверяет соответствующую маску в слове управления и либо реализует маскированную реакцию, дающую стандартный результат, либо оповещает ЦП сигналом прерывания.

Маскированные реакции сопроцессора, приведенные в табл. 4.7, тщательно проработаны с тем, чтобы продолжить программу с сохранением наиболее безопасных и осмысленных результатов. Управляя масками, программист может возложить обработку большинства особых случаев на сопроцессор, предусмотрев программную обработку наиболее серьезных ситуаций. Для большинства прикладных программ маскирование всех особых случаев, кроме недействительной операции, дает приемлемые результаты с минимумом усилий. Особый случай недействительной операции говорит о серьезной (фатальной) ошибке в программе, которую

Таблица 4.7. **Маскирование реакции на особые случаи**

Условие возникновения	Маскированная реакция
Недействительная операция	
Регистр-источник не отмечен как пустой	Возвращает вещественную неопределенность
Регистр-получатель не отмечен как пустой	Возвращает (с записью в регистр) вещественную неопределенность
Один или два операнда являются нечислами	Возвращает не-число с большей мантиссой
Один или два операнда в командах проверки и сравнения являются не-числами	Устанавливает условие «не сравнимы»
Операция сложения: указано аффинное замыкание, а операндами являются бесконечности с разными знаками; указано проективное замыкание, а операндами являются бесконечности	Возвращает вещественную неопределенность
Операция вычитания: указано аффинное замыкание, а операндами являются бесконечности с одинаковыми знаками; указано проективное замыкание, а операндами являются бесконечности	То же
Операция умножения: $(\infty)\times(0)$ или $(0)\times(\infty)$	»
Операция деления: $(\infty):(\infty)$, $(0):(0)$, $(0):$ (псевдонуль), делитель ненормализован или денормализован	»
Команда FPREM: модуль (делитель) денормализован или ненормализован	Возвращает вещественную неопределенность. Условие «остаток получен»
Команда FSQRT: ненулевой отрицательный операнд; денормализованный или ненормализованный операнд; операнд $(-\infty)$ с аффинным замыканием или (∞) с проективным замыканием	Возвращает вещественную неопределенность
Команды сравнения: указано проективное замыкание и (∞) сравнивается с (0), (∞) или ненормализованным числом	Устанавливает условие «не сравнимы»
Команда FTST: указано проективное замыкание, а операндом является (∞)	То же
Команды FIST, FISTP: регистр-источник отмечен как пустой, содержит не-число, (∞), денормализованное или ненормализованное число или его содержимое превышает диапазон получателя	Запоминает целочисленную неопределенность
Команда FBSTP: регистр-источник является пустым, содержит не-число, (∞), денормализованное или ненормализованное число или его содержимое превышает 18 цифр	Запоминает десятичную неопределенность
Команды FST, FSTP: регистр-источник содержит ненормализованное чис-	Запоминает вещественную неопределенность

Условие возникновения	Маскированная реакция
ло (порядок в диапазоне), а получатель имеет короткий или длинный вещественный формат	
Команда FXCH: один или оба регистра отмечены как пустые	Помещает в пустой(ые) регистр(ы) вещественную неопределенность и производит обмен
Денормализованный операнд	
Команда FLD: операнд-источник денормализован	Обычная загрузка
Арифметические операции: один или два операнда денормализованы	Преобразует операнд в эквивалентное ненормализованное число и продолжает
Операции сравнения и проверки: один или два операнда денормализованы или ненормализованы (но не псевдонуль)	Преобразует денормализованное число в эквивалентное ненормализованное число; по возможности число нормализуется и операция продолжается
Деление на нуль	
Операции деления: делитель равен нулю	Возвращает бесконечность со знаком, учитывающим знаки операндов
Переполнение	
Арифметические операции: указано округление к ближайшему или усечение, а порядок истинного результата больше 16383	Возвращает бесконечность с правильным знаком и сигнализирует особый случай точности
Команды FST, FSTP: указано округление к ближайшему или усечение, а порядок истинного результата больше 127 (короткий вещественный получатель) или больше 1023 (длинный вещественный получатель)	То же
Антипереполнение	
Арифметические операции: порядок истинного результата меньше —16382	Денормализует до тех пор, пока порядок не станет —16382, округляет мантиссу до 64 бит. Если денормализованная округленная мантисса равна нулю, возвращает истинный нуль; в противном случае возвращает денормализованное число
Команды FST, FSTP: получатель имеет короткий (длинный) вещественный формат, а порядок истинного результата меньше —126 (—1022)	См. выше, но с учетом минимального допустимого порядка (—126 и —1022) и длины мантиссы (24 и 53 бит)

Продолжение табл. 4.7

Условие возникновения	Маскированная реакция
Точность	
Возникает погрешность округления	Операция продолжается без специальных действий
Ранее в команде сопроцессор выполнил маскированную реакцию на переполнение	Операция продолжается без специальных действий

необходимо исправлять. Поэтому данный особый случай маскировать не рекомендуется, хотя сопропроцессор обеспечивает встроенную реакцию и на этот особый случай.

Флажки в слове состояния как бы накапливают особые случаи, возникшие после того, как флажки были последний раз сброшены. Установленные флажки можно сбросить только командой FCLEX, посредством инициализации сопроцессора или «перезаписью» флажков с помощью команд FRSTOR или FLDENV. Следовательно, программист может замаскировать все особые случаи (кроме недействительной операции), выполнить программу, а затем проверить по слову состояния, были ли особые случаи.

Незамаскированный особый случай инициирует прерывание ЦП. В ходе реакции на прерывание ЦП опрашивает программируемый контроллер прерываний и «выходит» на процедуру обработки особого случая. Эта довольно сложная программа обычно входит в состав системного программного обеспечения.

Действия типичной процедуры обработки особого случая:

передать в память среду сопроцессора на момент возникновения особого случая;

сбросить флажки особых случаев в слове состояния сопроцессора;

разрешить восприятие прерываний центральным процессором (они запрещаются в ходе реакции на прерывание);

идентифицировать особый случай, анализируя слова управления и состояния в запомненной среде;

предпринять некоторые системно-зависимые действия по исправлению особого случая;

возвратиться в прерванную программу и возобновить ее выполнение.

Иногда при выполнении команды могут возникнуть несколько особых случаев. Сопроцессор сигнализирует о них в соответствии со следующим старшинством: денормализованный операнд (незамаскированный особый случай), недействительная операция, деление на нуль, денормализованный операнд (замаскирован), пере-

полнение/антипереполнение, потеря точности. Например, при делении нуля на нуль возникает особый случай недействительной операции, а не особый случай деления на нуль.

Упомянутые выше системно-зависимые действия определяются требованиями конкретного применения. В наиболее простом варианте они состоят в следующем:

осуществить инкремент счетчика особых случаев для последующей индикации;

показать диагностическую информацию, например среду и регистры сопроцессора;

запомнить диагностическое значение (не-число) как результат операции и продолжить программу.

Чтобы исправить ошибку, вызванную особым случаем, процедура его обработки должна иметь в своем распоряжении точное состояние сопроцессора и «уметь» восстанавливать его перед возникновением особого случая. Для восстановления состояния программист должен четко понимать, когда фактически распознаются особые случаи при выполнении команд сопроцессором.

Особые случаи недействительной операции, деления на нуль и денормализованного операнда обнаруживаются до начала операции, а особые случаи переполнения/антипереполнения и потери точности не фиксируются до получения истинного результата. В первой ситуации регистры сопроцессора и память еще не модифицировались и выглядят так, как будто «подозрительная» команда не выполнялась. Во второй ситуации регистры сопроцессора и память содержат такие значения, которые получены по окончании команды. Однако в командах запоминания FST и запоминания с извлечением из стека FSTP незамаскированные особые случаи переполнения и антипереполнения генерируются так, как будто команды не выполнялись, т. е. память не модифицирована и извлечения из стека не было.

4.6. АЛГОРИТМЫ И ПРОГРАММЫ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ

Программирование системы с арифметическим сопроцессором особых трудностей не вызывает. Здесь программисту доступны все регистры и команды обоих процессоров, а также все режимы адресации памяти ЦП. Необходимо внимательно следить за синтаксисом команд сопроцессора и идентифицировать правильные форматы его данных с помощью указателей PTR. Кроме того, в нужных местах следует вводить команды FWAIT. Приведенные в этом параграфе ассемблерные программы призваны помочь читателю разобраться в особенностях программирования сопроцессора и ускорить переход к разработке собственных прикладных программ. Вначале рассмотрим несколько элементарных программ с подробными пояснениями.

Суммирование элементов массива (программа 4.1). Предположим, что в памяти находится массив чисел в коротком вещественном формате (длина чисел 4 байт). Количество элементов массива равно N, а адрес первого элемента равен ARRAY. Необходимо вычислить сумму всех элементов массива, представить ее в длинном вещественном формате и поместить по адресу SUM (SUM — это начальный адрес блока из 8 смежных байт).

Программа 4.1. *Суммирование элементов массива:*

```
        ; Адрес массива ARRAY, число элементов  N, числа
        ; в коротком вещественном формате. Сумма в длинном
        ; вещественном формате помещается по адресу SUM.
        ;
        MOV     CX,N            ; Образовать счетчик элементов
        FLDZ                    ; Подготовить место для суммы
        JCXZ    DONE            ; Проверить, что массив не пустой
        XOR     SI,SI           ; Подготовить индекс элементов
MORE:   FADD    DWORD PTR ARRAY [SI]  ; Прибавить элемент
        ADD     SI,4            ; Перейти к следующему элементу
        LOOP    MORE            ; Повторять до завершения
DONE:   FSTP    QWORD PTR SUM   ; Сохранить сумму
```

В этом фрагменте фигурируют три команды сопроцессора. Команда FLDZ включает в его стек нуль, подготавливая ST (0) к накоплению суммы элементов массива. Предполагается, что стек имеет хотя бы один свободный (пустой) регистр. Если уверенности в этом нет, нужно ввести команду FINIT, которая инициализирует сопроцессор и, в частности, очищает стек.

Команда FADD прибавляет к ST (0) текущий элемент массива. Указатель типа DWORD PTR и мнемоника FADD означают, что операндом является число с плавающей точкой в коротком вещественном формате. После передачи в сопроцессор операнд преобразуется во временный вещественный формат и прибавляется к ST (0).

Наконец, команда FSTP при выходе из цикла записывает в ячейку SUM накопленную сумму. Последняя буква P в мнемонике FSTP сообщает, что после выдачи в память содержимого ST (0) производится извлечение из стека. Следовательно, по завершении этого фрагмента стек сопроцессора возвращается в первоначальное состояние.

Интересно проанализировать временные соотношения при выполнении приведенного фрагмента. Ассемблер автоматически введет перед командой FADD команду WAIT, заставляющую ЦП ожидать готовности численного операционного устройства к выполнению команды FADD, и цикл несколько изменится (программа 4.2).

Программа 4.2. *Действительная форма цикла:*

```
MORE:   WAIT
        FADD    DWORD PTR ARRAY [SI]
        ADD     SI,4
        LOOP    MORE
```

При входе в цикл действует сигнал BUSY=1, так как команда FLDZ еще не закончена. Действительно, она выполняется 3 мкс, а команды JCXZ и XOR выполняются 1,6 и 0,8 мкс соответственно. Поэтому ЦП ожидает примерно в течение 1 мкс. Отсюда следует, что команду FLDZ целесообразно поместить перед командой MOV CX, N, что гарантирует завершение команды FLDZ перед входом в цикл.

Затем оба процессора одновременно дешифрируют и начинают выполнение команды FADD. Центральный процессор «выполнит» ее за 3 мкс, а сопроцессору требуется 25 мкс. ЦП дешифрирует команду ADD SI,4, выполняет ее за 1 мкс, переходит к команде LOOP MORE и выполняет ее за 3.4 мкс (предполагается, что осуществляется переход в начало цикла). Сопроцессор же продолжает выполнять команду FADD. После этого ЦП переводится в состояние ожидания и по завершении сопроцессором команды FADD оба процессора начинают выполнять новую команду FADD, операндом которой будет следующий элемент массива. Таким образом, одно прохождение цикла занимает 25 мкс, а не 25+1+ +3.4=29.4 мкс, как представляется без учета параллельной работы обоих процессоров. При этом сопроцессор занят на 100%, так как он одну за другой (без промежутков) выполняет команды FADD с различными операндами, а ЦП занят только 33% времени. Остальное время он бесполезно ожидает завершения выполнения сопроцессором команды FADD.

Ассемблер введет команду WAIT, и перед командой FSTP. Поэтому при выходе из цикла, когда (CX)=0, ЦП подождет окончания выполнения сопроцессором последней команды FADD и оба процессора одновременно начнут выполнять команду FSTP. Конечно, ЦП «выполнит» ее быстрее сопроцессора и, если его следующая (или другая близкая) команда обращается к ячейке SUM, перед ней необходимо указать команду FWAIT.

Рассмотренный фрагмент показывает практический прием использования параллельной работы процессоров для повышения скорости выполнения программы: после продолжительной команды сопроцессора следует по возможности ввести большее число команд ЦП, не обращающихся к результату команды сопроцессора.

Реализация условных переходов. Уже говорилось, что сопроцессор не может непосредственно влиять на ход выполнения программы в зависимости от результатов его команд сравнения, проверки, анализа и вычисления частичного остатка. Хотя эти коман-

ды формируют код условия в слове состояния сопроцессора, выполнить условный переход может только ЦП. Для этого код условия необходимо передать в регистр флажков МП К1810ВМ86.

Реализация условного перехода по результату операции сопроцессора включает в себя три этапа (рис. 4.5):

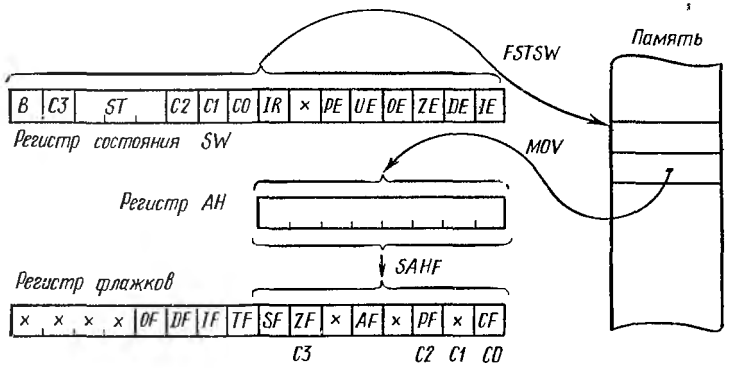


Рис. 4.5. Реализация условных переходов

выполнить команду сопроцессора, формирующую код условия;
передать код условия через память в регистр флажков ЦП;
выполнить команду условного перехода ЦП.

МП К1810 не имеет четырех отдельных команд условных переходов, соответствующих четырем комбинациям C3 и C0, т. е. тех битов кода условия, на которые воздействуют команды сравнения и проверки. Команда JB осуществляет переход, если C0=1 (т. е. если CF=1), а команда JE — если C3=1 (т. е. если ZF=1). Программный фрагмент, учитывающий все комбинации кода условия, приведен в программе 4.3. Предполагается, что STATUS — рабочее слово в памяти.

Программа 4.3. *Полный анализ результата сравнения:*

```
        ; Предполагается, что сравнение выполнено.
        FSTSW   STATUS          ; Передать код условия в память
        FWAIT                   ; Ожидать завершения передачи
        MOV     AH,STATUS+1     ; Код условия в регистре AH
        SAHF                    ; Передать его в регистр флажков
        JB      LORNON          ; Перейти, если меньше или не сравним
        JE      EQUL            ; Перейти, если равны
        ...
EQUAL:  ...                     ; Условие равенства
        ...
LORNON: JE      NOCOMP          ; Перейти, если не сравнимы
        ...                     ; Продолжать, если меньше
        ...
NOCOMP: ...                     ; Не сравнимы
```

Как видно из рис. 4.5, бит условия C1 не попадает ни в один из флажков ЦП. Поэтому для его проверки приходится применять две команды (считая, что код условия сопроцессора находится в регистре AH):

```
        CMP     AH,10B          ; Выделить бит C1
        JE      C1OFF           ; Перейти, если C1 = 0
        .                       ; Продолжать, если C1 = 1
        .
        .
C1OFF:
```

Статистическая обработка. При разработке прикладных программ для систем с сопроцессором необходимо стремиться к максимальному использованию его регистрового стека и минимизации числа обращений к памяти, так как они выполняются гораздо медленнее, чем обращения к регистрам. Эту рекомендацию иллюстрирует следующий пример.

Предположим, что требуется вычислить статистические величины для двух наборов данных $(x_1, x_2, ..., x_n)$ и $(y_1, y_2, ..., y_n)$:

$$m_x = \Sigma x_i,\ m_y = \Sigma y_2,\ s_x = \Sigma x_i^2,\ s_y = \Sigma y_2^2,\ c_{xy} = \Sigma x_i y_i.$$

В программе 4.4 каждый элемент наборов x_i и y_i считывается из памяти только один раз. Действие регистрового стека сопроцессора показано на рис. 4.6.

START:

								Y(i)	Y(i)^2
			X(i)	X(i)^2		Y(i)	Y(i)	Y(i)	Y(i)
	X(i)	X(i)	X(i)	X(i)	X(i)	X(i)	X(i)	X(i)	X(i)
Cxy=0	Cxy	Cxy	Cxy	Cxy	Cxy	Cxy	Cxy	Cxy	Cxy
Sy=0	Sy	Sy	Sy	Sy	Sy	Sy	Sy	Sy	Sy
Sx=0	Sx	Sx	Sx	Sx	Sx+X(i)^2	Sx+X(i)^2	Sx+X(i)^2	Sx+X(i)^2	Sx+X(i)^2
My=0	My	My	My	My	My	My	My+Y(i)	My+Y(i)	My+Y(i)
Mx=0	Mx	Mx+X(i)	Mx+X(i)	Mx+X(i)	Mx+X(i)	Mx+X(i)	Mx+X(i)	Mx+X(i)	Mx+X(i)
Исходное состояние	FLD X[SI]	FADD ST(0),ST	FLD ST(0)	FMUL ST,ST(0)	FADDP ST(1),ST	FLD Y[SI]	FADD ST(0),ST	FLD ST(0)	FMUL ST,ST(0)

Y(i)			
X(i)	X(i) Y(i)		
Cxy	Cxy	Cxy+X(i)Y(i)	
Sy+Y(i)^2	Sy+Y(i)^2	Sy+Y(i)^2	
Sx+X(i)^2	Sx+X(i)^2	Sx+X(i)^2	
My+Y(i)	My+Y(i)	My+Y(i)	
Mx+X(i)	Mx+X(i)	Mx+X(i)	
FADDP ST(1),ST	FMUL	FADD	Стек готов к обработке следующих элементов

Рис. 4.6. Действие регистрового стека в программе статистической обработки

Программа 4.4. *Расчет статистических величин для двух наборов чисел:*

```
        ; Заданы наборы  (X1,X2,...,XN) и (Y1,Y2,...,YN) чисел
        ; в коротком вещественном формате с начальными
        ; адресами X и Y. Вычисляются MX, MY, SX, SY, CXY
        ; и сохраняются в памяти в длинном вещественном
        ; формате.
        ;
        FINIT                   ; Инициализировать сопроцессор
        FLDZ                    ; Подготовить место в стеке
        FLDZ
        FLDZ
        FLDZ
        MOV     CX,N            ; Образовать счетчик элементов
        XOR     SI,SI           ; Начальный индекс наборов
        ;
        ; Подготовка закончена.
START:  FLD     DWORD PTR X[SI]   ; Текущий элемент XI
        FADD    ST(5),ST        ; Накопление MX
        FLD     ST(0)           ; Продублировать XI
        FMUL    ST,ST(0)        ; Образовать XI**2
        FADDP   ST(4),ST        ; Накопление SX
        FLD     DWORD PTR Y[SI]   ; Текущий элемент YI
        FADD    ST(5),ST        ; Накопление MY
        FLD     ST(0)           ; Продублировать YI
        FMUL    ST,ST(0)        ; Образовать YI**2
        FADDP   ST(4),ST        ; Накопление SY
        FMUL                    ; Образовать XI*YI
        FADD                    ; Накопление CXY
        ADD     SI,4            ; Продвинуть индекс
        LOOP    START           ; Повторять до завершения
        ;
        ; Передать результат в память.
        FSTP    QWORD PTR CXY   ; Сохранить CXY
        FSTP    QWORD PTR SY    ; Сохранить SY
        FSTP    QWORD PTR SX    ; Сохранить SX
        FSTP    QWORD PTR MY    ; Сохранить MY
        FSTP    QWORD PTR MX    ; Сохранить MX
```

Логарифмирование. Трансцендентная команда FYL2X вычисляет значение $Y\log_2 X$, т. е. позволяет найти логарифм любого числа по основанию 2. Число X находится в вершине стека ST (0), а Y — в следующем регистре стека ST (1). После извлечения из стека, т. е. удаления X, результат помещается на место Y.

В вычислительных задачах часто требуется находить натуральные и десятичные логарифмы. Переход от двоичных логариф-

мов к логарифмам по любому основанию осуществляется в соответствии с тождеством $\log_n X = \log_n 2 \times \log_2 X$. Необходимые для вычисления натуральных и десятичных логарифмов константы $\log_e 2$ и $\log_{10} 2$ предусмотрены в командах загрузки констант.

В следующих двух программах 4.5 и 4.6 находятся $\log_e X$ и $\log_{10} X$ числа X из вершины стека. Предполагается, что в стеке сопроцессора есть минимум один свободный регистр.

Программа 4.5. *Вычисление натурального логарифма:*

```
        ; Подпрограмма вычисления натурального логарифма
        ; числа, находящегося в ST(0).
        ;
LN      PROC   FAR
        FLDLN2              ; Включить в стек LN 2
        FXCH                ; Обменять с аргументом
        FYL2X               ; Вычислить натуральный логарифм
        RET                 ; Возврат
LN      ENDP
```

Программа 4.6. *Вычисление десятичного логарифма:*

```
        ; Подпрограмма вычисления десятичного логарифма
        ; числа, находящегося в ST(0).
        ;
LOG     PROC   FAR
        FLDLG2              ; Включить в стек нужную константу
        FXCH                ; Обменять с аргументом
        FYL2X               ; Вычислить десятичный логарифм
        RET                 ; Возврат
LOG     ENLP
```

Возведение в степень. В арифметическом сопроцессоре имеется единственная команда FX2M1 возведения в степень. Она воспринимает аргумент X из диапазона $0<X<\sqrt{2}/2$ в вершине стека ST (0) и вычисляет значение $Y=2^X-1$, возвращая его на место аргумента: $ST(0) \leftarrow 2^{ST(0)}-1$. При необходимости вычисления общей степенной функции Y^X следует пользоваться основным тождеством для изменения основания степени: $Y^X=2^{X \log_2 Y}$. Из этого тождества видно, что реализация общей степенной функции требует возведения 2 в произвольную степень. Здесь на помощь приходит команда FSCALE умножения числа в вершине стека на целую степень 2: $ST(0) \leftarrow ST(0) \times 2^{ST(1)}$. Следовательно, произвольный показатель степени Z целесообразно разбить на две части: целое число Z_1 (для команды FSCALE) и правильную дробь Z_2 (для команды F2XM1). Когда $Z_2>1/2$, т. е. находится вне допустимого диапазона аргумента команды F2XM1, следует вычесть 1/2 из Z_2, а затем умножить результат на $2^{1/2}$.

Алгоритм возведения 2 в произвольную степень Z состоит из следующих этапов:

1. Образовать Z_1 — наибольшее целое, которое меньше или равно Z. На этом этапе встречаемся с необходимостью изменения в сопроцессоре режима округления. По умолчанию в нем принят наиболее благоприятный для вычислений режим округления к ближайшему (поле RC в слове управления содержит 00), а нахождение Z_1 требует округления вниз (поле RC=01).

Программа 4.7. *Возведение 2 в произвольную степень:*

```
        ; Показатель степени находится в ST(0).
        ; Стек сопроцессора имеет два свободных регистра.
TWOZ    PROC    FAR
        PUSH    AX              ; Сохранить содержимое AX
        FSTCW   CONTROL         ; Сохранить слово управления
        FSTCW   TEMP            ; Передать слово управления
        FWAIT                   ; Ожидать завершения передачи
        AND     TEMP,0F3FFH ; Сбросить биты RC
        OR      TEMP,0400H  ; Задать округление вниз
        FLDCW   TEMP            ; Вернуть слово управления с RC=01
        FLD     ST(0)           ; Образовать копию Z
        FRNDINT                 ; Образовать Z1
        FLDCW   CONTROL         ; Восстановить режим округления
        FSUB    ST(1),ST        ; Образовать Z2
        FXCH                    ; Обменять местами Z1 и Z2
        FLD     HALF            ; Включить в стек 0.5
        FXCH                    ; Обменять Z2 и 0.5
        FPREM                   ; При необходимости вычесть 0.5
        FSTSW   STATUS          ; Передать слово состояния в память
        FWAIT                   ; Ожидать завершения передачи
        FSTP    ST(1)           ; Удалить из стека 0.5
        F2XM1                   ; Образовать 2**Z2 - 1
        FLD1                    ; Прибавить к результату 1
        FADD    ST(1),ST        ;    для коррекции
        TEST    BYTE PTR STATUS+1,00000010B
                                ; Проверить бит C1
        JZ      NOCORR          ; Вычитания 0.5 не было
        FLD1                    ; Включить в стек
        FADD    ST,ST(0)        ; Образовать 2.0
        FSQRT                   ; Найти квадратный корень из 2
        FMULP   ST(1),ST        ; Скорректировать результат
NOCORR: FSCALE                  ; Возвести в целую степень
        FSTP    ST(1)           ; Удалить Z1
        POP     AX              ; Восстановить AX
        RET                     ; Возврат
TWOZ    ENDP
```

2. Сформировать правильную дробь $Z_2=Z-Z_1$, которая обязательно будет положительной.

3. Если $Z>1/2$, необходимо вычесть 1/2 из Z_2 и отметить факт вычитания.

4. Возвести 2 в степень Z_2 (командой F2XM1) и масштабировать на 2^{Z_1} (командой FSCALE).

5. Если из Z_2 вычиталась 1/2, следует умножить результат на $2^{1/2}$.

В приведенной выше программе 4.7 возведения 2 в произвольную степень Z предполагается, что Z находится в вершине стека и определены следующие ячейки памяти:

STATUS — два байта для передачи в ЦП слова состояния сопроцессора;

CONTROL — два байта для сохранения текущего слова управления;

TEMP — рабочее слово;

HALF — двойное слово, инициализированное на 0,5 в коротком вещественном формате.

Вычисление тригонометрических функций. Основой для вычисления сопроцессором всех тригонометрических функций является команда FPTAN. Напомним, что она находит для угла α, который содержит в вершине стека и лежит в диапазоне $0<\alpha<\pi/4$, два таких числа X и Y, что $\operatorname{tg}\alpha=Y/X$. При этом Y замещает значение аргумента, а X включается в стек. Если угол α находится в требуемом диапазоне, вычисление $\operatorname{tg}\alpha$ не представляет трудностей и реализуется всего двумя командами:

```
FPTAN                ; Образовать частичный тангенс
FDIVP  ST(1),ST      ; Найти тангенс угла
```

После выполнения этих команд значение $\operatorname{tg}\alpha$ замещает собой аргумент α в вершине стека.

Имея числа X и Y, на основе тригонометрических тождеств нетрудно получить значения других тригонометрических функций. Ниже приведен фрагмент вычисления $\sin\alpha$, если α по-прежнему находится в диапазоне от 0 до $\pi/4$.

Программа 4.8. *Вычисление синуса угла:*

```
; Аргумент находится в вершине стека
; и удовлетворяет требованиям команды FPTAN.
;
FPTAN                ; Образовать частичный тангенс
FMUL   ST,ST(0)      ; Образовать X**2
FLD    ST(1)         ; Включить Y в стек
FMUL   ST,ST(0)      ; Образовать Y**2
FADD                 ; Сложить квадраты X и Y
FSQRT                ; Извлечь квадратный корень
FDIVP  ST(1),ST      ; Найти синус угла
```

Если допустить произвольный угол α, а не ограничивать его первым октантом, вычисление тригонометрических функций заметно усложняется. Во-первых, приходится учитывать знак исходного угла. Это не вызывает трудностей при использовании тождеств вида $\operatorname{tg}(-\alpha)=-\operatorname{tg}\alpha$, $\cos(-\alpha)=\cos\alpha$ и др. Во-вторых, угол требуется приводить в диапазон команды FPTAN, т. е. в первый (а точнее — в нулевой) октант. Приведение реализуется с помощью команды FPREM, которая ориентирована именно на эту задачу. Напомним, что она образует частичный остаток от деления ST (0) на модуль из ST (1) и, кроме того, фиксирует в коде условия (C0, C3, C1) три младших бита частного. Полное приведение требует организации программного цикла, повторяемого до тех пор, пока в бите C2 кода условия не будет зафиксирован нуль. Очевидно, в качестве модуля следует использовать $\pi/4$, так как при этом биты (C0, C3, C1) определяют октант, в котором находится исходный угол. К сожалению, команда FPREM дает точные значения битов частного, если число вычитаний при приведении не превышает 62. В-третьих, в зависимости от фактического октанта потребуется вычислять тригонометрические функции по различным формулам.

Покажем на примере функции тангенса, как можно вычислять тригонометрические функции, не пользуясь тремя младшими битами частного от команды FPREM. Обозначим произвольный аргумент функции через z.

Если значение z отрицательно, тангенс вычисляется с привлечением тождества $\operatorname{tg}(-z)=-\operatorname{tg}z$. Следовательно, далее можно считать, что аргумент z положителен. Если в команде FPREM задать модуль π, то она сформирует остаток r в соответствии с равенством

$$z=q\times\pi+r,\ 0\leqslant r<\pi,$$

где q — частное.

Благодаря тождеству $\operatorname{tg}(n\pi+x)=\operatorname{tg}x$ при любых целых n получаем $\operatorname{tg}z=\operatorname{tg}r$ и задача свелась к вычислению тангенса угла, находящегося между 0 и π.

Если $r>\pi/2$, то можно воспользоваться тождеством $\operatorname{tg}z=-1/\operatorname{tg}(z-\pi/2)$ и, следовательно, аргумент приведен в диапазон между 0 и $\pi/2$. Наконец, если аргумент больше $\pi/4$, привлекается тождество $\operatorname{tg}x=1/\operatorname{tg}(\pi/2-x)$ и аргумент оказывается в диапазоне команды FPTAN.

Рассмотренные выше этапы приведения аргумента реализованы в программе 4.9 вычисления тангенса.

Дадим некоторые пояснения программы. Последовательность команд до метки REM осуществляет проверку знака аргумента z. Кроме того, здесь же инициализируются два программных флажка в регистрах BX и CX, состояния которых проверяются на за-

Программа 4.9. *Вычисление тангенса произвольного аргумента:*

```
        ; Аргумент Z находится в вершине стека,
        ; тангенс  также возвращается в вершине стека.
        ; Анализ аргумента на точные 0 и PI/4 не производится.
        ;
TAN:    FLDPI                   ; Включить в стек PI
        FXCH                    ; Обменять аргумент и PI
        FTST                    ; Проверить знак аргумента
        MOV     BX,0            ; Флажок обратной величины
        MOV     CX,BX           ; Флажок изменения знака
        FSTSW   STATUS          ; Передать состояние для проверки.
        FWAIT                   ; Ожидать завершения передачи
        MOV     AH,STATUS+1     ; Передать код условия в AH
        SAHF                    ; Код условия в  регистре флажков
        JAE     REM             ; Аргумент положительный
        FCHS                    ; Аргумент отрицательный,
        NOT     CX              ;     изменить знак и отметить
        ;
        ; Приведение аргумента в диапазон от 0 до PI.
REM:    FPREM                   ; Вычислить остаток R
        FSTSW   STATUS          ;     от деления
        FWAIT                   ;     аргумента на PI
        MOV     AH,STATUS+1
        SAHF
        JP      REM             ; Повторять до завершения
        ;
        ; Привести аргумент в диапазон от 0 до PI/2.
LEPI:   FLD     PINA2           ; Включить в стек PI/2
        FSTP    ST(2)           ; R в ST(0), PI в ST(1)
        FCOM                    ; Сравнить R с PI/2
        FSTSW   STATUS          ; Проверить отношение
        FWAIT                   ;     между R и PI/2
        MOV     AH,STATUS+1
        SAHF
        JBE     LEPI2           ; R меньше PI/2
        FSUB    ST(1)           ; Вычесть PI/2 из R
        NOT     BX              ; Изменить флажок обратной величины
        NOT     CX              ; Изменить флажок знака
        ;
        ; Привести аргумент в диапазон от 0 до DPI/4.
LEPI2:  FLD     PINA4           ; Включить в стек PI/4
        FCOMP                   ; Сравнить R с PI/4
        FSISW   STATUS          ; Проверить отношение
        FWAIT                   ;     между R и PI/4
        MOV     AH,STATUS+1
        JAE     LEPI4
        FSUBR   ST,ST(1)        ; Вычесть PI/4 из R
```

```
        NOT     BX              ; Изменить флажок обратной величины
        ; Приведение аргумента закончено.
        ; Заключительные действия по вычислению тангенса.
LEPI4:  FPTAN                   ; Найти частичный тангенс
        AND     CX,CX           ; Проверить флажок знака
        JZ      NONEG           ; Изменять знак не нужно
        FCHS                    ; Изменить знак X
NONEG:  AND     BX,BX           ; Проверить флажок обратной величины
        JZ      NOBR            ; Ее находить не нужно
        FXCH                    ; Обменять числитель и знаменатель
NOBR:   FDIV                    ; Вычислить тангенс
```

ключительном этапе нахождения тангенса. Нулевое содержимое регистра BX показывает, что при вычислении тангенса не нужно находить обратную величину; в противном случае потребуется эта операция. Нулевое содержимое регистра CX показывает, что изменять знак тангенса не требуется; в противном случае знак необходимо изменить.

Шесть команд, начиная с метки REM, осуществляют приведение аргумента в диапазон от 0 до π. Для этого применяется команда FPREM с модулем π. Она находится в цикле, условием повторения которого является получение C2=1. При передаче кода условия в регистр флажков ЦП битов C2 попадает во флажок PF, поэтому зацикливание приведения осуществляет команда JP (перейти, если PF=1).

Последовательности команд, начиная с меток LEPI1 и LEPI2, выполняют примерно аналогичные действия по приведению аргумента в диапазоны $0 - \pi/2$ и $0 - \pi/4$ соответственно. Здесь предполагается, что требуемые для этого константы находятся в ячейках с адресами PINA2 и PINA4. В процессе приведения при необходимости модифицируются флажки нахождения обратной величины в регистре BX и изменения знака в регистре CX.

Заключительные восемь команд, начиная с метки LEPI4, формируют в вершине стека значение тангенса исходного аргумента. При этом учитываются состояния флажков в регистрах BX и CX и, когда это требуется, производятся корректирующие действия.

В программе 4.10 вычисления всех тригонометрических функций предположим, что в вершине стека ST (0) находится значение угла α, удовлетворяющее диапазону команды FPTAN. Необходимо вычислить значения всех тригонометрических функций и поместить их в область памяти, адресуемую регистрам BX (в длинном вещественном формате). Вычисления тангенса и котангенса производятся обычным образом, а для остальных функций используются тождества, приведенные в п. 4.4.4.

Программа 4.10. *Вычисление всех тригонометрических функций:*

```
        ; Аргумент находится в вершине стека ST(0).
        ; Значения всех его тригонометрических функций
        ; передаются в область памяти, адресуемую регистром BX
        ; (в длинном вещественном формате).
        ;
TFUNC:  FLD     ST(0)           ; Продублировать аргумент
        FPTAN                   ; Вычислить частичный тангенс
        FDIV                    ; Образовать тангенс
        FST     QWORD PTR [BX]  ;     и запомнить его
        FLD1                    ; Включить в стек 1.0
        FDIVR                   ; Образовать котангенс
        FSTP    QWORD PTR [BX+8] ;    и запомнить его
        FLD4                    ; Включить в стек 1.0
        FCHS                    ; Теперь в ST(0)   -1.0
        FXCH                    ; Обменять ее с аргументом
        FSCALE                  ; Поделить аргумент на 2
        FSTP    ST(1)           ; Удалить   -1.0
        FPTAN                   ; Вычислить частичный тангенс
        FDIV                    ; Вычислить  Y/X
        FLD     ST(0)           ;    и продублировать три раза
        FLD     ST(0)
        FLD     ST(0)
        FADD    ST(2),ST        ; Образовать 2*(Y/X)
        FADD    ST(3),ST        ; Образовать 2*(Y/X)
        FMUL                    ; Вычислить (Y/X)**2
        FLD     ST(0)           ;    и образовать копию
        FLD1                    ; Два раза
        FLD1                    ;    включить в стек 1.0
        FADDP   ST(2),ST        ; Вычислить 1+(Y/X)**2
        FSUBRP  ST(2),ST        ; Вычислить 1-(Y/X)**2
        FDIV    ST(2),ST        ; Вычислить синус
        FDIV    ST(3),ST        ; Вычислить косеканс
        FDIV                    ; Вычислить косинус
        FLD1                    ; Включить в стек 1.0
        FDIV    ST,ST(1)        ; Вычислить секанс
        FSTP    QWORD PTR [BX+16]  ; Запомнить секанс,
        FSTP    QWORD PTR [BX+24]  ;     косинус,
        FSTP    QWORD PTR [BX+32]  ;     синус,
        FSTP    QWORD PTR [BX+40]  ;     косеканс
```

Вычисление обратных тригонометрических функций. Вычисление в сопроцессоре обратных тригонометрических функций основывается на команде FPATAN. Она находит arctg z, где $z=Y/X$, причем X находится в вершине стека ST (0), а Y — в следующем регистре стека ST (1). Числа X и Y должны удовлетворять тре-

бованию $0<Y<X<\infty$. Команда FPATAN производит извлечение из стека и возвращает результат в новой вершине стека, т. е. он замещает число Y.

Для вычисления арктангенса произвольного аргумента необходимо проверить следующие варианты: $z=0$, $z>0$, $z<1$, $|z|=1$ и $|z|>1$. Приведение z в диапазон аргументов команды осуществляется в соответствии с тождествами:

$$\operatorname{arctg} z=-\operatorname{arctg}(-z),\ \operatorname{arctg} z=\pi/2-\operatorname{arctg}(1/z).$$

Отсюда следует, что знак аргумента учитывается отдельно и далее можно работать с абсолютным значением z. Если $|z|<1$, то $|z|$ принимается за X, а Y полагается равным 1. Когда же $|z|>1$, за X принимается 1, а за $Y=|z|$, результат команды FPATAN корректируется по второму тождеству.

Программа 4.11 вычисляет арктангенс произвольного аргумента z, передаваемого в вершине стека ST (0). Предполагается, что в стеке сопроцессора имеются минимум три свободных регистра. Регистр CX используется как флажок знака аргумента.

Особых пояснений эта программа не требует. Отметим только, что в ней предусмотрена проверка аргумента на нуль и единицу. При равенстве его одному из этих чисел результат будет абсолютно точным. Кроме того, необходимые константы $\pi/2$ и $\pi/4$ формируются в программе из числа π, а не считываются из памяти.

Вычисление скалярного произведения векторов. Во многих задачах линейной алгебры, например при умножении матриц, приходится вычислять скалярное произведение двух векторов: $X=(x_1, x_2, \ldots, x_n)$ и $Y=(y_1, y_2, \ldots, y_n)$:

$$s=\sum_{i=1}^{n} x_i y_i.$$

Эту задачу решает элементарная программа 4.12, в которой предполагается, что элементы векторов представлены в коротком вещественном формате, а результат — в длинном вещественном формате.

Рассмотрим теперь более общую подпрограмму для вычисления скалярного произведения, в которой используется одна интересная возможность сопроцессора. Автоматическое преобразование всех входных данных во временный вещественный формат, а выходных данных в формат получателя позволяет разработать в некотором смысле универсальные подпрограммы. В них тип обрабатываемых данных передается как параметр и учитывается в командах загрузки и запоминания, а внутренняя обработка в сопроцессоре оказывается одинаковой во всех типах данных.

Подпрограмма GINPR вызывается оператором

```
CALL GINPR(X,Y, TYPEX, TYPEY, N),
```

Программа 4.11. *Вычисление арктангенса произвольного аргумента:*

```
        ; Аргумент Z находится в вершине стека,
        ; арктангенс также возвращается в вершине стека.
        ; Осуществляется проверка аргумента Z на 0 и 1.
        ;
ATAN:   ; Проверить знак и равенство нулю исходного аргумента.
        MOV     CX,0            ; Флажок изменения знака
        FTST                    ; Проверить аргумент
        FSTSW   STATUS          ; Передать в память код условия
        FWAIT                   ; Ожидать завершения передачи
        MOV     AH,STATUS+1
        SAHF                    ; Код условия в регистре флажков
        JA      PLUS            ; Аргумент положительный
        JZ      ZERO            ; Аргумент равен нулю
        JMP     NEG             ; Аргумент отрицательный
ZERO:   FSTP    ST(0)           ; Удалить аргумент из стека
        FLDZ                    ; Включить в стек 0.0
        JMP     DONE            ; Результат готов - он равен 0
NEG:    FCHS                    ; Теперь аргумент положительный
        NOT     CX              ; Отметить отрицательный аргумент
        ; Сравнить аргумент с единицей.
PLUS:   FLD1                    ; Включить в стек 1.0
        FCOM                    ; Произвести сравнение Z и 1
        FSTSW   STATUS          ; Передать в память код условия
        FWAIT                   ; Ожидать завершения передачи
        MOV     AH,STATUS+1
        SAHF                    ; Код условия в регистре флажков
        JA      LESS1           ; Аргумент меньше единицы
        JC      GRT1            ; Аргумент больше единицы
        ;
        ; Аргумент равен 1, результат равен PI/4.
        FCHS                    ; В ST(0) находится  -1.0
        FADD    ST(0),ST        ; Образовать  -2.0
        FLDPI                   ; Включить в стек PI
        FSCALE                  ; Образовать PI/4
        FSTP    ST(1)           ; Удалить из стека  -2.0
        JMP     SIGN            ; Перейти к учету знака
        ;
        ; Использовать тождество arctg Z = PI/2 - arctg (1/Z)
GRT1:   FXCH                    ; Z в st(0), 1 в ST(1)
        FPATAN                  ; Вычислить арктангенс
        FLD1                    ; Включить в стек 1.0
        FCHS                    ; Образовать  -1.0 в ST(0)
        FLDPI                   ; Включить в стек PI
        FSCALE                  ; Образовать PI/2
        FSTP    ST(1)           ; Удалить из стека  -1.0
        FSUBRP  ST(1),ST        ; Образовать результат
```

```
                JMP     SIGN            ; Перейти к учету знака

                ; Аргументы в диапазоне команды FPATAN.
LESS1:          FPATAN                  ; Вычислить арктангенс
SIGN:           ; Учесть знак Z.
                TEST    CX,CX           ; Проверить знак аргумента Z
                JZ      DONE            ; Аргумент положительный
                FCHS                    ; Результат отрицательный
DONE:                                   ; Результат в ST(0)
```

Программа 4.12. *Вычисление скалярного произведения векторов:*

```
                ; Начальный адрес вектора X находится в регистре SI,
                ; вектора Y - в регистре DI. Число элементов векторов N.
                ; Результат помещается в ячейку INPROD.
                ;
INPR:           FLDZ                    ; Подготовить в стеке место
                MOV     CX,N            ; Образовать счетчик элементов
                XOR     BX,BX           ; Подготовить BX для адресации
PLOOP:          FLD     DWORD PTR [BX][SI]  ; Текущий элемент XI
                FMUL    DWORD PTR [Bx][DI]  ; Умножить на YI
                FADD                    ; Накапливать результат
                LOOP    PLOOP           ; Повторять до завершения
                FSTP    QWORD PTR INPROD    ; Запомнить результат
```

который предполагает передачу в стеке адресов векторов X и Y, типов элементов векторов TYPEX и TYPEY, а также числа N элементов векторов. На рис. 4.7 показано, как размещаются в стеке параметры и как осуществляется доступ к ним с помощью регистра BP.

Целочисленные параметры TYPEX и TYPEY имеют значения 4 (элементы вектора представлены в коротком вещественном формате). Каждый из них определяется независимо друг от друга.

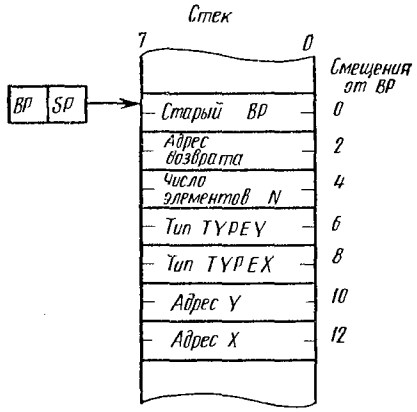


Рис. 4.7. Стек в подпрограмме GINPR

Программа 4.13. *Вычисление скалярного произведения с задаваемыми типами элементов векторов:*

```
                ; Параметры передаются подпрограмме в стеке.
                ; Результат возвращается в вершине стека сопроцессора.
                ;
BINPP   PROC    NEAR
        PUSH    BP              ; Образовать стековый кадр
        MOV     BP,SP           ;    для доступа к параметрам
        FLDZ                    ; Подготовить место для результата
        MOV     CX,[BP+4]       ; Образовать счетчик элементов
        MOV     SI,[BP+12]      ; В SI адрес вектора X
        MOV     DI,[BP+10]      ; В DI адрес вектора Y
        MOV     BX,[BP+6]       ; В BX тип элементов Y
        MOV     AX,[BP+8]       ; В AX тип элементов X
        JCXZ    DONE            ; Проверить число элементов
PLOOP:  CMP     AX,4            ; Проверить тип элементов X
        JNE     XDOUBL          ; X - длинный вещественный формат
        FLD     DWORD PTR [SI]  ; Текущий элемент XI (KB)
        JMP     MULT            ; Перейти к умножению
XDOUBL: FLD     QWORD PTR [SI]  ; Текущий элемент XI (DB)
MULT:   CMP     BX,4            ; Проверить тип элементов Y
        JNE     YDOUBL          ; Y - длинный вещественный формат
        FMUL    DWORD PTR [DI]  ; Умножить на YI (KB)
        JMP     NEXT            ; Перейти к суммированию
YDOUBL: FMUL    QWORD PTR [DI]  ; Умножить на YI (DB)
NEXT:   FADD                    ; Накапливать результат
        ADD     SI,AX           ; Перейти к следующим элементам
        ADD     DI,BX           ;    векторов X и Y
        LOOP    PLOOP           ; Повторять до завершения
        POP     BP              ; Восстановить старый BP
DONE:   RET     10              ; Возврат с удалением параметров
BINPR:  ENDP
```

4.7. ОСОБЕННОСТИ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ В ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРАХ

Профессиональные персональные компьютеры (ПК) можно классифицировать на ПК с базовым микропроцессором К1810ВМ86 (ЕС-1840, ЕС-1841, Искра-1030 и др.) и ПК с системой команд СМ ЭВМ (ДВК-3, ДВК-4 и др.). ПК внутри каждого класса совместимы на уровне машинных команд, но совместимости между классами нет даже на уровне языков программирования высокого уровня. Рассмотрим особенности представления чисел в ПК первого класса.

ПК оснащаются трансляторами с нескольких языков программирования. Среди них специально отметим язык Ассемблера и

язык Бейсик. Язык Ассемблера дает возможность программисту разрабатывать наиболее компактные и «быстрые» программы, обеспечивая доступ ко всем ресурсам ЦК. Кроме того, разработку программ с привлечением арифметического сопроцессора К1810ВМ86 пока можно вести только на языке Ассемблера. Этот язык, по-видимому, будет сохранять свою актуальность еще долгое время, по крайней мере на нем будут разрабатываться критические секции программ, для которых языки высокого уровня оказываются неадекватными.

Версия Бейсика для ПК претерпела существенные изменения по сравнению с его предыдущими версиями. Теперь Бейсик стал довольно мощным языком и обладает некоторыми конструкциями структурного программирования. Вместе с тем Бейсик сохранил свою привлекательность в части простоты изучения и удобной отладки программ. Наличие интерпретатора и компилятора Бейсика позволяет совместить достоинства обоих способов трансляции.

В языке Бейсик применяются три типа числовых данных: целые числа (INTeger) и числа с плавающей точкой в коротком и длинном форматах, называемые числами с одинарной (SiNGle) и двойной (DouBLe) точностью. Имеется несколько способов определения типов переменных и констант.

Явная идентификация типов переменных и констант осуществляется с помощью символов определения типа, которые завершают их имена: символ процента (%) обозначает целое число, восклицательный знак (!) — число с одинарной точностью и символ номера (#) — число с двойной точностью. Примеры типизации с помощью символов определения типа:

```
PI#, FUNC#     - двойная точность,
MIN!, MAX!     - одинарная точность
LIMIT%, INDEX  - целые числа.
```

Второй способ объявления типов переменных и констант заключается в использовании оператора DEF*type*, где *type*=INT, SNG или DBL. Этот оператор имеет следующий общий формат:

```
DEFtype буква[-буква][,буква[-буква]]...
```

Здесь квадратные скобки ограничивают необязательные элементы, а многоточие — возможность повторения предыдущей конструкции любое число раз. Как видно из приведенного формата, оператор DEF*type* может задавать либо отдельные буквы, либо диапазоны букв латинского алфавита и устанавливает, что переменные, имена которых начинаются с указанных букв, будут иметь тип *type*. Примеры оператора DEF*type*:

```
DEFINT  I-N
DEFDBL  A,B,X-Z
```

Первый оператор показывает, что всем переменным, имена которых начинаются с букв I, J, K, L, M, N, назначается тип INT; второй оператор назначает тип DBL переменным, имена которых начинаются с букв A, B, X, Y, Z.

Если тип переменной не объявлен ни символом определения типа, ни оператором DEF*type*, т. е. по умолчанию, принимается тип SNG. Следует помнить, что символ определения типа «перевешивает» указание оператора DEF*type*.

Операторы DEF*type* рекомендуется размещать в начале программы. В частности, они должны предшествовать первым употреблениям тех переменных, типы которых они определяют. Целесообразно типизировать с помощью операторов DEF*type* все переменные, имеющиеся в программе.

Важно знать, в каком формате переменные разных типов представляются в памяти и участвуют в операциях, а также какую точность обеспечивает каждый тип. Форматы чисел разных типов показаны на рис. 4.8.

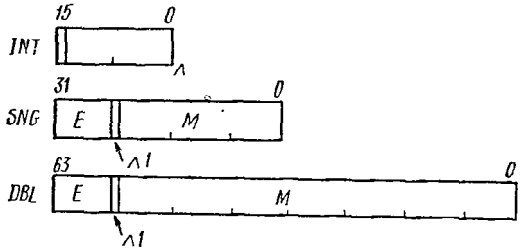


Рис. 4.8. Форматы чисел в языке Бейсик

Целые числа занимают два байта (слово) и представлены в дополнительном коде, имея диапазон от —32 768 до +32 767. Этот формат полностью аналогичен формату слова МП К1810ВМ86 и формату целого слова арифметического сопроцессора К1810ВМ87. Бейсик не поддерживает такую разновидность формата слова, как целые беззнаковые числа, которые применяются для адресов. Поэтому в операциях с адресами необходимо быть очень внимательным.

Числа с плавающей точкой типов SNG и DBL различаются только длиной мантиссы. Тип SNG обеспечивает точность 7 десятичных цифр, а тип DBL — точность 16—17 десятичных цифр (хранятся 17 цифр, а выводятся 16 старших цифр). В обоих форматах порядок E занимает левый байт и представлен в смещенной форме, причем смещение равно 128. Мантисса M считается нормализованной правильной дробью с неявным старшим битом, равным 1. Знаковый бит S занимает место неявного старшего бита мантиссы. Таким образом, форматы чисел с плавающей точкой в Бейсике не совпадают с «внешними» форматами аналогичных чисел в сопроцессоре К1810ВМ87. Это обстоятельство необходимо учитывать при разработке ассемблерных программ с привлечением сопроцессора.

Приведем четыре программы преобразований форматов чисел, применяемых в Бейсике и арифметическом сопроцессоре

Программа 4.14. *Преобразование числа с одинарной точностью Бейсика в короткий вещественный формат сопроцессора:*

```
        ; Регистр SI адресует исходное число,
        ; регистр DI адресует результат.
        ;
        MOV     AX,[SI]     ; Скопировать
        MOV     [DI],AX     ;    младшее слово мантиссы
        MOV     DX,[SI+2]   ; Старшее слово числа в DX
        MOV     AH,DL       ; Выделить знаковый бит
        AND     AH,80H      ;    в регистре AH
        SUB     DH,2        ; Учесть разницу смещений
        JBE     ZERO1       ; Число равно нулю
        SHR     DH,1        ; Выдвинуть младший бит порядка
        JC      SET1        ; Он равен единице
        AND     DL,7FH      ; Он равен нулю
        JMP     L1          ; Обойти формирование единицы
SET1:   OR      DL,80H      ; Младший бит порядка равен единице
L1:     OR      DH,AH       ; Знак числа на месте
        MOV     [DI+2],DX   ; Запомнить старшее слово числа
        JMP     DONE        ; Преобразование закончено
ZERO1:  MOV     WORD PTR [DI],0    ; Результат
        MOV     WORD PTR [DI+2],0  ;    равен нулю
DONE:   <Следующая команда>
```

Программа 4.15. *Преобразование короткого вещественного формата сопроцессора в одинарный формат Бейсика:*

```
        ; Регистр SI адресует исходное число,
        ; регистр DI адресует результат.
        ;
        MOV     AX,[SI]     ; Скопировать
        MOV     [DI],AX     ;    младшее слово мантиссы
        MOV     DX,[SI+2]   ; Старшее слово числа в DX
        MOV     AH,DH       ; Выделить знаковый бит
        AND     AH,80H      ;    в регистре AH
        SHL     DH,1        ; Убрать знаковый бит в порядке
        TEST    DL,80H      ; Проверить младший бит порядка
        JZ      L2          ; Он равен нулю
        OR      DH,1        ; Он равен единице
L2:     CMP     DH,0        ; Проверить число на нуль
        JE      ZERO2       ; Число равно нулю
        ADD     DH,2        ; Учесть разницу смещений
        AND     DL,7FH      ; Сформировать
        OR      DL,AH       ;    знаковый бит
        MOV     [DI+2],DX   ; Запомнить старшее слово числа
        JMP     DONE        ; Преобразование закончено
ZERO2:  MOV     WORD PTR [DI],0    ; Результат
        MOV     WORD PTR [DI+2],0  ;    равен нулю
DONE:   <Следующая команда>
```

Программа 4.16. *Преобразование числа с двойной точностью Бейсика в длинный вещественный формат сопроцессора:*

```
        ; Регистр SI адресует исходное число,
        ; регистр DI адресует результат.
        ;
        MOV     AX,[SI]       ; Скопировать
        MOV     [DI],AX       ;    исходное число
        MOV     AX,[SI+2]     ;    в получатель
        MOV     [DI+2],AX
        MOV     AX,[SI+4]
        MOV     [DI+4],AX
        MOV     AX,[SI+6]
        MOV     [DI+6],AX
        MOV     DH,[SI+6]     ; Выделить знаковый бит
        AND     DH,80H        ;   в регистре DH
        MOV     AL,[SI+7]     ; Передать порядок в AL
        XOR     AH,AH         ; Сбросить регистр AH
        CMP     AL,0          ; Проверить исходное число
        JE      ZERO3         ;    на нуль
        ADD     AX,(1023-129)  ; Учесть разницу смещений
        MOV     CL,4          ; Передать знаковый бит
        SHR     DH,CL         ;    в правильную позицию
        OR      AH,DH
        AND     BYTE PTR [DI+6],7FH  ; Сбросить старый
        SHR     AX,1          ; знаковый бит и передать на его место
        JNC     L3            ; младший бит порядка
        OR      BYTE PTR [DI+6],80H
L3:     MOV     CX,3          ; Образовать счетчик сдвигов
SHIFT:  SHR     AX,1          ; Сдвинуть число на 3 бита вправо
        RCR     BYTE PTR [DI+6],1
        RCR     WORD PTR [DI+4],1
        RCR     WORD PTR [DI+2],1
        RCR     WORD PTR [DI],1
        LOOP    SHIFT
        MOV     BYTE PTR [DI+7],AL  ;Старший байт
        JMP     DONE          ; Преобразование закончено
ZERO3:  MOV     WORD PTR [DI],0     ; Результат
        MOV     WORD PTR [DI+2],0   ;    равен нулю
        MOV     WORD PTR [DI+4],0
        MOV     WORD PTR [DI+6],0
DONE:   <Следующая команда>
```

К1810ВМ87. Приведенные фрагменты образуют основную часть (тело) соответствующих подпрограмм, которые можно вызывать из Бейсик-программы. Чтобы превратить их в подпрограммы, потребуются некоторые обрамляющие директивы. Особых по-

Программа 4.17. *Преобразование длинного вещественного формата сопроцессора в двойной формат Бейсика:*

```
        ; Регистр SI адресует исходное число,
        ; регистр DI адресует результат.
        ;
        MOV     AX,[SI]      ; Скопировать исходное число
        MOV     [DI],AX      ;     в получателе
        MOV     AX,[SI+2]
        MOV     [DI+2],AX
        MOV     AX,[SI+4]
        MOV     [DI+4],AX
        MOV     AX,[SI+6]
        MOV     [DI+6],AX
        MOV     DH,[SI+7]    ; Выделить знаковый бит
        AND     DH,80H       ;     в регистре DH
        MOV     AX,[SI+6]    ; Передать старшее слово в AX
        AND     AX,7FF0H     ; Выделить порядок
        MOV     CL,4         ; Сдвинуть порядок
        SHR     AX,CL        ;     вправо на 4 бита
        CMP     AX,(1023-129)  ; Проверить исходное число
        JBE     ZERO4        ;     на нуль
        SUB     AX,(1023-129)  ; Учесть разницу смещений
        MOV     [DI+7],AL    ; Запомнить порядок
        SHR     DH,1         ; Передать знаковый бит
        SHR     DH,1         ;     в правильную позицию
        SHR     DH,1
        AND     BYTE PTR [DI+6],0FH
        OR      [DI+6],DH    ; Знак на месте
        MOV     CX,3         ; Образовать счетчик сдвигов
SHIFT:  RCL     WORD PTR [DI],1     ;Сдвинуть мантиссу
        RCL     WORD PTR [DI+2],1         на 3 бита влево
        RCL     WORD PTR [DI+4],1
        RCL     BYTE PTR [DI+6],1
        LOOP    SHIFT
        JMP     DONE         ; Преобразование закончено
ZERO4:  MOV     WORD PTR [DI],0     ; Результат
        MOV     WORD PTR [DI+2],0   ;     равен нулю
        MOV     WORD PTR [DI+4],0
        MOV     WORD PTR [DI+6],0
DONE:   <Следующая команда>
```

яснений программы не требуют, так как в них нет каких-либо сложных операций. Отметим только необходимость учета разницы в смещениях, участвующих в образовании смещенного порядка E. При этом приходится принимать во внимание, что в числах сопроцессора имеется бит F_0 целой части мантиссы, а в числах Бейсика он отсутствует. Кроме того, Бейсик не рассчитан на какие-

либо специальные числа, за исключением нуля. Предполагается также, что переполнение невозможно.

Контрольные вопросы и упражнения

1. Назовите достоинства и недостатки сопроцессорных конфигураций.
2. Перечислите несколько других прикладных областей, в которых могут эффективно применяться сопроцессоры.
3. Необходим ли сопроцессорной конфигурации специальный арбитр шины, «регулирующий» доступ к шине отдельных процессоров?
4. Можно ли применять сопроцессор К1810ВМ87 с центральным процессором, отличающимся от микропроцессора К1810ВМ86?
5. Поясните основные моменты синхронизации ЦП и сопроцессора по командам и данным. Приведите примеры гипотетических программных фрагментов.
6. Каким образом сопроцессор считывает из памяти операнды, длина которых больше 16 бит?
7. Рассмотрите, какие форматы команд может иметь сопроцессор с учетом кода ESC (11011).
8. Изобразите круговую диаграмму, поясняющую принцип действия регистрового стека сопроцессора.
9. С какой целью в тэгах регистрового стека сопроцессора предусмотрен специальный код, отмечающий пустой регистр?
10. Позволяет ли наличие тэгов повысить производительность сопроцессора?
11. Поясните причины применения в сопроцессоре стековой организации вместо обычной организации с регистрами общего назначения.
12. Для каждого из форматов чисел сопроцессора покажите, как кодируются минимальные и максимальные числа, и оцените их величину.
13. Почему в формате упакованных десятичных чисел сопроцессора принят прямой код, а не дополнительный?
14. В форматах вещественных чисел сопроцессора порядок имеет разную длину, хотя обычно его длина фиксирована, а изменяется только длина мантиссы. Назовите причины выбора порядков различной длины.
15. Приведите кодирование чисел —31 5, 2048, —3/512, —16384 во всех вещественных форматах сопроцессора.
16. Имеет ли смысл кодировать входные и выходные числовые данные во временном вещественном формате?
17. В чем заключается особенность обратных форм команд вычитания и деления? Как смоделировать эти формы, если бы указанных команд не было?
18. Объясните необходимость наличия у сопроцессора команд загрузки констант.
19. Покажите кодирование специальных чисел во всех форматах сопроцессора.
20. Рассмотрите все особые случаи, которые могут возникнуть в каждой арифметической команде.
21. Изучите табл. 4.7 и кратко сформулируйте смысл всех маскированных реакций сопроцессора на особые случаи.
22. Какие изменения потребуются в программе 4.1 для того, чтобы сохранить сумму элементов массива в коротком вещественном формате? во временном вещественном формате?
23. Исследуйте, что произойдет, если в программе 4.1 вместо команды FSTP указана команда FST и эта программа несколько раз вызывается как подпрограмма.
24. Разработайте такие рекомендации для программиста, которые обеспечивают максимальный параллелизм работы ЦП и сопроцессора.
25. Составьте граф-схемы алгоритмов, реализованных в программах 4.7, 4.9 и 4.11.
26. При каких условиях получается минимальное и максимальное время выполнения программы 4.13?

ПРИЛОЖЕНИЕ

СИСТЕМА КОМАНД АРИФМЕТИЧЕСКОГО СОПРОЦЕССОРА К1810ВМ87

Ниже дается описание системы команд арифметического сопроцессора К1810ВМ87. Команды сгруппированы по функциональному назначению. Для каждой команды приводятся ее мнемоника, машинное кодирование (объектный код), формат операндов, время выполнения в тактах синхронизации (время вычисления эффективного адреса EA берется из табл. 3.3), возникающие при выполнении команды особые случаи и символическое описание функции команды.

Приняты следующие условные обозначения:

src — операнд-источник, т. е. операнд, не изменяющийся при выполнении команды,

dst — операнд-получатель, т. е. операнд, который замещается результатом операции,

mod — режим адресации операнда,

r/m — регистр-память (кодирование см. в табл. 3.2),

ESC — первые пять битов (11011) кода операции всех команд сопроцессора,

disp — 8- или 16-битное смещение в команде (*dispL* — младший байт, *dispH* — старший байт),

d — направление,

R — прямая (0) или обратная (1) операция вычитания и деления;

П — память.

Обозначения форматов чисел:

КВ — короткое вещественное (32 бит),

ДВ — длинное вещественное (64 бит),

ВВ — временное вещественное (80 бит),

ЦС — целое слово (16 бит),

КЦ — короткое целое (32 бит),

ДЦ — длинное целое (64 бит).

Обозначения особых случаев:

I — недействительная операция,

D — денормализованный операнд,

Z — деление на нуль,

O — переполнение,

U — антипереполнение,

P — потеря точности.

Звездочка обозначает неявный операнд.

Мнемоника	Объектный код	Длина, байт	Операнды src	Операнды dst	Время, тактов	Особые случаи I	D	Z	O	U	P	Операция
				Команды передач данных								
FLD *src*	ESC001 11000ST(*i*)	2	ST(0)*	ST(*i*)	20	×	×	*				ST←(ST)—1 ST(0)←(*src*)
	ESC001 *mod*000*r*/*m* (*displ*L) (*disp*H)	2, 3, 4	ST(0)*	П—КВ	43+EA	×	×					
	ESC101 *mod*000*r*/*m* (*displ*L) (*disp*H)	2, 3, 4	ST(0)*	П—ДВ	46+EA	×	×					
	ESC011 *mod*101*r*/*m* (*displ*L) (*disp*H)	2, 3, 4	ST(0)*	П—ВВ	57+EA	×	×					
FILD *src*	ESC111 *mod*000*r*/*m* (*displ*L) (*disp*H)	2, 3, 4	ST(0)*	П—ЦС	50+EA	×						ST←(ST)—1 ST(0)←(*src*)
	ESC011 *mod*000*r*/*m* (*displ*L) (*disp*H)	2, 3, 4	ST(0)*	П—КЦ	56+EA	×						
	ESC111 *mod*101*r*/*m* (*displ*L) (*disp*H)	2, 3, 4	ST(0)*	П—ДЦ	64+EA	×						
FBLD *src*	ESC111 *mod*100*r*/*m* (*displ.*) (*disp*H)	2, 3, 4	ST(0)*	П-УПК	300+EA	×						ST←(ST)—1 ST(0)←(*src*)

FST *dst*	ESC101 11010ST(*i*)	2	ST(*i*)
	ESC001 *mod*010*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	П—КВ
	ESC101 *mod*010*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	П—ДВ
FIST *dst*	ESC111 *mod*010*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	П—ЦС
	ESC011 *mod*010*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	П—ДЦ
FBSTP *dst*	ESC111 *mod*110*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	П—УПК
FSTP *dst*	ESC101 11011ST(*i*)	2	ST(*i*)
	ESC001 *mod*011*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	П—КВ
	ESC101 *mod*011*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	П—ДВ
	ESC011 *mod*111*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	П—ВВ

ST(0)*	18							*dst*←ST(0)
ST(0)*	87+EA	×			×	×	×	
ST(0)*	100+EA	×			×	×	×	
ST(0)*	86+EA	×					×	*dst*←ST(0)
ST(0)*	88+EA	×					×	
ST(0)*	530+EA	×						*dst*←ST(0) ST←(ST)+1
ST(0)*	20							*dst*←ST(0) ST←(ST)+1
ST(θ)*	89+EA	×			×	×	×	
ST(0)*	102+EA	×			×	×	×	
ST(0)*								

Мнемоника	Объектный код	Длина, байт	Операнды *src*	Операнды *dst*	Время, тактов	Особые случаи I	D	Z	O	U	P	Операция
FISTP *dst*	ESC111 *mod*011*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	П—ЦС	ST(0)*	88+ЕА	×					×	*dst*←ST(0) ST←(ST)+1
	ESC011 *mod*011*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	П—КЦ	ST(0)*	90+ЕА	×					×	
	ESC111 *mod*111*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	П—ДЦ	ST(0)*	100+ЕА	×					×	
FXCH //*dst*	ESC001 11001ST(*i*)	2	ST(1)*	ST(0)*	12							(*dst*) ↔ ST(0)
			ST(*i*)	ST(0)*	12							
				Арифметические команды								
FADD //*src*/*dst*, *src*	ESC110 11000001	2	ST(1)	ST(0)*[1]	85	×	×		×	×	×	*dst*←(*dst*)+(*src*)
	ESC*d*00 11000ST(*i*)	2	ST(*i*)	ST(0)	85	×	×		×	×	×	
			S(0)*	ST(*i*)	85	×	×		×	×	×	
	ESC000 *mod*000*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	S(0)*	П—КВ	105+ЕА	×	×		×	×	×	
	ESC100 *mod*000*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	S(0)*	П—ДВ	110+ЕА	×	×		×	×	×	
FADDP *dst*, *src*	ESC110 11000ST(*i*)	2	ST(*i*)	ST(0)	90	×	×		×	×	×	*dst*←(*dst*)+(*src*) ST←(ST)+1

FIADD *src*	ESC110 *mod*000*r/m* (*dispL*) (*dispH*)	2, 3, 4	S(0)*	П—ЦС	120+EA	×	×		×		×	ST(0)←ST(0)+(*src*)
	ESC010 *mod*010*r/m* (*dispL*) (*dispH*)	2, 3, 4	ST(0)*	П—КЦ	125+EA	×	×		×		×	
FSUB //*src*/*dst*, *src*	ESC110 1110R001	2	ST(1)*	ST(0)*[1]	85	×	×		×	×	×	*dst*←(*dst*)—(*src*)
	ESC*d*00 11100ST(*i*)	2	ST(*i*)	ST(0)	85	×	×		×	×	×	
			ST(0)*	ST(*i*)	85	×	×		×	×	×	
	ESC000 *mod*10R*r/m* (*dispL*) (*dispH*)	2, 3, 4	ST(0)*	П—КВ	105+EA	×	×		×	×	×	
	ESC100 *mod*10R*r/m* (*dispL*) (*dispH*)	2, 3, 4	ST(0)*	П-ДВ	110+EA	×	×		×	×	×	
FSUBP *dst*, *src*	ESC110 1110RST(*i*)	2	ST(*i*)	ST(0)	90	×	×		×	×	×	*dst*←(*dst*)—(*src*) ST←(ST)+1
FISUB *src*	ESC110 *mod*10R*r/m* (*dispL*) (*dispH*)	2, 3, 4	ST(0)*	П—ЦС	120+EA	×	×		×	×	×	ST(0)←ST(0)—(*src*)
	ESC010 *mod*10R*r/m* (*dispL*) (*dispH*)	2, 3, 4	ST(0)*	П—КЦ	125+EA	×	×		×	×	×	
FSUBR //*src*/ *dst*, *src*	См. команду FSUB											*dst*←(*src*)—(*dst*)

Мнемоника	Объектный код	Длина, байт	Операнды	
			src	*dst*
FSUBRP *dst, src* FISUBR *src*	См. команду См. команду	FSUBP FISUB		
FMUL[2] //*src/ dst, src*	ESC110 11001001	2	ST(1)*	ST(0)*[1]
	ESC*d*00 11001ST(*i*)	2	ST(*i*) ST(0)*	ST(0) ST(*i*)
	ESC000 *mod*001*r/m* (*dispL*) (*dispH*)	2, 3, 4	ST(0)*	П—КВ
	ESC100 *mod*001*r/m* (*dispL*) (*dispH*)	2, 3, 4	ST(0)*	П—ДВ
FMULP *dst, src*	ESC110 11001ST(*i*)	2	ST(*i*)	ST(0)
FIMUL *src*	ESC110 *mod*001*r/m* (*dispL*) (*dispH*)	2, 3, 4	ST(0)*	П—ЦС
	ESC010 *mod*001*r/m* (*dispL*) (*dispH*)	2, 3, 4	ST(0)*	П—КЦ
FDIV //*src/ dst, src*	ESC110 1111R001	2	ST(1)	ST(0)*[1]

Время, тактов	Особые случаи						Операция
	I	D	Z	O	U	P	
							dst←(*src*)—(*dst*) ST←(ST)+1 ST(0)←(*src*)—ST(0)
97	×	×		×	×	×	*dst*←(*dst*)×(*src*)
97	×	×		×	×	×	
97	×	×		×	×	×	
118+EA	×	×		×	×	×	
160+EA	×	×		×	×	×	
142	×	×		×	×	×	*dst*←(*dst*)×(*src*) ST←(ST)+1
130+EA	×	×		×	×	×	ST(0)←ST(0)×(*src*)
136+EA	×	×		×	×	×	
198	×	×	×	×	×	×	*dst*←(*dst*)/(*src*)

	ESC*d*00 1111RST(*i*)	2	ST(*i*) ST(*i*)	ST(0) ST(*i*)
	ESC000 *mod*11R*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	ST(0)*	П—КВ
	ESC100 *mod*11R*r*/*m* (*dispL*) (*disH*)	2, 3, 4	ST(0)*	П—ДВ
FDIVP *dst, src*	ESC110 1111RST(*i*)	2	ST(0)	ST(0)
FIDIV *src*	ESC110 *mod*11R*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	ST(0)*	П—ЦС
	ESC010 *mod*11R*r*/*m* (*dispL*) (*dispH*)	2, 3, 4	ST(0)*	П—КЦ
FDIVR //*src*/ *dst, src*	См. команду FDIV			
FDIVRP *dst, src*	См. команду FDIVP			
FIDIVR *src*	См. команду FIDIV			
FABS	ESC001 11100001	2	ST(0)*	ST(0)*
FCHS	ESC001 11100000	2	ST(0)*	ST(0)*

198 198 220+EA	×	×	×	×	×	×	
225+EA	×	×	×	×	×	×	
202	×	×	×	×	×	×	*dst*←(*dst*)/(*src*) ST←(ST)+1
230+EA	×	×	×	×	×	×	ST(0)←ST(0)/(*src*)
236+EA	×	×	×	×	×	×	
							dst←(*src*)/(*dst*)
							dst←(*src*)/(*dst*) ST←(ST)+1 ST(0)←(*src*)/ST(0)
14	×						ST(0)←|ST(0)|
15	×						ST(0)←—ST(0)

Мнемоника	Объектный код	Длина, байт
FPREM	ESC001 11111000	2
FRNDINT	ESC001 11111100	2
FSCALE	ESC001 11111101	2
FSQRT	ESC001 11111010	2
FXTPACT	ESC001 11110100	2
FCOM//*src*	ESC000 11011001	2
	ESC000 11010ST(*i*)	2, 3, 4
	ESC000 *mod*010*r*/*m* (*dispL*) (*dispH*)	2, 3, 4
	ESC100 *mod*010*r*/*m* (*dispL*) (*dispH*)	2, 3, 4
FCOMP//*src*	ESC000 11011001	2
	ESC000 11011ST(*i*)	2
	ESC000 *mod*011*r*/*m* (*dispL*) (*dispH*)	2, 3, 4

Операнды		Время, тактов	Особые случаи						Операция
src	*dst*		I	D	Z	O	U	P	
ST(0)*		125	×	×			×		ST(0)←ST(0) MOD ST(1)
ST(0)*	ST(0)*	45	×		*			×	ST(0)←целая часть ST(0)
ST(0)*		35	×			×	×		ST(0)←ST(0)×2ST(1)
ST(0)*	ST(0)*	183	×	×				×	ST(0)←$\sqrt{ST(0)}$
	ST(0)*		×						ST(0)←порядок ST(0) ST(1)←мантисса ST(0)
Команды сравнения									
ST(0)*	ST(1)*	45	×	×					ST(0)—(*src*)
ST(0)*	ST(*i*)	45	×	×					
ST(0)*	П—КВ	65+ЕА	×	×					
ST(0)*	П—ДВ	70+ЕА	×	×					
ST(0)*	ST(1)	45	×	×					ST(0)—(*src*) ST←(ST)+1
ST(0)*	ST(*i*)	45	×	×					
ST(0)*	П—КВ	65+ЕА							

	ESC100 *mod*011*r/m* (*dispL*) (*dispH*)	2, 3, 4
FCOMPP	ESC110 11011001	2
FICOM *src*	ESC110 *mod*010*r/m* (*dispL*) (*dispH*) ESC010 *mod*010*r/m* (*dispL*) (*dispH*)	2, 3, 4 2, 3, 4
FICOMP	ESC110 *mod*010*r/m* (*dispL*) (*dispH*) ESC010 *mod*011*r/m* (*dispL*) (*dispH*)	2, 3, 4 2, 3, 4
FTST	ESC001 11100100	2
FXAM	ESC001 11100101	2
F2XM1	ESC001 11110000	2

ST(0)*	П—ДВ	70+ЕА				
ST(0)*	ST(1)*	50	×	×		ST(0)—ST(1) ST←(ST)+2
ST(0)*	П—ЦС	80+ЕА	×	×		ST(0)—(*src*)
ST(0)*	П—КЦ	85+ЕА	×	×		
ST(0)*	П—ЦС	82+ЕА	×	×		ST(0)—(*src*) ST←(ST)+1
ST(0)*	П—КЦ	87+ЕА				
ST(0)*	0.0	42	×	×		ST(0)—0.0
	ST(0)*	17				Установить С3—С0

Трансцендентные команды

ST(0)*	ST(0)*	500			×	×	ST(0)←$2^{ST(0)}$—1

Мнемоника	Объектный код	Длина, байт	Операнды *src*	Операнды *dst*	Время, тактов	Особые случаи I	D	Z	O	U	P	Операция
FPATAN	ESC001 11110011	2	ST(0)*	ST(0)* ST(1)*	650					×	×	Частичный арктангенс $z=\mathrm{arctg}(ST(1)/ST(0))$ ST←(ST)+1 ST(0)←z
FPTAN	ESC001 11110010	2	ST(0)* ST(0)*	ST(0)*	450	×					×	Частичный тангенс $\mathrm{tg}\,ST(0)=Y/X$ ST(0)←Y, ST←(ST)−1 ST(0)←X
FYL2X	ESC001 11110001	2	ST(0)*	ST(0)* ST(1)*	950						×	$z=ST(1)\log_2 ST(0)$ ST←(ST)+1, ST(0)←z
FYL2XP1	ESC001 11111001	2	ST(0)*	ST(0)* ST(1)*	850						×	$z=ST(1)\log_2(ST(0)+1)$ ST←(ST)+1, ST(0)←z
Команды загрузки констант												
FLDZ	ESC001 11101110	2	ST(0)*		14	×						ST←(ST)−1 ST(0)←+0.0
FLD1	ESC001 11101000	2	ST(0)*		14	×						ST←(ST)−1 ST(0)←+1.0
FLDP1	ESC001 11101011	2	ST(0)*		19	×						ST←(ST)−1 ST(0)←π
FLDL2T	ESC001 11101001	2	ST(0)*		19	×						ST←(ST)−1 ST(0)←$\log_2 10$
FLDL2E	ESC001 11101010	2	ST(0)*		18	×						ST←(ST)−1 ST(0)←$\log_2 e$
FLDLG2	ESC001 11101100	2	ST(0)*		21	×						ST←(ST)−1 ST(0)←$\log_{10} 2$
FLDLN2	ESC001 11101101	2	ST(0)*		20	×						ST←(ST)−1 ST(0)←$\log_e 2$
Команды управления сопроцессором												
FINIT[3]	ESC011 11100011	2			5							Инициализировать сопроцессор

FENI[3]	ESC011 11100000	2			5		Разрешить прерывания
FDISI[3]	ESC011 11100001	2			5		Запретить прерывания
FLDCW *src*	ESC001 *mod*101*r/m* (*dispL*) (*dispH*)	2, 3, 4	CW	*src*	10+EA		CW←(*src*)
FSTCW[3] *dst*	ESC001 *mod*111*r/m* (*dispL*) (*dispH*)	2, 3, 4	*dst*	CW	15+EA		*dst*←(CW)
FCLEX	ESC011 11100010	2			5		Сбросить флажки особых случаев
FINCSTP	ESC001 11110111	2	ST*		9		ST←(ST)+1
FDECSTP	ESC001 11110110	2	ST*		9		ST←(ST)—1
FFREE	ESC101 11000ST(*i*)	2	Тэг ST(*i*)		11		Тэг ST(*i*)←11
FNOP	ESC001 11010000	2			13		Холостая команда
FLDENV *src*	ESC001 *mod*001*r/m* (*dispL*) (*dispH*)	2, 3, 4		*src*	40+EA		Загрузить среду
FSTENV[3] *dst*	ESC001 *mod*110*r/m* (*dispL*) (*dispH*)	2, 3, 4	*dst*		45+EA		Запомнить среду
FRSTOR *src*	ESC101 *mod*100*r/m* (*dispL*) (*dispH*)	2, 3, 4		*src*	210+EA		Восстановить полное состояние
FSAVE *dst*	ESC101 *mod*110*r/m* (*dispL*) (*dispH*)	2, 3, 4	*dst*		210+EA		Запомнить полное состояние
FSTSW[3] *dst*	ESC101 *mod*111*r/m* (*dispL*) (*dispH*)	2, 3, 4			15+EA		Запомнить слово состояния

Примечания: 1) В безоперандном формате команд FADD, FSUB, FSUBR, FMUL, FDIV производится извлечение из стека.

2) Время выполнения команды FMUL может быть несколько меньше, если мантисса имеет нулевые младшие биты.

3) Команды допускают мнемонику в форме «без ожидания».

В учебном пособии рассмотрены алгоритмы и программы арифметических операций в 8- и 16-битных микропроцессорах, а также в арифметическом сопроцессоре К1810ВМ87. Для автономного изучения пособия приведены необходимые сведения по двоичной системе счисления, современным форматам числовых данных и особенностям выполнения арифметических операций в микропроцессорах.

Расширение применения микропроцессоров в сложных системах управления, интеллектуальных роботах, графических станциях и т. д., требующих математической обработки больших объемов данных в условиях дефицита времени, будет стимулировать интерес к машинной арифметике. Этому будет способствовать разработка 32-битных микропроцессоров и совместимых с ними арифметических сопроцессоров. За рубежом такие устройства уже появились — это микропроцессор 80386 и сопроцессор 80387, выпущенный фирмой Intel. В них предусмотрена архитектурная совместимость с 16-битными предшественниками. Поэтому материал гл. 3 и 4 настоящего учебного пособия будет способствовать освоению студентами новых микропроцессорных средств и внедрению их в новые разработки.

Так как микропроцессор 80386 и сопроцессор 80387 уже широко применяются в зарубежных персональных компьютерах, приведем основные сведения, относящиеся к этим микросхемам. Естественно, главное архитектурное отличие МП 80386 от рассмотренного в данном учебном пособии МП К1810ВМ86 заключается в расширении длины внутренних регистров до 32 бит. Вместе с тем значительно увеличено и число этих регистров: в составе МП 80386 имеется 16 пользовательских регистров и 15 системных регистров. Среди новых регистров отметим два дополнительных сегментных регистра FS и GS, системные регистры глобальной и локальной дескрипторных таблиц GDTR и LDTR, регистр дескрипторной таблицы прерываний IDTR, регистр задачи TR, три регистра управления CRi, 6 регистров отладки DRi и два регистра проверки TRi.

В составе МП имеется очень гибкое устройство управления памятью, поддерживающее сегментную и страничную организацию памяти, и развитые средства защиты по 4 уровням привилегий. Конвейерная архитектура, включающая в себя 6 параллельно ра-

ботающих устройств, позволила повысить производительность МП до 4—5 млн. операций в секунду.

МП 80386 может работать в нескольких режимах. В одном из них, называемом режимом реального адреса или R-режимом, он полностью моделирует МП К1810ВМ86. Поэтому практически все программное обеспечение, разработанное для МП К1810ВМ86, может быть использовано для МП 80386. Более того, еще в одном из своих режимов (режим виртуального МП 8086 или V-режим) МП 80386 одновременно может выполнять несколько задач МП К1810ВМ86.

О расширенных возможностях МП 80386 свидетельствует краткий обзор его системы команд, большинство из которых могут оперировать байтами, словами и двойными словами:

— команды передач данных (MOV, XCHG и др.);

— команды ввода — вывода (IN/OUT, INS/OUTS);

— стековые команды (PUSH/POP, PUSHA/POPA);

— команды преобразований данных (XLAT, MOVSX, MOVZX, CBW, CWD, CDQ);

— команды арифметических операций (ADD, SUB, MUL, DIV);

— команды десятичной коррекции (AAA, AAS, AAM, AAD, DAA, DAS);

— команды логических операций (AND, OR, XOR, TEST, NOT);

— команды сдвигов (SHR/SHL, SAR/SAL, ROR/ROL, RCR/RCL, SHRD/SHLD);

— команды манипуляций битами (BT/BTC/BTR/BTS, BSF/BSR);

— команды управления флажками (CLD/STD, CLI/STI, CLC/CMC/STC, LAHF/SAHF, PUSHF/POPF);

— команды операций с цепочками (MOVS, CMPS, STOS, LODS, SCAS);

— команды передачи управления (JMP, Jcc, CALL/RET, LOOP, LOOPcc, INT, INTO, IRET);

— команды поддержки языков высокого уровня (BOUND, ENTER, LEAVE);

— команды загрузки адресов (LEA, Lsreg);

— системные команды (LGDT/SGDT, LIDT/SIDT и др.);

— команды загрузки и сохранения содержимого регистров управления, отладки и проверки.

Освоение и умелое использование системы команд процессора 80386 позволит разрабатывать эффективные прикладные программы.

В сопроцессоре 80387 сохранена архитектурная совместимость с сопроцессором К1810ВМ87. Вместе с тем в нем реализованы следующие усовершенствования:

— улучшен интерфейс с центральным процессором (с применением дополнительного сигнала ошибки ERROR);

— значительно повышена производительность сопроцессора как

за счет увеличения частоты синхронизации, так и благодаря новым алгоритмам выполнения операций;

— во многих трансцендентных командах ослаблены ограничения на диапазоны аргументов;

— введено несколько новых команд, например вычисления синуса FSIN, косинуса FCOS и одновременно синуса и косинуса FSINCOS;

— от программиста или программы-ассемблера не требуется вставлять команду ожидания WAIT перед каждой командой сопроцессора; она требуется только для синхронизации по данным (когда сопроцессор записывает в память результат, требующийся близкой последующей команде центрального процессора).

Отметим также, что совсем недавно фирма Intel выпустила МП 80486. На его кристалле, который содержит около 1 млн. транзисторов, совмещены центральный процессор и арифметический сопроцессор. Появление этого микропроцессора будет стимулировать дальнейший интерес к численной обработке данных и способствовать разработке новых систем с очень широкими возможностями.

1. *Григорьев В. Л.* Программное обеспечение микропроцессорных систем. — М.: Энергоатомиздат, 1983.
2. *Григорьев В. Л.* Программирование однокристальных микропроцессоров. — М.: Энергоатомиздат, 1987.
3. *Левенталь Л., Сэйвилл Д.* Программирование на языке ассемблера для микропроцессоров 8080 и 8085. — М.: Радио и связь, 1987.
4. *Савельев А. Я.* Прикладная теория цифровых автоматов. — М.: Высшая школа, 1987.
5. *Brey B. B.* The 8085 microprooessor. Software, programming and architecture. — Prentice — Hall, 1984.
6. *Morse S. P., Albert D. J.* The 80286 architecture. — Wiley, 1986.
7. *Palmer J. F., Morse S. P.* The 8087 primer. — Wiley, 1984.
8. Proposed standard for floating point arithmetic. — Computer, v. 14, № 3, 1981, p. 52—62.
9. *Scanlon L. J.* 8086/8088 assembly Language programming. Prentico — Hall, 1984.

В.К.ЗЛОБИН
В.Л.ГРИГОРЬЕВ

Программирование арифметических операций в микропроцессорах

Допущено Государственным комитетом СССР
по народному образованию в качестве учебного пособия
для студентов высших учебных заведений

Москва
«Высшая школа» 1991

ББК 32.973—01
З-68
УДК 681.3

Рецензенты:
кафедра вычислительной техники
Московского института электронной техники;
проф. Б. М. Каган (Московский институт инженеров транспорта)

Злобин В. К., Григорьев В. Л.
З-68 Программирование арифметических операций в микропроцессорах: Учеб. пособие для технических вузов.— М.: Высш. шк., 1991. — 303 с.: ил.
ISBN 5-06-002052-5

Приведены форматы числовых данных для современных микропроцессоров и профессиональных персональных компьютеров. Рассмотрены алгоритмы выполнения арифметических операций и особенности их программной реализации в микропроцессорных системах.

З $\frac{2405000000(4309000000)-393}{001(01)-91}$ 169—91

ББК 32.973—01
6Ф7.8

ISBN 5-06-002052-5