Алла Солонина, Дмитрий Улахович, Лев Яковлев

bhv
www.bhv.ru
www.bhv.kiev.ua

Алгоритмы и процессоры цифровой обработки сигналов

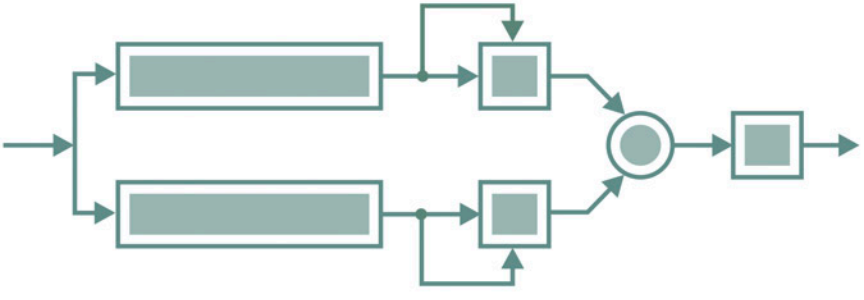
Учебное пособие

Методы и базовые алгоритмы

Архитектура процессоров

Представление данных

Основы проектирования систем цифровой обработки сигналов



Алла Солонина
Дмитрий Улахович
Лев Яковлев

Алгоритмы и процессоры цифровой обработки сигналов

Рекомендовано УМО по образованию в области телекоммуникаций в качестве учебного пособия для студентов, обучающихся по напрвлению 654400 "Телекоммуникации"

Дюссельдорф ◆ Киев ◆ Москва ◆ Санкт-Петербург

УДК 681.3.06

Учебное пособие посвящено базовым алгоритмам ЦОС и архитектуре ЦПОС на примерах процессоров фирм Texas Instruments, Analog Devices и Motorola. Рассматриваются принципы построения и характеристики цифровых процессоров обработки сигналов, представление и обработка данных, команды, типы адресации операндов. Описывается работа различных устройств внутрикристальной периферии. Обсуждаются вопросы подготовки программ пользователя: этапы разработки и отладки, особенности и элементы языков ассемблера различных процессоров, состав пакетов программного обеспечения, а также использование языка С. Изложение сопровождается многочисленными примерами и иллюстрациями.

Для студентов, преподавателей, инженеров и научных работников, разрабатывающих и применяющих ЦПОС

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зав. редакцией	*Наталья Таркова*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Солонина А. И., Улахович Д. А., Яковлев Л. А.
Алгоритмы и процессоры цифровой обработки сигналов. — СПб.: БХВ-Петербург, 2001. — 464 с.: ил.

ISBN 5-94157-065-1

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 24.08.01.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 37,41.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар, № 77.99.1.953.П.950.3.99 от 01.03.1999 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с диапозитивов
в Академической типографии "Наука" РАН.
199034, Санкт-Петербург, 9-я линия, 12.

Содержание

Введение

Цифровые процессоры обработки сигналов (ЦПОС) или их равнозначное название — цифровые сигнальные процессоры (ЦСП или просто сигнальные процессоры), англоязычное сокращение — DSP (Digital Signal Processor), предназначены для реализации алгоритмов цифровой обработки сигналов (ЦОС) и систем управления в реальном времени. ЦПОС являются одной из разновидностей микропроцессоров, четко выделившейся в отдельное направление развития электронной индустрии.

Первый ЦПОС TMS320C10 с производительностью 5 млн операций в секунду фирмы Texas Instruments появился в начале 80-х годов XX века. В последующие годы аналогичная продукция была разработана другими фирмами: Analog Devices, Motorola, Lucent Technologies и т. д. Современный рынок представлен большим разнообразием ЦПОС с производительностью, достигающей нескольких миллиардов операций в секунду. Фантастические, непрерывно возрастающие темпы совершенствования ЦПОС в свою очередь активизируют разработку новых методов и алгоритмов ЦОС все возрастающей вычислительной сложности. В настоящее время динамически развивающаяся взаимосвязь "алгоритм ЦОС $\leftrightarrow$ ЦПОС" в значительной мере определяет научно-технический прогресс в области телекоммуникаций, включая глобальную сеть Internet, и систем мультимедиа.

В условиях, когда жесткая конкуренция диктует высочайшие темпы производства ЦПОС и реализации на их базе новых алгоритмов ЦОС, появилась возможность, благодаря сети Internet, одновременного, столь же быстрого распространения соответствующей технической информации. В последнее время руководства для пользователей ЦПОС все реже выпускаются в виде книг, все чаще их размещают на сайтах фирм и дублируют на компакт-дисках. Поэтому основным способом изучения ЦПОС становится *индивидуальная* и *самостоятельная* работа с информацией, предоставляемой фирмами в Сети.

Главная трудность для начинающих самостоятельно изучать ЦПОС — колоссальный объем информации в Internet. Можно потратить массу времени и усилий, а в результате получить лишь набор фрагментарных знаний, не дающий общего представления об основах построения ЦПОС, причинах, по которым они выделены в отдельный класс микропроцессоров, признаках, отличающих архитектуры различных ЦПОС, и т. д. В этих условиях очень важна начальная (базовая) подготовка.

В настоящее время отечественных книг по ЦПОС очень немного и предназначены они для изучения конкретных процессоров (или базовых семейств).

Учебные пособия по изучению основ ЦПОС отсутствуют. Существует американский вариант подобной книги [4], приводимый в списке литературы.

Авторы полагают, что базовая подготовка должна предусматривать:

- знакомство с типовыми методами и алгоритмами ЦОС, оценку их вычислительной сложности, анализ возможности разработки на базе ЦПОС соответствующих устройств ЦОС, работающих в реальном времени;
- изучение единых или сходных принципов формирования архитектуры ЦПОС, основных вариантов и характерных особенностей их различных реализаций.

Авторы считают, что такая подготовка, *поддерживаемая данным пособием*, будет способствовать развитию общей эрудиции в области ЦОС и ЦПОС, позволит создать цельное представление о принципах организации ЦПОС и познакомит с многообразием их новейших технических решений.

Пособие может стать основой для последующего более обстоятельного изучения конкретных ЦПОС различных фирм. Оно научит понимать, какие характеристики, в каком приложении являются определяющими, ориентироваться в оценке сильных и слабых сторон различных ЦПОС и принимать правильные компромиссные решения при выборе сигнальных процессоров для требуемых приложений.

Данное пособие может использоваться при изучении дисциплин, включенных в государственные стандарты специальностей направления "Телекоммуникации" (654400):

- вычислительная техника и информационные технологии (раздел "Сигнальные процессоры и их применение в системах цифровой обработки сигналов");
- микропроцессоры и цифровая обработка сигналов;
- цифровая обработка сигналов и сигнальные процессоры в системах подвижной радиосвязи.

Пособие может быть также полезно для инженеров и разработчиков цифровых систем связи на основе ЦПОС.

Изложение основ ЦПОС сопровождается многочисленными примерами общего или конкретного характера в зависимости от иллюстрируемого положения. В приводимых примерах в основном использовались процессоры фирм Texas Instruments, Analog Devices и Motorola, получившие на российском рынке наибольшее распространение. Ссылки на источники информации по продукции данных фирм, а также о российских процессорах, содержатся в *приложении 3*. При написании пособия использовались многочисленные руководства пользователя по конкретным процессорам. Однако все эти руководства не включены в список литературы из-за их большого количества и постоянной модификации.

Читатель заметит определенные повторы материала в разных главах. Авторы сохранили их намеренно в случаях, когда желательно не отвлекать внимание читателя на поиск информации в других главах, либо, когда уместно подчеркнуть, что один и тот же вопрос может рассматриваться в разном контексте.

Учебное пособие содержит 11 глав; главы 1, 8, 10 написаны Д. А. Улаховичем, главы 2, 9, 11, приложения — Л. А. Яковлевым, главы 3, 4, 5, 6, 7 — А. И. Солониной.

- В *главе 1* рассматриваются основные направления ЦОС, приводятся примеры базовых методов и алгоритмов ЦОС, анализируются методы их реализации и делается вывод о требованиях, которым должны отвечать цифровые процессоры обработки сигналов для обеспечения эффективной реализации алгоритмов ЦОС.
- *Глава 2* посвящена принципам построения и архитектуре ЦПОС; в ней также рассматриваются основные типы существующих архитектур и их сравнение, организация памяти в ЦПОС.
- *Глава 3* посвящена подробному знакомству с представлением данных в ЦПОС с фиксированной и плавающей точками и арифметическим операциям с данными.
- В *главе 4* рассматриваются специальные операции с данными: округление, арифметика насыщения при переполнении, обработка особых случаев для данных с плавающей точкой.
- В *главе 5* рассматриваются основные типы адресации операндов.
- В *главе 6* изучаются принципы формирования системы команд и основные группы команд.
- В *главе 7* рассматривается работа процессора в состоянии прерывания.
- *Глава 8* посвящена изучению принципов работы разнообразных устройств внутрикристальной периферии: компандеров, таймеров, контроллеров прямого доступа к памяти, портов, синхронизации и др.; приводятся примеры процессоров, в которых применяются эти устройства.
- *Глава 9* посвящена вопросам подготовки программ пользователя: этапы разработки, используемые для программирования ЦПОС языки, общие особенности и элементы языков ассемблера различных процессоров, состав пакетов программного обеспечения процессоров, а также использование языка С.
- В *главе 10* рассматриваются программные и аппаратные средства разработки устройств ЦОС: симуляторы, внутрисхемные и внутрикристальные эмуляторы, отладочные модули, приводятся конкретные примеры.
- *Глава 11* посвящена рассмотрению основных характеристик ЦПОС, сравнению производительности различных процессоров (по зарубежным ма-

териалам), а также существующим разновидностям ЦПОС с точки зрения назначения.

- В *приложениях* приводятся сравнительная таблица параметров некоторых упоминаемых в пособии процессоров, список книг на русском языке, посвященных описаниям различных ЦПОС, а также список источников информации: периодические издания, сайты дистрибьюторов и фирм-разработчиков.

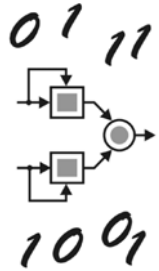
Предполагается, что читатели данного пособия уже знакомы с основами микропроцессорных систем, цифровой и вычислительной техники.

При написании пособия использован опыт чтения лекций и проведения лабораторно-практических занятий по соответствующим курсам на кафедре цифровой обработки сигналов Государственного университета телекоммуникаций им. проф. М. А. Бонч-Бруевича.

Авторы выражают благодарность заведующему кафедрой ЦОС и руководителю Центра ЦОС профессору Артуру Абрамовичу Ланне за помощь в подготовке книги.

Все предложения и замечания по книге просим присылать по адресу: 191186, Санкт-Петербург, наб. р. Мойки, 61, Государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича, кафедра цифровой обработки сигналов или электронному адресу **info@dsp-sut.spb.ru**.

Глава 1



Методы и алгоритмы цифровой обработки сигналов

Прежде чем изучать принципы, лежащие в основе конструкции цифровых процессоров обработки сигналов (ЦПОС), целесообразно рассмотреть базовые методы и алгоритмы цифровой обработки сигналов (ЦОС), которые оказали и продолжают оказывать непосредственное влияние как на всю элементную базу ЦОС, так, в частности, и на архитектуру ЦПОС. Знание методов и алгоритмов ЦОС, их особенностей, сложности, возможностей применения в разнообразных задачах позволяет оценить вычислительные проблемы, свойственные данному алгоритму в конкретном приложении, и выбрать оптимальную элементную базу для его реализации. Кроме того, это знание помогает понять, почему развитие цифровой вычислительной техники пришло к цифровым процессорам обработки сигналов.

Общие же проблемы ЦОС, характерные для любых алгоритмов, нетрудно обнаружить на обобщенной схеме цифровой обработки аналоговых сигналов, которую и рассмотрим.

1.1. Обобщенная схема цифровой обработки аналоговых сигналов

Цифровая обработка принципиально связана с представлением любого сигнала в виде последовательности чисел. Это означает, что исходный аналоговый сигнал должен быть преобразован в исходную последовательность чисел (рис. 1.1), которая вычислителем по заданному алгоритму преобразуется в новую последовательность, однозначно соответствующую исходной. Из полученной новой последовательности формируется результирующий аналоговый сигнал.

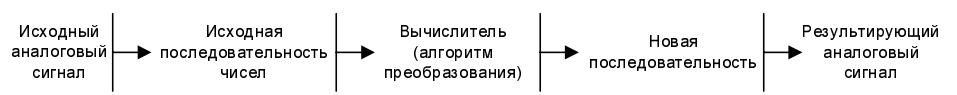


Рис. 1.1. Этапы цифровой обработки

Перечисленные преобразования должны происходить по определенным правилам, смысл которых отображен на рис. 1.2, где показаны основные элементы обобщенной схемы цифровой обработки аналоговых сигналов:

- ❒ аналоговый антиэлайсинговый фильтр низких частот (АФНЧ);
- ❒ аналого-цифровой преобразователь (АЦП);
- ❒ устройство цифровой обработки сигналов (вычислитель);
- ❒ цифро-аналоговый преобразователь (ЦАП);
- ❒ аналоговый сглаживающий фильтр низких частот (СФНЧ).

На этом же рисунке приведен пример цифровой обработки аналогового сигнала и временные и спектральные диаграммы на входе и выходе основных элементов. Устройство, объединяющее АФНЧ и АЦП, называется *кодером*. Как следует из рис. 1.2, *а*, кодер формирует последовательность чисел, соответствующую обрабатываемому аналоговому сигналу. Устройство, объединяющее ЦАП и СФНЧ, называется *декодером*. Декодер по принятому цифровому сигналу формирует аналоговый сигнал, т. е. производит преобразования, обратные происходившим в кодере.

На вход системы поступает ограниченный по длительности сигнал $x(t)$, имеющий в своем составе постоянную составляющую $x_0 = 1$, явно выраженные высокочастотные составляющие, которые несколько затеняют сигнал небольшими всплесками (рис. 1.2, *б*). Спектр амплитуд, или просто спектр, такого сигнала представлен на рис. 1.2, *в*. В силу конечной длительности сигнала его спектр бесконечен!

Бесконечность спектра является препятствием для преобразования сигнала $x(t)$ в цифровую форму, что связано с природой аналого-цифрового преобразования, осуществляемого в два этапа: дискретизации по времени и квантования по уровню.

Дискретизация по времени (или *дискретизация*) представляет собой процедуру взятия мгновенных значений сигнала $x(t)$ через равные промежутки времени T. Мгновенные значения $x(nT)$ называются *выборками*, или *отсчетами*, время T — *периодом дискретизации*, n указывает порядковый номер отсчета. Ясно, что чем чаще брать отсчеты, т. е. чем меньше период дискретизации T, тем точнее последовательность отсчетов $x(nT)$ будет отображать исходный сигнал $x(t)$. Период дискретизации T определяет *частоту дискретизации*.

$$f_д = \frac{1}{T}; \qquad T = \frac{1}{f_д}, \tag{1.1}$$

откуда чем меньше T, тем выше $f_д$. С другой стороны, чем выше частота дискретизации, тем труднее вычислителю выполнять большое количество операций над отсчетами в темпе их поступления на переработку и тем сложнее должно быть его устройство.

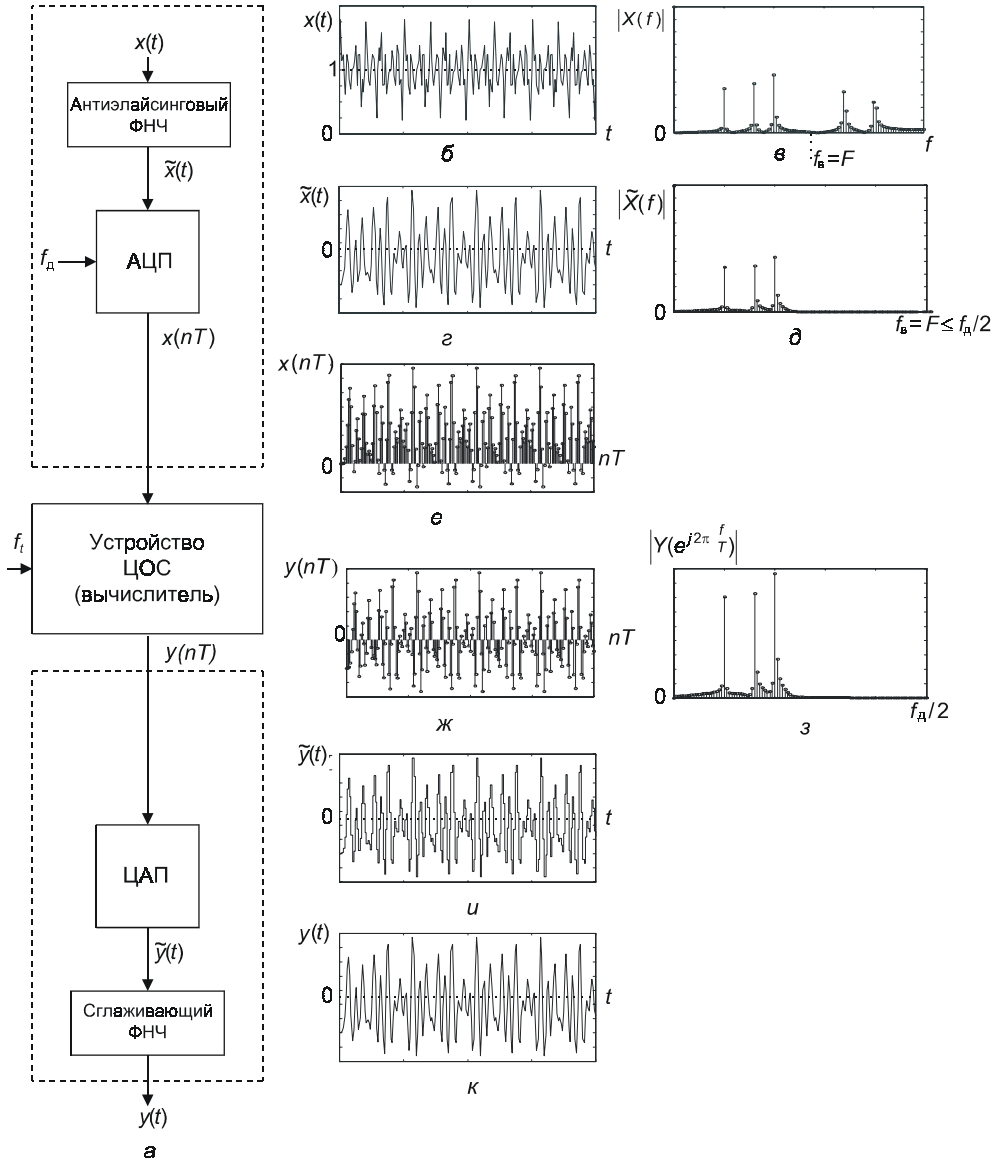


Рис. 1.2. Обобщенная схема цифровой обработки аналоговых сигналов

Таким образом, точность представления сигнала требует увеличивать $f_д$, а стремление сделать вычислитель как можно более простым приводит к желанию понизить $f_д$. Однако существует ограничение на минимальное значение $f_д$: для полного восстановления непрерывного сигнала по его отсче-

там $x(nT)$ необходимо и достаточно, чтобы частота дискретизации $f_д$ была, как минимум, в два раза больше наивысшей частоты F в спектре передаваемого сигнала $x(t)$, т. е.

$$f_д \geq 2F, \qquad T \leq 1/2F. \tag{1.2}$$

Отсюда следует, что при бесконечном спектре, когда $F \to \infty$, дискретизация невозможна. Тем не менее, в спектре любого конечного сигнала есть такие высшие составляющие, которые, начиная с некоторой верхней частоты $f_в$, имеют незначительные амплитуды, и потому ими можно пренебречь без заметного искажения самого сигнала. Значение $f_в$ определяется конкретным типом сигнала и решаемой задачей. Например, для стандартного телефонного сигнала $f_в$ = 3,4 кГц минимальная стандартная частота его дискретизации $f_д$ = 8 кГц. Ограничение спектра до частоты $F = f_в$ осуществляется аналоговым **ФНЧ**, получившим название *антиэлайсингового*, поскольку он предотвращает искажения спектра типа *наложения (aliasing)*, которые возникают в спектре цифрового сигнала при недостаточной частоте дискретизации. Во временной области эффект наложения означает необратимую потерю возможности точного восстановления аналогового сигнала по его отсчетам.

Антиэлайсинговый фильтр формирует аналоговый сигнал со значительно подавленными верхними частотными составляющими (рис. 1.2, *г*, *д*) в полосе задерживания, начиная с частоты $F = f_в$. Это дает основание считать сигнал практически ограниченным по частоте и неподверженным эффекту наложения при частоте дискретизации не менее $2F$.

Квантование отсчетов по уровням (или *квантование*) производится с целью формирования последовательности чисел: весь диапазон изменения величины отсчетов разбивается на некоторое количество дискретных уровней (рис. 1.2, *е*), и каждому отсчету по определенному правилу присваивается значение одного из двух ближайших уровней квантования, между которыми оказывается данный отсчет. В результате получается последовательность чисел $x(nT) = x(n)$, представляемых в двоичном коде. Количество уровней определяется разрядностью b АЦП; так, если $b = 3$, всего можно иметь $k = 2^b = 2^3 = 8$ уровней квантования, а минимальное и максимальное значения отсчетов равны соответственно $0 \Leftrightarrow 000$ и $7 \Leftrightarrow 111$. Ясно, что квантованный отсчет отличается от выборки $x(nT)$. Это отличие выражается ошибкой квантования

$$\varepsilon_{кв} = x_ц(nT) - x(nT), \tag{1.3}$$

которая тем больше, чем меньше b. Максимальная ошибка квантования при использовании округления в качестве приближения равна половине шага квантования Q

$$\max|\varepsilon_{кв}| = Q/2, \quad \text{где} \quad Q = q_1 = 2^{-b}. \tag{1.4}$$

Отсюда следует, что чем больше разрядность b АЦП, тем точнее представляется отсчет, но тем сложнее и дороже оказывается АЦП, который необходим для решения поставленной задачи. Современные АЦП имеют разрядность от 8 до 20.

Последовательность $x(nT) = x(n)$ поступает на вычислитель, который по заданному алгоритму каждому отсчету $x(n)$ ставит в однозначное соответствие выходной отсчет $y(nT) = y(n)$

$$x(n) \Rightarrow y(n). \tag{1.5}$$

Результатом переработки исходного сигнала является новая цифровая последовательность — цифровой сигнал (рис. 1.2, *ж*), уже не имеющий постоянной составляющей и существенно отличающийся от $x(n)$. Амплитудный спектр (рис. 12, *з*) оказывается более обостренным на частотах, близких к частоте $f_д/4$. Количество операций (умножений, сложений, пересылок и т. д.) для получения одного отсчета $y(n)$ может исчисляться тысячами, поэтому вычислитель должен работать на более высокой тактовой частоте f_τ, чтобы успеть произвести все необходимые действия до поступления очередного отсчета $x(n)$, т. е. какой бы сложности ни был алгоритм, время переработки $t_{пер}$ не должно превышать периода дискретизации T

$$t_{пер} \le T. \tag{1.6}$$

Но это может быть обеспечено лишь в случае, когда тактовая частота f_τ вычислителя существенно превышает частоту дискретизации $f_д$. Именно при этих условиях возможна работа вычислителя в реальном времени, т. е. в темпе поступления входных отсчетов. Например, при обработке стандартного телефонного сигнала с частотой дискретизации 8 кГц для обеспечения работы вычислителя в реальном времени тактовая частота должна быть равной, по крайней мере, 6 МГц, как в процессоре первого поколения TMS320C10.

Полученные выходные отсчеты подаются на цифро-аналоговый преобразователь (ЦАП), формирующий ступенчатый сигнал $\tilde{y}(t)$ — рис. 1.2, *и*, который затем с помощью сглаживающего фильтра НЧ преобразуется в аналоговый непрерывный сигнал $y(t)$ — рис. 1.2, *к*.

Из всего сказанного вытекает ряд ограничений, влияющих на характер и выбор элементной базы для реализации вычислителя:

- разрядность регистров должна быть большой и превышать разрядность ЦАП во избежание дополнительных ошибок при округлении результатов вычислений;

- тактовая частота, на которой работает вычислитель, должна в сотни раз превосходить частоту дискретизации, если предъявляется требование реального времени;
- малое энергопотребление;
- компактность.

1.2. Основные направления, задачи и алгоритмы ЦОС

Среди многочисленных задач, решаемых на базе ЦОС, можно выделить группу наиболее полно характеризующих как традиционные, так и нетрадиционные области применения ЦОС. Каждая задача — в зависимости от конкретного приложения — может решаться с использованием различных методов и алгоритмов; например, задача выделения сигнала из помех может решаться методами линейной, адаптивной и нелинейной фильтрации. В свою очередь, цифровая линейная фильтрация может быть осуществлена алгоритмами КИХ- или БИХ-фильтрации.

В настоящее время выделяют следующие основные направления ЦОС (табл. 1.1):

- линейная фильтрация;
- спектральный анализ;
- частотно-временной анализ;
- адаптивная фильтрация;
- нелинейная обработка;
- многоскоростная обработка.

Таблица 1.1. *Основные задачи ЦОС*

№ п/п	Направление	Примеры задач
1	Линейная фильтрация	Селекция сигнала в частотной области; синтез фильтров, согласованных с сигналами; частотное разделение каналов; цифровые преобразователи Гильберта и дифференциаторы; корректоры характеристик каналов
2	Спектральный анализ	Обработка речевых, звуковых, сейсмических, гидроакустических сигналов; распознавание образов
3	Частотно-временной анализ	Компрессия изображений, гидро- и радиолокация, разнообразные задачи обнаружения

Таблица 1.1 *(окончание)*

№ п/п	Направление	Примеры задач
4	Адаптивная фильтрация	Обработка речи, изображений, распознавание образов, подавление шумов, адаптивные антенные решетки
5	Нелинейная обработка	Вычисление корреляций, медианная фильтрация; синтез амплитудных, фазовых, частотных детекторов, обработка речи, векторное кодирование
6	Многоскоростная обработка	Интерполяция (увеличение) и децимация (уменьшение) частоты дискретизации в многоскоростных системах телекоммуникации, аудиосистемах

1.2.1. Фильтрация

Фильтрация может осуществляться с помощью цифровых фильтров (ЦФ), описываемых во временной области линейными разностными уравнениями вида:

$$y(n) = \sum_{i=0}^{N-1} b_i x(n-i); \tag{1.7}$$

$$y(n) = \sum_{i=0}^{N-1} b_i x(n-i) - \sum_{k=1}^{M-1} a_k y(n-k), \tag{1.8}$$

где $x(n)$ — отсчеты воздействия; $y(n)$ — отсчеты реакции; $\{b_i, a_k\}$ — вещественные коэффициенты, полностью определяющие свойства ЦФ; M и N — константы, задающие сложность ЦФ; $x(n-i)$ и $y(n-k)$ — отсчеты воздействия и реакции, задержанные на i и k периодов дискретизации T соответственно.

Фильтр, описываемый выражением (1.7), называют *нерекурсивным*, или *КИХ-фильтром* (фильтр с конечной импульсной характеристикой); фильтр, описываемый выражением (1.8), называют *рекурсивным*, или *БИХ-фильтром* (фильтр с бесконечной импульсной характеристикой).

Передаточные функции КИХ- и БИХ-фильтров определяются из (1.7) и (1.8) с помощью Z-преобразования и имеют вид соответственно

$$H(z) = \sum_{i=0}^{N-1} b_i z^{-i}; \tag{1.9}$$

$$H(z) = \frac{\sum_{i=0}^{N-1} b_i z^{-i}}{1 + \sum_{k=1}^{M-1} a_k z^{-k}}, \tag{1.10}$$

откуда после подстановки $z = e^{j\omega T} = e^{j\tilde{\omega}}$ получают комплексные частотные характеристики

$$H(e^{j\tilde{\omega}}) = \sum_{i=0}^{N-1} b_i e^{-j\tilde{\omega}i}; \tag{1.11}$$

$$H(e^{j\tilde{\omega}}) = \frac{\sum_{i=0}^{N-1} b_i e^{-j\tilde{\omega}i}}{1 + \sum_{k=1}^{M-1} a_k e^{-j\tilde{\omega}k}}. \tag{1.12}$$

Из (1.11) и (1.12) нетрудно получить выражения для амплитудно-частотной характеристики (АЧХ) $A(\tilde{\omega})$ и фазо-частотной характеристики (ФЧХ) $\varphi(\tilde{\omega})$:

$$A(\tilde{\omega}) = \left| H(e^{j\tilde{\omega}}) \right|, \tag{1.13}$$

$$\varphi(\tilde{\omega}) = arg\left| H(e^{j\tilde{\omega}}) \right|, \tag{1.14}$$

или в явном виде

$$A(\tilde{\omega}) = \frac{\sqrt{\left[\sum_{i=0}^{N-1} b_i \cos(\tilde{\omega}i)\right]^2 + \left[\sum_{i=0}^{N-1} b_i \sin(\tilde{\omega}i)\right]^2}}{\sqrt{\left[\sum_{k=0}^{M-1} a_k \cos(\tilde{\omega}k)\right]^2 + \left[\sum_{k=0}^{M-1} a_k \sin(\tilde{\omega}k)\right]^2}}; \tag{1.15, а}$$

$$\varphi(\tilde{\omega}) = \operatorname{arctg}\frac{-\sum_{i=0}^{N-1} b_i \sin(\tilde{\omega}i)}{\sum_{i=0}^{N-1} b_i \cos(\tilde{\omega}i)} - \operatorname{arctg}\frac{-\sum_{k\neq 0}^{M-1} a_k \sin(\tilde{\omega}k)}{\sum_{k=0}^{M-1} a_k \cos(\tilde{\omega}k)}. \tag{1.15, б}$$

Связь между воздействием и реакцией фильтра устанавливается не только разностными уравнениями, но и с помощью *свертки*

$$y(n) = \sum_{i=0}^{K} h(i)x(n-i) = \sum_{i=0}^{K} x(i)h(n-i), \tag{1.16}$$

где $h(k)$ — импульсная характеристика фильтра. Для КИХ-фильтра $K = N - 1$, отсчеты импульсной характеристики равны коэффициентам фильтра $h(i) = b_i$; для БИХ-фильтра $K = \infty$, а отсчеты импульсной характеристики определяются через коэффициенты a_k и b_i сложным образом.

Из (1.7), (1.8) и (1.16) видно, что для вычисления результатов фильтрации необходимо многократно выполнять следующие операции:

- сложение;
- вычитание;
- умножение;
- сдвиг, реализующий задержку сигнала на один период дискретизации.

Перечисленные операции называются *основными* (или *базовыми*) *операциями*.

1.2.2. Преобразование Гильберта

Среди задач обработки сигналов важное место принадлежит модуляции и демодуляции узкополосных сигналов и задачам сдвига частоты (например, в модемах в режиме подстройки), что характерно для систем радиосвязи. Примером такой задачи является демодуляция однополосного сигнала, который получается выделением одной из боковых полос амплитудно-модулированного сигнала. Результатом демодуляции является низкочастотный сигнал, представляющий собой *огибающую* узкополосного сигнала. Демодулируемый сигнал $x(n)$ можно представить в комплексном виде:

$$\dot{x}(n) = x(n) + j\tilde{x}(n); \tag{1.17}$$

$$x(n) = s(n)\cos\varphi(\omega n); \tag{1.18}$$

$$\tilde{x}(n) = s(n)\sin\varphi(\omega n), \tag{1.19}$$

где $x(n)$ — вещественный сигнал; $\tilde{x}(n)$ — мнимый сигнал; $s(n)$ — огибающая сигнала $x(n)$, которая, как следует из (1.17), определяется по формуле

$$s(n) = \sqrt{x^2(n) + \tilde{x}^2(n)}. \tag{1.20}$$

Из (1.18) и (1.19) видно, что $x(n)$ и $\tilde{x}(n)$ находятся в квадратуре относительно друг друга, т. е. их фазы отличаются на $\pi/2$. Следовательно, необходимо иметь фазовращатель на $\pi/2$. Такие сигналы называются сопряженными по Гильберту, а устройство, формирующее пару сопряженных сигналов, называется *цифровым преобразователем Гильберта* (ЦПГ) (рис. 1.3), который позволяет организовать вычисление огибающей $s(n)$ сигнала $x(n)$. Согласующая линия задержки (СЛЗ) обеспечивает временное согласование сигналов $x(n)$ и $\tilde{x}(n)$.

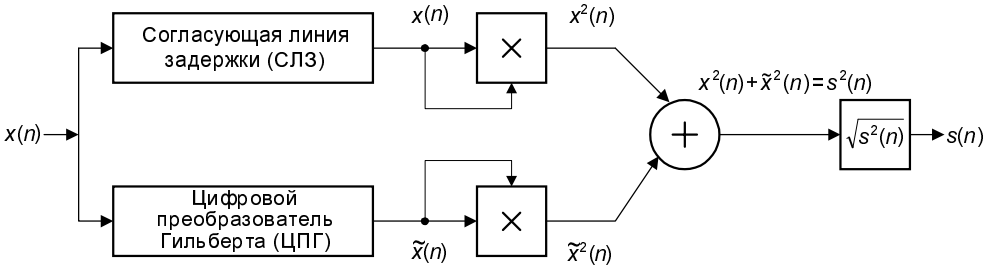


Рис. 1.3. Структурная схема выделения огибающей $s(n)$ сигнала $x(n)$

Собственно ЦПГ представляет собой полосовой КИХ-фильтр с линейной ФЧХ, имеющий антисимметричную импульсную характеристику нечетной длины. ФЧХ такого фильтра имеет постоянную составляющую $\pi/2$ (рис. 1.4, *а*, *б*).

Антисимметричность означает

$$h(n) = -h(N-1-n)\,. \qquad (1.21)$$

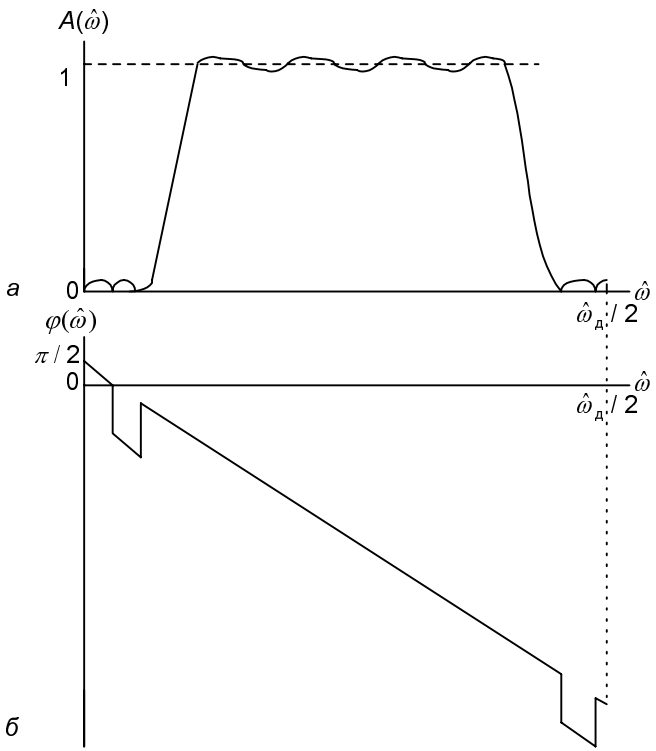


Рис. 1.4. Частотные характеристики ЦПГ: *а* — АЧХ, *б* — ФЧХ

Структурная схема ЦПГ изображена на рис. 1.5, где каждый второй коэффициент (или отсчет импульсной характеристики) равен нулю. Коэффициенты ЦПГ рассчитываются по формулам:

$$b(n) = h(n) = \begin{cases} \dfrac{2\sin 2(\pi n/2)}{\pi n}, & n \neq 0; \\ 0, & n = 0. \end{cases} \tag{1.22}$$

В качестве СЛЗ используются первые $(N - 1)/2$ регистров всей линии задержки фильтра, содержащей N регистров.

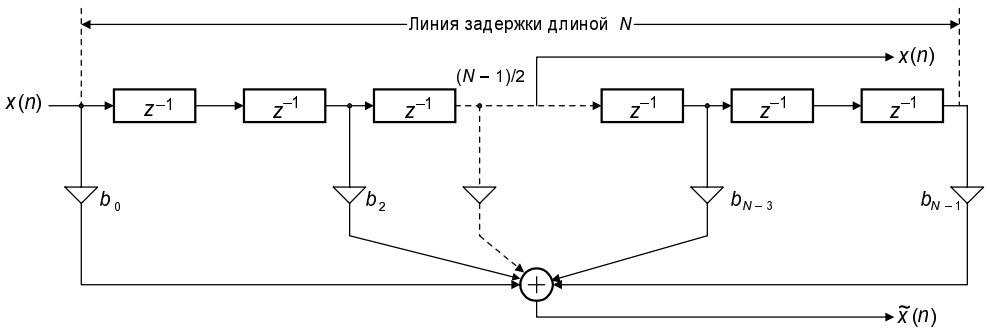


Рис. 1.5. Структурная схема цифрового преобразователя Гильберта

1.2.3. Дифференциатор

Дифференциаторы — это КИХ-фильтры с линейной ФЧХ, имеющие антисимметричную импульсную характеристику четной длины, линейно возрастающую АЧХ в рабочем диапазоне частот $0 \le \omega \le \omega_p$ и постоянную составляющую ФЧХ, равную $\pi/2$ (рис. 1.6).

Свое название такие КИХ-фильтры получили благодаря свойству дифференцирования сигнала. Покажем это.

На рис. 1.6 видно, что отношение приращения АЧХ $\Delta A(\omega)$ к приращению частоты $\Delta\omega$ постоянно

$$K = \frac{\Delta A(\omega)}{\Delta\omega} = \text{const}. \tag{1.23}$$

При устремлении $\Delta\omega \to 0$ получаем

$$A'(\omega) = \frac{dA(\omega)}{d\omega}, \tag{1.24}$$

откуда

$$dA(\omega) = A'(\omega)d\omega. \tag{1.25}$$

Аналогично можно записать для сигнала $s(\omega)$:

$$ds(\omega) = s'(\omega)\,d\omega. \qquad (1.26)$$

Последнее равенство означает, что сигнал на выходе рассматриваемого фильтра представляет собой производную входного сигнала, задержанного на некоторое время, определяемое фазочастотной характеристикой, т. е. такое устройство действительно обладает свойством дифференцирования.

В системах управления часто требуется линейное изменение коэффициента управления $K(\omega)$ (например, усиление или ослабление нагрузки) в зависимости от частоты ω воздействия. Для этой цели используют дифференциаторы, линейно возрастающую АЧХ которых можно истолковать как частотнозависимый коэффициент управления $K(\omega)$ в этом же диапазоне частот: подавая на вход дифференциатора гармонические колебания, получаем линейно зависимые от частоты коэффициенты управления, рассчитываемые согласно (1.15, *а*).

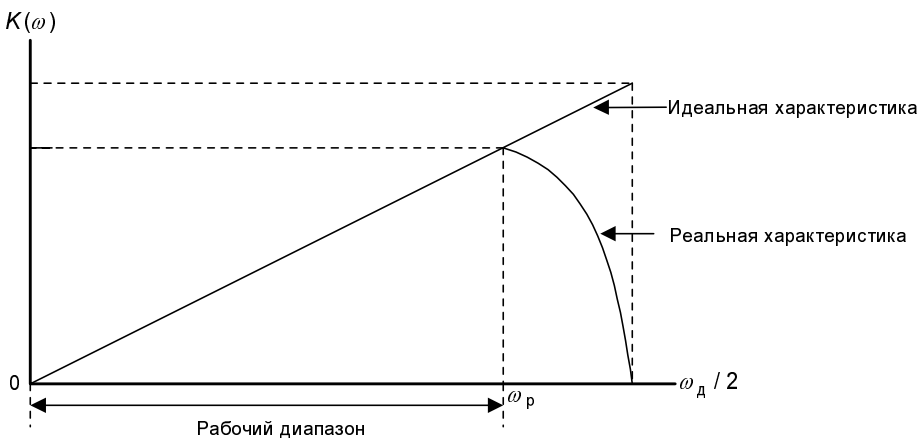


Рис. 1.6. АЧХ дифференциатора $[K(\omega)]$

Заметим, что вычисление огибающей сигнала и частотнозависимого коэффициента управления в дополнение невозможно без многократных вычислений тригонометрических функций и извлечения квадратного корня из суммы двух величин.

1.3. Цифровой спектральный анализ

Цифровой спектральный анализ — это совокупность разнообразных методов обработки цифровых сигналов, которые позволяют оценить частотный состав (спектр) исследуемого сигнала. Задача спектрального анализа может

носить как самостоятельный характер (например, в сейсмологии для определения типа сейсмического события, в геофизике для поиска месторождений полезных ископаемых и разработки новых методов поиска и т. п.), так и вспомогательный (в системах компрессии речи и изображений, компенсации помех и фильтрации).

В ЦОС важнейшими сигналами являются периодические последовательности с периодом отсчетов N и последовательности конечной длины в N отсчетов.

Для периодических последовательностей вводится *дискретное преобразование Фурье* (*ДПФ*):

прямое ДПФ
$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad 0 \le k \le N-1; \tag{1.27}$$

обратное ДПФ (ОДПФ)
$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk}, \quad 0 \le n \le N-1, \tag{1.28}$$

где $X(k)$ — k-я комплексная амплитуда (составляющая) спектра (ДПФ); $x(n)$ — отсчеты дискретного сигнала (периодического с периодом N или конечной длины N); W_N^{nk} — поворачивающий множитель (или ядро преобразования), равный

$$W_N^{nk} = e^{-j\frac{2\pi}{N}nk}. \tag{1.29}$$

Составляющие спектра $X(k)$ имеют период N и располагаются по частотной оси с интервалом

$$\Delta\omega = 2\pi / NT, \quad T = 1/f_{\text{Д}}. \tag{1.30}$$

Этот интервал называется *частотой преобразования*. Составляющая с номером k располагается на частоте

$$\omega_k = \Delta\omega k, \quad X(k) = X\left(e^{jk\Delta\omega}\right). \tag{1.31}$$

Говорят, что ДПФ является N-точечным, если оно содержит N составляющих спектра. Модуль ДПФ $|X(k)|$ называется *спектром амплитуд* (часто просто *спектром*, если нет каких-либо оснований для уточнения), аргумент ДПФ называется спектром фаз

$$\varphi(k) = \arg X(k). \tag{1.32}$$

Это же преобразование можно применить и к последовательности конечной длины, рассматривая ее как один период повторяющейся последовательности.

Замечание

Следует иметь в виду, что смысл ДПФ для дискретных периодических сигналов и сигналов конечной длины различен. Спектр периодического дискретного сигнала с периодом N является также периодическим и дискретным, имеющим на одном периоде ровно N комплексных составляющих, поэтому ДПФ *точно выражает его спектр*. С дискретным сигналом конечной длины N дело обстоит несколько сложнее: его спектр, являясь периодическим с периодом $\omega_д$, представляет собой непрерывную функцию частоты, однако по N равноотстоящим отсчетам спектра $X(k)$, т. е. по ДПФ, гарантируется возможность *точного восстановления* как непрерывного спектра $X(e^{j\tilde{\omega}})$, так и последовательности $x(n)$.

1.3.1. Быстрое преобразование Фурье

Прямое вычисление ДПФ по формулам (1.27) и (1.28) для больших N (например, при обработке речевых сигналов длина одного фрагмента N достигает значения $2^{10} = 1024$) крайне неэффективно и может стать препятствием для обеспечения реального времени. Действительно, для вычисления N-точечного преобразования требуется произвести $(N-1)^2$ комплексных умножений и $N(N-1)$ комплексных сложений, т. е. объем вычислений имеет порядок N^2 операций сложения и умножения комплексных чисел.

Для уменьшения вычислительных затрат разработаны *алгоритмы* быстрого вычисления ДПФ, называемые *быстрым преобразованием Фурье (БПФ)*. Эти алгоритмы основаны на периодичности ядра преобразования W_N^{nk}. Идея БПФ состоит в том, чтобы разделить N-точечную последовательность на две, из ДПФ которых можно получить ДПФ исходной последовательности, и продолжать такое деление каждой новой последовательности до тех пор, пока не останутся последовательности, состоящие только из двух элементов. Конечно, такое деление возможно лишь при $N = 2^m$.

Алгоритм БПФ с прореживанием по времени

Исходная N-точечная последовательность $x(n)$ делится на две $N/2$-точечных последовательности, одна из которых содержит отсчеты с нечетными номерами, а другая — с четными номерами:

- четная последовательность $x_1(n) = x(2n)$;
- нечетная последовательность $x_2(n) = x(2n+1)$

при $n = 0, 1, ..., (N/2)-1$. Тогда N-точечное ДПФ исходной последовательности $x(n)$ преобразуется в два $N/2$-точечных ДПФ:

$$X(k) = \begin{cases} X_1(k) + W_N^k X_2(k); \\ X_1(k) - W_N^k X_2(k), \end{cases} \qquad (1.33)$$

где верхняя строка дает первые $N/2$ составляющих ДПФ $X(k)$, а нижняя строка — вторые $N/2$ составляющих, $0 \le k \le N/2-1$. Далее аналогичным образом $N/2$-точечные ДПФ заменяются двумя $N/4$-точечными каждое, и т. д. Такая сортировка осуществляется до тех пор, пока не образуются $N/2$ последовательностей по два элемента в каждой. В результате N-точечное ДПФ сводится к $m = \log_2 N$ этапам, на каждом из которых требуется вычислить N коэффициентов. Выражения (1.33) показывают, что на каждом этапе требуется N комплексных сложений и $N/2$ комплексных умножений. Это легко видеть из направленного графа (рис. 1.7), напоминающего крылья бабочки, чем и объясняется название *"бабочка"* самой операции (1.33) и графа, на котором стрелкой обозначено умножение на W_N^{nk}. Использование базовой операции "бабочка" снижает количество требуемых для вычисления N-точечного ДПФ комплексных сложений с N^2 до $\frac{3}{2} N \log_2 N$, что является существенной экономией вычислительных, а потому и временных ресурсов.

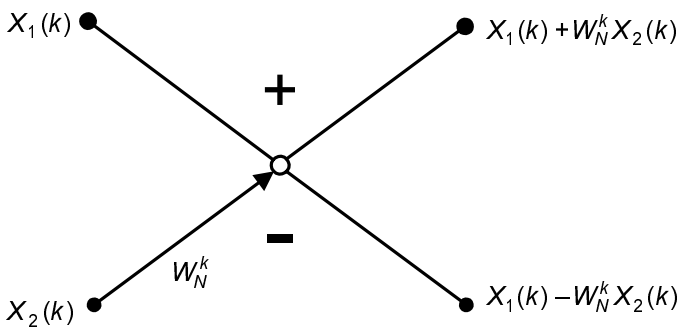


Рис. 1.7. Операция "бабочка" с прореживанием по времени

В результате сортировки отсчетов $x(n)$ по нечетным и четным номерам входные данные записываются в необычном порядке, который называется *двоичной инверсией*, или *бит-реверсией*: например, если трехразрядное двоичное представление номера n отсчета $x(n)$ имеет вид $n = \{n_2\ n_1\ n_0\}$, то отсчет $x\{n_2\ n_1\ n_0\}$, должен располагаться на месте $x\{n_0\ n_1\ n_2\}$. Это означает, что для правильного выполнения БПФ необходимо в исходном двоичном номере заменить порядок расположения разрядов на обратный (инверсный). Например, входной отсчет $x(6) = x(110)$ должен разместиться на $011 \Rightarrow 3$ третьем месте, а третий отсчет $x(3)$ окажется на шестом месте. При двоичной инверсии входной последовательности составляющие $X(k)$ ДПФ будут расположены в естественном порядке.

Алгоритм БПФ с прореживанием по частоте

В этом случае входная последовательность $x(n)$ делится пополам на $N/2$ первых и $N/2$ последних отсчетов и так до тех пор, пока не сформируются $N/2$ двухэлементных последовательностей. Базовая операция "бабочка" будет описываться выражением (1.34), а ее направленный граф примет вид, показанный на рис. 1.8.

$$X(k) = \begin{cases} X_1(k) + X_2(k); \\ (X_1(k) - X_2(k))W_N^k. \end{cases} \quad (1.34)$$

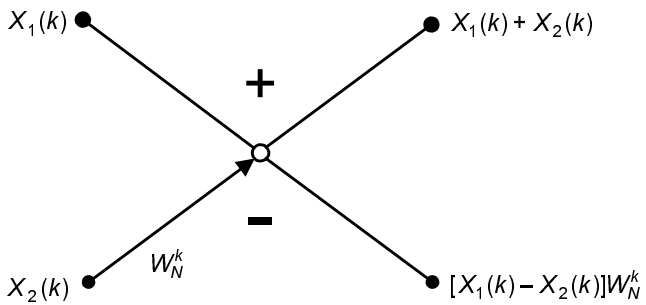


Рис. 1.8. Операция "бабочка" с прореживанием по частоте

Вычисление согласно данному алгоритму приводит к тому, что составляющие $X(k)$ ДПФ располагаются в порядке, соответствующем бит-реверсии, поэтому их необходимо пересортировать согласно естественному порядку. Например, $X(3) = X(011)$ следует разместить на шестом месте, т. к. инверсный номер оказывается равным $110 \Rightarrow 6$.

Замечание

Алгоритмы БПФ являются рекурсивными: невозможно рассчитать $N/2$-точечное ДПФ, не рассчитав предварительно $N/4$-точечное ДПФ.

1.3.2. Дискретное преобразование Хартли (ДПХ)

Дискретное преобразование Фурье отображает последовательность вещественных данных в комплексную область, где хорошо разработаны методы анализа, существенно облегчающие изучение и трактовку колебательных процессов. Однако обработку вещественных данных желательно выполнять в вещественной области. Эту задачу решает дискретное преобразование Хартли (ДПХ), которое, как и ДПФ, может применяться в задачах спектрального анализа и цифровой фильтрации.

Прямое и обратное дискретное преобразование Хартли вещественной последовательности $x(n)$ длины N определяются соотношениями:

$$H(k) = \sum_{n=0}^{N-1} x(n)\mathrm{cas}\left(\frac{2\pi nk}{N}\right); \tag{1.35}$$

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} H(k)\mathrm{cas}\left(\frac{2\pi nk}{N}\right), \tag{1.36}$$

где $\mathrm{cas}\theta = \cos\theta + \sin\theta$. Между ДПФ и ДПХ существует простая прямая связь:

$$\mathrm{Re}\{X(k)\} = \frac{H(k) + H(-k)}{2}; \tag{1.37}$$

$$\mathrm{Im}\{X(k)\} = \frac{H(k) - H(-k)}{2}. \tag{1.38}$$

С другой стороны, по известным составляющим $X(k)$ ДПФ можно получить ДПХ

$$H(k) = \mathrm{Re}\{X(k)\} - \mathrm{Im}\{X(k)\}. \tag{1.39}$$

Симметричность формул прямого и обратного ДПХ, отсутствие комплексного представления данных и ряд других свойств ДПХ обеспечивают по сравнению с БПФ более высокую вычислительную эффективность при обработке вещественных данных, что особенно важно при двумерном анализе.

1.3.3. Дискретное косинусное преобразование

Одной из весьма сложных задач ЦОС является построение видеокодеков для сжатия изображения с целью уменьшения скорости передачи, а потому и уменьшения спектра, занимаемого видеосигналом. Рассмотрим существо проблемы на примере телевизионного сигнала.

Применяемая система со строчно-переменной фазой PAL (Phase Alternate Line) для образования одного кадра черно-белого изображения использует 625 строк, а на каждой строке 625 элементов (пикселов). Следовательно, для получения одного кадра изображения всего необходимо 380 625 пикселов. Для воспроизведения движения приемлемого качества и исключения эффекта дрожащего экрана требуется формировать около 50 кадров в секунду. Отсюда нетрудно рассчитать скорость передачи

$$380\ 625 \times 50 = 19\ 031\ 250 \text{ пикселов/с},$$

что соответствует полосе пропускания около 20 МГц. Практически используется более узкая полоса около 6 МГц, что достигается, во-первых, сокращением числа передаваемых строк до 575 и, во-вторых, распределением строк по двум кадрам: нечетные строки выводятся в первом кадре, четные —

во втором. При этом частота дискретизации составляет 12 МГц. Однако добиться изображения хорошего качества таким способом не удается.

С цветными изображениями дело обстоит еще сложнее, поскольку необходима дополнительная информация о цвете, яркости, а также применение помехозащитного кодирования. Поэтому скорость передачи в канале цветного телевидения возрастает до 200 Мбит/с, что для практических целей совершенно недопустимо.

Целью сжатия как раз и является снижение скорости передачи изображения без существенных, т. е. незаметных глазу, искажений. Все изображения делят на два обширных класса: *статические* (например, фотографии) и *подвижные*, которые имеют свои особенности.

В связи с этим был разработан ряд международных стандартов компрессии изображений:

- стандарт JPEG (Joint Photographic Experts Group, Объединенная фотографическая экспертная группа) предназначен для сжатия статических (неподвижных) изображений;
- стандарты H.320 (и его часть H.261), H.263 предназначены для сжатия подвижных изображений в видеофонах и системах видеоконференций;
- серия стандартов MPEG*n* (Moving Pictures Experts Group, Экспертная группа по движущимся изображениям; *n* — номер стандарта) предназначен для сжатия подвижных изображений в более развитых системах, в частности, MPEG4 в цифровых носителях памяти, таких как CD-ROM.

Все стандарты основаны на *кодировании преобразованием*, существо которого состоит в следующем. Цветное или черно-белое изображение расщепляется на блоки из 64 = 8×8 пикселов. К каждому блоку применяется *частотное преобразование*. Таким частотным преобразованием могло бы служить ДПФ, однако оно формирует мнимые коэффициенты, которые настолько усложняют вычисления, что исключается возможность применения ДПФ для целей сжатия изображений.

В указанных стандартах используется другое, подобное ДПФ, преобразование — *дискретное косинусное преобразование (ДКП)*, которое имеет только вещественные частотно-зависимые коэффициенты. Смысл ДКП рассматривается ниже. На блоке из 64 пикселов формируется двумерный массив коэффициентов, которые переупорядочиваются и преобразуются в одномерный массив. Полученные коэффициенты подвергаются дальнейшей статистической обработке, и соответствующие им параметры передаются по каналу связи. На приеме осуществляется восстановление изображения (синтез) путем обратного преобразования. Эта процедура обеспечивает существенное сокращение скорости передачи вплоть до 20 раз (коэффициент сжатия достигает 20).

Дискретное косинусное преобразование представляет сигнал рядом гармонических функций. Двумерное $N \times N$ ДКП (2-ДКП) имеет вид:

$$X(k,n) = \frac{2}{N} C(k)C(n) \sum_{m=0}^{N-1} \sum_{l=0}^{N-1} x(m,l) \cos\left[\frac{(2l+1)k\pi}{2N}\right] \cos\left[\frac{(2m+1)n\pi}{2N}\right]; \quad (1.40)$$

обратное двумерное ДКП (2-ОДКП) определяется по формуле:

$$x(l,m) = \frac{2}{N} \sum_{n=0}^{N-1} \sum_{k=0}^{N-1} C(k)C(n)X(k,n) \cos\left[\frac{(2l+1)k\pi}{2N}\right] \cos\left[\frac{(2m+1)n\pi}{2N}\right], \quad (1.41)$$

где

$$C(i) = \begin{cases} \frac{1}{\sqrt{2}}, & i = 0; \\ 1, & i \neq 0. \end{cases} \quad (1.42)$$

Здесь $x(l, m)$ — яркость пиксела по координатам x и y соответственно; $X(k, n)$ — коэффициенты 2-ДКП этого пиксела.

Одним из свойств 2-ДКП является его разделимость. Это означает, что двумерное ДКП блока размерности $N \times N$ можно представить через одномерные ДКП по строкам и столбцам этого блока. Одномерные прямое и обратное ДКП задаются выражениями:

ДКП
$$X(k) = \sqrt{\frac{2}{N}} C(k) \sum_{n=0}^{N-1} x(n) \cos\left[\frac{(2n+1)k\pi}{2N}\right]; \quad (1.43)$$

ОДКП
$$x(n) = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} C(k)X(k) \cos\left[\frac{(2n+1)k\pi}{2N}\right]. \quad (1.44)$$

Видно, что все коэффициенты ДКП являются вещественными.

Разделение двумерного ДКП на одномерные приводит к существенному снижению количества умножений. Действительно, можно показать, что для вычисления двумерного ДКП блока размерности $N \times N$ при разделении необходимо $2N^3$ умножений, в то время как непосредственное применение 2-ДКП требует N^4 умножений. Например, при стандартном расщеплении изображения 256×256 на 1024 блока по 8×8 пикселов количество умножений снижается с $1024 \times 8^4 = 4\,194\,304$ до $1024 \times 2 \times 8^3 = 1\,048\,576$.

Из сказанного следует, что дискретное косинусное преобразование может обеспечить обработку изображений в реальном времени, если элементная база позволяет быстро производить умножение.

Замечание

Основная информация об изображении содержится в низкочастотной области и передается соответствующими коэффициентами ДКП $X(k)$; поэтому коэффициенты $X(k)$, отображающие высокочастотную область, оказываются нулевыми или близкими к нулю. Применение ДКП к цветному изображению позволяет снизить скорость передачи по каналу связи с 200 Мбит/с до 34 Мбит/с, т. е. коэффициент сжатия приблизительно равен 6.

Сказанное позволяет сделать вывод: задачи цифрового спектрального анализа связаны с многократной обработкой комплексных чисел, больших массивов данных, их особым упорядочиванием для обеспечения быстрых вычислений.

1.4. Нелинейная обработка

В следующих разделах приводятся примеры широко используемых алгоритмов нелинейной обработки.

1.4.1. Вычисление квадратного корня

Одной из типичных практических задач ЦОС является многократное вычисление АЧХ фильтра с передаточной функцией $H(z) = 1/A(z)$ за достаточно короткий временной отрезок, что характерно для вокодеров с линейным предсказанием *(см. разд. 1.5)*. Для этого потребуется на большом множестве частот $\{\omega_i\}$ вычислять величину

$$\left|H(e^{j\omega T})\right| = 1/\left|A(e^{j\omega T})\right| = \left(x_1^2 + x_2^2\right)^{-1/2}, \qquad (1.45)$$

где

$$z = e^{j\omega T}, \quad x_1 = \mathrm{Re}\left\{A(e^{j\omega T})\right\}, \quad x_2 = \mathrm{Im}\left\{A(e^{j\omega T})\right\}. \qquad (1.46)$$

Если аппаратная реализация операции извлечения корня в процессорах отсутствует, ее воплощают программно, что требует нескольких десятков командных циклов процессора. В задачах реального времени это может оказаться недопустимым, в связи с чем было предложено аппроксимировать функцию $\sqrt{x_1^2 + x_2^2}$ полиномом невысокого порядка F, удобным для программной реализации на языке ассемблера и отвечающим необходимой точности.

Решение минимаксной задачи приводит к полиному второго порядка

$$F = 0{,}828427(x_1 + x_2) - 0{,}3431452x_1x_2, \qquad (1.47)$$

дающему погрешность $\delta \in [0;\ 0{,}171573]$, откуда $\varepsilon_{\max} = 1{,}636$ дБ, что удовлетворяет требованиям, предъявляемым к системам обработки речи.

1.4.2. Вычисление функции $\cos(\hat{\omega})$

Вычисление тригонометрических функций нетрудно организовать через вычисление $\cos(\hat{\omega})$, для чего чаще всего в памяти процессора формируют таблицы, к которым обращаются в ходе вычислений. Когда создание таблиц затруднено или нецелесообразно вследствие недостаточной точности, необходимо организовывать специальную вычислительную процедуру, которая отвечала бы двум критериям: необходимой точности и минимуму команд. С этих позиций наилучшей оказалась аппроксимация в виде отрезка ряда по полиномам Чебышева

$$\cos(\lambda x) = \sum_{n=0}^{\infty} c_n(\lambda) T_{2n}(x), \tag{1.48}$$

где $x = \hat{\omega} / \lambda$; $\lambda = \pi$, т. е. $x \in (0; 1)$. Исследования, выполненные по отношению к компрессии речи, показали, что при допустимом отклонении частотной характеристики достаточно иметь $\max\{n\} = 3$. В этом случае получаем

$$\cos(x) = 0{,}998568 - 4{,}888184x^2 + 3{,}819208x^4 - 0{,}930944x^6. \tag{1.49}$$

1.4.3. Вычисление полиномов

Алгебраические и тригонометрические полиномы, используемые в цифровой обработке, могут быть представлены в алгебраическом виде простой заменой переменных. Поэтому алгоритм вычисления всякого полинома нетрудно свести к вычислению алгебраического полинома

$$y(x) = \sum_{i=0}^{M} a_i x^i. \tag{1.50}$$

Ясно, что вычисление полинома непосредственно по формуле (1.50) требует M операций сложения и $(M - 1)M/2$ операций умножения, т. е. всего необходимо произвести $(M + 1)M/2$ арифметических операций, что при больших M, характерных для ЦОС, может стать критическим относительно обеспечения реального времени. Кроме того, при использовании арифметики ФТ *(см. главу 4)* в процессе вычисления могут появиться две опасности: одна — переполнение в сумматорах, другая — потеря значимости вследствие возведения в большую степень числа x, по абсолютной величине не превышающего 1.

Во избежание перечисленных недостатков прямого вычисления полинома используют *алгоритм Горнера*

$$y(x) = a_0 + x(a_1 + x(a_2 + \ldots + a_M)\ldots), \tag{1.51}$$

который требует M умножений и M сложений, т. е. общее количество арифметических операций составляет всего $2M$, что в $(M + 1)/4$ раза меньше, чем

в случае прямого вычисления. Данный алгоритм также не приводит к переполнению и потере значимости.

Алгоритм Горнера нетрудно применить и для вычисления многомерных полиномов. Процедуру поясним на примере двумерного полинома

$$y(x)=\sum_{i}^{M/2}\sum_{j}^{M/2}a_{ij}x_1^i x_2^j. \qquad (1.52,\ а)$$

Раскроем этот полином, т. е. представим его в виде сумм:

$$\begin{aligned} y(x)=&a_{00}+\\ &+a_{10}x_1+a_{01}x_2+\\ &+a_{20}x_1+a_{11}x_1x_2+a_{02}x_2^2+\\ &\ldots\ldots\ldots\ldots\ldots\ldots\\ &+a_{M0}x_1^M+a_{M-1,1}x_1^{M-1}x_2+\ldots+a_{0M}x_2^M, \end{aligned} \qquad (1.52,\ б)$$

откуда, взяв суммирование по столбцам, получим более удобную запись

$$y(x)=\left[A_0(x_1)\right]x_2^0+\left[A_1(x_1)\right]x_2^1+\left[A_2(x_1)\right]x_2^2+\ldots+A_M(x_1)x_2^M. \qquad (1.52,\ в)$$

Таким образом, можно записать

$$y(x)=\sum_{i=0}^{M}A_i(x_1)x_2^i, \qquad (1.52,\ г)$$

т. е. исходный двумерный полином можно рассматривать как одномерный, коэффициенты которого

$$A_i(x_1)=\sum_{j=0}^{M-i}a_{j,i}x_1^j \qquad (1.53)$$

также одномерные полиномы. Разумеется, вычисление одномерных полиномов следует осуществлять с помощью алгоритма Горнера.

1.4.4. Медианные фильтры

Среди разнообразных аддитивных помех во временной области часто встречаются импульсные помехи, которые действуют на очень коротких интервалах времени и накладываются на отсчет сигнала. Одним из эффективных методов борьбы с подобными помехами является применение *медианной фильтрации*, алгоритм которой состоит в следующем:

1. Выбирается нечетное количество k отсчетов принятого сигнала, начиная с i-го отсчета.
2. Отсчеты располагаются в порядке возрастания по амплитуде.

3. Центральный отсчет данной выборки, называемый *медианой*, берется в качестве представителя этой выборки.
4. Выбираются новые k отсчетов принятого сигнала, начиная с $(i+1)$-го отсчета, и выполняются действия 2—4.

Замечание

Алгоритм начинается с $i = 0$.

Алгоритм медианной фильтрации поясняется на примере рис. 1.9, а результат отображается в табл. 1.2.

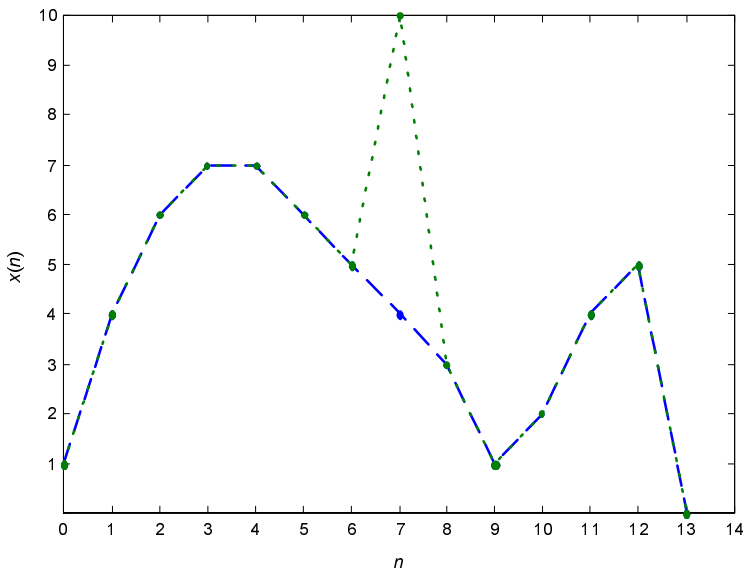


Рис. 1.9. Исходная и принятая последовательности

На рис. 1.9 точками отмечены значения отсчетов передаваемого сигнала, который, как видно, является гладким. Положим, что во время передачи в канале действовала импульсная помеха, в результате чего седьмой отсчет получил значение 10, и в принятой последовательности образовался выброс:

Значение отсчета	1	4	6	7	7	6	5	**10**	3	1	2	4	5	0
Номер отсчета	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Создадим выборки из этого ряда по три последовательных отсчета, переставим их в порядке возрастания и возьмем медиану, как показано в табл. 1.2. Результат фильтрации отображен на рис. 1.10, где точками отмечены отсчеты итогового сигнала. Из рис. 1.10 и табл. 1.2 видно, что устранение импульсной помехи сделало сигнал более плоским в областях максимумов и

минимумов, расположенных справа вблизи отсчета, подвергшегося воздействию импульсной помехи.

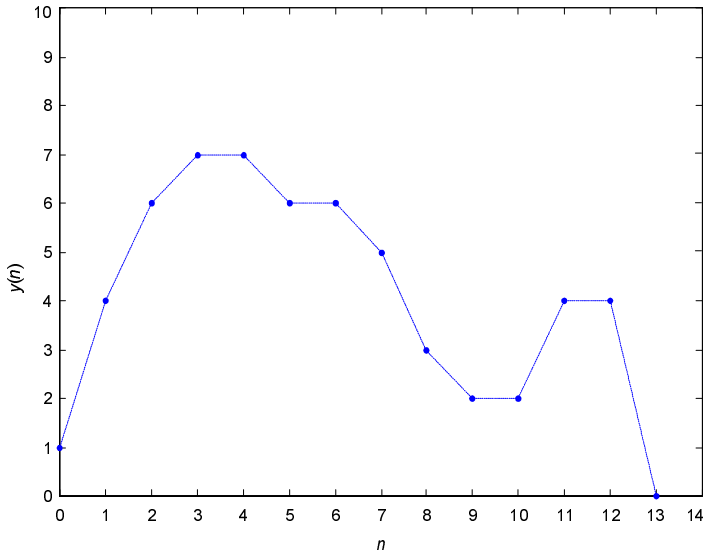


Рис. 1.10. Последовательность на выходе медианного фильтра

Таблица 1.2. *Представление медианной фильтрации*

Три последовательных отсчета		Медиана	Номер отсчета фильтрованного сигнала	Результат фильтрации
Принято	Перестановка			
1 4 6	1 4 6	4	1	4
4 6 7	4 6 7	6	2	6
6 7 7	6 7 7	7	3	7
7 7 6	6 7 7	7	4	7
7 6 5	5 6 7	6	5	6
6 5 10	5 6 10	6	6	5
5 10 3	3 5 10	5	7	10
10 3 1	1 3 10	3	8	3
3 1 2	1 2 3	2	9	1
1 2 4	1 2 4	2	10	2
2 4 5	2 4 5	4	11	4
4 5 0	0 4 5	4	12	5
5 0 0	0 0 5	0	13	0

1.4.5. Векторное квантование

Напомним, что известное скалярное квантование, осуществляемое аналого-цифровым преобразователем, состоит в том, что каждому отсчету сигнала $x(t)$ ставится в соответствие двоичное число $x(n)$; в канал связи передается также двоичное число $y(n)$, получаемое в вычислителе. Количество таких чисел определяется частотой дискретизации $f_д$, а скорость c их передачи в канале зависит еще и от разрядности b представления чисел

$$c = b \cdot f_д \tag{1.54}$$

и при передаче по каналу, например, речевого сигнала при стандартных $b = 12$, $f_д = 8000$ Гц получаем $c = 96\,000$ бит/с, что препятствует использованию стандартных телефонных каналов. Можно поступить иначе (рис. 1.11): определить в кодируемой последовательности L блоков по k отсчетов в каждом при условии, что эти отсчеты не сильно отличаются друг от друга. Эти k отсчетов могут быть отображены с заданной ошибкой (приближением) своим представителем y_i. Такие блоки называют *кластерами*, а представителя i-го кластера y_i — *центроидом*. Тогда каждый отсчет $x(n)$, принадлежащий i-му кластеру, заменяется соответствующим центроидом, и в канал связи передается только номер кластера (центроида). На передаче и приеме необходимо иметь множество из L центроидов, которое называется *кодовой книгой*, а параметр L — размером кодовой книги.

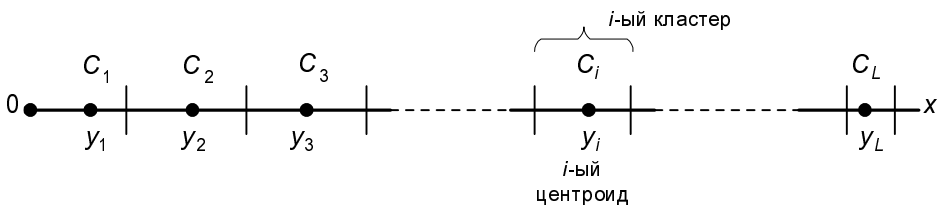


Рис. 1.11. Кластеры и центроиды одномерной последовательности

В общем случае входной последовательностью могут быть k-мерные векторы (на рис. 1.12 $k = 2$), что характерно для линейного предсказания *(см. разд. 1.5.2)*, когда $k \geq 10$ и на каждом кадре линейного предсказания приходится передавать десять параметров (линейных спектральных корней), на которые отводится не более 40 битов.

Типовые размеры кодовых книг в речевых технологиях составляют 256, 512, 1024 и 2048 центроидов (кластеров). Положим $L = 1024 = 2^{10}$; это означает, что кодовая книга содержит 1024 центроида размерностью $k = 10$ каждый. Следовательно, вместо набора из десяти параметров линейного предсказания передается только номер кластера (а потому и номер центроида), которому принадлежит данный набор, на что потребуется всего 10 битов, т. е. обеспечивается сжатие сигнала в $K_{сж} = 40/10 = 4$ раза.

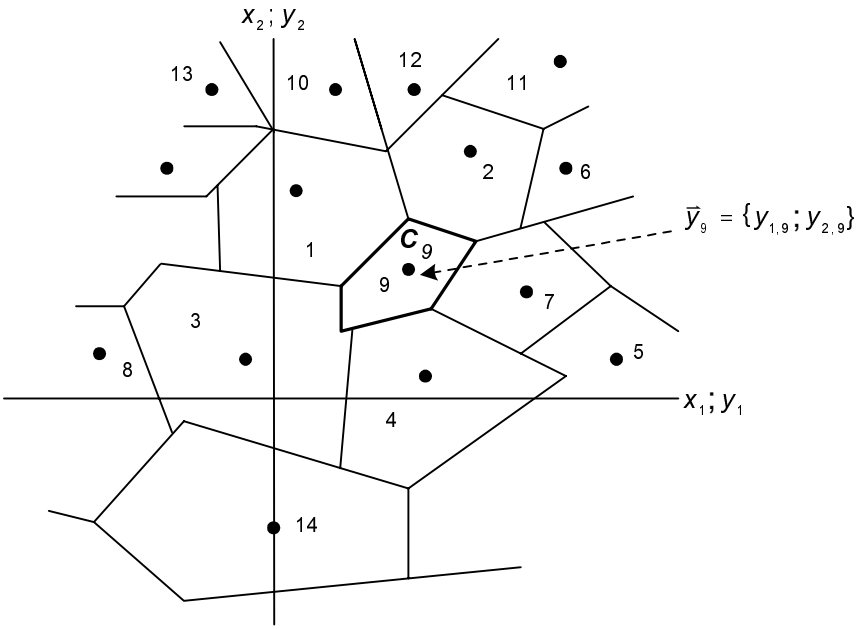


Рис. 1.12. Разбиение двумерного массива на кластеры (точками обозначены двумерные центроиды)

На практике $K_{сж}$ может быть и существенно большим, что определяется соотношениями:

$$K_{сж} = \frac{b}{R}, \quad R = \frac{\log_2 L}{k}, \tag{1.55}$$

где b — разрядность одного отсчета k-мерного вектора; R — количество битов, затрачиваемых на передачу одного отсчета. Поиск соответствующего центроида в кодовой книге простым перебором неэффективен. Поэтому ее задают в виде *бинарного дерева* с расщеплением каждого узла на два (рис. 1.13). Центроиды $\bar{y}_j = \{y_{j1}, y_{j2}, ..., y_{jk}\}$ в узлах дерева нумеруются сверху вниз и слева направо.

Нерасщепленному узлу (третьего уровня поиска в нашем случае) присваивается признак $S = 0$. Переход по левой ветви с верхнего уровня на нижний соответствует записи нуля в регистр номера искомого центроида; иначе записывается 1. Отсюда получаем следующий алгоритм поиска центроида (рис. 1.14):

1. Формируется текущий входной вектор $\vec{x} = \{x_1, x_2, ..., x_k\}$; устанавливается указатель адреса центроидов $j = 1$, признак $S = M - 1$, регистр R_N номера искомого центроида $N = 0$.

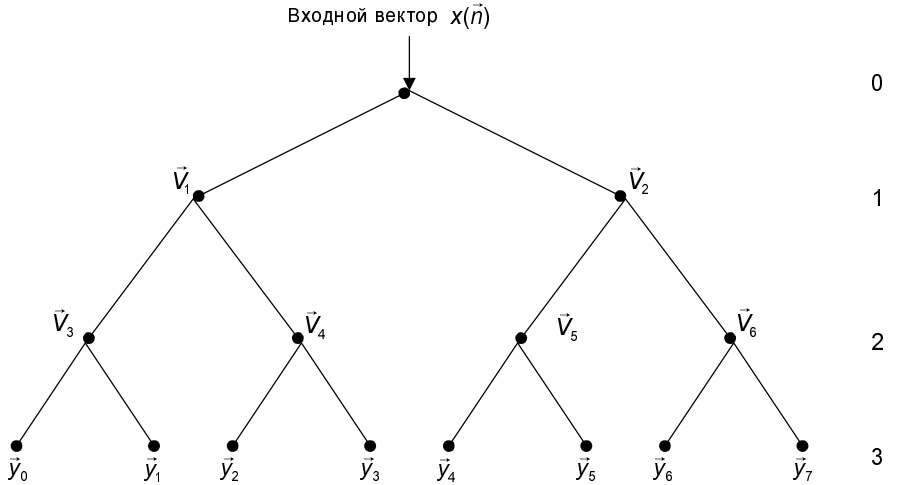


Рис. 1.13. Бинарное дерево (j = 1, ..., 8)

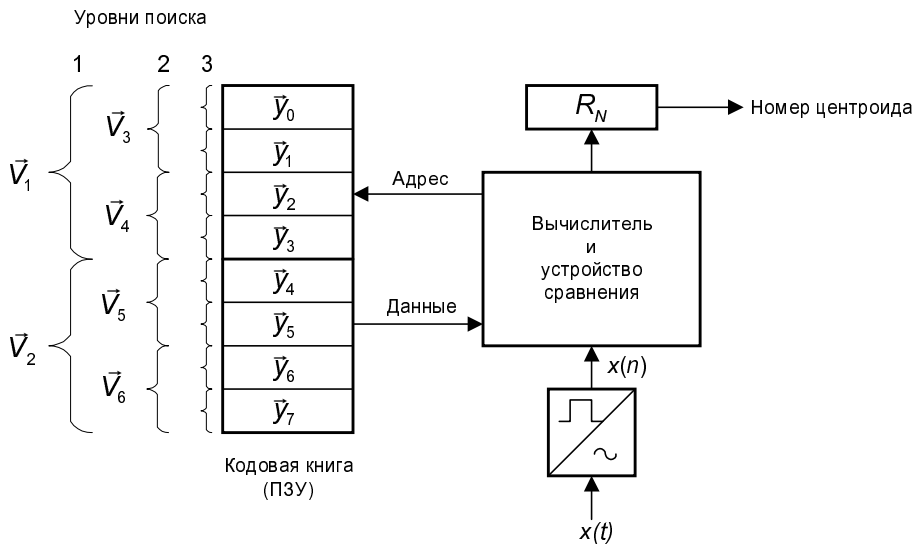


Рис. 1.14. Структурная схема векторного квантователя

2. Определяются расстояния между $\vec{x}$ и центроидами $\vec{v}_j$ и $\vec{v}_{j+1}$, например по СКО:

$$d_1 = \frac{1}{k}\sum_{i=1}^{k}\left(x_i - v_{ji}\right)^2 \qquad \text{и} \qquad d_2 = \frac{1}{k}\sum_{i=1}^{k}\left(x_i - v_{j+1,i}\right)^2. \tag{1.56}$$

3. Если $d_1 < d_2$ (движение пойдет по левой ветви), адреса центроидов равны $j = j + 2$, иначе $j = j + 3$.
4. Значение признака уменьшается на 1: $S = S - 1$. Если $S \neq 0$, содержимое R_N сдвигается влево на один бит; если $d_1 > d_2$ (движение по правой ветви), следует записать 1 в нулевой разряд регистра R_N, перейти к п. 2.
5. Если $S = 0$, $d_1 > d_2$, то необходимо записать единицу в нулевой разряд и списать номер центроида из регистра R_N.

Рассмотренные примеры показывают, что нелинейная обработка включает в себя алгоритмы, содержащие не только перечисленные ранее операции и функции, но и более сложные процедуры обработки больших массивов векторов, а также логические операции.

1.5. Адаптивная фильтрация

1.5.1. Адаптивные фильтры

Адаптивными называют фильтры, частотные характеристики которых зависят от спектров обрабатываемых сигналов. Основная задача адаптивного фильтра (АФ) — повысить качество приема или обработки сигнала. Требования к АЧХ адаптивных фильтров не задаются, поскольку их характеристики *изменяются во времени.*

Процедура конструирования АФ состоит в выборе класса фильтра (КИХ, БИХ, одномерный, двумерный) и оптимального алгоритма корректировки (адаптации) переменных коэффициентов. Именно выбор и построение оптимального алгоритма является наиболее сложной задачей, связанной с большими затратами вычислительных ресурсов и обеспечением работы устройства в реальном времени.

АФ состоит из трех элементов (рис. 1.15):

- цифрового фильтра с переменными коэффициентами;
- устройства определения ошибки (сумматор на схеме);
- устройства, реализующего алгоритм адаптации.

Принцип работы АФ ясен из рисунка. Выходной сигнал фильтра $y(n)$, отличающийся от эталонного $y_0(n)$, вычитается из $y_0(n)$. Получаемая ошибка $e(n)$ подается на устройство адаптации, которое так изменяет коэффициенты ЦФ, чтобы свести $e(n)$ к минимуму.

В более сложных системах с целью получения лучших характеристик сигнала $y(n)$ используется иной принцип адаптации, получившей название *обратной.* Этот вариант изображен на рис. 1.15 штриховыми линиями. По сигналу $y(n)$ восстанавливается сигнал $\tilde{x}(n)$, который будет отличаться от входного

сигнала $x(n)$ на величину ошибки $e(n)$, которая и управляет адаптацией. Введение линии задержки необходимо для временного согласования сигналов $x(n)$ и $\tilde{x}(n)$.

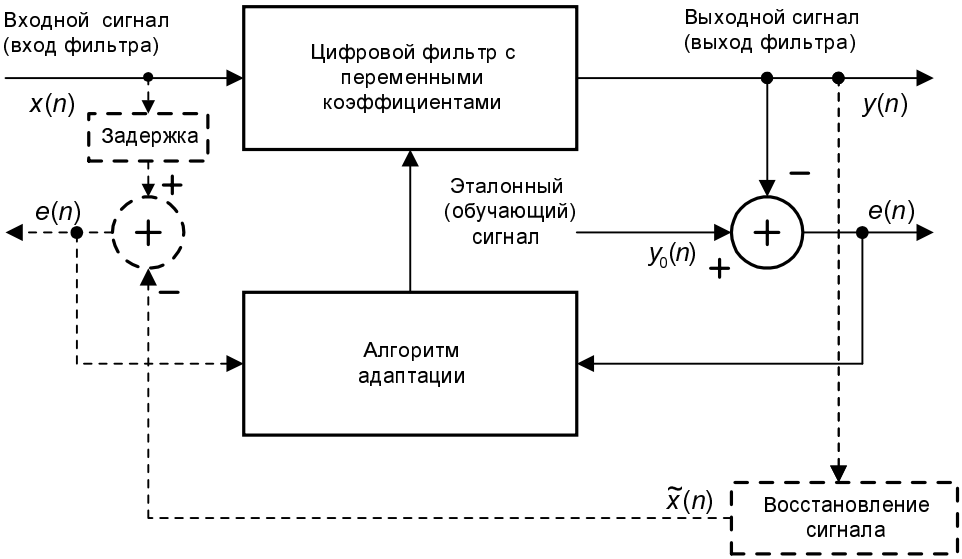


Рис. 1.15. Структурная схема адаптивного фильтра

Процесс адаптации может быть как одноцикловым (одношаговым), так и итеративным, когда адаптация осуществляется шаг за шагом. Основными характеристиками алгоритма адаптации являются скорость сходимости при заданной ошибке и сложность (объем вычислений). На практике из множества алгоритмов адаптации наиболее часто применяются алгоритмы, основанные на одном из двух критериев: минимума среднеквадратической ошибки (СКО) и метода наименьших квадратов (МНК).

В зависимости от характера усреднения ошибки фильтрации по заданному критерию выделяют *глобально-адаптивные* и *локально-адаптивные* фильтры. Если ошибка усредняется по всему обрабатываемому сигналу, фильтр называется глобально-адаптивным (обычно это КИХ-фильтры); если адаптация осуществляется в пределах отдельных фрагментов (кадров) сигнала, фильтр называется локально-адаптивным (обычно это БИХ-фильтры).

Среди многочисленных областей применения АФ можно выделить основные:

- *коррекция искажений* при передаче сигнала по каналам связи; в этом случае АФ моделирует обратную характеристику системы: известный на передаче и приеме эталонный сигнал подается на вход канала связи, его искаженная копия с выхода канала связи проходит через ЦФ, далее из

сигнала, полученного на выходе ЦФ, вычитается эталонный сигнал; в результате перестройки коэффициентов частотная характеристика цифрового фильтра оказывается обратной относительно частотной характеристики канала связи;

- *подавление шумов* — в этом случае сигнал, содержащий помеху, подается непосредственно на сумматор, на вход ЦФ подается образец помехи, которая после прохождения через ЦФ вычитается из сигнала, содержащего помеху; в результате на выходе получается искомый сигнал;
- *компрессия (сжатие)* речевых сигналов в системах с линейным предсказанием (вокодерах), которые рассматриваются ниже.

1.5.2. Линейное предсказание

Линейное предсказание (ЛП) — это вычислительная процедура, позволяющая по некоторой линейной комбинации L предшествующих взвешенных отсчетов сигнала предсказать (с некоторой точностью) будущее значение отсчета. Практическая важность линейного предсказания для спектрального анализа состоит в получении оценки спектра исследуемого сигнала на его отрезке (кадре) длиной в L отсчетов, а с точки зрения фильтрации — в получении рекурсивного адаптивного фильтра порядка $M-1$ на участке квазистационарности, т. е. на том временном отрезке длительностью LT (T — период дискретизации), где коэффициенты фильтра остаются постоянными. Итогом решения задачи ЛП является получение коэффициентов адаптивного фильтра, АЧХ которого с хорошей степенью приближения соответствует спектру сигнала на кадре.

Задача линейного предсказания может быть сформулирована следующим образом: на выходе некоторой системы наблюдается сигнал $y(n)$; известно, что это система полюсного типа с передаточной функцией вида

$$H(z) = \frac{b_0}{1 - \sum_{k=1}^{M-1} a_k z^{-k}} \tag{1.57}$$

имеет порядок $K = M - 1$ и возбуждается белым шумом. Требуется найти коэффициенты a_k.

Суть процедуры решения состоит в следующем (рис. 1.16). Согласно (1.57) отсчеты сигнала $y(n)$ на выходе системы определяются выражением

$$y(n) = b_0 x(n) + \sum_{k=1}^{M-1} a_k y(n-k). \tag{1.58}$$

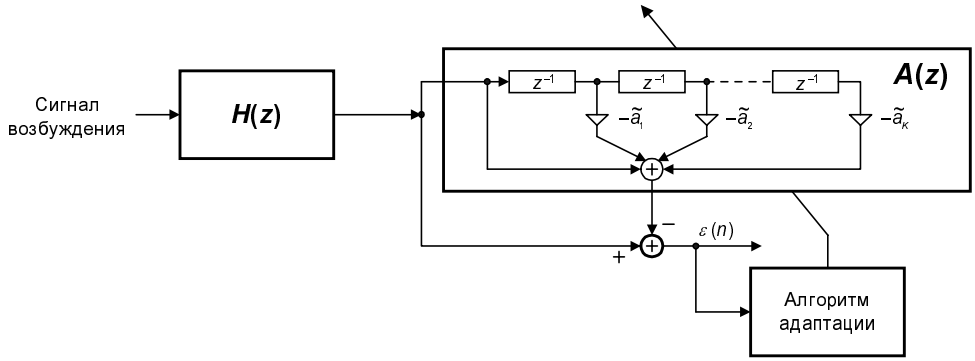


Рис. 1.16. Решение задачи линейного предсказания

Включим последовательно с искомой системой КИХ-фильтр с передаточной функцией

$$A(z) = 1 - \sum_{k=1}^{K} \tilde{a}_k z^{-k}, \tag{1.59}$$

коэффициенты которой $\tilde{a}_k = a_k$. Общая передаточная функция получит вид:

$$H_{\text{общ}}(z) = \frac{b_0}{1 - \sum_{k=1}^{M-1} a_k z^{-k}} \left(1 - \sum_{k=1}^{M-1} a_k z^{-k} \right) = b_0 = \text{const}. \tag{1.60}$$

Фильтр с передаточной функцией $A(z)$ называется фильтром линейного предсказания или *фильтром-предсказателем*.

В действительности коэффициенты $\tilde{a}_k$ будут отличаться от точных a_k, поэтому предсказываемое значение сигнала $\tilde{y}(n)$ будет отличаться от точного $y(n)$ на величину ошибки предсказания (при $n > 0$)

$$\varepsilon(n) = y(n) - \tilde{y}(n) = y(n) - \sum_{k=1}^{K} \tilde{a}_k y(n-k), \tag{1.61}$$

которую называют *остатком*. Отсюда нетрудно получить передаточную функцию КИХ-фильтра линейного предсказания (*фильтр-предсказатель*)

$$A(z) = 1 - \sum_{k=1}^{K} a_j z^{-k}, \tag{1.62}$$

сигнал на выходе которого представляет собой остаток $\varepsilon(n)$. Выражения (1.57—1.60) показывают, что передаточная функция искомой системы

$$H(z) = \frac{b_0}{A(z)}, \tag{1.63}$$

с точностью до коэффициента b_0 представляет собой обратную передаточную функцию (а потому и частотную характеристику) фильтра-предсказателя.

Коэффициенты линейного предсказания a_k вычисляются согласно критерию минимума среднеквадратической ошибки (СКО) предсказания:

$$E = \sum_{n=1}^{K} \varepsilon^2(n) = \sum_{n-1}^{K} \left[y(n) - \sum_{k=1}^{K} a_k y(n-k) \right]^2. \tag{1.64}$$

Коэффициенты a_k можно найти, положив

$$\frac{\partial E}{\partial a_k} = 0; \qquad k = 1,..., K, \tag{1.65}$$

что приводит к системе из L уравнений для определения K коэффициентов:

$$\sum_{l}^{L} y(l)y(l-1) = \sum_{l=1}^{L} \sum_{k=1}^{K} a_k y(l-k)y(l-1), \qquad 1 \le l \le L, \ 1 \le k \le K, \tag{1.66}$$

где L — количество отсчетов на кадре, которое может существенно превосходить порядок предсказания K (например, для стандарта LPC-10 $K = 10$, $L = 120$). Это означает, что в процессе вычисления искомых коэффициентов необходимо на каждом кадре речи решать переопределенную систему уравнений. Переопределенные системы после ряда преобразований сводятся к симметричным системам уравнений вида

$$\sum_{k=1}^{K} a_k R_{ik} = R_{i0}, \tag{1.67}$$

где

$$R_{ik} = \sum_{l=1}^{L} y(l-i)y(l-k) \tag{1.68}$$

коэффициент корреляции; $i = 1, 2, ..., K$; $k = 1, 2, ..., K$.

Известен быстрый итеративный алгоритм Левинсона-Дарбина, в котором решение уравнения порядка K вида (1.67) выражается через решение уравнения того же вида порядка $K-1$.

Линейное предсказание является чрезвычайно эффективным при построении вокодеров — систем сжатия речи. Оно позволяет получать на приеме

синтезированный речевой сигнал по качеству, очень близкому к естественному звучанию. Линейное предсказание нашло также широкое применение в обработке изображений для сжатия видеоданных.

Вокодеры с линейным предсказанием

Важнейшей областью применения линейного предсказания является сжатие речевого сигнала с целью снижения скорости передачи речи по каналам телекоммуникации. Необходимость постановки такой задачи объясняется следующим. Передача стандартного телефонного сигнала, ограниченного полосой (0,3—3,4) кГц, по цифровым каналам связи при стандартной частоте дискретизации 8 кГц и несложном АЦП с разрядностью 12 битов потребует скорости передачи

$$C = 12 \times 8000 = 96\,000 \text{ бит/с}$$

и, следовательно, в идеальном случае полосы пропускания канала 48 кГц.

Замечание

Здесь и далее под битом понимается один элемент передаваемого цифрового сигнала.

В то же время, желательно более экономно использовать частотный ресурс канала; кроме того, КВ-каналы вообще не допускают таких скоростей, их возможности значительно скромнее: скорость передачи в КВ-каналах не превосходит 2400 бит/с. Этот пример показывает, что необходимо так преобразовать информацию, содержащуюся в речевом сигнале, чтобы скорость передачи сократилась в 40 раз (!), т. е. коэффициент сжатия должен быть по крайней мере равен 40. Вообще, ничего удивительного в самой возможности сжатия речевого сигнала нет, поскольку в процессе сжатия устраняется некоторая избыточность, содержащаяся в речевом сигнале. Степень устранения избыточности при передаче отражается на качестве восстанавливаемого (синтезируемого) сигнала.

Устройства кодирования речи называются *вокодерами* (от англ. *voice* — голос, *coder* — кодировщик). Для их построения используются свойства голосового тракта.

Основным элементом модели голосового тракта (рис. 1.17) является адаптивный фильтр с дискретно меняющимися во времени коэффициентами, фильтр подстраивает свою частотную характеристику под спектр короткого отрезка передаваемого речевого сигнала. Таким адаптивным фильтром является фильтр ЛП порядка K. Возбуждение подобного шума возможно основным тоном или шумом. Моделирование возбуждения осуществляется перестраиваемым генератором частот. Для моделирования сигнала возбуждения служит перестраиваемый генератор частот (генератор основного тона — частоты колебаний голосовых связок) и специальный генератор белого шума (генератор шума).

Вокодер (рис. 1.17) состоит из двух частей: анализатора и синтезатора.

Анализатор определяет параметры речи, *синтезатор* по принятым параметрам восстанавливает речь.

Анализатор обрабатывает цифровой речевой сигнал *покадрово*. Кадры вырезаются друг за другом с помощью гладкой функции типа "окна" (треугольного или Хэмминга). Длительность одного кадра и количество отсчетов речевого сигнала, содержащихся в одном кадре, определяется стандартом и находится в пределах от 15 мс до 30 мс. При частоте дискретизации 8 кГц в одном кадре содержится от 120 до 240 отсчетов соответственно.

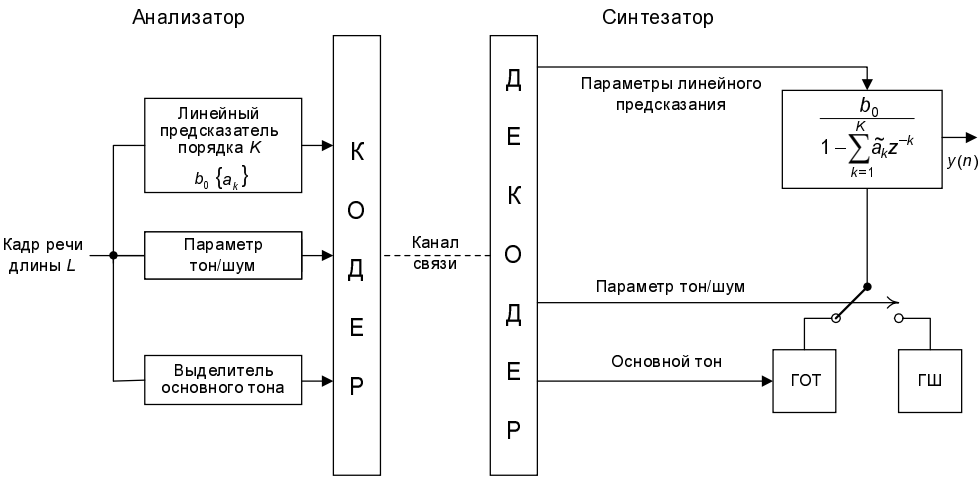


Рис. 1.17. Обобщенная структурная схема вокодера с линейным предсказанием

К параметрам речевого сигнала, анализируемого на кадре, относятся:

- параметры линейного предсказания (математически эквивалентные коэффициентам ЛП);
- тип возбуждения голосового тракта тон/шум;
- период основного тона и энергия сигнала возбуждения.

В ряде вокодеров новейших моделей применяется *векторное квантование параметров ЛП* по кодовой книге размерностью 1024 центроида.

В синтезаторе происходит обратный процесс: по параметрам линейного предсказания восстанавливаются коэффициенты $\{\tilde{a}_k\}$ (конечно, они будут несколько отличаться от вычисленных на передаче, но устойчивость гарантируется), формируется полюсный фильтр, возбуждаемый либо шумом от генератора шума (если передавался неогласованный звук), либо основным тоном от генератора основного тона, вырабатывающим частоту ОТ по принятым параметрам.

Известно несколько типов вокодеров, отличающихся друг от друга способами представления параметров речевого сигнала на кадре, битовой скоростью в канале и, в связи с этим, качеством синтезируемого на приеме сигнала. Ниже дается краткая характеристика вокодеров согласно принятым международным стандартам и рекомендациям.

- **Стандарт LPC-10** (Linear Prediction Coder). В вокодерах этого типа используется фильтр-предсказатель 10-го порядка на кадрах речи, длительностью ≈ 20 мс; коэффициенты предсказания преобразуются в математически эквивалентные параметры — спектральные корни, значения которых передаются по каналу связи. Вокодеры данного типа обеспечивают хорошую (≈ 92%) словесную разборчивость; используются на скоростях 1,2; 2,4 и 4,8 Кбит/с.
- Известна группа чрезвычайно сложных вокодеров с векторным квантованием параметров CELP (Code Excited Linear Prediction), из которой приведем три примера.
  - **Стандарт *ITU-T G*.723.1 —** двухскоростной вокодер для мультимедийных коммуникаций, является частью семейства стандартов H.324. Вокодер работает на скоростях 5,3 и 6,3 Кбит/с. Применяется линейное предсказание LPC-10 с векторным квантованием параметров ЛП.
  - Помимо собственно рекомендации G.723.1 существует **Приложение *A***, согласно которому в кодер добавляется классификатор входного сигнала VAD (Voice Activity Detector, определитель активности голоса). Классификатор выясняет, какой сигнал присутствует на входе: речь или пауза. Во время пауз скорость передачи понижается с 6,3 (или 5,3) Кбит/с до 1 Кбит/с и менее.
  - **Стандарт *ITU-T G*.728.1 —** вокодер с векторным квантованием сигналов возбуждения и параметров линейного предсказания, предназначен для работы на скорости 16 Кбит/с. Входной сигнал (частота дискретизации 8 кГц) подвергается компандированию по *A*- или μ-закону *(см. главу 7)*. Для вычисления параметров линейного предсказания служит предсказатель 50-го порядка, для вычисления коэффициентов усиления сигнала возбуждения используются линейные предсказатели 10-го порядка. Кодовая книга имеет размерность 1024.

Как видно из приведенных примеров, алгоритмы адаптивной фильтрации так или иначе включают в себя рассмотренные ранее алгоритмы и потому являются очень сложными как в функциональном, так и в вычислительном плане.

1.6. Способы реализации алгоритмов ЦОС

Среди алгоритмов ЦОС, как следует из изложенного выше, с точки зрения организации вычислений можно выделить как простые, так и чрезвычайно сложные. Тем не менее, независимо от сложности алгоритма вычисления

осуществляются с помощью *базовых операций*: сложения, вычитания и умножения. Возведение в степень — это многократное умножение, а деление — многократное вычитание, причем частное может быть как целым, так и дробным числом, поэтому при организации деления необходимо задавать желаемую точность частного. Поскольку вычислительные операции производятся с данными, задерживаемыми относительно друг друга на один и более периодов дискретизации T с помощью элементов задержки, представляющих собой регистры (ячейки памяти), объединяемые в линии задержки, необходимо иметь возможность осуществлять пересылки и сдвиги данных. Кроме того, для управления вычислительным процессом необходимо предусмотреть и логические операции.

Достаточно ли всего перечисленного для построения алгоритма? Оказывается, достаточно. В математической логике доказывается, что алгоритм любой сложности может быть построен и вычислен с использованием конечного числа только простейших математических и логических операций, а также операций сдвига и пересылки, причем вычисление одного отсчета выходной последовательности осуществляется за ограниченное число шагов. Сама процедура может или продолжаться исполняться, повторяясь и никогда не останавливаясь, или по какой-либо причине прерваться на каком-либо шаге, прекратить текущие вычисления и обратиться к другому алгоритму. Такое обращение называют *прерыванием*, а сигнал, вызывающий прерывание, называется *запросом на прерывание*. Принципиальная возможность вычислимости того или иного алгоритма вовсе не означает, что ограниченное количество шагов всегда удовлетворит практическим нуждам. Важнейшим фактором, определяющим пригодность созданной процедуры, является время вычисления одного отсчета. Действительно, пусть один отсчет $y(n)$ формируется за 0,1 с. Тогда, если это отсчет речевого сигнала, то процедура, будучи математически верной, практически абсолютно бесполезна; если же это — отсчеты сигнала, поступающего от датчика температуры воздуха, изменяющейся очень медленно, используемая процедура с лихвой удовлетворит самого взыскательного метеоролога. Таким образом, определяющим свойством процедуры становится ее практическая вычислимость, т. е. ее способность вычислять отсчет $y(n)$ за разумное время, или, как принято говорить, за *реальное время*, при этом имеется в виду обязательное достижение заданной точности.

1.6.1. Реальное время

Два рассматриваемых далее примера показывают, что определение реального времени зависит от конкретной задачи и связано с объемом вычислений алгоритма, точностью вычислений и частотой дискретизации (периода дискретизации). Пусть T — период дискретизации (рис. 1.18), τ_a — время выполнения алгоритма.

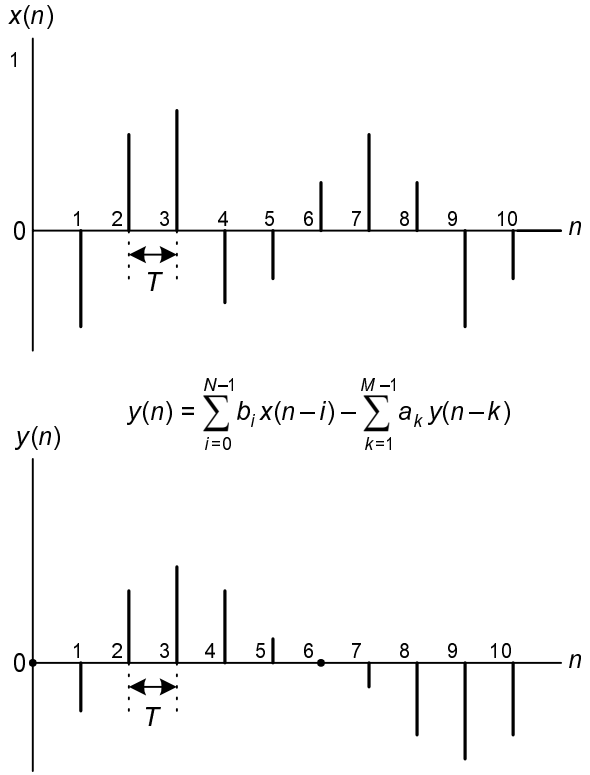


Рис. 1.18. К определению реального времени

Определение. Говорят, что цифровая система работает в реальном времени, если время выполнения алгоритма τ_a не превышает периода дискретизации.

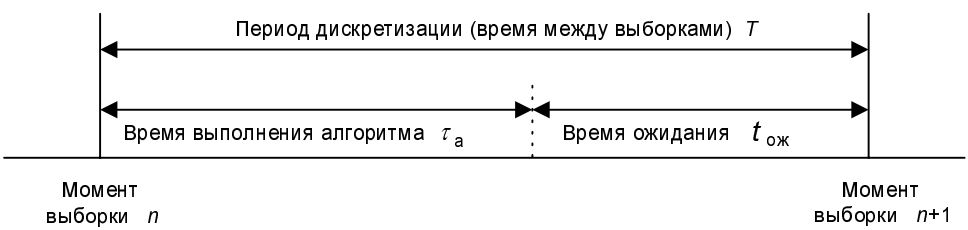


Рис. 1.19. Определение реального времени

Это означает, что остается еще некоторый запас времени, обычно называемый *временем ожидания* $t_{ож}$. Найти время выполнения алгоритма можно, если знать время выполнения элементарной (одноцикловой) команды $\tau_к$, называемое *командным циклом*, и количество командных циклов N_a, необхо-

димое для выполнения алгоритма (это можно определить в процессе *отладки*). Тогда

$$\tau_{\text{а}} = \tau_{\text{к}} N_{\text{а}}, \quad t_{\text{ож}} = T - \tau_{\text{а}}. \tag{1.69}$$

На рис. 1.19 показано, что вычисление отсчета последовательности в рекурсивной системе происходит за время, не превосходящее периода дискретизации.

Тактовая частота

Тактовая частота $f_{\text{такт}}$ (тактовый период $\tau_{\text{такт}} = 1/f_{\text{такт}}$) показывает, как быстро процессор выполняет простейшую единицу работы, например пересылку в регистре из разряда в разряд. Тактовая частота должна существенно превышать частоту дискретизации. Отношение тактовой частоты к частоте дискретизации относится к наиболее важным характеристикам, определяющим, каким образом будет реализована система. Это отношение частично определяет количество аппаратных средств, необходимое для реализации алгоритма заданной сложности в реальном времени. Если отношение указанных частот падает, то количество и сложность аппаратных средств, требуемых для реализации алгоритма, увеличивается. Например, вокодеры, являясь чрезвычайно сложными устройствами, при частоте дискретизации 8 кГц могут быть реализованы только на нескольких процессорах младшего поколения TMS320C10 (тактовая частота 6 МГц, командный цикл 200 нс, отношение частот 750) и всего на одном (!) современном процессоре TMS320C67xx (тактовая частота 167 МГц, командный цикл 1 нс, отношение частот 20 875) можно реализовать несколько вокодеров.

Далее будет показано, что вместо тактовой частоты используются более универсальные единицы измерения производительности процессора.

Время выполнения алгоритма $\tau_{\text{а}}$ зависит не только от процедуры, представляющей этот алгоритм, но и от способа реализации алгоритма. Возможны три способа реализации алгоритмов ЦОС (рис. 1.20):

- ❒ аппаратный;
- ❒ программный;
- ❒ аппаратно-программный.

1.6.2. Аппаратная реализация

Аппаратная реализация подразумевает использование разнообразных функциональных блоков: регистров, сумматоров, шифраторов и дешифраторов, счетчиков, линий задержек, устройств памяти, умножителей, сдвигателей, логических элементов, интегральных и больших интегральных схем, программируемых логических матриц и т. п. Совокупность функциональных блоков и связей между ними определяет реализуемый алгоритм.

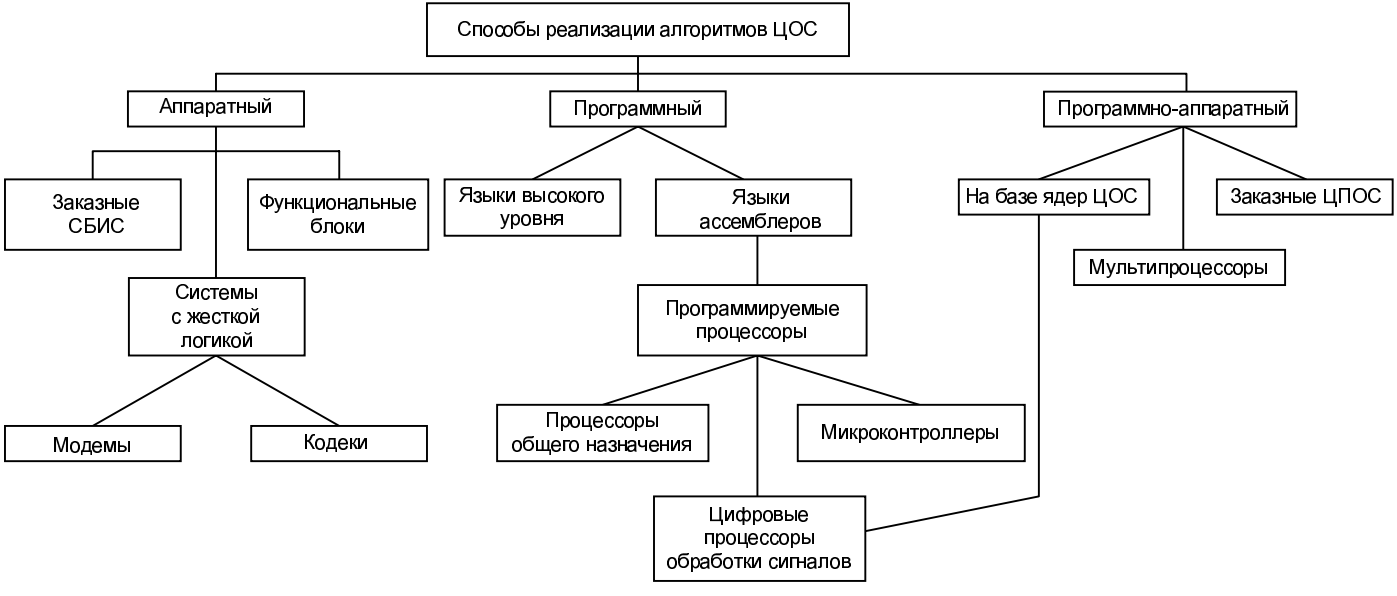


Рис. 1.20. Способы реализации алгоритмов и систем ЦОС

Рассмотрим пример аппаратной реализации БИХ-звена второго порядка, описываемого разностным уравнением

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + a_1 y(n-1) + a_2 y(n-2). \qquad (1.70)$$

Вычисления по этому уравнению можно организовать несколькими алгоритмами, один из которых, называемый *прямой формой 1*, в виде структурной схемы изображен на рис. 1.21, где задержка на один период дискретизации обозначена через z^{-1}, умножение — треугольником с соответствующим коэффициентом, а суммирование — кружком с внутренним знаком "+".

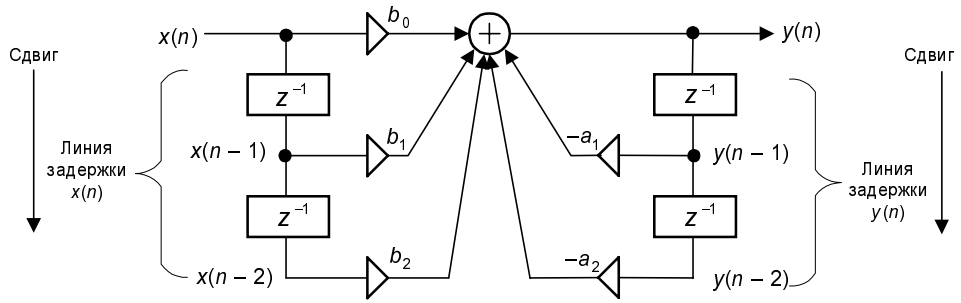


Рис. 1.21. Прямая форма 1

Из уравнения и структурной схемы следует, что цифровое устройство, реализующее данный алгоритм, должно иметь:

- ❒ память коэффициентов и память исходных и промежуточных данных;
- ❒ систему ввода и вывода исходных данных $x(n)$ и результата $y(n)$ соответственно;
- ❒ умножители;
- ❒ сумматор;
- ❒ генератор тактовой частоты (ГТЧ) с периодом следования импульсов $\tau_{\text{ти}} << T$.

Такое цифровое устройство изображено на рис. 1.22. Вместо нескольких умножителей используется один быстродействующий умножитель, на один вход которого последовательно согласно логике вычислений поступают данные $x(n)$ или $y(n)$, на другой — соответствующие коэффициенты. Получаемые произведения подаются в накапливающий сумматор, результат с которого считывается один раз за период дискретизации T.

Вместо линий задержек обычно организуют *X*-память и *Y*-память данных, а также память коэффициентов. По окончании вычислений очередного отсчета $y(n)$ осуществляется его пересылка в *Y*-память, откуда подается на устройство вывода. В *X*-памяти и *Y*-памяти происходит сдвиг данных, что означает готовность устройства к приему очередного отсчета $x(n)$. Ответственным за последовательность выполнения всех операций является логическое устройство. Обычно умножитель, сумматор и логическое устройство объединяются в один

блок, называемый арифметико-логическим устройством (АЛУ). Связи между всеми блоками осуществляются с помощью шин, разрядность которых соответствует разрядности ячеек *X*-памяти и *Y*-памяти.

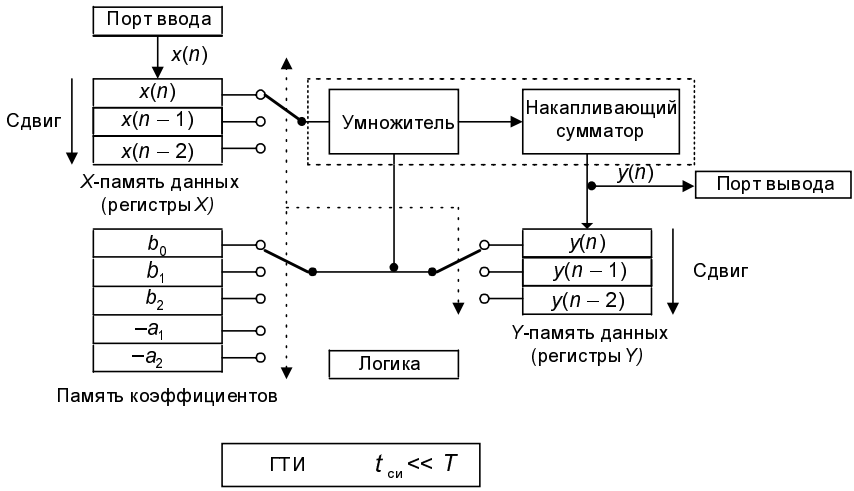


Рис. 1.22. Аппаратная реализация цифрового устройства

Еще раз отметим, что для обеспечения работы полной системы в реальном времени все вычисления должны происходить за время $\tau_a << T$.

Достоинство аппаратной реализации состоит в очень высоком быстродействии, что позволяет обрабатывать сигналы при частоте дискретизации в десятки мегагерц. Это достигается применением функциональных блоков на базе ТТ-логики, распараллеливанием операций и узкой направленностью (специализацией) создаваемых устройств (например, для реализации алгоритма БПФ в радиолокационных системах).

С другой стороны, аппаратная реализация, ориентированная на решение узкоспециальных задач, подразумевает создание систем с жесткой логикой, когда любое изменение алгоритма требует изменения структуры устройства, т. е. введения дополнительных функциональных блоков, что, конечно, является недостатком. Кроме того, аппаратная реализация приводит к большому потреблению энергии и к необходимости организовывать теплоотвод. Все это вместе определяет высокую стоимость аппаратной реализации, причем проектирование, изготовление и отладка оказываются весьма трудоемкими при больших временных затратах.

1.6.3. Программная реализация

Программная реализация подразумевает представление алгоритма в виде программы, которую последовательно от команды к команде выполняет один или одновременно несколько независимых блоков. Программа должна быть напи-

сана на языке программирования, соответствующем конкретному операционному блоку. Так, для персонального компьютера это будет любой из языков высоко уровня (Pascal, C++, Java и др.), а для микропроцессорного комплекта или цифрового процессора — соответствующий язык ассемблера.

Конечно, все команды, составляющие программу, должны быть представлены в виде, понятном процессору и непосредственно им воспринимаемом. Команды предъявляются процессору как комбинация нулей и единиц; такие комбинации составляют машинный язык. Например, в процессорах семейства TMS320Cxxx команде сложения соответствует комбинация 0000, команде умножения — комбинация 0011. Написание программы на машинном языке, что требовалось в 60-е годы XX века для микроЭВМ типа "Электроника" — дело крайне трудоемкое. Поэтому создаются специальные средства, облегчающие подготовку и отладку программ для процессора, когда каждой машинной команде или группе команд ставится в соответствие понятный человеку символ на основе мнемоники (правил и приемов, облегчающих запоминание) так, чтобы символ отражал смысловое содержание команды. Например:

- `ADD` — сложить (от англ. *add*);
- `SUB` — вычесть (от англ. *subtract*);
- `MUL` — умножить (от англ. *multiply*).

Определение. Язык программирования, в котором каждой машинной команде или совокупности машинных команд соответствует сокращенная символическая запись, называется языком ассемблера.

Перевод программы на машинный язык называется *трансляцией* и выполняется автоматически с помощью специальной программы — *ассемблера* (от англ. *assembler* — сборщик).

Разработаны и все большее значение приобретают специальные весьма эффективные программы-трансляторы с языков высокого уровня на языки ассемблеров. Эти программы называются *компиляторами*. Ассемблеры и компиляторы более подробно рассматриваются в *главе 9*.

К достоинствам программной реализации относятся:

- неизменная структура системы при различных алгоритмах и областях применения;
- хорошая гибкость, позволяющая достаточно легко изменять алгоритмы работы системы за счет коррекции или изменения программы;
- существенное ускорение, облегчение и удешевление проектирования, изготовления и отладки системы, поскольку вместо прибора разрабатывается программа.

Недостатком программной реализации является относительно низкое быстродействие по причине последовательного выполнения операций программы в одном процессоре: как бы ни увеличивали скорость выполнения команд, она будет оставаться ниже производительности соответствующего

устройства, реализованного аппаратно. Отсюда вытекает задача обеспечения реального времени, которая подразумевает два обстоятельства:

- во-первых, время обработки одного отсчета или группы отсчетов сигнала $t_{об}$ должно быть меньше допустимого времени задержки $t_{доп}$: $t_{об} < t_{доп}$; контроль за выполнением этого условия осуществляется как при написании программы, так и при ее отладке;
- во-вторых, цикл работы программы и моменты поступления отсчетов входного сигнала $x(n)$ должны быть строго согласованы по времени, т. е. начало обработки очередного отсчета $x(n)$ должно совпадать или следовать за поступлением этого отсчета; с другой стороны, результат обработки $y(n)$ должен быть выведен согласно темпу работы внешнего устройства, который не обязан совпадать с темпом поступления отсчетов $x(n)$, что видно на примере вокодеров.

1.6.4. Аппаратно-программная реализация

Аппаратно-программная реализация подразумевает, что часть функций системы ЦОС выполняется аппаратно (аналого-цифровое и цифро-аналоговое преобразования, умножение, умножение с накоплением, прием/передача данных и др.), а другая часть функций выполняется программно. Пример аппаратно-программной реализации показан на рис. 1.23, где к процессору, работающему по заданной программе, подключены:

- аналого-цифровой (АЦП) и цифро-аналоговый (ЦАП) преобразователи;
- модули внешней памяти, хранящие программы, разнообразные константы и таблицы функций (например, sin и cos), что позволяет заменять длительное их вычисление быстрым обращением к памяти (такая реализация является прекрасным примером обмена скорости вычисления на дополнительное оборудование, т. е. действует закон, согласно которому невозможно получить абсолютный выигрыш: любой выигрыш требует платы за себя);

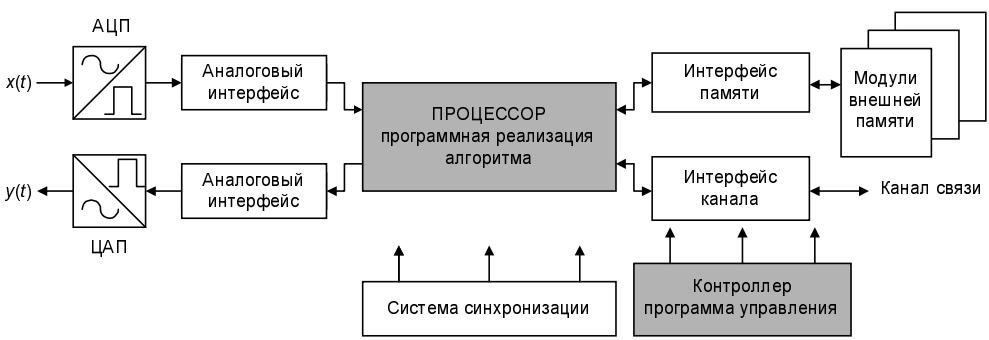


Рис. 1.23. Аппаратно-программная реализация системы ЦОС

- интерфейс — специальные вспомогательные схемы, обеспечивающие согласование сигналов на стыках с модулями внешней памяти, АЦП, ЦАП, каналом связи (регистры стыковки также могут быть отнесены к интерфейсу);
- система синхронизации, обеспечивающая временное согласование всех элементов системы.

Аппаратно-программная реализация сочетает положительные свойства аппаратной и программной реализаций. Разумное сочетание аппаратных и программных средств позволяет снизить требования к вычислительным возможностям элементной базы и упростить реализацию систем ЦОС в целом, для отладки которой требуются специальные средства отладки. С наших позиций обязательность отладочных средств не является недостатком: средства отладки создаются под конкретную элементную базу и по сути являются *инструментом разработки* многочисленных систем ЦОС на этой элементной базе.

1.7. Особенности ЦОС, влияющие на элементную базу

Все сказанное в этой главе позволяет выделить ряд особенностей цифровой обработки сигналов, которые, с одной стороны, предъявляют достаточно жесткие требования к элементной базе, а с другой — облегчают разработку самой элементной базы, ориентированной на реализацию цифровых систем. Ниже дается краткая характеристика особенностей и основных свойств ЦОС.

1.7.1. Характеристика особенностей ЦОС

Итак, особенности таковы.

1. *Высокая скорость поступления данных.* Например, пусть отсчеты аудиосигнала поступают в устройство обработки со скоростью от 8000 до 20 000 отсчетов в секунду, каждый из них может содержать от 8 до 16 битов (в зависимости от разрядности АЦП). Отсчеты согласно выбранному алгоритму преобразуются в кадры, параметры которых и скорость в канале связи показаны в табл. 1.3. Ясно, что чем больше битов содержит кадр и чем меньше его длительность, тем естественнее звучит синтезируемый на приеме сигнал.

Таблица 1.3. *Параметры кадров РПУ*

Длина кадра (бит)	Длительность кадра (мкс)	Скорость в канале (бит/с)
53	22,5	2400
144	30	4800
80	10	8000

Скорость обработки данных определяется *производительностью* процессора, которая выражается количеством миллионов условных одноцикло-

вых команд, выполняемых в секунду (табл. 1.4): в MIPS (Million Instructions Per Second) для процессоров с ФТ и в MFLOPS (Million Float Operations Per Second) для процессоров с ПТ.

Таблица 1.4. *Пример характеристик семейств процессоров по тактовым частотам и производительности*

Процессоры	Тактовая частота (МГц)	Производительность (MIPS)
TMS320C2xxx	20—80	20—40
TMS320C5xxx	30—133	30—532
TMS320C5xxx	167—250	до 2000
ADSP-21xx	40—100	75—150

Производительность, выражаемая в MIPS (FLOPS), является пиковой, т. е. предельно возможной для данного процессора. *Реальная производительность* может быть значительно меньшей и потому ее оценивают временем выполнения стандартных алгоритмов; в частности, временем выполнения 1024-точечного БПФ. По этому показателю процессор ADSP-21160 (100 МГц, 600 MFLOPS) имеет преимущество перед процессором TMS320C6701 (167 МГц, 1000 MFLOPS), поскольку выполняет такое БПФ за 90 мкс, а его конкурент — за 120 мкс. Такая неожиданность объясняется разной полосой пропускания системы ввода/вывода, размером и типом внутренней памяти данных, количеством поддерживаемых циклических буферов и т. д.

Другой способ определения реальной производительности, называемый BDTImark (см. **http://www.bdti.com/**), состоит в тестировании ЦПОС на группе специальных задач. Результат тестирования выражается в относительных условных единицах (табл. 1.5): чем выше производительность, тем большим количеством единиц оценивается процессор.

Таблица 1.5. *Производительность процессоров в единицах BDTImark*

Процессор	Производительность	
	Пиковая (MIPS)	Реальная в единицах BDTImark
LUSENT DSP161210	100	36
Motorola DSP56303	100	25
TMS320VC549	100	25
ADSP-2189M	75	19
TMS320C6201	1000	600

Из табл. 1.5 следует, что:

- нет пропорциональной зависимости реальной производительности от пиковой;

- процессоры с одинаковой пиковой производительностью не обязательно имеют одинаковую реальную производительность.

2. *Широкий диапазон изменения значений входных/выходных данных.* Обычно диапазон данных составляет 40—80 дБ, а в радиоприемных устройствах может доходить до 100 дБ. Следовательно, в ряде случаев необходимо иметь такую элементную базу, которая обеспечивала бы организацию обработки данных большой разрядности. Если учесть, что один бит соответствует ≈ 6 дБ, то разрядность регистров сомножителей при различных диапазонах обязана быть такой, как указано в табл. 1.6, а регистры произведений должны иметь удвоенную разрядность.

Таблица 1.6. *Динамический диапазон и разрядность*

Динамический диапазон (дБ)	Разрядность регистров сомножителей	Разрядность регистра произведения
40	7	14
50	9	18
60	10	20
70	12	24
80	14	28
100	17	34

Динамический диапазон данных определяется в первую очередь разрядностью АЦП, которая на современном этапе достигает 20—24, т. е. предел динамического диапазона по АЦП составляет около 120—144 дБ. В действительности за счет эффектов квантования динамический диапазон оказывается несколько меньшим, нежели при указанной в табл. 1.4 разрядности.

Разрядность в 7—10 битов вполне удовлетворяет контроллеры, используемые в системах управления. Для систем обработки речи и звука минимально допустимой является разрядность в 13—14 битов.

Динамический диапазон, точность вычислений и мощность собственного шума цифровой цепи зависят не только от разрядности, но и типа арифметики — с фиксированной точкой (ФЗ) или с плавающей точкой (ПТ). Большинство фирм (табл. 1.7) выпускают процессоры с обеими типами арифметики.

Таблица 1.7. *Тип арифметики и производительность процессоров*

Поставщик	Семейство процессоров	Тип арифметики	Разрядность данных (бит)	Производительность (MIPS)
Analog Devices	ADSP-21xx	ФТ	16	33,3
	ADSP-21xxx	ПТ	32	40,0

Таблица 1.7 *(окончание)*

Поставщик	Семейство процессоров	Тип арифметики	Разрядность данных (бит)	Производительность (MIPS)
Motorola	DSP5600x	ФТ	24	40,0
	DSP563xx	ФТ	24	80,0
	DSP96002	ПТ	32	20,0
Texas Instruments	TMS320C2xx	ФТ	16	40,0
	TMS320C3x	ПТ	32	25,0
	TMS320C4x	ПТ	32	30,0
	TMS320C5x	ФТ	16	50,0
	TMS320C54x	ФТ	16	50,0
	TMS320C8x	ФТ	8/16	50,0
	TMS320C5000	ФТ	16	от 40 до 2000
	TMS320C662x	ФТ	32	от 1200 до 2400
	TMS320C67x	ПТ	32	от 600 до 1000
AT&T	DSP16xx	ФТ	16	70,0
	DSP32xx	ПТ	32	20,0
NEC	μPD7701x	ФТ	16	33,3
Zoran	ZR3800x	ФТ	20	33,3

3. *Большое количество операций сложения, умножения и логических операций.* Как показывалось ранее, эти операции требуются для вычисления одного выходного отсчета. Кроме того, все виды сложной обработки могут быть представлены композицией рассмотренных выше операторов: свертки, рекурсии, ДПФ, нелинейных и логических преобразований. Отсюда следует, что элементная база должна быть ориентирована на быстрое выполнение таких операторов. В частности, должно быть организовано аппаратное умножение с накоплением (сложение локальных произведений) и создана большая память данных и память программ с удобным и быстрым доступом к ним.
4. *Необходимость обеспечения гибкости и перестройки* цифровых систем обработки сигналов, что связано с изменением разнообразных параметров, коэффициентов и данных в регулируемых и адаптивных системах. Именно адаптивные системы находят все бóльшее применение в телекоммуникации для подавления эхо-сигналов разнообразной природы, коррекции модемов (устранение сдвига частоты и дрожания фазы — джиттера), коррекции характеристик канала связи, построения вокодеров с линейным предсказанием и т. д.
5. *Параллелизм алгоритмов*, проявляющийся в том, что для каждого набора входных данных выполняются такие действия, которые могут совмещаться по времени. Например, параллельная обработка стереоканалов в процессорах платформы TMS320C6xxx за счет особой организации архитектуры.
6. *Регулярность алгоритмов*, т. е. повторяемость отдельных операций. Типичными примерами являются операция "бабочка" в БПФ и алгоритм Горнера для вычисления полиномов.

1.7.2. Основные свойства ЦПОС

Сказанное позволяет выделить основные свойства ЦПОС, обеспечивающие эффективную реализацию алгоритмов ЦОС:

- быстрое выполнение типовых операций ЦОС;
- аппаратная реализация комплексной операции умножения с накоплением (суммирование локальных произведений);
- применение арифметики с ФТ и ПТ с разнообразной разрядностью;
- параллельное выполнение отдельных частей алгоритма, которое достигается аппаратной реализацией ряда типовых алгоритмов;
- большая внутрикристальная память данных и память программ;
- разнообразие режимов адресации применительно к различным задачам: организация буферов, поддержка бит-реверсивной адресации в БПФ и т. д.;
- обработка в реальном времени данных, поступающих с высокой скоростью;
- наличие внутрикристальной периферии (последовательных и параллельных интерфейсов, портов ввода/вывода, таймеров);
- малое время обращения к элементам внешней периферии.

Обобщение перечисленных свойств, характерных для разнообразных ЦПОС, и краткое описание их роли в системах цифровой обработки сигналов приведены в табл. 1.8.

Таблица 1.8. *Общие свойства ЦПОС*

Свойства	Применение
Быстрое умножение с накоплением	Большинство алгоритмов ЦОС (фильтрация, преобразования, спектральный анализ, нелинейная обработка и т. д.) насыщены операциями сложения и умножения
Архитектура с параллельным доступом к памяти	Увеличение производительности, поскольку многие операции ЦОС, работающие с большими объемами данных, требуют чтения команд программы и многократного обращения к данным во время каждого командного цикла
Режимы специальной адресации	Эффективная поддержка массивов данных и буферов типа FIFO ("первым вошел — первым вышел")
Управление специальными программами	Эффективное управление циклами в многоитеративных алгоритмах ЦОС; быстрое прерывание, поддерживающее часто повторяемые команды типа ввода/вывода
Внутрикристальная периферия и интерфейсы ввода/вывода	Внутрикристальная периферия, включающая в себя разнообразные устройства (компандеры, кодеки, таймеры, интерфейсы ввода/вывода, приспособленные к внешней периферии общего назначения и др.), позволяет разрабатывать компактные системы малой стоимости

Повсеместное распространение идей ЦОС ставит перед разработчиками новые виды задач, которые либо не могут быть обеспечены одним процессором, либо это такие узкоспециализированные задачи, решение которых возможно только на более сложных процессорах.

В подобных случаях создаются специальные ЦПОС, которые по требованию заказчика могут содержать дополнительные функциональные блоки: блок битманипуляций, блок помехоустойчивого кодирования, устройства, автоматически реализующего БПФ и т. д. Кроме заказных процессоров (рис. 1.24), создаются мультипроцессоры на базе однокристальных.

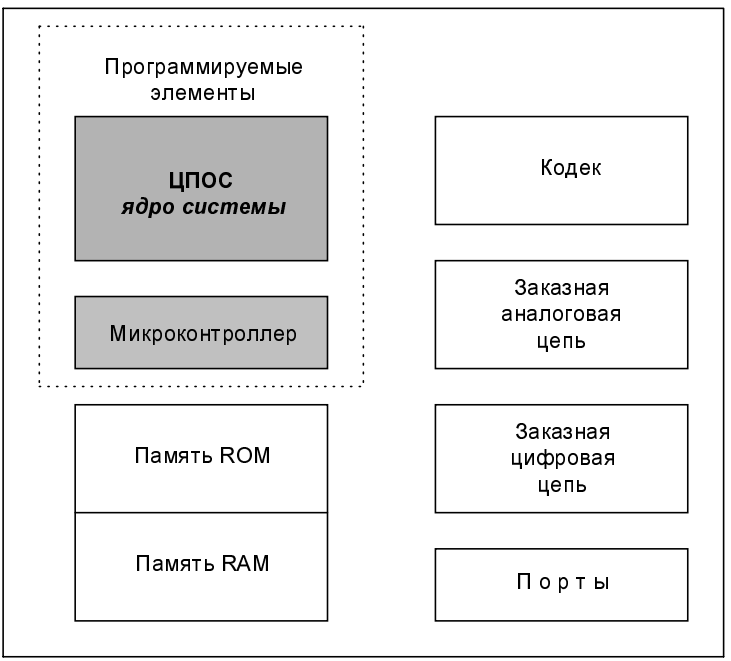


Рис. 1.24. Схема ASIC

Мультипроцессоры обладают:

- большим количеством внешних шин;
- логикой совместного доступа к шинам;
- выделенными параллельными портами, предназначенными для межпроцессорной связи.

Наиболее перспективными являются интегральные схемы, которые строятся на базе *ядер*. Такие схемы в одном кристалле содержат собственно ЦПОС и пользовательскую схему, благодаря чему сочетаются достоинства ЦПОС (программируемость, наличие инструментальных средств разработки и биб-

лиотек программ) с достоинствами заказных схем (высокая производительность, малые габариты, малое энергопотребление).

Замечание

Определение ядра еще не устоялось; поставщики по-разному представляют, что именно включается в ядро. Так, фирма Texas Instruments включает не только собственно процессор, но также память и периферию; фирмы Clarcspur Design и DSP Group добавляют память, но не включают периферию, а фирма SGS-Thomson имеет в виду только собственно процессор без периферии и памяти.

Здесь под ядром понимается ЦПОС, на основе которого строится заказная интегральная схема.

Одним из вариантов конструирования подобных схем являются прикладные специализированные интегральные схемы ASIC (Application-Specific Integrated Circuits), получившие название проблемно-ориентированных ИС.

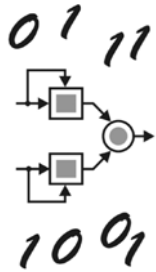
Схема ASIC (см. рис. 1.24) включает в себя ЦПОС-ядро как один из элементов общего кристалла, а также разнообразные порты, заказные цепи, память, микроконтроллеры, кодеки.

В настоящее время известно несколько компаний — поставщиков ядер для ASIC (табл. 1.9).

Таблица 1.9. *Характеристика ядер*

Поставщик	Ядерное семейство	Разрядность	Производительность (MIPS)
Clarcspur Design	CD2400 CD2450	16 16—24	25,0 50,0
DSP Group	PineDSPCore OakDSPCore	16 16	40,0 40,0
SGS–Thomson	D950–C0RE	16	40,0
Tensleep Design	A/DSCx21	16	30,0
Texas Instruments	T320C2xLP TEC320C52	16 16	40,0 50,0
3 Soft	M320C25	16	15,0

Глава 2



Архитектура цифровых процессоров обработки сигналов

2.1. Реализация цифровой обработки сигналов

Реальные сигналы, как правило, являются аналоговыми (речь, музыка, информация с датчиков, например, температурных, и т. д.). Обработка сигналов может производиться аналоговой и цифровой системами. Достоинства цифровых систем обработки информации и сигналов общеизвестны, и в данной книге не обсуждаются. Общая структура цифровой системы обработки информации приведена на рис. 2.1.

Рис. 2.1. Система цифровой обработки сигнала

АЦП (аналого-цифровой преобразователь) формирует из аналогового сигнала цифровой. ЦАП (цифро-аналоговый преобразователь) осуществляет обратное преобразование цифрового сигнала в аналоговый. Последнее преобразование не требуется в системах, где на основе обработки сигнала выносится некоторое решение, и нет необходимости восстанавливать исходный аналоговый сигнал. Порты ввода/вывода осуществляют ввод цифрового сигнала для обработки в вычислитель и, при необходимости, вывод результатов. Если цифровой вычислитель и источник сигнала (или получатель сигнала) разнесены территориально, в систему обработки входят устройства передачи сигнала по каналам связи, причем может передаваться как цифровой (с выхода АЦП по цифровым каналам связи), так и аналоговый сигнал. В последнем случае на вход АЦП поступает сигнал из канала связи.

Цифровая обработка сигнала (ЦОС) в вычислителе может выполняться разными способами с помощью самой разнообразной элементной базы.

Вычислитель может быть реализован аппаратным способом (устройство с жесткой логикой) и программным методом. Элементная база включает различные непрограммируемые (работающие не под управлением программы) и программируемые устройства. К непрограммируемым элементам относятся интегральные схемы ASICs (Application Specific Integrated Circuits, специализированные или проблемно-ориентированные интегральные схемы), PLDs (Programmable Logic Devices, программируемые логические устройства), FPGA (Field Programmable Gate Arrays). Программируемыми элементами являются микроконтроллеры, универсальные процессоры общего назначения разного типа (RISC и CISC), и, наконец, ЦПОС (цифровые процессоры обработки сигнала). Следует отметить, что устройства типа PLD, FPGA "программируются" для реализации ими определенной функции. В результате получается некоторая специализированная ИС. На основе схем типа PLD возможно получение и программируемого процессора, в том числе процессора цифровой обработки сигнала.

Не существует и невозможно построить цифровой процессор, эффективно выполняющий любые прикладные программы. Для каждой задачи можно подобрать наиболее подходящий и лучший по какому-то критерию процессор.

Первые ЦПОС, появившиеся в начале 80-х годов XX века, представлялись потребителям сугубо специализированными. В настоящее время они используются в очень многих конструктивных решениях: автомобилях, ЭВМ, музыкальных инструментах, видеотехнике, медицинской аппаратуре. Этот список может быть бесконечным, причем сфера использования ЦПОС непрерывно расширяется.

В [1] отмечается (см. **www.ednmag.com**), что сегмент мирового рынка процессоров, занятый ЦПОС, в 2000 г. был самым большим и динамичным, традиционно уступая только 8-разрядным микроконтроллерам. Продажи ЦПОС в долларах в 2000 г. возросли по сравнению с 1999 г. на 40%, в то время как продажи всех процессоров — только на 30%.

Эффективность и скорость решения задач ЦОС при использовании различной элементной базы будут отличаться. Решение задачи ЦОС при примерно равных тактовых частотах занимает приблизительно в три раза больше времени на универсальных процессорах по сравнению с процессорами ЦПОС. Например, в процессоре i8086 операция сложения занимает три такта, операция умножения больше 100 тактов, в то время как в ЦПОС для этих операций необходимо по одному такту.

На сайте компании Berkeley Design Technology, Inc. (BDTI) (**http://www.bdti.com/**) приводятся следующие данные: решение задачи БПФ на 256 точек в процессоре TMS320C6701 занимает время в два раза больше, чем при использовании процессора Pentium III, в то время как тактовые частоты работы процессоров отличаются в 6 раз (167 МГц и 1000 МГц соответственно).

Цифровой процессор обработки сигналов (ЦПОС) чаще всего определяется как процессор, предназначенный для выполнения обработки данных и ре-

шения задач анализа и управления в реальном масштабе времени с использованием "оцифрованных" реальных сигналов [21], в том числе и реализации алгоритмов ЦОС в реальном масштабе времени. Примером различия задачи решаемой в реальном и в нереальном масштабе времени, может служить задача обработки изображения. Обработка фотографий небесных объектов, передаваемых из космоса, может производиться достаточно долго на универсальных ЭВМ, в то время как обработка кадров кинофильма или кадров телевизионного изображения должна выполняться за некоторый промежуток времени, определяемый частотой смены кадров.

Для повышения производительности ЦПОС в них используются общие методы повышения производительности типа увеличения тактовой частоты работы. Однако главным образом применяются структурные и архитектурные решения, учитывающие специфику задач и алгоритмов ЦОС. Таким образом, ЦПОС являются специализированными или проблемно-ориентированными процессорами. Они во многих случаях обеспечивают преимущества при решении задач ЦОС по сравнению с другими вариантами реализации систем.

2.2. Общие принципы построения ЦПОС и особенности их архитектуры

Термин "архитектура" [40] обычно используется для описания состава, принципа действия, конфигурации и взаимного соединения основных узлов вычислительной системы. Этот термин включает в себя также изложение возможностей программирования, форматов данных, системы команд, способов адресации и т. д. Таким образом, термин "архитектура" относится как к аппаратным средствам или программному обеспечению, так и к их комбинации.

С момента появления самых первых ЦПОС (1982 г.) их архитектура формировалась алгоритмами ЦОС. Любые особенности этих процессоров определяются требованиями, возникающими при реализации алгоритмов цифровой обработки сигналов (ЦОС). Исследование типичных алгоритмов ЦОС и их вычислительных требований является лучшим способом для изучения и понимания развития архитектуры ЦПОС [3].

Традиционным простым примером, на котором иллюстрируются особенности алгоритмов ЦОС и процессоров ЦПОС, является алгоритм реализации КИХ-фильтра.

Выходной сигнал фильтра определяется выражением

$$y(n) = \sum_{i=0}^{N-1} x(n-i) \cdot h(i),$$

где $x(n)$ — отсчеты входного сигнала; $h(i)$ — коэффициенты фильтра.

В соответствии с алгоритмом, выборки входного сигнала умножаются на коэффициенты фильтра и суммируются. Подобные вычисления используются и во многих других алгоритмах ЦОС *(см. главу 1)*. Таким образом, базовой операцией ЦОС является операция умножения и добавление (накопление) результата умножения. Подобную операцию часто обозначают при описаниях мнемоникой MAC.

Для того чтобы работать с высокой производительностью, процессор должен выполнять операцию MAC за один цикл (такт) работы процессора. Отсчеты сигнала, коэффициенты фильтра и команды программы хранятся в памяти. Для выполнения операции требуется произвести три выборки из памяти — команды и двух сомножителей. Следовательно, для работы с высокой производительностью эти три выборки необходимо произвести за один такт работы процессора. При этом подразумевается, что результат операции остается в устройстве выполнения операции (в центральном процессорном устройстве ЦПУ), а не помещается в память. В более общем случае нужна еще операция записи результата в память, т. е. необходимы четыре обращения к памяти за цикл. Таким образом, производительность процессора прежде всего определяется возможностями обмена данными между ЦПУ и памятью процессора и организацией их взаимодействия.

Ниже рассматриваются различные идеи и методы, используемые в ЦПОС для повышения производительности системы. Эти идеи являются общими для большинства процессоров и в той или иной форме сводятся к параллелизму — одновременной работе разнообразных модулей, одновременному выполнению разных операций в нескольких модулях.

2.2.1. Архитектура фон Неймана и гарвардская архитектура

На рис. 2.2 показана традиционная структура вычислительной системы, соответствующая "фон-неймановской" вычислительной машине. Американский математик Д. фон Нейман (1903—1957) предложил концепцию вычислительной машины (и в частности, хранимой в памяти программы), которая лежит в основе большинства современных машин [40]. Одним из основных моментов этой концепции является то, что система обладает единой памятью, в которой хранятся и команды программы и данные. Система содержит одну шину данных (ШД), по которой передаются и команды программы, и данные. Следовательно, в такой системе требуется три цикла для выборки команды и двух сомножителей (то есть для выполнения операции MAC).

В процессорах ЦПОС применяется гарвардская архитектура вычислительной системы, приведенная на рис. 2.3. Подобная архитектура названа по работе, выполненной в 40-х годах XX века в университете Гарварда под руководством Г. Айкена (1900—1973) [7]. В соответствии с этой концепцией для хранения программы (команд) и данных используются различные уст-

ройства памяти. Соответственно в системе имеется два комплекта шин для этих устройств: шина адреса памяти программ (ШАПП), шина данных памяти программ для работы с памятью программ (ПП) и шина адреса памяти данных (ШАПД), шина данных памяти данных (ШДПД) для работы с памятью данных (ПД). В системе с гарвардской архитектурой можно одновременно производить операции обращения к различным устройствам памяти, т. е. синхронно выбирать команду из памяти программ ПП по шине ШДПП и сомножитель из памяти данных ПД по шине ШДПД. Соответственно при этом для выполнения операции МАС требуется два цикла работы процессора. Реально за счет различных дополнительных мер почти всегда время операции МАС сводится к одному циклу. Различные варианты реализации операции рассмотрены ниже. В частности, ПП иногда используется не только для хранения команд, но и данных. Поэтому при описаниях ЦПОС говорят о модифицированной гарвардской архитектуре.

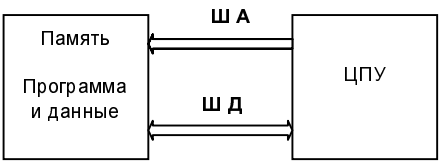


Рис. 2.2. Архитектура фон Неймана

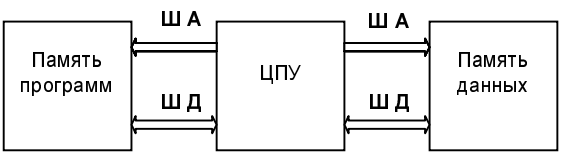


Рис. 2.3. Гарвардская архитектура

Следует лишь иметь в виду, что несколько комплектов шин для одновременной выборки данных и команд из ПД и ПП используются только внутри кристалла ЦПОС при работе с внутренней памятью процессора. Для обращения к внешней памяти во всех процессорах применяется один комплект внешних шин — ВША и ВШД. Это определяется ограничениями, накладываемыми технологией на количество выводов ИС. Поэтому разработчики систем ЦОС стремятся использовать только внутреннюю память, и процессоры выпускаются с большой внутренней памятью как ПП, так ПД. При использовании внешней памяти, особенно для хранения программы и данных, увеличивается время, затрачиваемое на выполнение операций. Некоторые вопросы использования внешней памяти рассмотрены в *разд. 2.5.*

2.2.2. Структура процессора ЦПОС

На рис. 2.4 представлена обобщенная структура процессора ЦПОС, отражающая только основной состав узлов процессора и их взаимодействие.

Для хранения информации в процессоре используются память программ ПП и память данных ПД, которые связаны с другими устройствами шинами. ЦПОС имеет шины различного назначения. Шина адреса ПП ШАПП предназначена для передачи адресов ячеек памяти программ. Шина данных ПП ШДПП служит для передачи команд, хранимых в памяти программ, а также данных при использовании ПП для хранения данных (например, таблиц коэффициентов цифровых фильтров). Шины адреса ПД ШАПД и данных ПД ШДПД применяются для передачи адреса и данных памяти данных. Количество шин, особенно шин данных в различных процессорах существенно отличается. Увеличение количества ШДПД связано с увеличением производительности процессоров за счет одновременной передачи данных для использования в различных модулях. Это дает возможность модулям одновременно выполнять определенные операции. В некоторых процессорах (например, TMS320C5000) производится разделение функций шины данных: используются различные шины данных для чтения и записи информации.

Все процессоры ЦПОС имеют внутреннюю (внутрикристальную) память. Однако внутренней памяти процессора иногда оказывается недостаточно для хранения программ и данных. Кроме того, процессор может не иметь внутренней памяти типа ПЗУ для хранения программы, которая не изменяется. В этих случаях может использоваться внешняя память, связь с которой осуществляется через интерфейс внешних шин и внешние шины адреса ВША и данных ВШД. Как уже указывалось ранее, количество внешних шин ограничено двумя, поэтому при использовании внешних ПП и ПД одновременное обращение к ним может вызвать конфликт и задержку операций чтения или записи. Внешние шины могут использоваться не только для обращения к внешней памяти, но и к другим адресуемым устройствам, например параллельным портам.

Устройство управления выполнением программы в соответствии с командами, читаемыми из ПП, вырабатывает сигналы управления работой всех узлов процессора. Оно связано с регистрами состояния и управления. В эти регистры, адресуемые как ячейки памяти, заносится на этапе инициализации системы различная информация, управляющая работой процессора, например информация об используемой конфигурации памяти и распределении адресов между внешней и внутренней памятью. В эти же регистры заносится информация о текущем состоянии процессора, например, о наличии на входе запроса на прерывание.

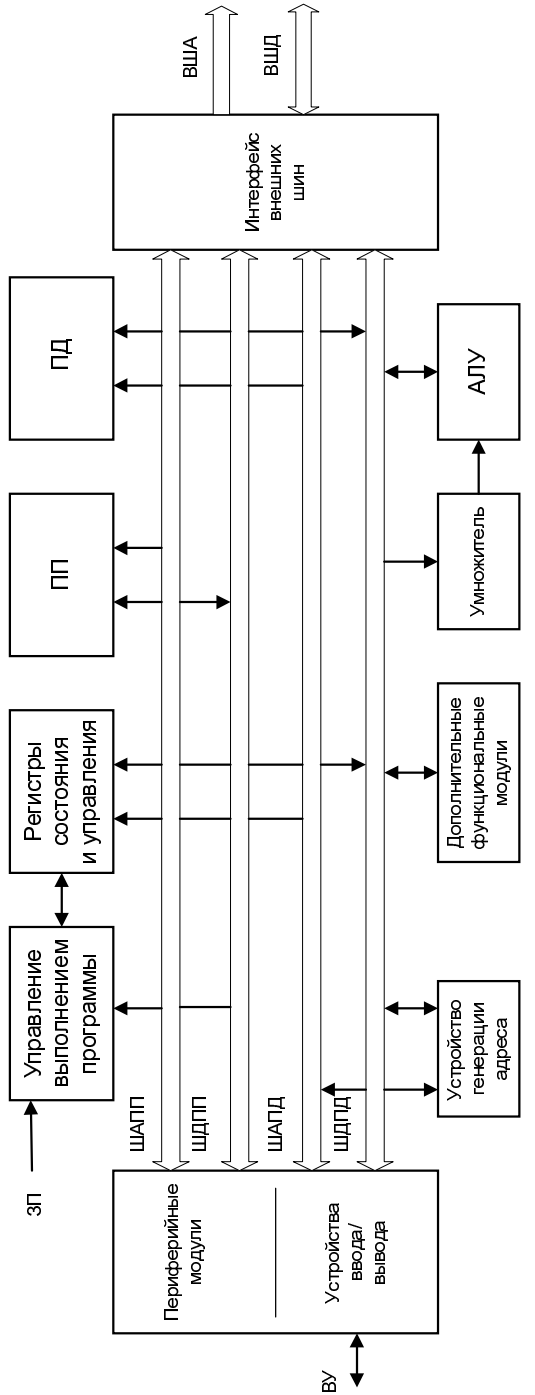


Рис. 2.4. Обобщенная функциональная схема ЦПОС

На устройство управления от внешних устройств ВУ поступают запросы на прерывания основной программы работы. Как уже отмечалось, ЦПОС, как правило, работают с ВУ и их сигналами в реальном масштабе времени. Работа в режиме с прерываниями и, в частности, ввод/вывод информации по прерываниям позволяют процессору согласовывать свою работу и прием/выдачу сигналов со скоростью (частотой) работы ВУ. Ввод/вывод информации от ВУ осуществляется через устройства ввода/вывода, в качестве которых в основном применяются различного вида последовательные порты *(см. главу 8)*.

ЦПОС имеют большое количество периферийных модулей, состав и количество которых определяются назначением процессора. Все процессоры имеют различные таймеры, предназначенные, скажем, для генерации сигналов необходимых частот (например, частоты дискретизации внешнего АЦП) и внутренних запросов на прерывания по таймеру, с помощью которых можно организовать временной опрос внешних датчиков информации.

АЛУ, умножитель и дополнительные функциональные узлы предназначены для выполнения операций над обрабатываемой информацией. Принципы их построения определяют производительность процессора, и они описываются ниже.

Устройство генерации адреса (УГА) формирует адреса данных, извлекаемых из ПД. Для одновременной выборки нескольких операндов необходимо формировать одновременно несколько адресов. Для этого и процессор может иметь несколько устройств УГА. Эти устройства включают в себя арифметические модули для вычисления адресов при различных сложных методах адресации *(см. главу 5)*.

2.2.3. Конвейерное выполнение команд

Процесс выполнения команды во всех процессорах разбивается на несколько этапов. Конвейерный принцип выполнения команд состоит в том, что различные этапы разных команд выполняются одновременно. Необходимо подчеркнуть: выполняются *одновременно* этапы *различных* команд в *разных* функциональных устройствах.

Количество этапов, на которые разбивается процесс выполнения команды, отличается в разнообразных процессорах.

- В процессорах TI C2X существует три этапа конвейера: выборка, декодирование, выполнение команды. В процессорах TI C24X, C5X, C20X — четыре этапа: выборка команды, декодирование команды, подготовка операнда, выполнение. В процессорах TI C5000 — шесть этапов. В процессорах C6000 количество этапов переменное и доходит до 11.
- В процессорах ADI AD2100 и AD2106x — три этапа.
- В процессорах Motorola DSP56K — три этапа, а в процессоре DSP56300 — семь.

Следует отметить, что приведенные названия этапов не отражают полностью характер выполняемых действий, т. к. на определенном этапе команды могут осуществлять несколько действий, например на этапе декодирования может начинаться модификация содержимого регистров для подготовки операндов и т. д.

Время выполнения одного этапа команды называют *командным циклом* работы процессора (иногда — тактом работы процессора). Время командного цикла, как правило, равно периоду внутренней тактовой частоты работы процессора. (Внешняя частота сигнала управления, подаваемая на выводы процессора, может отличаться от внутренней частоты работы процессора.)

На рис. 2.5 приведена схема выполнения команды во времени и обозначения этапов при разделении ее на четыре этапа.

Рис. 2.5. Выполнение команды по этапам

На рис. 2.6 представлена схема неконвейерного (полностью последовательного) способа выполнения команд. В этом случае время выполнения команды равно 4-м циклам; время выполнения m команд равно $4m$ циклов.

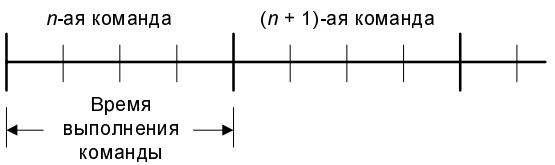


Рис. 2.6. Неконвейерное выполнение команд

На рис. 2.7 изображен конвейерный способ работы команд, при котором различные этапы выполнения разных команд совмещаются. При этом время выполнения каждой отдельной команды остается по-прежнему равным 4-м циклам, однако результаты выполнения отдельных команд будут появляться через интервал времени, равный одному циклу (при большом количестве последовательно выполняемых команд). Период появления результатов команд и считается в этом случае временем *выполнения команды*.

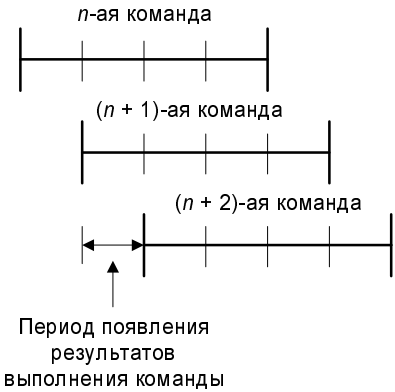


Рис. 2.7. Конвейерный способ выполнения команд

В табл. 2.1 приведено распределение во времени различных этапов разных команд при конвейерном способе выполнения.

Таблица 2.1. *Распределение этапов команд при конвейерном способе выполнения*

Номер команды	Номер цикла						
	i – 2	*i* – 1	*i*	*i* + 1	*i* +	*i* + 3	*i* + 4
n – 1		ВбК	ДК	ПО	ВпК		
n			ВбК	ДК	ПО	ВпК	
n + 1				ВбК	ДК	ПО	ВпК
n + 2					ВбК	ДК	ПО

Конвейерное выполнение команд возможно при условии, что в процессоре имеются несколько функциональных узлов для выполнения одновременно различных этапов с разными данными, относящимися к разным командам. В частности использование гарвардской архитектуры позволяет одновременно выбирать команду из памяти программ и данные из памяти данных. Виды одновременно работающих функциональных узлов будут рассмотрены далее.

На стационарность функционирования конвейера, изображенного на рис. 2.7, последовательность выполнения этапов которого торажена в табл. 2.1, влияет несколько факторов.

- **Длина команды**. Для нормальной работы конвейера все команды должны иметь одно и то же количество слов, чтобы их выборка всегда занимала одинаковое время.
- **Вид исполняемой программы**. Для того чтобы команды программы выполнялись строго в соответствии с порядком, приведенном в табл. 2.1, про-

грамма должна носить линейный характер (без команд условных переходов), иными словами порядок следования команд должен быть предопределен. Если же после данной команды осуществляется команда условного перехода, то следующая команда определится только после последнего этапа ее выполнения, т. е. вычисления условия. При этом может оказаться, что должна выполняться команда, не следующая по порядку в записи программы (и уже выбранная). Тогда выборку команды необходимо начинать заново, и происходит потеря нескольких циклов. Кроме того, последовательность выполнения этапов разных команд может нарушаться при·безусловных переходах, переходах к выполнению подпрограмм и возвратах в основную программу, прерываниях основной программы и возвратах в нее, выполнении циклов.

При нарушении стационарного последовательного характера выполнения этапов команд, т. е. нарушениях нормального функционирования конвейера, возникают так называемые *конфликты конвейера.*

Рассмотрим пример. Пусть в процессоре TMS320C5x требуется выполнить команды

```
LACC  *+     ; загрузка аккумулятора
ADD   #2000h ; прибавление к содержимому аккумулятора числа
SACL  *+     ; сохранение в памяти результата
```

Первая и третья команды имеют длину одно слово, а вторая — два слова. Во втором слове этой команды записано непосредственное число 2000h, которое прибавляется к содержимому аккумулятора. Порядок выполнения и распределение этапов, рассматриваемых трех команд по циклам, приведены в табл. 2.2. В результате того, что для выборки из памяти второго слова команды `ADD` требуется дополнительный цикл, происходят задержка выборки и декодирования третьей команды и затягивание выполнения всей последовательности команд. Подобный вариант затягивания выполнения команд при использовании команд разной длины имеет место и в других процессорах TI, процессорах Motorola DSP56K. В процессорах ADI все команды имеют одинаковую длину.

Таблица 2.2. *Порядок выполнения и распределение этапов команд по циклам*

Номер цикла	Этап конвейера			
	ВбК	**ДК**	**ПО**	**ВпК**
1	LACC			
2	ADD	LACC		
3	#2000h	ADD	LACC	
4	SACL	Пусто	ADD	LACC

Таблица 2.2 *(окончание)*

Номер цикла	Этап конвейера			
	ВбК	**ДК**	**ПО**	**ВпК**
5		SACL	Пусто	ADD
6			SACL	Пусто
7				SACL

Иногда при выполнении команд конфликты конвейера возникают на аппаратном уровне. Примерами могут служить следующие ситуации.

- **Требуется на одном этапе осуществить выборку из внешней памяти команды и данных**.

  Все процессоры имеют одну шину данных для обращения к внешней памяти и в подобных ситуациях возникают циклы ожидания. Циклы ожидания появляются также при обращении к "медленной" внешней памяти *(см. разд. 2.5)*. Для предотвращения подобных конфликтов необходимо знать возможности процессора и размещать секции команд и данных по различным блокам памяти, например, допускающим двойной доступ за один цикл.

- **Возникновение необходимости на разных этапах выполнения команд использовать одни и те же функциональные узлы или одни и те же данные**.

  Например, в процессорах Motorola DSP56K нельзя загружать регистр адреса в данной команде и использовать его же в качестве счетчика следующей.

Другой пример. Пусть в процессоре TMS320C3x должны последовательно выполняться команды

```
LDI  9, AR2     ; загрузка вспомогательного регистра AR2 константой
MPYF *AR3, R0   ; умножение с использованием для адресации
                ; вспомогательного регистра AR3
```

В процессорах C3X имеет место условие: если загружается вспомогательный регистр, использование любого другого вспомогательного регистра задерживается, пока не закончится процесс записи. Таким образом, декодирование второй команды будет задержано до окончания выполнения первой.

При возникновении конфликта на уровне эксплуатации функциональных устройств процессор в большинстве случаев автоматически добавляет пустые такты или циклы. При использовании процессоров Motorola в подобных случаях имеет место следующее. Ассемблер Motorola имеет опцию `RP`, при использовании которой на этапе трансляции в программу автоматически вставляются пустые команды `NOP`. В противном случае генерируется сооб-

щение об ошибке. Тем самым предоставляется возможность самому программисту выбрать способ программы.

Как уже отмечалось выше, с конвейерным принципом выполнения команд связан вопрос определения времени выполнения команды.

Время выполнения этапа команды — длительность командного цикла обычно совпадает с периодом основной частоты работы процессора. В этом случае время выполнения команд, разбиваемых на четыре этапа в нормально работающем (без конфликтов, см. табл. 2.1) конвейере, фактически равно 4-м циклам или периодам частоты работы процессора. Однако такие команды считаются одноцикловыми (однотактовыми), т. е. время их выполнения равно длительности командного цикла. Таким образом, под временем выполнения команды понимается не время обработки одиночной команды, а время появления результатов выполнения команды в длинной последовательности других. (Оно подобно периоду появления автомобилей на конце сборочного конвейера; в то время как время движения автомобиля по конвейеру может быть существенно больше.) Если же выполнение команды требует дополнительных этапов (типа выборки второго слова команды) по сравнению с другими командами процессора, то время ее выполнения увеличивается на соответствующее количество циклов.

В некоторых случаях, например при конфликтах использования одних функциональных узлов последовательно выполняемыми командами, фактическое время выполнения команды зависит от применяемой последовательности команд. Это имеет место, например, в процессоре TI TMS320C3X.

2.2.4. Аппаратная реализация программных функций. Параллельная работа различных функциональных узлов

В ЦПОС операции, которые обычно в процессорах общего назначения выполняются программным образом, реализуются аппаратным способом, т. е. для их выполнения предназначены дополнительные функциональные узлы и специализированные модули. Эти модули работают одновременно с основным АЛУ, повышая тем самым общую производительность системы.

Рассмотрим некоторые используемые специализированные модули.

Умножитель

Первое и основное место среди подобных модулей занимает *умножитель*, используемый во всех ЦПОС. Он производит операцию умножения данных в формате "слово" процессора (16×16 для 16-разрядных процессоров или 32×32 для 32-разрядных) за один цикл (в отличие от программных методов реализации операции умножения, которые требуют многих циклов). Умножитель

является основным элементом при выполнении операции умножения/накопления MAC. Варианты использования умножителя в процессорах с различной архитектурой будут рассмотрены ниже.

Сдвигатели

Сдвиги операндов в ту или иную сторону на определенное количество разрядов можно производить в АЛУ, однако при этом для выполнения сдвига требуется отдельная команда. В процессорах Motorola, TI и некоторых других имеются аппаратно реализованные модули сдвига, расположенные в цепях передачи операндов между различными модулями[1]. Они позволяют производить сдвиги при передаче и загрузке операндов без использования дополнительных команд.

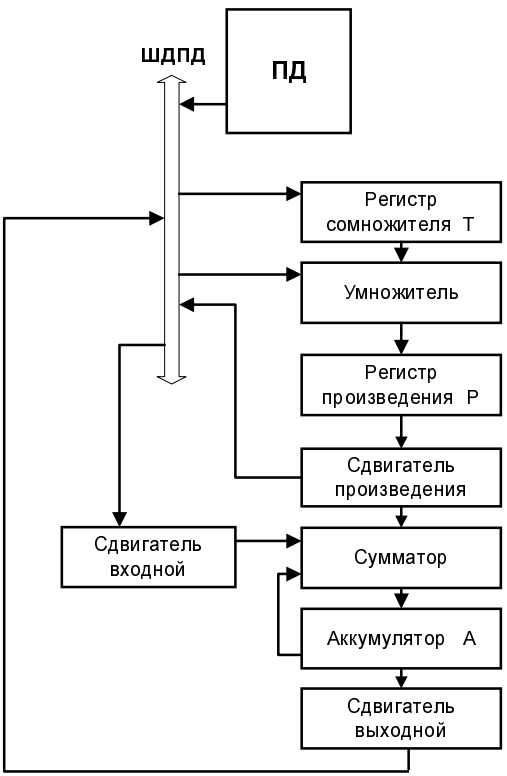


Рис. 2.8. Функциональная схема цепей передачи данных в ЦПУ процессоров TI

[1] Следует отметить, что в фирменных описаниях конкретных процессоров названия различных модулей могут отличаться от используемых в данной книге. Отличаются также названия модулей, используемых для одних и тех же целей в процессорах различных фирм.

Рассмотрим пример. На рис. 2.8 приведена функциональная схема части ЦПУ процессоров TI C2X, C20X, C5X, на которой показаны цепи передачи данных между памятью и узлами ЦПУ. В этих процессорах операнд при загрузке из памяти данных ПД в сумматор проходит через "Сдвигатель входной", результат выполнения операции из аккумулятора в ПД передается через "Сдвигатель выходной", произведение из регистра P в сумматор или в ПД — через "Сдвигатель произведения".

Дополнительные арифметические устройства

Дополнительно к основному АЛУ, выполняющему различные арифметические и логические операции, в процессорах ЦПОС применяют вспомогательные арифметические устройства. Они позволяют осуществлять различные математические операции одновременно с основным АЛУ, повышая тем самым производительность системы.

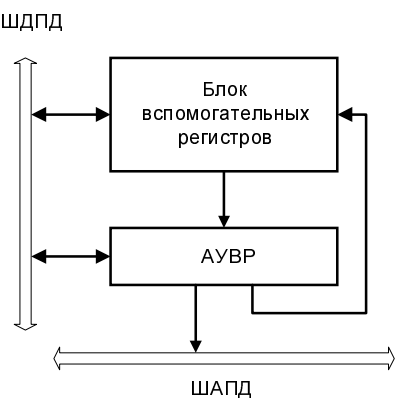


Рис. 2.9. Арифметическое устройство вспомогательных регистров АУВР процессоров TI

Например, в процессорах TMS320 фирмы TI используется второе арифметическое устройство АУВР (арифметическое устройство вспомогательных регистров) для работы со вспомогательными регистрами (рис. 2.9). Вспомогательные регистры применяются в основном для косвенной адресации данных, а также могут служить в качестве регистров общего назначения. Входной информацией для АУВР может быть содержимое регистров и операнды, передаваемые из памяти данных по шине данных ШДПД. АУВР используется для модификации содержимого регистров и получения адресов данных, которые передаются на шину адреса памяти данных ШАПД и для сохранения в регистры.

На рис. 2.10 приведена функциональная схема умножителя и АЛУ процессоров фирмы ADI. В этих процессорах для накопления результатов умножения

(то есть для выполнения операций MAC) применяется отдельный сумматор — накопитель. Результат накопления из регистра MR по отдельной шине P (шине результата P) передается для дальнейшей обработки в общее АЛУ.

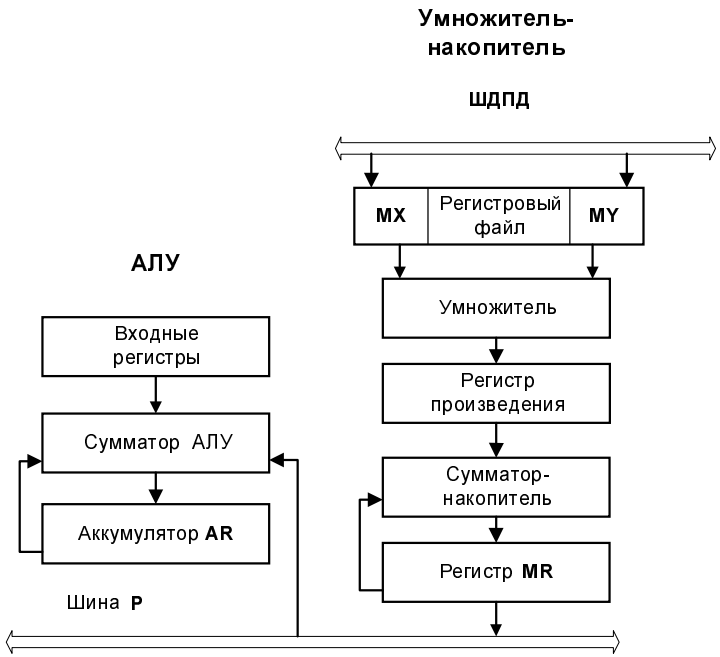


Рис. 2.10. Дополнительный сумматор устройства умножения процессоров ADI

Иногда в ЦПОС добавляют дополнительное устройство для выполнения только логических операций, например, устройство PLL в процессорах TI TMS320C5X.

Большинство последних процессоров имеет как минимум два аккумулятора, позволяющих сохранять несколько результатов вычислений в АЛУ.

Специализированные устройства генерации адреса УГА (AGU)

В процессорах ЦПОС используют специализированные устройства для генерации адресов данных в памяти данных. Эти устройства формируют или модифицируют адреса для обращения к операндам, размещенным в памяти данных ПД. Формирование адресов при различных методах адресации, особенно специальных *(см. главу 5)*, связано с выполнением вычислений. УГА функционируют параллельно с другими модулями и позволяют одновременно с выполнением операций в АЛУ вычислять адреса операндов для следующей команды. В них используются наборы регистров для хранения адре-

сов и некоторых данных (например, значений индексов или приращений при модификации адреса) и специализированные АЛУ. По общей структуре и назначению УГА совпадают с устройством, приведенным на рис. 2.10 (блок вспомогательных регистров и АУВР). Отдельно как устройства генерации адресов данных УГА выделяются в процессорах ADI (DAG), Motorola (AGU), TMS320C55X (DAGEN). В этих процессорах используются два устройства, позволяющих формировать одновременно два адреса для двух операндов.

Аппаратная организация циклов

Циклические процессы, т. е. повторение одиночных команд и их блоков, занимают значительное место среди алгоритмов ЦОС. Обычная организация циклов программным образом требует использования команд формирования и проверки условий окончания циклов, которые должны выполняться при каждом прохождении "тела" цикла. На выполнение этих команд затрачивается время. Поэтому в ЦПОС используются устройства, которые позволяют организовать циклы с "нулевыми потерями" времени на организацию (проверку условий окончания).

В процессорах TI для повторения одиночной команды применяется счетчик повторений RC, который загружается командой RPT и содержимое которого декрементируется на каждом шаге. Пока содержимое RC не равно нулю, значение программного счетчика PC не изменяется. Для повторения блока команд дополнительно существуют регистры начала (RSA) и конца (REA) блока команд, в которые записываются адреса первой и последней повторяемой команды. На каждом шаге значение программного счетчика PC сравнивается с REA; если значение PC становится больше REA, в PC переписывается значение RSA и декрементируется счетчик количества повторений. В процессорах C55 имеется два комплекта таких регистров для организации циклов.

В процессорах Motorola используется команда цикла `DO`, которая также работает с регистрами начала и конца цикла (LC и LA). Аналогичным образом циклы реализуются в процессорах ADI.

2.2.5. Использование нескольких АЛУ

Идея эксплуатации нескольких одновременно работающих функциональных устройств находит свое полное воплощение в некоторых высокопроизводительных процессорах в форме использования для выполнения программы нескольких АЛУ, в некоторых случаях полностью идентичных.

Приведем примеры использования нескольких АЛУ в различных ЦПОС.

- Процессоры TI C55 имеют два устройства умножения/накопления, которые позволяют выполнять одновременно: перемножение двух пар опе-

рандов или одно перемножение с накоплением и одно перемножение и некоторые другие операции. Эти действия в программе оформляются как параллельно выполняемые команды.

- Процессор LUCENT DSP16xxx имеет функциональный блок, включающий два умножителя, два сдвигателя, два устройства выполнения операций над отдельными разрядами, АЛУ и аккумулятор. Описание работы процессора приведено в *разд. 2.4.2.*
- Процессоры ADI ADSP-21160 и ADSP-TS001 TigerSHARC имеют по два узла, каждый из которых включает регистровый файл, умножитель и АЛУ с возможностями одновременного выполнения операций умножения/накопления или выполнения независимых команд *(см. разд. 2.4.2).*
- Процессоры Motorola MSC8101 используют ядро ЦПОС Star Core SC140. Это ядро имеет четыре параллельно работающих АЛУ (DALU), каждое из которых состоит из MAC — устройства умножения/накопления, устройства обработки отдельных разрядов BFU. Общими для всех АЛУ являются 8 сдвигателей и 16 регистров, используемых в качестве источников и получателей операндов и результатов для всех АЛУ. Команда для каждого АЛУ и некоторых других функциональных устройств представляет собой 16-разрядное слово. От одной до шести команд отдельных узлов могут объединяться в одну общую команду. Команды отдельных АЛУ способны включать различные операции над двумя операндами, в том числе и умножение с накоплением. Более подробное описание архитектуры процессора *см. в разд. 2.4.3.*
- Наибольшее количество АЛУ имеют процессоры платформы C6000 фирмы TI. В них имеется восемь АЛУ, объединенных в две группы по четыре — типа S, L, M, D. АЛУ внутри группы имеют специализацию: M предназначен для выполнения операций умножения и умножения с накоплением, S, L и D ориентированы на выполнение арифметических операций *(см. разд. 2.4.3).*

Следует отметить, что эффективное использование и полная загрузка нескольких АЛУ достигаются при использовании простых команд типа "регистр, регистр → регистр" *(см. разд. 2.2.6).*

Дальнейшим развитием идеи использования нескольких АЛУ является объединение нескольких ЦПОС в одном кристалле.

- Процессор ADI ADSP-2192 объединяет два независимых ЦПОС, имеющих общую и локальные области памяти.
- Процессор TI C5441 имеет внутри кристалла четыре ядра C54x с локальными и общей областями памяти.
- Процессоры TI C5420 и C5421 содержат внутри кристалла два независимых процессора с ядром C54x с внутренней магистралью обмена данными между ними.

2.2.6. Регистровые файлы

Все ЦПОС имеют *регистровые файлы* — наборы регистров, предназначенных для выполнения различных функций. Количество наборов и количество регистров в наборе в различных процессорах меняется в достаточно широком диапазоне и имеет тенденцию увеличиваться в современных (новых) процессорах.

Выделим основные функции регистровых файлов.

- Использование регистров для хранения и модификации адреса операнда при косвенной адресации. Примером могут служить наборы вспомогательных регистров AR*n* процессоров TI. Их количество меняется от 2 в TMS320C1X до 8 в TMS320C5X. Следует отметить, что эти регистры могут также использоваться как регистры общего назначения, например для временного сохранения данных.
- Применение набора регистров для задания и сохранения конфигурации системы (например, конфигурации памяти, маскирования прерываний) и контроля режимов работы системы (скажем, наличие на входе сигналов запроса на прерывание). В эти регистры (и отдельные разряды регистров) пользователь программным образом на этапе инициализации системы записывает конкретные управляющие слова, определяющие конфигурацию системы и режимы ее работы. Процессор при выполнении программы фиксирует изменяющуюся информацию о своем состоянии в некоторый момент времени. Эта информация может считываться программным образом и использоваться для управления работой и изменения состояния процессора. Управляющие регистры и регистры состояния, как правило, являются регистрами, отображенными на память, т. е. помимо имени они имеют определенный адрес в пространстве памяти процессора и к ним можно обращаться как к ячейкам памяти для считывания и записи информации. Примерами таких регистров являются регистры состояния ST*n*, PMST процессоров TMS320 фирмы TI, регистры конфигурации памяти OMR процессоров DSP56K Motorola и др.
- Использование регистров в качестве источников операндов, получателей результатов, источников/получателей операндов и результатов.

Здесь следует отвлечься и отметить следующее.

В любых процессорах (не только ЦПОС) существуют команды различных типов: "регистр, регистр → регистр", "память, память → память" "память, память → регистр" "регистр → память" и др. [21]. Тип команды "регистр, регистр → регистр" означает, что источником двух операндов при ее выполнении являются регистры, и результат операции помещается также в регистр. Компиляторы языков высокого уровня, вообще говоря, наиболее эффективно используют простые команды типа "регистр, регистр → регистр",

которые выполняются также достаточно быстро и существуют в высокопроизводительных процессорах с несколькими АЛУ *(см. разд. 2.4)*.

Применение регистровых файлов в качестве источников/получателей операндов позволяет широко использовать команды типа "регистр, регистр → регистр" и строить быстродействующие процессоры с архитектурой, "дружественной к языку С" *(см. разд. 9.6)*.

В ЦПОС различных фирм (и в процессорах разных семейств одной фирмы) используются (в качестве основных) все из упомянутых выше типов команд.

В качестве получателей результатов операций практически всегда используются регистры (наборы регистров). Примерами могут служить:

- аккумулятор А в процессорах TMS320C1x, TMS320C2X;
- два аккумулятора А и В в процессорах TMS320C5x, TMS320C54X, DSP16XXX, Motorola DSP 56K и др.;
- регистры AR, MR в процессорах ADI.

В качестве источников операндов в процессорах с фиксированной запятой фирмы TI (TMS320C1X, TMS320C2X, TMS320C20X) в основном используется память.

В процессорах Motorola, ADI источником операндов являются регистры MX, MY при умножении и AX, AY при сложении (ADI), X0, X1, Y0, Y1 (Motorola).

В процессорах TI C55X источниками операндов и получателями результатов служат наборы регистров AC0—AC3, TRN0, TRN1, кроме того, источником операндов может служить также и память (как в более ранних моделях ЦПОС этой фирмы).

В процессорах TI с плавающей точкой TMS320C3X имеется регистровый файл R*n* на 8 регистров, который служит всегда получателем и для некоторых команд также и источником операндов. Кроме того, источником операндов может служить память. Вид источника операнда определяется используемым методом адресации.

Ядро ЦПОС Star Core SC140 использует для своих четырех АЛУ регистровый файл на 16 регистров для операндов и результатов.

В процессорах семейства С6000 TI для 8 АЛУ применяются два регистровых файла А и В по 16 регистров каждый.

2.2.7. Специальные методы адресации

Для реализации алгоритмов ЦОС в ЦПОС существуют специальные методы адресации. К ним, прежде всего, относятся бит-реверсивная и циклическая (циркулярная) адресации. Подробно методы адресации рассмотрены в *главе 5*. Здесь опишем организацию циркулярных буферов, используемых при циклической адресации.

Циркулярный буфер представляет собой набор ячеек в памяти данных, обращение к которым производится по циклу, т. е. при достижении последней ячейки буфера ячейкой, к которой производится очередное обращение, является не следующая по порядку, а начальная ячейка. На рис. 2.11 изображен буфер длиной 10 с физическими адресами ячеек памяти ФА ($n - 1$) — ($n + 8$). Адреса ячеек в буфере обозначены "Ад буф" и имеют номера 1—10. Обращение к ячейкам буфера может идти в любом направлении: "вниз"и "вверх". При движении "вниз" после ячейки с номером ФА, равным ($n + 8$), будет осуществляться обращение к ячейке с адресом ($n - 1$), а при движении "вверх" после ячейки ($n - 1$) — обращение к ячейке ($n + 8$). Изменение адреса при движении по буферу может происходить с любым индексом (шагом).

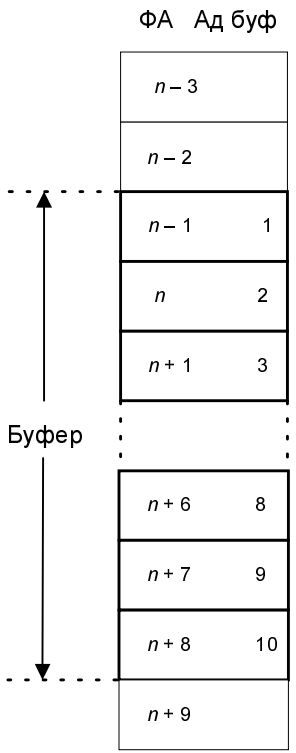


Рис. 2.11. Циркулярный буфер

Циркулярный буфер может использоваться, например, для организации линии задержки. На рис. 2.12 показана организация линии задержки на 9 тактов частоты дискретизации. Точки входа и выхода линии задержки, т. е. точки записи и считывания отсчетов сигнала, циклически перемещаются по буферу с расстоянием между ними, определяемым величиной задержки. На

рис. 2.12 приведены точки записи двух последовательных отсчетов $x(10)$, $x(11)$ и считывания задержанных отсчетов $x(1)$, $x(2)$. Отсчет $x(11)$ записывается на место отсчета $x(1)$, который больше не потребуется. Для адресации данных внутри буфера применяется косвенный метод адресации.

В процессорах TI для организации буферов существуют специальные регистры, куда записываются адреса начала и конца буфера.

В процессорах Motorola и ADI адресация буферов производится регистрами в УГА. Сильная поддержка буферов осуществляется специальными директивами на уровне языка ассемблера, например ассемблера процессоров Motorola.

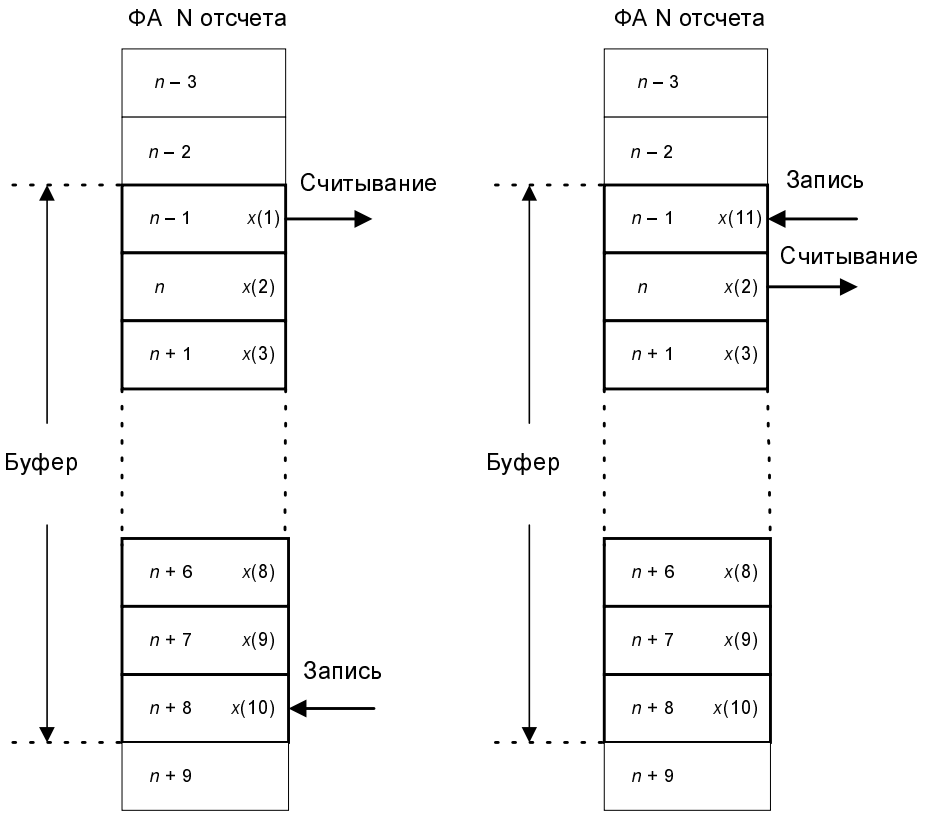


Рис. 2.12. Линия задержки на 9 тактов с использованием циркулярного буфера

2.2.8. Комбинированные и специализированные команды

Большое количество разнообразных дополнительных к основному АЛУ функциональных узлов дает возможность производить в процессорах одно-

временно несколько действий. Это, в свою очередь, предоставляет возможность ввода и широкого использования комбинированных команд, осуществляющих одновременно несколько действий. Комбинированные команды применяются в основном в процессорах первого и второго поколений со "стандартной" и "улучшенной стандартной" архитектурой *(см. разд. 2.4)*. В самых мощных высокопроизводительных процессорах выбрана архитектура RISC с набором упрощенных команд типа "регистр, регистр → регистр" *(см. разд. 2.2.6)*. Это объясняется тем, что сложные комбинированные команды плохо реализуются в компиляторах языка С и процессорах с несколькими АЛУ. В этом случае программы, написанные на языке С, проигрывают по эффективности программам на ассемблере. Следует также отметить, что грамотное использование сложных комбинированных команд на ассемблере требует от программиста хорошего знания системы команд и архитектуры конкретного процессора.

Комбинированные команды, прежде всего, используются для выполнения основной операции ЦОС — умножения с накоплением и различных ее вариантов. Реализация этих команд будет рассмотрена в *разд. 2.4.1*.

Другим примером комбинированных команд могут служить команды, связанные с действиями в АЛУ, умножителе и сдвигателе процессоров ADI, которые способны иметь условие выполнения. Например:

```
IF  AC  AR=AX0+AY0+C
```

где `IF AC` — условие выполнения: бит переноса не равен нулю; `AR`, `AX0`, `AY0` — регистры процессора.

В соответствии с командой будет выполнено сложение содержимого регистров при выполнении условия, которое является необязательным элементом команды.

К комбинированным командам можно также отнести команды, использующие косвенную адресацию с модификацией адреса *(см. главу 5)*. При выполнении этих команд одновременно с основным действием в АЛУ осуществляются вычисления адреса в УГА.

Некоторые команды ЦПОС фирмы Motorola допускают использование дополнительных пересылок данных (поля параллельных пересылок в командах). Например:

```
MAC X0,Y0,A X0,X:(R0)+Y:(R4)+,Y0
```

В соответствии с данной командой осуществляется операция умножения содержимого регистров `X0`, `Y0`, и одновременно содержимое регистра `X0` пересылается в X-память, а из ячейки Y-памяти выполняется пересылка в регистр `Y0`.

Процессоры ADI и TI применяют специальные команды для выполнения одновременно с основной операцией дополнительных пересылок данных. В описаниях процессоров такие команды называют также *многофункциональными* [32].

Процессоры TI TMS320C3X и TMS320C55X допускают существование некоторых команд в качестве параллельных, при этом одновременно выполняются несколько действий, описываемых этими командами, в различных функциональных узлах ЦПУ, и производительность процессора при использовании в качестве единицы измерения количества операций (MOPS) увеличивается в два раза. Такие, одновременно выполняемые, команды также иногда называют *комбинированными*.

Разновидностью комбинированных команд являются команды, выполняемые в процессорах TI C6000 и StarCore SC140. Как уже было отмечено выше, в этих процессорах присутствуют несколько АЛУ (8 и 4 соответственно). Команды, определяющие действия в отдельных АЛУ, выполняются параллельно *(см. разд. 2.3)*.

В некоторых процессорах применяются специализированные команды, ориентированные на выполнение операций ЦОС. Примерами таких команд могут служить:

- в процессоре TMS320C55X — вычисление сигнала на выходе симметричного/ассиметричного фильтра КИХ (конечная импульсная характеристика) FIRSADD/FIRSSUB;
- в процессоре TMS320C54X — вычисление сигнала на выходе симметричного фильтра КИХ (конечная импульсная характеристика) FIRS.

При повторении эти команды позволяют вычислять реакцию фильтра на входной сигнал, в конечном итоге они естественно сводятся к выполнению операций умножения/накопления.

2.2.9. Разнообразные устройства ввода/вывода и периферии

Решение задач ЦОС при использовании ЦПОС существенно облегчается из-за наличия в этих процессорах самых разнообразных устройств ввода/вывода информации и периферийных устройств.

К устройствам ввода/вывода относятся:

- параллельные и последовательные порты ввода/вывода, использующие различные протоколы передачи информации;
- каналы прямого доступа в память DMA, позволяющие вводить/выводить информацию в память системы без использования мощностей ЦПУ, т. е. без потери производительности системы;
- модули АЦП и ЦАП (аналого-цифрового преобразования и цифро-аналогового преобразования), разрешающие вводить/выводить в процессор аналоговый сигнал (Lucent DSP16XX, 16-разрядный дельта-сигма АЦП/ЦАП в процессорах DSP56156 Motorola, 16-канальные 10-разрядные АЦП в процессорах TMS320LF240X и TMS32024X);

- разнообразные модули, ориентированные на решение конкретных задач обработки сигналов: кодеки, декодеры, компрессоры;
- порты обмена информацией между процессорами при реализации многопроцессорной системы;
- сопроцессоры для решения специальных задач: сопроцессоры — декодеры избыточного кодирования Витерби (например, DSP16XX, DSP16XXX Lucent Technologies, TMS320C6416), сопроцессор — Turbo Decoder (TMS320C6416), сопроцессоры для реализации цифровых фильтров (DSP56307, DSP56311 Motorola);
- интерфейсы передачи данных в режиме ATM (UTOPIA), включающие контроллер ATM, приемопередатчик 8-разрядных комбинаций со скоростью передачи/приема до 50 Мбит (TMS320C6416);
- генераторы сигналов ШИМ (широтно-импульсной модуляции) в DSP контроллерах (TMS320C24X и TMS240X).

Более подробно некоторые из этих устройств рассмотрены в *главе 8*.

2.3. ЦПОС с фиксированной и плавающей точкой

Процессоры с фиксированной (ФТ) и плавающей точкой (ПТ) отличаются способностью обрабатывать сигналы и данные, использующие соответствующие формы представления *(см. главу 3)*. При этом следует иметь в виду, что все процессоры с ПТ имеют набор команд для обработки данных как с ФТ, так и с ПТ, т. е. являются в этом смысле универсальными.

С другой стороны, в процессорах с ФТ всегда можно организовать обработку данных с ПТ, но программным образом. Соответствующие программы преобразования и обработки данных требуют достаточно много времени для выполнения.

Основные отличия процессоров с ФТ и ПТ заключаются в следующем.

- Функциональные модули, выполняющие арифметические операции и операции умножения, в процессорах с ПТ по сравнению с ЦПОС с ФТ гораздо сложнее, т. к. алгоритмы выполнения операций над числами с ФТ и ПТ существенно отличаются.
- Процессоры с ПТ имеют более разнообразные типы представления данных, системы команд для обработки данных как с ФТ, так и ПТ и их взаимного преобразования.
- Разрядность внутреннего представления данных в процессорах с ПТ как правило 32 разряда, в некоторых ЦПОС возможно использование укороченной формы представления.

Усложнение процессоров с ПТ приводит к тому, что их цена становится выше, чем у процессоров с ФТ. Однако для многих применений это окупа-

ется большими преимуществами ЦПОС с ПТ. Основные преимущества сводятся к следующему.

- При использовании 32 разрядов и ПТ существенно повышается точность внутреннего представления данных.
- Существенно расширяется возможный динамический диапазон сигналов и данных, т. е. отношение максимально возможного к минимально возможному значению сигнала и соответственно отношение сигнал/шум.
- При использовании процессоров с ПТ снимается проблема масштабирования данных [14] с целью избежать переполнения при выполнении различных операций и особенно операций накопления[2].
- Большое разнообразие типов данных и особенно данных с ПТ приводит к тому, что архитектура ЦПОС с ПТ становится более дружественной для компиляторов с языка С *(см. главу 9)*; это в свою очередь позволяет получать более эффективные программы в ЦПОС с ПТ при использовании языков высокого уровня.

Достоинства процессоров с ПТ приводят к тому, что при их использовании построение системы ЦОС становится более легким и быстрым.

2.4. Основные типы ЦПОС

Как уже отмечалось выше, особенности архитектуры ЦПОС определяются тем, каким образом он реализует алгоритмы ЦОС. Исследование вариантов реализации типичных алгоритмов ЦОС и их вычислительных требований является лучшим способом для изучения и понимания развития архитектуры ЦПОС.

Традиционным простым примером, на котором иллюстрируются особенности алгоритмов ЦОС и процессоров ЦПОС является алгоритм реализации КИХ-фильтра, выходной сигнал которого определяется выражением

$$y(n) = \sum_{i=0}^{N-1} x(n-i) \cdot h(i),$$

где $x(n)$ — отсчеты входного сигнала; $h(i)$ — коэффициенты фильтра.

При определении типов процессоров в настоящей работе будем придерживаться терминологии, использованной в [3]. В соответствии ней существующие в настоящее время процессоры можно разделить, с точки зрения архитектуры, на следующие основные типы:

- стандартные процессоры (conventional);

---

[2] Отметим, что на входе любого ЦПОС мы имеем одинаковые входные данные в форме с ФТ, т. к. соответствующим образом работают все аналого-цифровые преобразователи (АЦП), которые нормированы к максимально возможному сигналу на входе как к 1. Первой операцией над входными данными в ЦПОС с ПТ является преобразование их в форму с ПТ.

- улучшенные стандартные процессоры (enhanced conventional);
- процессоры VLIW (очень длинное слово команды);
- суперскалярные процессоры (superscalar);
- гибриды ЦПОС/микроконтроллер.

Это деление естественно носит условный характер, особенно для процессоров первых двух типов. Вопрос, куда отнести тот или иной конкретный процессор, может вызвать затруднение. Однако такая классификация представляется полезной для определения особенностей построения и архитектуры процессоров.

Следует также отметить, что фирмы-производители объявляют о появлении новых процессоров, в том числе на основе новых архитектур. Например, фирма ADI анонсировала новый процессор. Так что возможно появление процессоров, не укладывающихся в приведенную классификацию.

Ниже рассматриваются особенности процессоров приведенных типов и примеры конкретных процессоров.

2.4.1. Стандартные процессоры ЦПОС (Conventional DSP)

Принципы построения процессоров этого типа рассмотрим на примерах реализации в них операции умножения с накоплением. Выполнение указанной операции отличается в процессорах различных фирм[3]. Как уже отмечалось, для выполнения этой операции требуется произвести три выборки из памяти — команды и двух сомножителей.

Процессоры TI

Структура блока АЛУ и пути передачи данных в процессорах первых поколений приведены на рис. 2.13 и 2.14.

Итак, в процессоре требуется вычислять выражение

$$y(n) = \sum_{i=0}^{N-1} x(n-i) \cdot h(i),$$

т. е. в цикле выполнять операцию $x(n - i) \cdot h(i)$ при различных значениях отсчетов сигнала $x(n - i)$ и коэффициентов фильтра $h(i)$.

[3] При сравнении способов реализации основной операции ЦОС в процессорах различных фирм полезно иметь в виду, что процессоры TI были первыми ЦПОС. Другие фирмы разрабатывали свои процессоры с учетом опыта TI. Следует также упомянуть, что при разработке любых процессоров в пределах определенного семейства всегда стремятся выполнить условие преемственности и совместимости новых процессоров с ранее выпущенными до тех пор, пока это условие не становится "тормозом" на пути получения новых свойств.

В процессорах с фиксированной запятой (TMS320C2X/2XX/24X/5X) можно использовать два варианта вычисления.

Вариант 1

Коэффициенты фильтра и отсчеты сигнала хранятся в памяти данных ПД и для вычисления приведенного выше выражения в цикле выполняются две последовательные команды:

```
LT    dma  ; загрузка 1-ого сомножителя в регистр T
MPYA  dma  ; умножение 2-ого сомножителя на содержимое
           ; регистра T, передача полученного произведения
           ; в регистр P, добавление предыдущего произведения
           ; (содержимого регистра P) к содержимому аккумулятора A
```

в которых адресуется память данных `dma` (data memory address). Схема выполнения операции изображена на рис. 2.13.

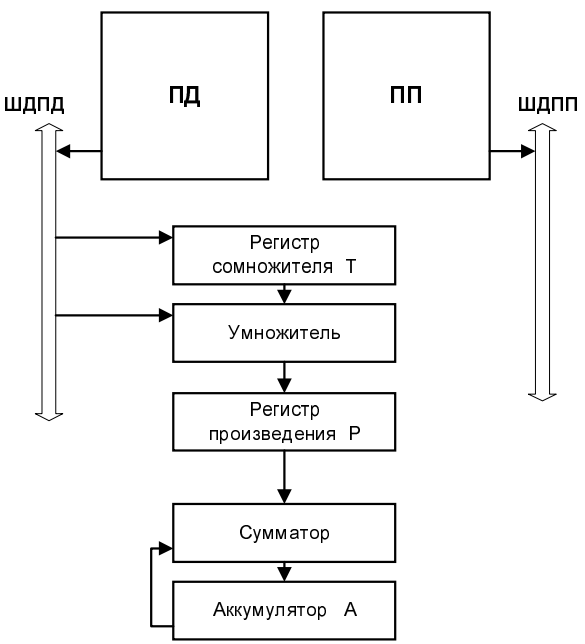


Рис. 2.13. Пути передачи данных при выполнении команд LT — MPYA

В цикле операция вычисления произведения с накоплением этим способом выполняется практически за два такта, т. к. на все вычисления добавляется одна команда прибавления последнего произведения APAC. Время вычисления суммы n произведений в тактах будет равно $(2n + 1)$. Такое время выполнения операции умножения с накоплением определяется тем, что из памяти данных ПД

за один такт можно произвести одну выборку, в то время как для однотактового выполнения операции требуется два обращения к памяти данных.

Вариант 2

Для хранения коэффициентов фильтра используется память программ ПП. В этом случае для вычисления отсчета на выходе фильтра можно применять одну комбинированную команду `MAC`, которая позволяет адресовать для двух сомножителей память данных (dma, data memory address) и память программ (pma, program memory address):

```
MAC  pma, dma  ; перемножение содержимого адресуемых ячеек
               ; ПП и ПД, передача полученного произведения
               ; в регистр P, добавление предыдущего произведения
               ; (содержимого регистра P) к содержимому аккумулятора A
```

Схема выполнения операции приведена на рис. 2.14.

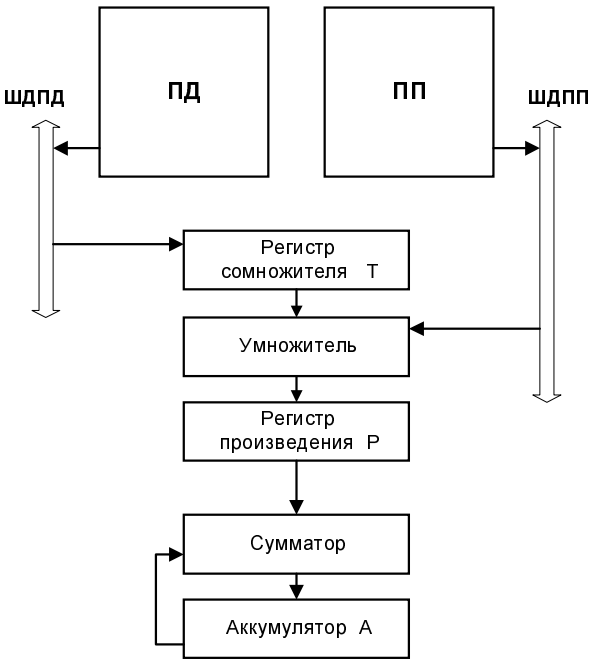


Рис. 2.14. Пути передачи данных при выполнении команды `MAC`

При повторении `MAC` в цикле с помощью команды повторения RPT нет необходимости выбирать каждый раз слово команды из ПП. Поэтому команда `MAC` становится практически однотактовой (при повторении). Время вычисления суммы n произведений в тактах будет равно $(n + 2)$. При использовании для вычисления команды `MAC` (с целью получения времени вычислений $(n + 2)$

для хранения коэффициентов необходимо задействовать только внутреннюю память программ. Таким образом, две выборки из ПД за один такт осуществляются за счет использования сложной команды с большим количеством действий, причем команда "работает хорошо" только при повторении.

Процессоры ADI (ADSP-2100)

В процессорах ADI операцию умножения с накоплением можно выполнить за один такт, используя команду[4] с параллельными пересылками данных:

```
MR=MR+MX*MY, DM->MX PM->MY ;перемножение содержимого регистров
    ; сомножителей MX и MY и добавление произведения
    ; к содержимому регистра результата MR,
    ; перемещение новых сомножителей из ПП и ПД в регистры MX и MY
```

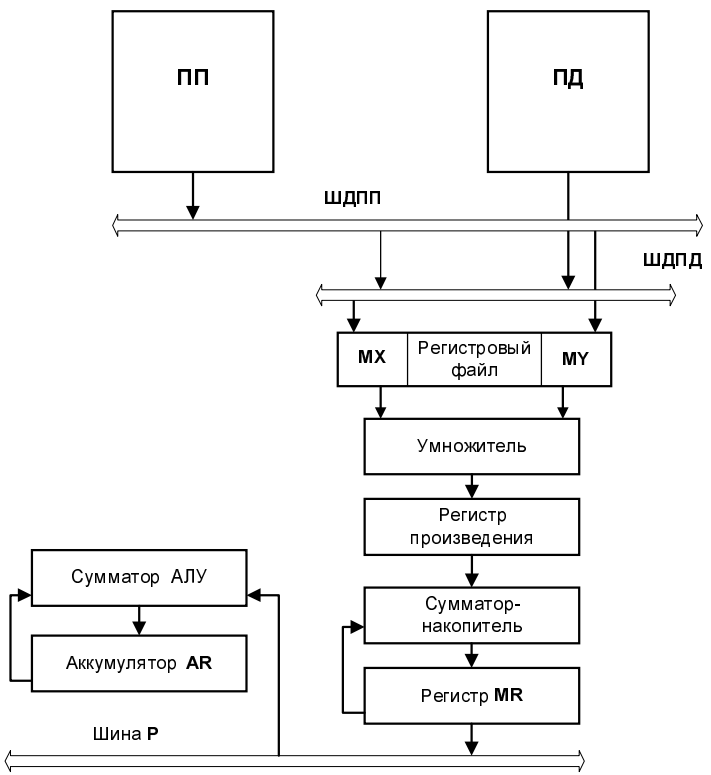


Рис. 2.15. Пути передачи данных при выполнении умножения с накоплением в процессорах ADI

[4] Запись команды носит условный характер, т. е. команда записана с нарушением правил языка ассемблера ADI.

Для хранения отсчетов сигнала используется ПД, а для хранения коэффициентов — память программ ПП. Приведенная команда является комбинированной. Одновременно с выполнением операций умножения/накопления производятся две выборки из ПП и ПД и запись их в регистры сомножителей MX и MY. Эти операции реализуются за один такт, т. к. считывание старого содержимого регистра (для выполнения умножения) происходит в начале такта, а запись нового содержимого — в конце такта работы процессора. Структура модуля, выполняющего операцию, и пути передачи данных приведены на рис. 2.15. При одновременном считывании данных из ПП и ПД для передачи данных из ПП в регистр MY служит отдельная магистраль.

Таким образом, в процессорах ADI для того, чтобы произвести две выборки данных из памяти, фактически используется добавочная шина передачи данных (но не называемая так), а также память и регистры с повышенным быстродействием.

Процессоры Motorola (DSP56000)

В процессорах Motorola (как и в некоторых других, например, Lucent DSP1600) задачу осуществления двух выборок данных за один такт для быстрого выполнения операции умножения с накоплением решили следующим образом [5]. Память данных делится на две: память данных X и память данных Y. Для обращения к каждой из них используется своя шина данных: шина данных памяти X — ШДПХ и шина данных памяти Y — ШДПУ.

Операцию умножения с накоплением можно выполнить с помощью следующей команды

```
MAC X,Y,A PX->RG PY->RG ; перемножение содержимого регистров
   ; сомножителей X и Y и добавление произведения
   ; содержимому регистра результата A, перемещение
   ; к новых сомножителей из ПХ (PX) и ПУ (PY) в регистры X и Y
```

Для хранения отсчетов сигнала используется, например, память ПХ, а для хранения коэффициентов — память ПУ. Команда допускает одновременное выполнение операций умножения/накопления, двух параллельных пересылок из ПХ и ПУ и запись их в регистры X и Y. Структура модуля, осуществляющего операцию, и пути передачи данных приведены на рис. 2.16.

Таким образом, в процессорах Motorola для того, чтобы можно было произвести две выборки операндов за один такт, увеличивается количество независимых модулей памяти и количество шин для передачи данных. Процессоры имеют три банка (модуля) памяти для трех выборок за один такт и соответствующее количество шин. Проблемы с быстродействием могут воз-

[5] Следует еще раз отметить, что эти процессоры представляют второе поколение ЦПОС, они были разработаны позже процессоров TI и естественно учитывали их опыт.

никнуть в случае нехватки внутренней памяти. По внешним шинам можно осуществить только одно обращение к памяти за такт.

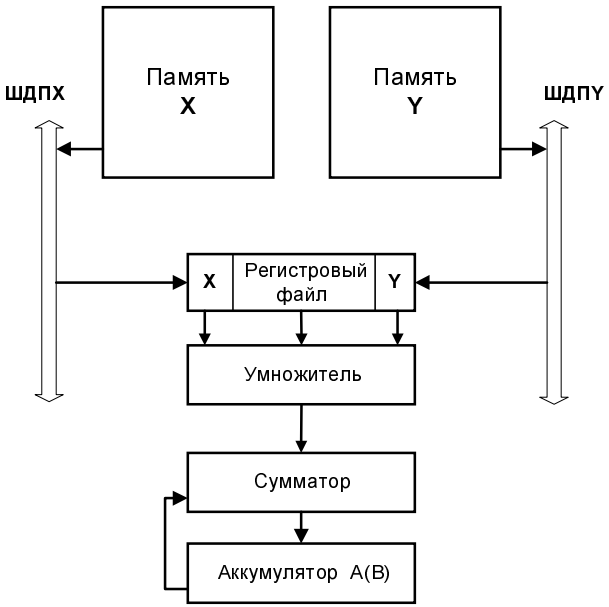


Рис. 2.16. Выполнение операции умножения с накоплением в процессорах Motorola

В некоторых процессорах, использующих фактически два независимых банка памяти данных, подобную структуру называют *двухпортовой памятью* (память с возможностью выполнения двух обращений через два порта).

Следует отметить, что в пределах стандартной архитектуры находятся ЦПОС первых двух поколений, которые отличаются производительностью. Повышение производительности достигалось за счет увеличения тактовой частоты работы, совершенствования системы команд и некоторых других изменений.

2.4.2. Улучшенные стандартные процессоры ЦПОС (Enhanced-Conventional DSP)

Методом повышения производительности (помимо обычного метода увеличения тактовых частот), который используется при разработке ЦПОС, является расширение параллелизма работы. При этом можно идти двумя основными путями:

- ❒ увеличивать количество операций, производимых одновременно;
- ❒ увеличивать количество команд, выполняемых одновременно.

Процессоры, использующие первый вариант повышения производительности, относят к *улучшенным стандартным процессорам*, а процессоры, использующие второй путь, — к *процессорам типа VLIW*.

Увеличение количества операций, производимых одновременно, в улучшенных стандартных ЦПОС достигается:

- увеличением количества дополнительных функциональных и операционных узлов и модулей (умножители, сумматоры, АЛУ и т. д.);
- увеличением количества различных специализированных устройств и специализированных сопроцессоров (декодеры Витерби, сопроцессоры для построения цифровых фильтров и т. д.);
- "расширением" шин передачи данных (увеличение ширины шин) для повышения количества передаваемой одновременно информации;
- использованием памяти с многократным доступом (памяти с возможностью выполнения нескольких обращений за один такт);
- расширением и усложнением системы команд, которые позволяют использовать дополнительные функциональные модули.

Перечисленные меры (как отмечалось в *разд. 2.2*) являются традиционными для ЦПОС и использовались, начиная с самых первых процессоров. Поэтому точно разделить процессоры на стандартные и улучшенные стандартные не всегда возможно, поскольку многие процессоры занимают промежуточное положение между ними. К улучшенным стандартным можно отнести процессоры DSP56301 (Motorola), TMS320C55x (TI), ADSP-2116x, DSP16xxx (Lucent) и некоторые другие.

Рассмотрим примеры организации вычислений в некоторых из этих процессоров.

Процессоры TMS320C55x фирмы TI

На рис. 2.17 приведена функциональная схема вычислительного модуля (модуля D) процессора фирмы TI TMS320C55x [15]. Модуль состоит из двух устройств MAC, каждый из которых включает умножитель и сумматор для накопления результатов умножения, 40-разрядного АЛУ (может выполнять две операции над 16-разрядными операндами), блока 4-х регистров — аккумуляторов и сдвигателя. Сдвигатель может осуществлять сдвиги при передаче данных между различными блоками, эти пути передачи на рис. 2.17 не показаны. Модуль работает с пятью 16-разрядными шинами данных — три шины чтения данных из памяти и две шины записи данных. Каждый модуль MAC может получать операнды по всем трем шинам чтения данных, но при параллельной работе один сомножитель у них будет общим.

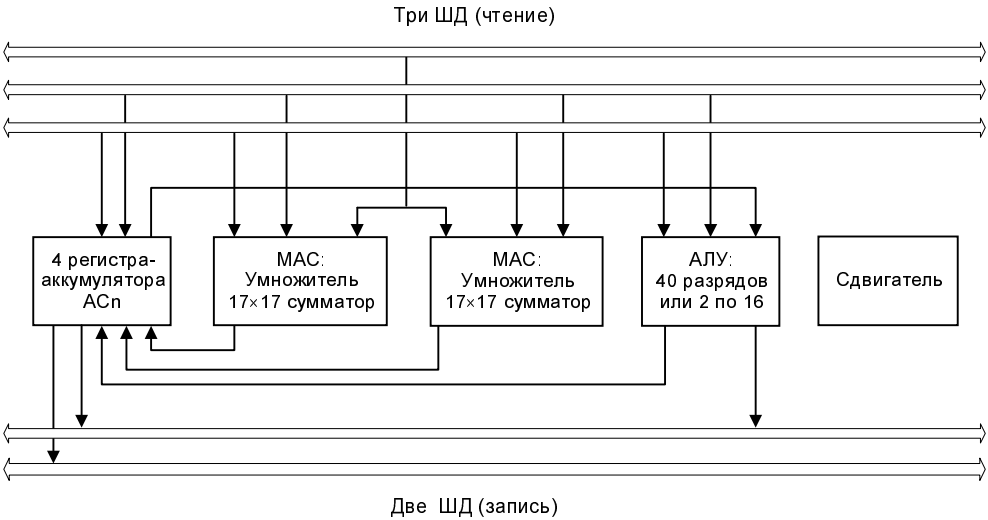


Рис. 2.17. Модуль вычислительный D процессора TMS320C55x

Рассмотрим выполнение двух параллельных команд умножения с накоплением (команды записаны условно — не по правилам языка), в которых один из сомножителей является общим:

```
MAC  dma1, dma2, AC1    ; перемножение сомножителей, выбираемых
                        ; из ячеек ПД с адресами dma1 и dma2,
                        ; произведение складывается
                        ; с содержимым аккумулятора AC1
                        ; и результат помещается туда же
|| MAC dma3, dma2, AC2  ; параллельное перемножение сомножителей,
                        ; выбираемых из ячеек ПД с адресами dma3 и dma2,
                        ; произведение складывается с содержимым
                        ; аккумулятора AC2 и результат помещается туда же
```

Действия, выполняемые приведенными двумя параллельными командами, условно можно записать также в алгебраической форме следующим образом:

```
(AC1)=(AC1)+(dma1)*(dma2)
||  (AC2)=(AC2)+(dma3)*(dma2)
```

Выражение в скобках означает содержимое соответствующего регистра или ячейки памяти.

К выполнению подобных параллельных команд с общим сомножителем можно свести многие алгоритмы [15].

- **КИХ-фильтр с симметричными коэффициентами**. На раздельные шины (отличные сомножители) подаются различные наборы отсчетов сигнала, в качестве общих сомножителей используется одинаковый набор коэффициентов, при этом параллельно рассчитываются половинки фильтра, которые затем суммируются.
- **Любая многоканальная обработка**. В качестве общих сомножителей используются отсчеты входного сигнала, в качестве раздельных коэффициентов, подаваемых по раздельным шинам, — два набора коэффициентов. Результаты накапливаются в аккумуляторе. При наличии запаса по производительности можно одновременно обрабатывать 4 канала, сохраняя результат в 4-х аккумуляторах.

Таким образом, время вычисления реакции КИХ-фильтра n-го порядка будет равно $n/2$ тактов.

Процессоры DSP16xxx фирмы Lucent

Процессор DSP16410 имеет внутри кристалла два ядра. Упрощенная функциональная схема вычислительного модуля ядра приведена на рис. 2.18.

Процессор является 16-разрядным с фиксированной точкой. Каждое ядро имеет локальную память с двумя портами X и Y, т. е. две раздельные памяти данных X и Y. За один такт через две 32-разрядные шины можно осуществлять 2 выборки двойных (32-разрядных) слов, два умножения и два накопления результатов умножения. За один такт вычислительный модуль может выполнить действия, отображенные следующей С-подобной командой (см. "DSP Processor overviews" на сайте **www.bdti.com**):

```
a0=a4+p0    a1=a5+p1    p0=xh*yl    p1=xl*yh    x=*pt0++    y=*r1++
```

Эти действия заключаются в следующем. В АЛУ и сумматоре/вычитателе производятся сложения содержимого регистров аккумуляторов An и регистров произведения p0 и p1:

```
a0=a4+p0    a1=a5+p1
```

Выбранные мультиплексором из памяти 32-разрядные двойные слова делятся распределителем каждое на две части `xh`, `xl` и `yh`, `yl` соответственно, полученные четыре слова передаются в умножители в качестве сомножителей, произведения записываются в регистры p0 и p1 в качестве нового содержимого:

```
p0=xh*yl    p1=xl*yh
```

Подготавливаются адреса для выбора в следующем такте новых операндов из памяти:

```
x=*pt0++    y=*r1++
```

В процессоре DSP16xxx реализуется метод, который называют SIMD (Single Instruction Multiple Data), в соответствии с которым одной командой выпол-

няются действия над различными данными (данные расщепляются для разных действий).

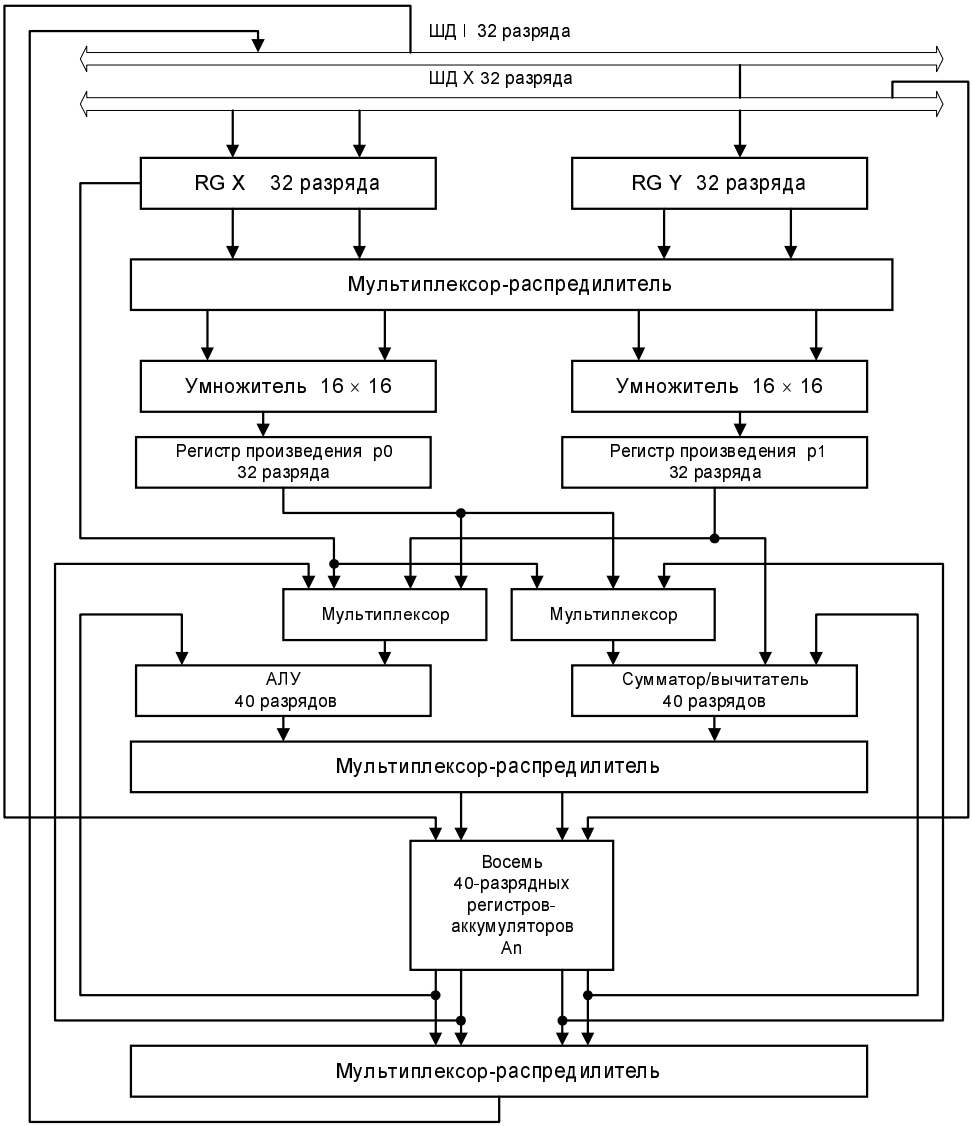


Рис. 2.18. Вычислительный блок ядра процессора DSP16xxx

Процессор может выполнить одновременно два умножения с накоплением без ограничения на наличие общих сомножителей (как в TMS320C55x).

Процессор ADSP-2116x фирмы ADI

Метод обработки данных SIMD (Single Instruction Multiple Data) в чистом виде реализуется в процессоре ADSP-2116x фирмы ADI. Функциональная схема вычислительного узла этого процессора приведена на рис. 2.19.

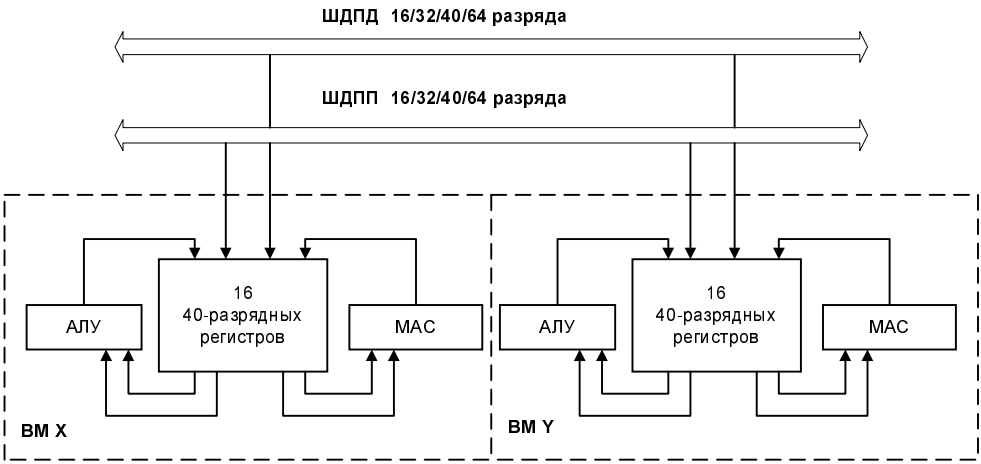


Рис. 2.19. Функциональная схема вычислительного узла процессора ADSP-2116x

Вычислительный узел процессора состоит из двух полностью идентичных вычислительных модулей BM X и BM Y, которые идентичны вычислительному модулю процессора ADSP-2106x (эти процессоры совместимы на уровне команд). Процессор может работать в режимах SISD (Single Instruction Single Data) и SIMD. В режиме SISD используется только модуль BM X, в таком режиме работают процессоры ADSP-2106x. В режиме SIMD применяются оба вычислительных модуля, и они в этом случае выполняют одну и ту же команду, однако данные, поступающие в модули по шинам данных в сопоставимые регистры, различны. Данные выбираются из памяти двойными словами и передаются по шинам с удвоенной разрядностью (64 разряда). Таким образом, процессор поддерживает обмен регистров вычислительных модулей одинарными словами по 32 разряда и двойными словами по 64 разряда.

Процессор использует различные форматы данных:

- с плавающей точкой одинарной точности (32 разряда);
- с плавающей точкой увеличенной точности (40 разрядов);
- с плавающей точкой с коротким словом (16 разрядов);
- с фиксированной точкой (16 разрядов);
- с фиксированной точкой с двойным словом (32 разряда).

В зависимости от формата данных и режима работы по шинам данных передаются операнды различной длины — 16/32/40/64 разряда. При этом процессор использует достаточно сложную организацию памяти и распределения данных в ней. Внутренняя память процессора поддерживает двойной доступ за один такт при пересылках "память ↔ регистр". В режиме SIMD из внутренней ПП и ПД можно прочитать два слова по 64 разряда.

Вычислительный модуль состоит из регистрового файла, имеющего шестнадцать 40-разрядных регистров, АЛУ, сдвигателя (не показанного на рис. 2.19) и устройства MAC. Модуль MAC включает умножитель, сумматор/вычитатель (для чисел с фиксированной точкой) и регистр результата MR. Он может выполнять операции умножения чисел с плавающей точкой, операции умножения/накопления чисел с фиксированной точкой в различных форматах. В модулях ВМ могут выполняться многофункциональные и параллельные операции с умножением (в MAC), операциями в АЛУ и передачей данных "память ↔ регистры", "регистр MR ↔ регистры".

При использовании представления с ФТ в каждом вычислительном модуле можно выполнить за один такт операции типа

```
MR=MR+Rx*Ry  dm -> Rx  pm -> Ry
```

где `Rx`, `Ry` — регистры регистрового файла.

Параллельно с выполнением операции умножения с накоплением производится перемещение новых данных из памяти данных `dm` и памяти программ `pm`.

В режиме SIMD подобная операция производится одновременно в двух модулях, и число операций удваивается (по сравнению, например, с ADSP-2106x). При этом из памяти выбираются двойные слова и передаются в оба модуля по шинам данных "шириной" в 64 разряда.

Выводы

Таким образом, во всех процессорах улучшенной стандартной архитектуры за счет удвоения количества вычислительных модулей и количества (или увеличения разрядности) шин передачи данных достигается увеличение количества одновременно выполняемых операций и увеличение производительности процессора. Все они могут производить до двух операций умножения с накоплением за один такт. Однако при этом существенно усложняется архитектура и система команд процессора за счет появления многофункциональных и комбинированных команд. Для того чтобы писать эффективные программы на ассемблере, необходимо "хорошее" знание архитектуры и системы команд процессора. Это в условиях усложнения системы становится более затруднительным. С другой стороны подобная архитектура является недружественной для компиляторов языков высокого уровня (языка С). Как уже отмечалось выше, компиляторы эффективно используют только простые команды, характерные для архитектур типа RISC (см. "Compiler-friendly Architectures for DSP" на сайте **www.zsp.com**, [21]).

2.4.3. Процессоры ЦПОС с архитектурой VLIW

Как уже отмечалось, возможным способом повышения производительности является увеличение количества команд, выполняемых одновременно. Подобный метод реализован в процессорах с архитектурой VLIW (Very Long Instructions Word, очень длинное слово команды). Этот способ построения ЦПОС называют также Multi-Issue Architectures (многократные команды в параллельном режиме)[3].

Подобные процессоры используют упрощенную систему команд (архитектура RISC [21]), каждая из которых определяет единственную операцию. Несколько простых команд выполняются параллельно (одновременно) в независимых операционных модулях. Общая команда процессора формируется как большая суперкоманда — набор (пакет) команд для отдельных модулей и, соответственно, имеет большую длину. Архитектура предполагает использование регистровых файлов большого размера для хранения операндов и результатов работы всех операционных модулей. При этом применяются команды типа "регистр, регистр → регистр", "память → регистр" "регистр → память". Команда типа "регистр, регистр → регистр" означает, что источником двух операндов являются регистры, и результат операции помещается в регистр. Длинные слова предполагают также существование многоразрядных шин передачи данных и слов команды.

Использование простой системы команд позволяет разрабатывать эффективные компиляторы программ на языке С и эффективные оптимизаторы *(см. разд. 9.6)*. В результате можно получить исполняемые программы высокого качества при использовании для написания исходных программ языка высокого уровня.

К недостаткам ЦПОС с архитектурой VLIW следует отнести большие объемы памяти, требуемой для записи программы с длинными командами, и нерациональное использование этой памяти.

Рассмотрим реализацию данной архитектуры на примере конкретных процессоров.

Процессоры TMS320C6xxx

Фирма TI в 1996 г. первой вывела на рынок процессор с архитектурой VLIW TMS320C62xx, предложив для архитектуры процессоров этого семейства термин VelosiTI.

Упрощенная функциональная схема операционных модулей ЦПУ процессоров TMS320C6xxx приведена на рис. 2.20.

ЦПУ имеет восемь операционных модулей L, S, M, D, разбитых на две идентичные группы — 1 и 2. Модули выполняют команды типа "регистр, регистр → → регистр". Источниками операндов и получателями результатов являются два набора 32-разрядных регистров A и B соответственно для операционных модулей группы 1 и 2. Однако возможно использование данных регистров B

модулями группы 1 и наоборот. Передача данных между группами отражена на рис. 2.20 пунктирными линиями. Все модули ориентированы на выполнение определенных операций, ниже перечислены некоторые из них.

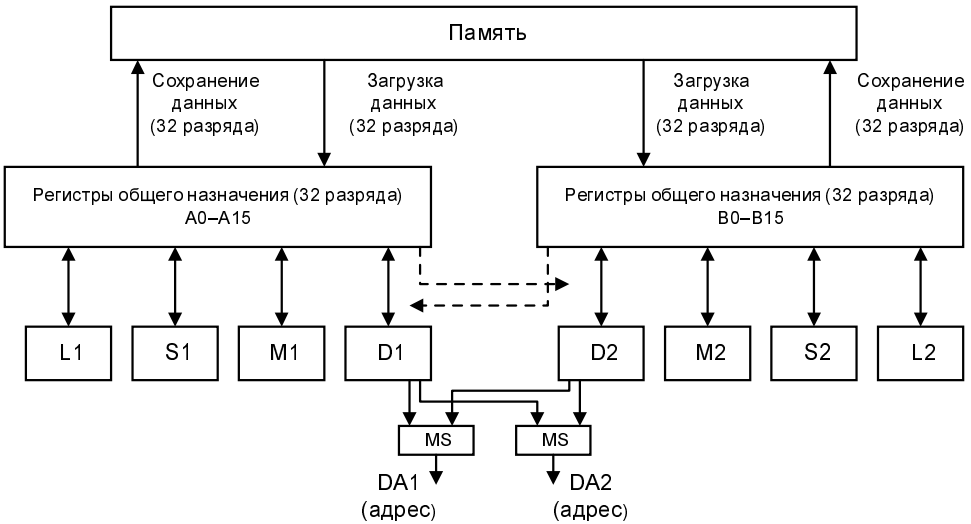


Рис. 2.20. Функциональная схема операционных модулей процессоров TMS320C6xxx

Модули L (L1, L2):

- 32/40-разрядные арифметические операции и операции сравнения;
- 32-разрядные логические операции;
- операции нормализации.

Модули S (S1, S2):

- 32-разрядные арифметические операции;
- 32/40-разрядные операции сдвига и операции с отдельными битами;
- 32-разрядные логические операции;
- генерация констант.

Модули M (M1, M2):

- операции умножения 32×32 с фиксированной точкой;
- операции умножения с плавающей точкой.

Модули D (D1, D2): 32-разрядные операции по вычислению адресов, в том числе адресов циклических и линейных буферов. Модули D1, D2, предназначенные для вычисления адресов, имеют выход на шины адресов.

ЦПУ процессора имеет две 32-разрядные шины для записи данных из регистров в память и две 32-разрядные шины для чтения из памяти, которые позволяют осуществлять по две операции чтения и записи данных за такт.

Для вычисления 16-разрядных отсчетов на выходе КИХ-фильтра процессор в цикле может параллельно выполнять 8 команд вида:

```
LOOP:
    ADD      .L1 A0,A3,A0          ; A0+A3 -> A0
    || ADD   .L2 B1,B7,B1          ; B1+B7 -> B1
    || MPYHL .M1X A2,B2,A3         ; A2(H)*B2(L) -> A3
    || MPYLH .M2X A2,B2,B7         ; A2(L)*B2(H) -> B7
    || LDW   .D2 *B4++,B2          ; DataMemory -> B2
    || LDW   .D1 *A7--,A2          ; DataMemory -> A2
    || [B0]   ADD .S2 -1,B0,B0     ; B0-1 -> B0
    || [BO]   B .S1 LOOP           ; условный переход
```

Данными командами за один такт в процессоре будут выполняться следующие действия:

- вычисляться два произведения 16-разрядных чисел, результаты в регистрах A3, B7;
- сомножители для произведений формируются из старших A2(H), B2(H) и младших B2(L), A2(L) слов содержимого регистров A2, B2;
- результаты перемножения будут накапливаться в регистрах A0, B1;
- пересылки из памяти DataMemory новых сомножителей в регистры B2(L), A2(L), пересылки выполняются 32-разрядными словами.

Как видно из приведенного примера, структура команд на языке ассемблера достаточно сложна. В каждой команде необходимо:

- указать операционный модуль, в котором будет выполняться операция (L1, L2 и т. д.);
- задать необязательное условие выполнения команды (содержимое регистра [B0] в последней команде);
- предусмотреть правильную последовательность операций по перемещению данных

и т. д.

Все перечисленное требует хорошего знания системы команд и архитектуры процессора для получения эффективной программы.

Для облегчения программирования ассемблер позволяет определять только функциональный тип (например, M), при этом конкретный модуль (например, M2) назначается на этапе трансляции. Если не будет вообще определен функциональный модуль, ассемблер назначит его на основании содержимого поля мнемоники команды и поля операнда.

Можно писать программу для процессоров TMS320C6xxx, используя так называемый линейный ассемблер; при этом нет необходимости указывать следующую информацию:

- команды, выполняемые параллельно;
- используемые регистры;
- используемые функциональные модули.

Для получения исполняемой программы (с определенными параллельными командами, регистрами, модулями и т. д.) применяется оптимизатор.

Семейство TMS320C6xxx состоит из трех процессоров: TMS320C62xx, TMS320C64xx с фиксированной точкой и TMS320C67xx с плавающей точкой. Процессоры совместимы на уровне кодов и используют общую систему команд. Все команды, допустимые для C62xx, также допустимы для C64xx и C67xx. Однако, поскольку C67xx является устройством с плавающей точкой, существуют некоторые команды, которые являются уникальными для него и не выполняются в устройстве с фиксированной точкой.

ЦПОС C64xx имеет некоторые расширенные функциональные возможности по сравнению с C62xx и некоторые дополнительные команды. Дополнительные возможности процессора C64xx включают:

- 64 регистра общего назначения;
- поддержку упакованных типов данных по 8 битов и данных 64 бита;
- дополнительные шины, обеспечивающие загрузку из памяти и запись в память до 4-х 32-разрядных слов за такт.

Процессор MSC810x (ядро SC140)

Ядро ЦПОС SC140 является совместной разработкой фирм Lucent Technologies и Motorola и используется в разных процессорах. Ядро может работать с операндами длиной 16 и 32 разряда. Оно включает несколько функциональных модулей, в которых могут выполняться одновременно различные операции. В состав этих модулей входят 4 блока АЛУ данных (DALU), 2 устройства генерации адреса УГА (AGU) с устройствами вычисления адреса и работы с отдельными разрядами. Модули содержат также большие регистровые файлы, являющиеся источниками и приемниками операндов при выполнении команд. Модули могут одновременно выполнять до шести различных простых команд типа RISC.

Для выполнения несколько команд группируются вместе, образуя общую команду (набор выполнения), которая содержит до шести элементарных команд с общей длиной до восьми слов по 16 разрядов. Разработчики назвали такую модель команд VLES (Variable Length Execution Set, команды переменной длины). Команды набора выполняются за один такт.

Функциональная схема блока обработки данных SC140 приведена на рис. 2.21. Блок состоит из четырех идентичных АЛУ, содержимое одного из которых раскрыто на указанной схеме. Все АЛУ используют в качестве источников и приемников операндов 16 регистров D0—D15 общего регистрового файла. Эти регистры связаны с памятью данных двумя 64-разрядными шинами ШДПД A и ШДПД B, по которым могут передаваться операнды длиной 8, 16, 32 разряда. Передача данных по шинам способна производиться через 8 сдвигателей, причем одновременно до 4-х 32-разрядных операндов за такт.

АЛУ состоит из умножителя, способного выполнять операции типа (16×16) + + 40 → 40-бит, сумматора и блока операций с отдельными разрядами операнда. Связи последнего на рис. 2.21 не показаны.

В блоке АЛУ может обрабатываться одновременно следующий набор команд:

```
MAC D0, D1, D7 MAC D3, D4, D6 MACR D0, D2, D5
ADR D3, D4 MOVE .L (R0)+N3, D2 MOVE .L D0, R
```

В соответствии с этими командами будут произведены действия:

1. Содержимое регистров `D0` и `D1` перемножено и произведение добавлено к содержимому регистра `D7`, результат будет сохранен в регистре `D7`.
2. Содержимое регистров `D3` и `D4` перемножено и произведение добавлено к содержимому регистра `D6`, результат будет сохранен в регистре `D6`.
3. Содержимое регистров `D0` и `D5` перемножено и произведение добавлено к содержимому регистра `D5`, результат с округлением будет сохранен в регистре `D5`.
4. Содержание регистров `D3` и `D4` прибавлено друг к другу, результат с округлением сохранен в регистре `D4`.
5. Содержание ячейки памяти данных, указанной регистром `R0`, перемещено в регистр `D2`.
6. Для изменения адреса операнда содержимое регистра `N3` добавлено к содержимому регистра `R0`, результат сохранен в регистре `R0`.
7. Содержимое регистра `D0` перемещено в регистр `R1`.

Последние три операции, соответствующие двум последним командам, будут выполнены в непоказанном на рис. 2.21 устройстве генерации адреса.

Группировка отдельных команд в набор производится ассемблером по определенным правилам при трансляции программы. При этом в общее командное слово включается информация о способе группирования и общей длине. Возможны два способа группирования. Например, при одном к коду каждой команды добавляется префикс, несущий информацию о том, является ли данная команда последней в группе. Ассемблер выбирает способ группировки и количество команд в группе, исходя из последовательности команд в

программе. При этом могут возникнуть недопустимые последовательности, о чем ассемблер выдаст сообщение. Таким образом, общая команда может иметь различную длину в зависимости от количества объединенных простых команд. Этим обеспечивается лучшее использование памяти программ по сравнению с процессором TMS320C6xxx.

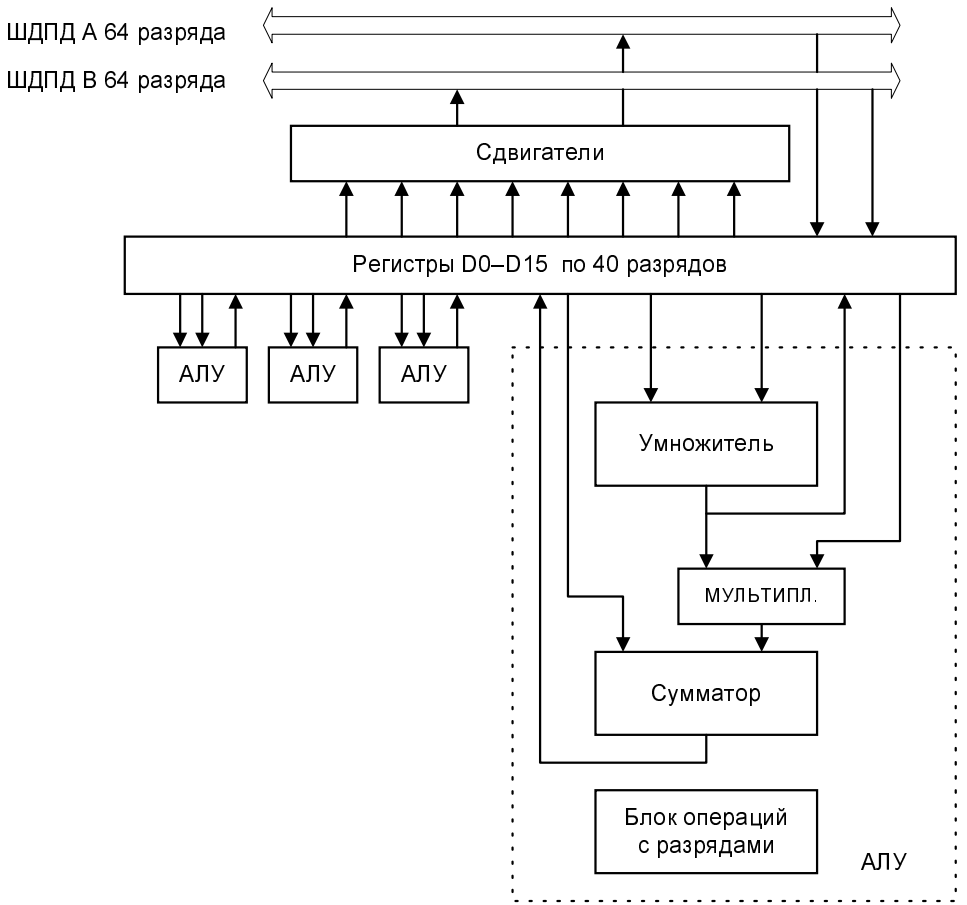


Рис. 2.21. Блок обработки данных и структурная схема АЛУ данных (DALU) ядра SC140

Конвейер SC140 состоит из пяти этапов:

- этап предвыборки общей команды;
- этап·выборки общей команды;
- этап·расщепления общей команды на простые;

- этап генерации адресов операнда;
- этап выполнения.

Первые три этапа выполняются в блоке управления программой, последние два — в модулях УГА и АЛУ данных (DALU) соответственно.

Выборки команд из памяти производятся по шине данных памяти программы "шириной" 128 битов. Память системы строится как общая память с ячейками в один байт, при этом ПП и ПД отличаются лишь адресами. При каждом доступе к памяти устройство управления посылает сигналы о "ширине" доступа — размере считываемых данных.

2.4.4. Суперскалярные процессоры

Суперскалярные процессоры отличаются от процессоров типа VLIW в двух связанных между собой моментах.

Команды суперскалярного процессора, предназначенные для выполнения в отдельных операционных модулях, не объединяются в одну общую суперкоманду, а выступают самостоятельно.

Процессор имеет модуль, который определяет, какие из команд могут быть выполнены параллельно, и группирует их в пакет. Правила группировки основаны на зависимостях данных, используемых в командах последовательно, и ресурсов процессора.

Таким образом, если в процессорах VLIW планирование параллельно выполняемых команд возлагается на программиста, то в суперскалярных процессорах эту задачу решает сам процессор.

На рис. 2.22 приведена функциональная схема основных элементов суперскалярного ЦПОС LSI40xZ (ZSP164xx) фирмы LSI Logic Corporation. Данный процессор является 16-разрядным, использующим представление данных с фиксированной точкой. Тактовая частота работы 200 МГц.

Модуль команд содержит узел предвыборки команд, кэш команд и узел отправки команд для выполнения. В течение каждого такта из памяти выбираются четыре команды и посылаются в кэш. При этом используются предсказания переходов и другие методы, уменьшающие конфликты и задержки конвейера выполнения команд. Если команда размещена в кэше, например, при выполнении цикла, для экономии мощности повторно из основной памяти она не выбирается.

Модуль данных содержит узлы предвыборки данных, кэш данных и систему управления/арбитража загрузки памяти. Модуль имеет возможность читать четыре слова данных за такт и записывать два слова в память, а также обладает аппаратными средствами для организации двух циклических буферов.

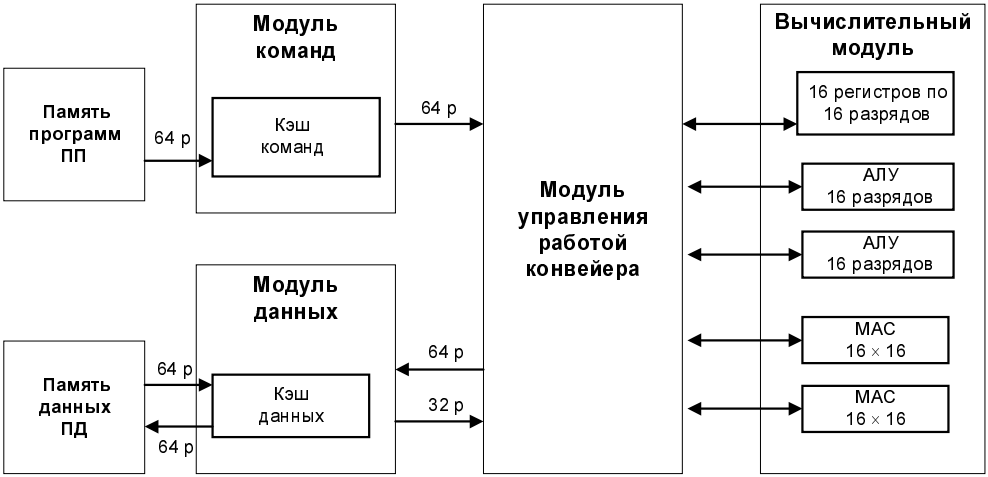


Рис. 2.22. Функциональная схема основных модулей процессора LSI40xZ

Вычислительный модуль содержит 4 функциональных узла: два узла АЛУ для выполнения арифметических и логических операций, два узла MAC умножения и накопления и набор регистров. Два АЛУ работают независимо или могут быть объединены в один узел для выполнения операций над 32-разрядными операндами. Узлы MAC способны выполнять две операции умножения 16×16 с накоплением или одну операцию умножения 32×32 разряда с накоплением 40-разрядного результата. При этом они используют общий накопитель и, соответственно, возможна только одна операция накопления за такт. Источником операндов и получателем результатов работы функциональных узлов являются регистры модуля. Обмен операндами между ними, а также памятью производится отдельными командами. Таким образом, в процессоре реализуется система команд RISC типа "регистр, регистр → регистр" и "регистр ↔ память".

Самым интересным модулем процессора, который прежде всего и определяет его особенности, является модуль управления работой конвейера. Этот модуль выбирает из последовательности команд четыре команды, которые могут выполняться параллельно, исходя из необходимого чередования данных, ресурсов вычислительного модуля и максимальной его загрузки. Тем самым программист или компилятор освобождаются от задачи группировки команд (при помощи языка C).

Процессор использует конвейер на пять этапов:

- ❒ выборка/декодирование команды (F/D);
- ❒ группирование команд (G);
- ❒ чтение (R);

- выполнение (E);
- записи (W).

Во время этапа (F/D) процессор выбирает команды из памяти и декодирует их.

На этапе (G) проверяются зависимости между командами, и они группируются для возможного параллельного выполнения. Группировка команд происходит в соответствии с набором определенных правил, например, не включается в группу любая команда, использующая АЛУ или MAC, если имеется более ранняя команда на этапе (G) или (R), которая производит операции с регистрами управления режимами работы функциональных модулей АЛУ или MAC.

На этапе (R) операнды считываются из модуля данных.

На этапе (E) команды выполняются, и результаты заносятся в регистры вычислительного модуля.

На этапе (W) модуль данных записывает необходимые данные в память.

Таким образом, выполнение команд происходит примерно следующим образом:

- модуль управления работой конвейера уведомляет модуль команд о том, какие четыре команды необходимы для следующей группы;
- модуль данных считывает из памяти данные (4 по 16 или 2 по 32 разряда) и посылает данные в вычислительный модуль;
- вычислительный модуль выполняет команды и записывает результаты в свой регистровый файл или посылает результаты в модуль данных для записи в память;
- необходимые данные переписываются из модуля данных в память.

В конвейере могут возникать конфликты, например при следующих ситуациях:

- предвыборка команд требует дополнительных тактов для заполнения кэша команд;
- предвыборка данных требует дополнительных тактов.

Модуль управления конвейером обрабатывает также запросы на прерывания.

Суперскалярные процессоры, подобные описанному, группируют команды, основываясь на зависимостях данных. Поэтому один и тот же набор команд может по-разному выполняться на различных этапах выполнения программы, например, на этапах цикла. Поэтому вопрос о расчете и предсказании времени выполнения программы, что очень важно для систем, работающих в реальном масштабе времени, является неопределенным. При расчете на худший случай потенциальные возможности процессора будут использованы не полностью.

2.4.5. Гибридные процессоры

Класс микропроцессоров, которые называются *микроконтроллерами*, ориентирован на управление объектами в реальном масштабе времени. Это наиболее широкий класс микропроцессоров, обладающий наибольшей специализацией, разнообразием функций и параметров, разнообразием различных периферийных устройств. Вычислительные требования, предъявляемые к микроконтроллерам, зачастую достаточно скромны. Микроконтроллеры широко применяют в качестве встроенных элементов в различные приборы.

Задачи управления некоторыми реальными объектами, например двигателями, требуют обработки сигналов и применения цифровых фильтров в цепях управления. Большие семейства ЦПОС различных фирм ориентированы на реализацию задач именно управления электроприводами. Примерами могут служить семейства TMS320C24xx фирмы TI, в основе которого лежит ядро ЦПОС C2000, встраиваемые контроллеры семейства ADMC3xx фирмы ADI, в основе которого лежит 16-разрядное ядро ADSP-2171, и некоторые другие.

Существует много задач, которые требуют сочетания возможностей решения классических задач ЦОС и задач управления. Примерами могут служить приборы мобильной телефонии, где требуется обработка голосовых сигналов и управление клавиатурой, дисплеем и т. д., задачи построения интеллектуальных кассовых аппаратов. Классические микроконтроллеры хорошо решают задачи управления и мало эффективны в задачах ЦОС и соответственно наоборот. Поэтому одним из методов решения подобных задач до недавнего времени было использование двух отдельных процессоров. В последнее время появились гибридные процессоры, объединяющие в одном кристалле возможности микроконтроллера и ЦПОС.

Рассмотрим в качестве примеров два подобных гибридных процессора.

Процессор DSP5665x фирмы Motorola

На рис. 2.23 приведена упрощенная функциональная схема процессора DSP5665x фирмы Motorola. Процессор содержит два независимых ядра — ядро микроконтроллера и ядро ЦПОС DSP56600. Каждое ядро обладает своей памятью данных и программ и работает под управлением собственной программы. Основные свойства микроконтроллера (МК):

- архитектура типа RISC;
- 32-разрядные операции типа "регистр, регистр → регистр", использующие 16 32-разрядных регистров общего назначения;
- 16-разрядные команды одинакового формата;
- 4-этапный конвейер команд;
- время выполнения большинства команд — один такт;

- время выполнения команд переходов и обращения к памяти — 2 такта;
- возможность работы с данными в 8/16/32 разряда.

Периферийные устройства МК включают в себя:

- сторожевой таймер;
- программируемый таймер прерываний;
- порт вспомогательной клавиатуры;
- генератор сигналов ШИМ;
- универсальный асинхронный приемопередатчик UART;
- последовательный аудиокодек и некоторые другие элементы.

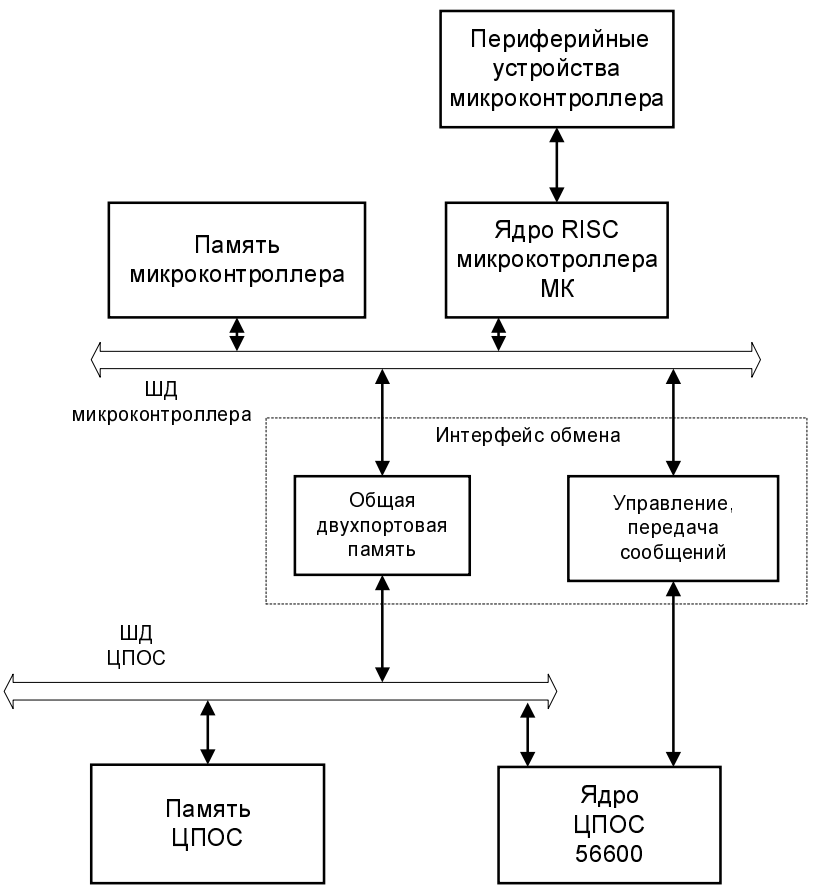


Рис. 2.23. Функциональная схема процессора DSP5665x

Взаимодействие ЦПОС и МК производится через интерфейс обмена. Он содержит общую память, включенную в адресное пространство и ЦПОС и МК, и блоки взаимного управления режимами и передачи сообщений. Протоколы сообщений могут устанавливаться программным образом. Передача сообщений может производиться через общую память, а также через группу регистров специального назначения. Эти регистры симметричны в двух ядрах и доступны на одной стороне для записи, а на другой — только для чтения.

МК способно вызывать любые прерывания ЦПОС, в том числе и сброс. Каждое ядро может вызвать другое из режима ожидания. Все это позволяет организовать взаимодействие и совместную работу двух устройств.

Ядро TMS320c27x фирмы TI

Данное ядро относится к классу cDSP процессоров (customizable Digital Signal Processors) — цифровых сигнальных процессоров с перестраиваемой конфигурацией или настраиваемых процессоров. Этот и подобные процессоры обеспечивают простой и эффективный метод сокращения разработки необходимой системы, при котором объединяется ЦПОС со схемами заказной логики или ASIS (Application Specific Integrated Circuit, проблемно-ориентированными интегральными схемами). C27x предназначен для реализации встраиваемых систем, содержащих ЦПОС и процессоры общего назначения, подобные микроконтроллерам. Примерами таких систем могут служить компьютерные периферийные устройства типа быстродействующих жестких дисков, игровые цифровые видеодиски и другие устройства хранения информации, роботоподобные индустриальные системы, бытовая электроника. В таких системах, интегрируя два процессора, можно уменьшить размер и стоимость изделия и повысить его эффективность.

По составу внутренних модулей C27x представляет дешевый 16-разрядный процессор обработки сигнала, однако по принципу работы и набору используемых команд он отличается от стандартных ЦПОС. Некоторые из отличий таковы:

- процессор объединяет несколько способов адресации, характерных для ЦПОС и МК;
- процессор использует систему команд RISC типа "регистр → регистр";
- добавлен новый класс операций RMW (Read-Modify-Write), которые позволяют действия типа `ADD`, `SUB`, `INC`, `DEC`, `AND`, `OR`, `XOR` выполнять непосредственно над содержимым памяти без переноса в регистры; это дает возможность рассматривать память как некий "виртуальный" регистр;
- добавлен способ адресации, называемый индексацией стека, позволяющий читать и записывать слова в любое место стека, что улучшает возможности компилятора при обработке функций.

2.5. Влияние архитектуры на возможности процессора

В качестве характеристики влияния особенностей архитектуры ЦПОС на его возможности и параметры в табл. 2.3—2.5 приведены сравнительные данные по реализации КИХ-фильтра при использовании различных процессоров. Источником для характеристик послужила работа [3].

Таблица 2.3. *Количество тактов, требуемое для реализации КИХ-фильтра*

Вид данных	Тип архитектуры	Процессор	Фирма	Кол-во тактов
ФТ 16р	Стандартная	C54	TI	730
ФТ 24р	Стандартная	DSP563xx	Motorola	943
ФТ 16р	Стандартная	DSP16xx	Lucent	1264
ФТ 16р	Улучшенная стандартная	DSP16xxx	Lucent	757
ФТ 16р	VLIW	SC140	Motorola/LSI	183
ФТ 16/32р	VLIW	C62xx	TI	347
ФТ 16р	Суперскалярная	LCI400	LSI	607
ПТ 32р	Стандартная	C3x	TI	1050
ПТ 32р	VLIW	C67x	TI	500
ПТ 32р	Улучшенная стандартная	ADSP-2116x	ADI	573
ПТ 32р	Стандартная	ADSP-2106x	ADI	812
ПТ	—	Pentium III	Intel	1498

Таблица 2.4. *Время вычисления отсчета КИХ-фильтра*

Вид данных	Тип архитектуры	Процессор	Частота, МГц	MIPS	Фирма	mc
ФТ 16р	VLIW	SC140	300	До 1800	Motorola/LSI	0,6
ФТ 16/32р	VLIW	C6202	250	До 2000	TI	1,4
ФТ 16р	Суперскалярная	LCI400	200	800	LSI	3,0
ФТ 24р	Стандартная	DSP56311	150	150	Motorola	6,3
ФТ 16р	Стандартная	C549	120	120	TI	6,1
ФТ 16р	Стандартная	DSP1620	120	120	Lucent	10,5

Таблица 2.4 *(окончание)*

Вид данных	Тип архитектуры	Процессор	Частота, МГц	MIPS	Фирма	mc
ФТ 16р	Улучшенная стандартная	DSP16219B	120	120	Lucent	6,3
ПТ	—	Pentium III	550	—	Intel	2,7
ПТ 32р	VLIW	C67x	167	TI	TI	2,5
ПТ 32р	Улучшенная стандартная	ADSP-2116	100	100	ADI	5,7
ПТ 32р	Стандартная	ADSP-21065	60	60	ADI	13,5

Таблица 2.5. *Объем использованной памяти для реализации КИХ-фильтра*

Тип данных	Тип архитектуры	Процессор	Фирма	К-во байт
ФТ 16р	Стандартная	C54	TI	54
ФТ 16р	Стандартная	DSP16xx	Lucent	66
ФТ 24р	Стандартная	DSP563xx	Motorola	81
ФТ 16р	Улучшенная стандартная	DSP16xxx	Lucent	94
ФТ 16р	Суперскалярная	LCI400	LSI	100
ФТ 16р	VLIW	SC140	Motorola/LSI	250
ФТ 16/32р	VLIW	C62xx	TI	540
ПТ 32р	Стандартная	ADSP-2106x	ADI	120
ПТ 32р	Улучшенная стандартная	ADSP-2116x	ADI	138
ПТ	—	Pentium III	Intel	179

2.6. Организация памяти ЦПОС

В следующих разделах будут отмечены некоторые особенности использования памяти в ЦПОС.

2.6.1. Доступ к блокам памяти. Блоки памяти

Как было сказано в *разд. 2.2*, организация быстрых вычислений при выполнении алгоритмов ЦОС требует архитектуры процессора с несколькими блоками памяти и несколькими комплектами шин. Для выполнения основ-

ной операции умножения с накоплением MAC требуется три обращения к памяти для выборки команды и двух сомножителей (при сохранении результата в каком-нибудь регистре).

В чистом виде на это ориентирована архитектура процессоров фирмы Motorola (и некоторых других), в которых выделяется память программ и память данных, которая в свою очередь разделена на две части — X и Y *(см. разд. 2.3)*. В процессорах фирм ADI и TI для хранения одного из сомножителей (например, коэффициентов фильтров) используется память программ. В конечном итоге, для организации эффективных вычислений определяющим является не название и количество реально независимых блоков памяти, а возможность организации необходимого для этих вычислений количества доступов к памяти (при которых производится чтение команд и операндов и запись результатов).

В некоторых последних процессорах (и в некоторых более ранних) разработчики, судя по всему, вообще стали отказываться от разделения на память программ и память данных.

Процессоры с ПТ семейства ADSP-21000 фирмы ADI имеют два блока памяти, которые могут быть сконфигурированы для сохранения различных комбинаций кодов и данных. Набор соответствующих шин и устройств генерации адреса УГА позволяют ядру процессора одновременно обращаться к командам и данным от обоих блоков памяти. Для максимального обращения без циклов ожидания необходимо соблюдение определенных условий, оговариваемых в описаниях процессоров. Процессор TigerSHARC имеет три блока внутренней памяти M0, M1, M2, каждый из которых содержит свой набор шин; распределение данных между ними указывается пользователем.

Ядро SC140 для процессоров, которые его используют (в частности, MSC8101 Motorola), определяет память как единое пространство без различия в расположении памяти программ и памяти данных. Общая память системы конфигурируется как память, состоящая из нескольких блоков с общим адресным пространством. Данные адресуются ячейками в 1 байт и доступны по трем шинам: шина кодов команд и две шины данных *(см. разд. 2.4.3)*. При правильном расположении данных может быть осуществлен доступ по всем трем шинам за один такт.

Процессоры семейства TMS320C3x фирмы TI, начиная с первого TMS320C30, имеют общее адресное пространство памяти. Распределение блоков внутренней памяти между различными данными производится пользователем, доступ к этим блокам осуществляется по нескольким шинам.

Сравнительно сложную организацию памяти имеют процессоры семейства C6000 фирмы TI. Процессоры TMS320C620x и TMS320C640x обладают выделенными внутренними памятью программ и памятью данных. Память данных включает два блока. Внутренняя память программ может работать как обычная адресуемая память программ и как кэш программ *(см. разд. 2.6.3)*.

Кэш в свою очередь имеет несколько режимов работы. Обычный режим — кэш программ между ЦПУ и внешней памятью. В этом случае происходит следующее: чтение по заданному адресу производится из кэша; если в нем нужной информации нет, то выполняется чтение данных из внешней памяти, запись ее в кэш и передача в ЦПУ. При повторном обращении к тем же командам, чтение осуществляется из кэша.

Процессоры TMS320C621x, TMS320C671x и TMS320C64xx имеют архитектуру памяти, называемую производителями L1/L2 Memory Architecture. Упрощенная функциональная схема памяти приведена на рис. 2.24. Внутренняя память имеет кэш двух уровней. *Первый уровень кэша L1* образуется из отдельных частей для программ и данных. Указанный кэш управляется двумя контроллерами — для кэша программ и для кэша данных. ЦПУ связано с кэшем программ через шины PD (program data) и PA (program address). Связь ЦПУ с кэшем данных происходит через два комплекта шин (для регистров общего назначения A и B соответственно, *см. разд. 2.4.3*) DA (data address), ST (store data) и LD (load data). Общий для данных и кодов программ кэш второго уровня управляется контроллером *кэша L2*. Он связан с внешней памятью процессора через расширенный контроллер DMA (прямой доступ к памяти) и интерфейс внешней памяти. Контроллер DMA позволяет загружать в кэш L2 данные через различные периферийные устройства, в том числе последовательные порты ввода/вывода.

Следует отметить некоторые особенности организации внутренней памяти процессоров TMS320 со стандартной и улучшенной стандартной архитектурой семейств C2x, C54x, C55x и некоторых других фирмы TI. Внутренняя память всех этих процессоров включает несколько блоков B0, B1 и т. д., которые программным образом могут включаться как в пространство памяти программ, так и в пространство памяти данных. Кроме того, эти блоки отличаются параметрами быстродействия — они классифицируются как SARAM (single access RAM) DARAM (dual access RAM), т. е. как память с одиночным и двойным доступом. Соответственно, блок SARAM допускает одно обращение к памяти за такт работы ЦПУ, а блок DARAM два обращения за один такт. В блоках DARAM время обращения к памяти составляет половину такта и, в этом случае, возможно два обращения в первой половине и во второй половине такта. Использование блоков DARAM важно для процессоров TMS320. Система команд этих ЦПОС включает большое количество комбинированных команд и команд, в которых адресуются ячейки памяти. Кроме того, команды могут иметь различную длину. Блоки с двойным доступом позволяют в необходимых случаях командам выбирать два операнда, расположенные в одном и том же блоке. При конвейерном выполнении команд может потребоваться одновременно (но на разных этапах двух последовательно выполняемых команд) обращение к одному и тому же блоку. Если идет обращение к блоку с двойным доступом, команды будут выполнены без задержки. Если же обращение происходит к блоку типа SARAM, неизбежены конфликт конвейера и задержка выполнения команд.

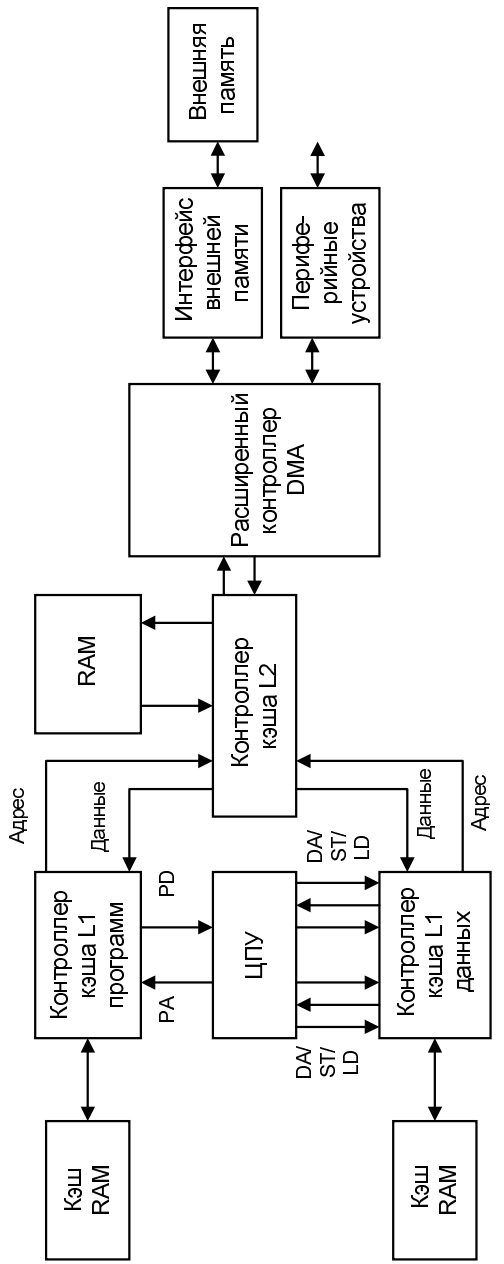


Рис. 2.24. Функциональная схема организации памяти в процессорах TMS C621x, C671x, C64xx

Проиллюстрируем сказанное на примере процессора C54. Данный процессор использует 8 шин, назначение которых приведено в табл. 2.6. Как уже было отмечено выше, в блоках типа DARAM два обращения происходят в разных половинах такта работы ЦПУ. В табл. 2.6 отмечено, на какой половине такта используется та или иная шина процессора.

Таблица 2.6. *Шины и выполняемые операции процессора TMS320C54*

Шина	Выполняемая операция	Операция выполняется в:
PAB	Загрузка адреса ПП	первой половине такта ЦПУ
PB	Чтение кода команды из ПП	первой половине такта ЦПУ
DAB	Загрузка адреса ПД	первой половине такта ЦПУ
DB	Чтение данного из ПД	первой половине такта ЦПУ
CAB	Загрузка адреса ПД	второй половине такта ЦПУ
CB	Чтение данного из ПД	второй половине такта ЦПУ
EAB	Загрузка адреса ПД	второй половине такта ЦПУ
EB	Запись данного в ПД	второй половине такта ЦПУ

Конвейер процессора имеет шесть этапов, приведенных на рис. 2.25, *а*.

При выполнении команды

```
MAC *AR2, *AR3 ,A   ; A + (*AR2)*(*AR3) -> A
```

производится выборка двух сомножителей из ячеек памяти, указанных во вспомогательных регистрах AR2 и AR3, их перемножение и сложение с содержимым аккумулятора. Если сомножители расположены в памяти типа DARAM, они выбираются одновременно по шинам данных DB и CB и так как эти шины используют для обращения разные половины цикла работы ЦПУ, то конфликта конвейера не происходит. Этапы конвейера, на которых выполняется обращение к памяти, приведены на рис. 2.25, *б*. Если бы сомножители располагались в памяти типа SARAM, обращение к памяти для выбора второго сомножителя потребовало бы еще одного цикла.

Пусть необходимо последовательно выполнить команды

```
ST  A,*AR1          ; (A) -> (*AR1) сохранение в памяти
LD  *AR3,B          ; (*AR3) -> B загрузка из памяти
```

В соответствии с первой командой содержимое аккумулятора A загружается в ячейку, адрес которой указан в вспомогательном регистре AR1, а второй командой содержимое ячейки памяти, адрес которой указан в регистре AR3, загружается в аккумулятор B. На рис. 2.25, *в* приведено распределение обращений к памяти по этапам выполнения команды и взаимное расположение во времени этих операций. Операции записи в память и чтения из памяти этих двух команд оказались совмещены во времени. Если обращение

идет к памяти типа DARAM, операции на шинах DB и EB выполняются на разных половинах цикла ЦПУ и не вызывают конфликта конвейера. Если же обращение производится к памяти типа SARAM, возникает конфликт конвейера и задержка выполнения второй команды на один цикл. Соответствующая ситуация приведена на рис. 2.25, *г*, где штриховкой отмечены пустые невыполняемые этапы команды.

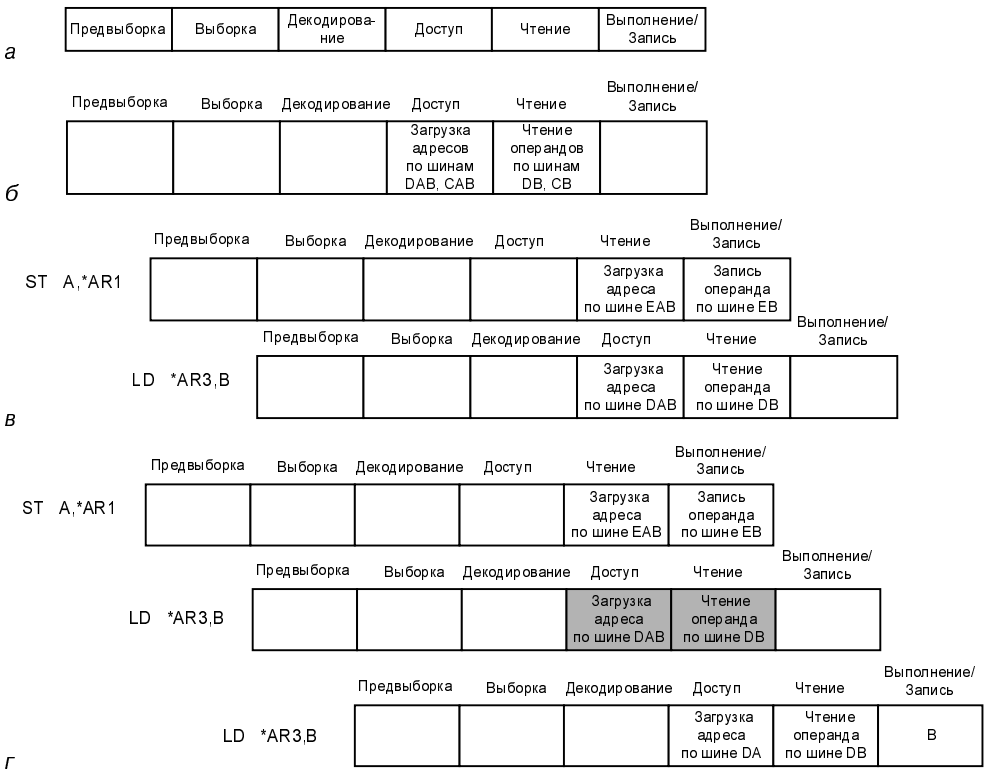


Рис. 2.25. Конвейер процессора TMS320C54x: *а* — этапы конвейера процессора; *б* — обращение к памяти при выполнении команды `MAC AR2,AR3,A`; *в* — обращение к памяти при выполнении двух команд (блок DARAM); *г* — обращение к памяти при выполнении двух команд (блок SARAM)

2.6.2. Внешняя память

Разделение на отдельные области и модули

Все приведенные ранее соображения о разделении памяти на ПП и ПД, а также на различные блоки, обращение к которым может происходить по нескольким шинам, относятся к внутренней памяти процессоров. Для обращения к внешней памяти во всех процессорах применяется один общий

комплект шин — шина адреса и шина данных, по которым при необходимости передаются как коды команд, так и данные. Это вызвано требованием минимизации количества внешних выводов процессора и значительным усложнением системы в противном случае.

Внешняя память по отношению к процессору рассматривается как единое пространство памяти, ячейки и отдельные части которого отличаются только адресами. Если в процессоре предусматривается разделение общей памяти на ПП и ПД и в свою очередь разделение ПД на блоки (например, память данных X и память данных Y), то эти блоки во внешней памяти могут отличаться только областью адресов. Конфигурация указанных блоков определяется пользователем и обычно записывается в те или иные регистры управления.

Например, в процессорах Motorola DSP5600x конфигурация внешней памяти, т. е. области адресов, принадлежащих памяти программ, памяти данных X и Y определяются содержимым регистра режима работы OMR. Полезно при этом обратить внимание на следующий момент. Формально модули памяти данных X и Y имеют совершенно одинаковые области адресов, однако обращение к той или иной области определяется сигналом на специальном выводе порта A X/Y, т. е. фактически область адресов памяти данных удваивается и в этой "удвоенной" области двоичные адреса памяти X и памяти Y отличаются значением в старшем разряде.

Обращение к памяти

При наличии одного комплекта шин адрес/данные возможно только одно обращение к внешней памяти за цикл работы ЦПУ. Если в соответствии с выполняемой программой потребуется бóльшее число обращений, то возникнет конфликт конвейера выполнения команд и, соответственно, задержки обработки очередной команды.

Генерация тактов ожидания для медленной внешней памяти

Задержки выполнения очередной команды при обращениях к внешней памяти могут возникать при использовании в качестве внешней БИС ЗУ с малым быстродействием. Для того чтобы обращение к ним происходило с быстродействием работы системы, они должны иметь определенное время обращения. Например, для БИС ЗУ в системах на основе процессоров TMS320 должно выполняться условие

$$t_{\text{обр ЗУ}} < T_{\text{ЦПУ}}/2,$$

где $t_{\text{обр ЗУ}}$ — время обращения к памяти (время чтения или записи в ЗУ); $T_{\text{ЦПУ}}$ — длительность цикла работы ЦПУ (длительность периода тактовой частоты).

Иногда для разрешения конфликтов, вызванных задержками при обращении к внешней памяти, может возникнуть необходимость в генерации тактов ожидания. Их генерация возможна, например, в процессорах TMS320C3x.

2.6.3. Кэш

Кэш (или кэш-память) используется в качестве буфера между собственно процессором и памятью системы [40]. Она имеет обычно небольшую емкость при высоком быстродействии и используется для промежуточного запоминания информации, считываемой из памяти, прежде всего кодов команд. При чтении сначала выполняется обращение к кэшу; если же необходимых данных там не оказалось, производится обращение к основной памяти, и полученные данные помещаются также в кэш [41]. Выгода от использования кэша возникает из-за того, что большинство прикладных программ носит циклический характер (в особенности это относится к алгоритмам ЦОС), и после первого обращения к основной программе в кэш (при достаточном его объеме) попадает все циклически повторяющиеся команды программы, для выполнения которых нет необходимости обращаться к основной памяти. Наибольшая эффективность кэша будет достигаться при использовании для хранения программы "медленной" внешней памяти. Кэш чаще всего работает по ассоциативному принципу, при котором в его ячейке хранится не только слово данных, но и адрес размещения этого слова в основной памяти, по которому и происходит поиск информации.

Организация кэш-памяти в различных ЦПОС существенно различается.

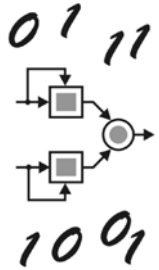
- В процессорах фирмы Lucent DSP16xx и DSP16xxx в кэше может сохраняться до 15 и 31 команды соответственно. Их запись в кэш для циклического повторения производится пользователем программным образом специальной командой `do K`, где `K` — необходимое число повторений. Таким образом, использование кэша совмещается с организацией цикла.
- В процессорах TMS320C55x фирмы TI кэш-память имеет объем 24 Кбайт для информации (код/данные), выбранный модулем буфера команд из памяти программы. Структура кэша может быть конфигурирована пользователем, возможно использование трех типов структуры.
- В процессорах с ПТ TMS320C3x кэш может сохранять до 64 повторяющихся команд.
- Процессоры семейства 56300 фирмы Motorola имеют кэш, который является буфером между процессором и только внешней памятью. Объем его — 1024 24-разрядных слова. Работа кэша определяется установками в регистрах управления и может изменяться некоторыми командами.
- Особенности использования кэша суперскалярного ЦПОС LSI40xZ (ZSP164xx) фирмы LSI Logic Corporation были отмечены в *разд. 2.4.4*.

- Структура памяти и кэша процессоров TI семейства TMS320C6000 описана в *разд. 2.6.1.*
- Оригинальными особенностями обладает кэш в некоторых процессорах фирмы ADI (ADSP-219x, ADSP-21xxx). В этих процессорах кэш упрощает проблему доступа к памяти при выполнении основной операции ЦОС — умножения с накоплением MAC *(см. разд. 2.1)*. Для ее выполнения требуется три обращения к памяти: чтение команды, коэффициента и отсчета сигнала. При использовании двух блоков памяти (память программ ПП и память данных ПД) процессоры ADI обычно используют ПП для хранения коэффициентов. При этом могут возникать конфликты при двух обращениях к ПП (чтение команды и чтение коэффициента). В кэше процессоров ADI сохраняются лишь команды, которые требуют одновременного чтения данных из ПП. Для циклически повторяющихся команд (а это большинство алгоритмов ЦОС) первое их выполнение требует обращения к памяти и занимает больше времени, затем команды выбираются из кэша, а данные — из ПП. При этом выборки сигнала идут по шине данных ПД, выборки коэффициентов — по шине данных ПП, а команды — из кэша. Объем кэша в ADSP-21xxx (Sharc) — 32 слова.

2.6.4. Защита содержимого памяти

В некоторых процессорах используются методы защиты содержимого памяти от чтения (сканирования). Например, в процессорах TI TMS320C54x применяется защита содержимого внутренней памяти от распаковки пользователем. Эта особенность может быть использована только для ЦПОС с внутренним масочным ПЗУ (ROM), содержимое которого записывается (заказывается) при изготовлении процессора. Защита не может быть отключена пользователем. Она имеет различные варианты: защита только ПЗУ, защита ПЗУ и ОЗУ, работа процессора только в режиме микрокомпьютера (невозможность отключения внутреннего ПЗУ).

Глава 3



Данные

Реализация устройств цифровой обработки сигналов (ЦОС) на базе цифровых сигнальных процессоров (ЦПОС) включает следующие основные этапы *(см. главу 9)*:

- разработка метода и алгоритма ЦОС;
- составление и отладка программы;
- выполнение программы процессором.

На всех этапах объектами цифровой обработки являются *данные* — совокупность констант, переменных и массивов. Различают *исходные*, *промежуточные* и *конечные* данные; значения исходных данных задаются, а промежуточных и конечных — вычисляются.

Указанным этапам соответствуют разные модели обработки данных: первому — математическая, второму — программная, а третьему — физическая. Каждому из этих этапов соответствует свое *представление* данных, обусловленное спецификой модели.

В настоящей главе рассматривается представление данных на уровне физической модели — в регистрах и ячейках памяти процессоров. Однако прежде кратко остановимся на основных характеристиках данных и особенностях их представления в алгоритме и программе.

3.1. Представление данных в алгоритме

Данные в схеме алгоритма представляются:

- символическими именами констант, переменных и массивов;
- непосредственно константами.

Важнейшей характеристикой данных, определяющей их представление и обработку в процессоре, является *тип данных*.

Различают следующие основные типы данных:

- целый;

- вещественный;
- комплексный;
- логический;
- литеральный (текстовый).

В алгоритмах ЦОС в вычислительных блоках, непосредственно относящихся к обработке сигналов, как правило, используются данные:

- *вещественного* типа — отсчеты входного и выходного сигналов, коэффициенты уравнений, отсчеты ДПФ и импульсных характеристик, значения частотных характеристик и т. д.;
- *комплексного* типа — в виде упорядоченных пар данных вещественного типа, соответствующих вещественной и мнимой частям;
- *целого* типа — количество и номера отсчетов и коэффициентов, размерность массивов, счетчики и т. д.;
- *логического* типа в операциях побитовой обработки.

Операции управления и вычисления адресов (номеров ячеек памяти, хранящих данные) выполняются специальными устройствами, предусмотренными в архитектуре процессоров, поэтому такие вычисления непосредственно к обработке данных не относятся и в этой главе не рассматриваются.

Наибольший объем вычислений в алгоритмах ЦОС связан с данными вещественного типа, поэтому в настоящей главе им уделено основное внимание. Обработка данных целого типа, занимающая, как правило, существенно меньший объем вычислений, также обсуждается. Особенности обработки данных логического типа и бит-последовательностей рассматриваются в *главе 6*.

3.2. Представление данных в программе

Данные в командах программы, составленной на языке ассемблера, представляются (указываются):

- символическими именами констант, переменных и массивов, соответствующими *адресам* ячеек памяти, в которых они хранятся;
- именами регистров, в которых они хранятся;
- непосредственно константами.

Правила указания данных в командах рассматриваются в *главе 5*.

Представление *исходных* данных в директивах инициализации констант, зависящее от их представления в процессоре, рассматривается в этой главе.

3.3. Представление данных в ЦПОС

Представление данных в процессоре обусловлено:

- разрядностью ячеек памяти и регистров, в которых они хранятся;
- условным функциональным распределением разрядов (на знаковые, значащие и т. п.) в ячейках и регистрах;
- спецификой выполнения арифметических операций в процессоре.

Соответственно, представление данных характеризуется:

- форматом;
- формой;
- кодом.

Кроме того, представление данных зависит от *типа арифметики*, используемой в процессоре.

Характеристики представления данных и типы арифметики подробно обсуждаются далее, но сначала кратко напомним особенности алгебраического представления чисел в двоичной системе и основные правила выполнения арифметических операций с двоичными числами.

3.4. Двоичная система счисления

Двоичная система счисления — это простейшая позиционная система, в которой:

- для записи чисел используются только две цифры — 0 и 1;
- разряды двоичного числа называют *битами*;
- веса битов зависят от их месторасположения (позиции) в двоичном числе.

Целые десятичные числа переводятся в двоичные числа *точно*. Правило перевода заключается в последовательном делении модуля десятичного числа на 2 и запоминании остатков (0 или 1). Процесс деления продолжается до тех пор, пока частное не окажется равным 1. Двоичное число составляется из последовательности последнего частного, которое является старшим разрядом (старшим битом) числа и остатков, начиная с последнего; знак числа восстанавливается. В табл. 3.1 приведен пример перевода числа $-123_{(10)}$ в двоичное число.

Таблица 3.1. *Пример перевода целого десятичного числа в двоичное*

Деление модуля числа	Остаток	Двоичное число
123 / 2 = 61	1 — младший бит	
61 / 2 = 30	1	–1111011
30 / 2 = 15	0	

Таблица 3.1 *(окончание)*

Деление модуля числа	Остаток	Двоичное число
15 / 2 = 7	1	
7 / 2 = 3	1	
3 / 2 = 1 — старший бит	1 — последний остаток	

Дробные десятичные числа переводятся в двоичные числа в общем случае *приближенно*, поэтому при переводе необходимо заранее указывать количество значащих цифр (количество битов) после запятой, определяющее точность представления десятичного числа. Правило перевода заключается в последовательном умножении на 2 модуля *дробной части* десятичного числа и запоминании значений произведения (0 или 1) целой части. Двоичное число составляется из последовательных значений целых частей произведений, начиная с первого, которое является старшим разрядом (старшим битом); знак числа восстанавливается. Пример перевода числа $0{,}65_{(10)}$ в двоичное число с точностью до 7 значащих цифр после запятой приведен в табл. 3.2.

Таблица 3.2. *Пример перевода дробного десятичного числа в двоичное*

Умножение модуля числа	Целая часть	Двоичное число
0,65 · 2 = 1,30	1 — старший бит	
0,30 · 2 = 0,60	0	
0,60 · 2 = 1,20	1	
0,20 · 2 = 0,40	0	0,1010011…
0,40 · 2 = 0,80	0	
0,80 · 2 = 1,60	1	
0,60 · 2 = 1,20	1 — младший бит	
…	…	

Пример точного перевода дробного числа $0{,}75_{(10)}$, как частного случая, дается в табл. 3.3.

Таблица 3.3. *Точный перевод дробного десятичного числа в двоичное*

Умножение модуля числа	Целая часть	Двоичное число
0,75 · 2 = 1,50	1 — старший бит	0,11
0,50 · 2 = 1,00	1	

Для *смешанных* чисел отдельно переводятся целая и дробная части; целая часть переводится точно, дробная — приближенно.

3.5. Форматы данных

Формат данных связан с разрядностью ячеек памяти и регистров, в которых хранятся данные; он определяет возможную длину представления данных в процессоре.

Различают следующие основные форматы представления данных:

- байт;
- полуслово;
- слово;
- двойное слово;
- расширенное слово.

Основными из этих форматов являются: слово, двойное слово и расширенное слово.

Замечание

Формат "двойное слово" часто называют *длинным словом*, подразумевая тот же смысл, однако, формат "короткое слово", также используемый для представления данных в процессорах, не обязательно соответствует полуслову. Короткое слово всегда больше байта и меньше слова.

Слово отображает содержимое одной из n-разрядных ячеек памяти или одного из n-разрядных регистров, поэтому слово обычно характеризует *внешнее представление* исходных и конечных данных.

Длина слова n измеряется в битах и по величине равна разрядности соответствующей ячейки памяти данных или регистра.

Двойное слово отображает содержимое пары соседних n-разрядных ячеек памяти, одного $2n$-разрядного регистра или пары соседних n-разрядных регистров, поэтому двойное слово характеризует представление данных *с удвоенной точностью*; *длина* двойного слова равна $2n$ битам.

Расширенное слово отображает содержимое k-разрядного аккумулятора или выходного регистра, и характеризует *внутреннее* представление данных — результатов промежуточных и конечных вычислений. Длина расширенного слова равна k битам; в зависимости от архитектуры процессора, *формы* представления данных и назначения регистра, для величины k выполняется условие: $k > 2n$ (в процессорах с фиксированной точкой) или $n < k < 2n$ (в процессорах с плавающей точкой). Длина расширенного слова всегда больше длины слова, что позволяет повысить точность промежуточных и конечных вычислений.

Содержимое аккумулятора отображается *словом аккумулятора*, которое, в зависимости от архитектуры процессора, имеет формат *двойного* или *расширенного* слова.

Байт и *полуслово* отображают содержимое соответствующих частей ячейки памяти данных или регистров.

Основные форматы (слово, двойное слово и расширенное слово) показаны на рис. 3.1.

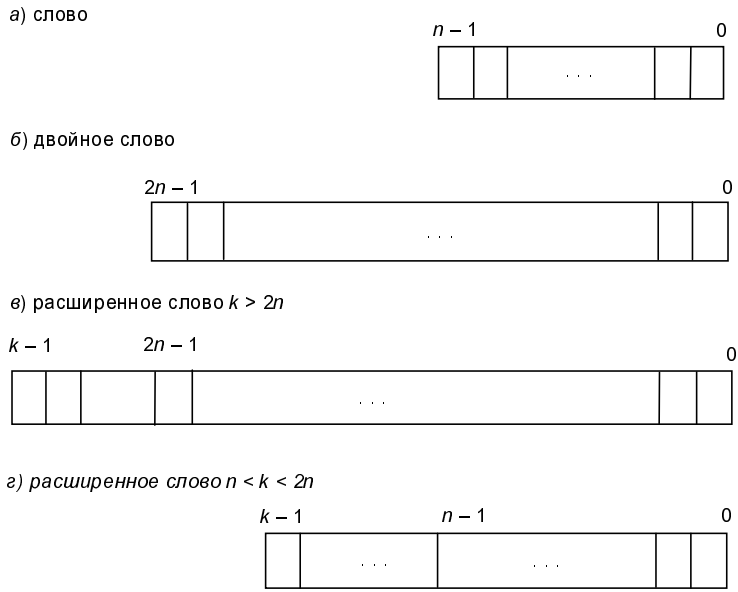


Рис. 3.1. Основные форматы данных

В табл. 3.4 приведены примеры форматов данных в процессорах фирм Texas Instruments, Motorola и Analog Devices; формы представления данных, также указанные в этой таблице, рассматриваются далее.

Таблица 3.4. *Форматы и формы представления данных в ЦПОС*

Фирма	Процессор	Формат			Форма представления
		Слово	Двойное слово	Расширенное слово	
		Длина в битах			
Texas Instruments	TMS320C3x*	32	64	40	ПТ и ФТ
	TMS320C2xxx	16	32	Нет	ФТ
	TMS320C54xx	16	32	40	ФТ
	TMS320C55xx	16	32	40	ФТ
	TMS320C64xx**	32	64	40	ФТ
	TMS320C62xx**	32	64	40	ФТ
	TMS320C67xx*	32	64	40	ФТ и ПТ

Таблица 3.4 *(окончание)*

Фирма	Процессор	Формат			Форма представления
		Слово	Двойное слово	Расширенное слово	
		Длина в битах			
Motorola	DSP560xx/563xx	24	48	56	ФТ
	DSP568xx	16	32	36	ФТ
	DSP566xx	16	32	40	ФТ
	MSC8100	16	32	40	ФТ
	DSP9600x*	32	64	44	ПТ и ФТ
	MSC810x	16	32	40	ФТ
Analog Devices	ADSP-21xx	16	32	40	ФТ
	ADSP-21xxx*	32	64	40	ПТ и ФТ

В табл. 3.4 символом * отмечены процессоры, использующие представление данных с плавающей точкой *(см. разд. 3.8)*, а символом ** — процессоры, в которых поддерживается обработка упакованных данных *(см. разд. 3.7.13)*.

Формат является важнейшей характеристикой представления данных. *Базовым* форматом в процессоре считается *слово*, его длина определяет диапазон и точность представления данных, объем памяти, разрядность шины данных и т. д. Пользователю, выбирающему процессор для конкретного приложения, желательно оценить минимально-достаточную длину слова.

3.6. Формы представления данных

Формат данных определяет длину последовательности из нулей и единиц — количество битов в *последовательности битов*. Такие последовательности могут рассматриваться как данные в операциях бит-манипуляций (операциях с отдельными битами) или как данные логического типа в логических операциях `AND`, `OR`, `NOT`. Однако для того, чтобы эти последовательности воспринимались как численные данные (*двоичные числа*), необходима дополнительная информация:

- ❒ о функциональном распределении битов в последовательности;
- ❒ о типе числа, представляемого в заданном формате;
- ❒ о форме представления числа заданного типа.

При этом именно форма представления является определяющей и для типа числа, и для функционального распределения битов в последовательности.

Прежде, чем рассматривать формы представления чисел в процессорах, напомним *алгебраические* формы записи чисел.

Для записи вещественных и целых чисел в алгебре используют *две* формы:

- обычную, например:

  5000; 5000,0; −3,77; 123; 13,784 и т. п.;

- показательную (или полулогарифмическую), например:

  $5 \cdot 10^3$; $5{,}0 \cdot 10^3$; $-0{,}377 \cdot 10^1$; $1{,}3 \cdot 10^2$; $137{,}84 \cdot 10^{-1}$ и т. п.

При этом в записи целых чисел запятая недопустима, т. е. целое число 5000, записанное как 5000,0, считается вещественным.

В цифровой вычислительной технике при записи чисел в программе принято целую часть от дробной отделять не запятой, а *точкой*.

Форма представления численных данных (чисел) в процессоре отображает алгебраическую форму записи числа.

Соответственно двум алгебраическим формам записи вещественных чисел — обычной и показательной — различают две формы представления численных данных в процессорах:

- с фиксированной точкой (ФТ);
- с плавающей точкой (ПТ).

Замечание

Терминология "фиксированная точка" (ФТ) и "плавающая точка" (ПТ), пришедшая из алгоритмических языков программирования, тождественна по смыслу "фиксированной запятой" (ФЗ) и "плавающей запятой" (ПЗ).

Данные *целого* типа (целые двоичные числа) представляются в процессорах *только в форме с ФТ*.

Данные *вещественного* типа могут представляться в форме с фиксированной или с плавающей точкой, в зависимости от чего различают (см. табл. 3.4):

- ЦПОС с фиксированной точкой;
- ЦПОС с плавающей точкой.

Рассмотрим подробнее представление данных с ФТ и ПТ.

3.7. Представление данных с фиксированной точкой

Представление данных вещественного типа (вещественных двоичных чисел) в форме с ФТ означает, что в рамках заданного формата для всех вещественных чисел логически фиксируется одинаковое местоположение точки, разделяющей целую и дробные части числа.

Представление целых двоичных чисел в форме с ФТ означает, что в рамках заданного формата для всех целых чисел точка логически фиксируется за правой границей формата, т. е. по существу отсутствует.

В дальнейшем для краткости вместо "числа, представленные в форме с ФТ или с ПТ" будем говорить "числа с ФТ" или "числа с ПТ".

Прежде чем обсуждать особенности представления целых и вещественных чисел в форме с ФТ, необходимо познакомиться с внутренней структурой тех форматов, в которых эти числа представляются.

3.7.1. Структура двойного и расширенного слова при представлении чисел с ФТ

Структура двойного и расширенного слова при представлении чисел с ФТ приведена на рис. 3.2. В рамках заданных форматов условно выделяют следующие части:

- в двойном слове (рис. 3.2, *а*):
  - старшее слово — MSP (Most Significant Portion);
  - младшее слово — LSP (Least Significant Portion);
- в расширенном слове (рис. 3.2, *б*):
  - старшее слово — MSP;
  - младшее слово — LSP;
  - расширение — EXT (Extension);
- в слове:
  - старшая часть слова — MSP;
  - младшая часть слова — LSP.

На рис. 3.2 показана сплошная нумерация битов, однако допустима нумерация внутри каждого из слов MSP, LSP, EXT.

Биты расширения EXT в процессорах называют "защитными" или "сторожевыми" (guard bits); смысл этого станет понятным далее из предназначения EXT.

Длины слов MSP и LSP равны между собой:

MSP = LSP = 1/2 длины двойного слова = длине слова.

Как видно из табл. 3.4, длина расширения EXT обычно равна байту, либо половине байта.

Двойное слово и расширенное слово удобно записывать в виде объединения соответствующих частей, а именно, MSP:LSP — для двойного слова, или EXT:MSP:LSP — для расширенного слова.

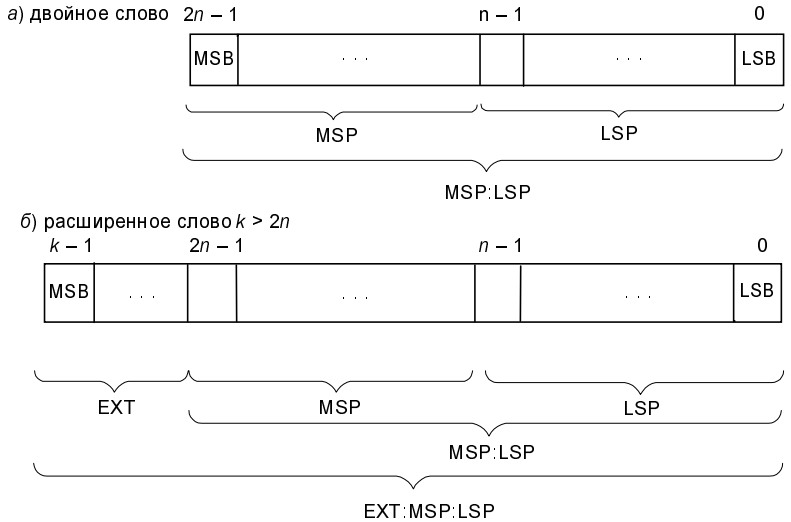


Рис. 3.2. Структура двойного и расширенного слов при представлении чисел с ФТ

Дополнительно в рамках каждого из форматов обычно именуют два крайних бита:

- старший бит — MSB (Most Significant Bit);
- младший бит — LSB (Least Significant Bit).

На рис. 3.2 обозначены биты MSB и LSB в двойном и расширенном словах. Аналогично можно именовать старший и младший биты в рамках каждого из слов MSP, LSP и EXT.

Рассмотрим назначение MSP, LSP и EXT слов при представлении целых и вещественных чисел.

3.7.2. Представление целых чисел

Представление целых чисел в форме с ФТ в форматах *слово* и *двойное слово* предполагает следующее функциональное распределение битов:

- старший бит MSB используется:
  - как *знаковый* при представлении *целых чисел со знаком*; значение MSB = 0 соответствует положительному знаку, а MSB = 1 — отрицательному знаку; ноль считается положительным; остальные биты являются значащими;
  - как старший *значащий* при представлении *беззнаковых* чисел; беззнаковыми называются целые числа, имеющие положительный знак по умолчанию;

- все биты, кроме знакового, считаются *значащими*; они выравниваются по *правому* краю формата, т. е. младший бит LSB соответствует младшему разряду целого двоичного числа; в "лишних" старших битах *целого со знаком* происходит *расширение знака*; это говорит о том, что все "лишние" старшие биты автоматически заполняются значением старшего знакового бита MSB; "лишние" старшие биты *беззнакового* целого обнуляются.

Операция *"расширение знака"* весьма широко используется в сигнальных процессорах. Подробно она рассматривается ниже, после знакомства с дополнительным кодом.

На рис. 3.3, *а*, *б* приведены примеры представления целых чисел со знаком в формате "слово" длиной 8 битов, а на рис. 3.3, *в* — пример беззнакового числа в том же формате; указаны *веса* битов и дано правило перевода двоичного целого числа в десятичное.

Рис. 3.3. Примеры представления целых чисел

При представлении целых чисел со знаком в формате "расширенное слово" происходит следующее:

- функциональное распределение битов в слове MSP:LSP сохраняется таким же, как и для двойного слова (старший бит MSB — знаковый, остальные — значащие с выравниванием по *правому краю*);
- в расширении EXT происходит *расширение знака*; это означает, что все биты EXT автоматически заполняются значением старшего знакового бита MSB слова MSP:LSP.

На рис. 3.4 приведен пример расширенного слова — слова аккумулятора длиной 56 битов в процессорах DSP5600x фирмы Motorola, где расширение EXT заполнено нулями — значением 47-го знакового бита.

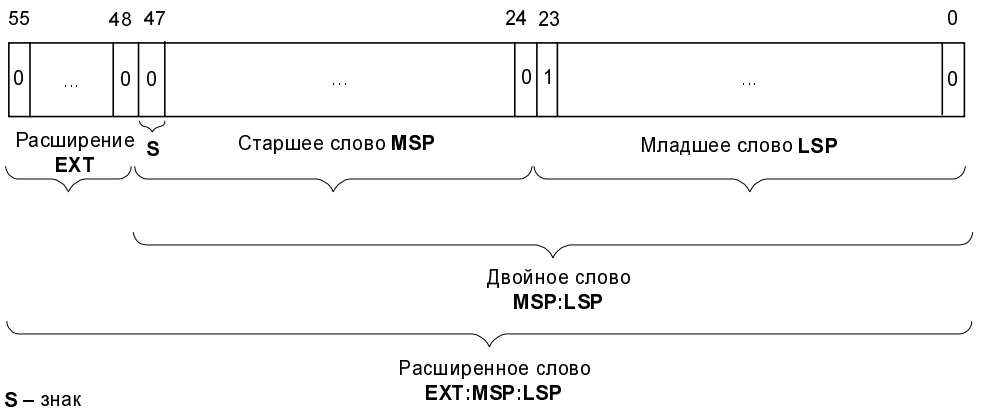


Рис. 3.4. Слово аккумулятора в процессорах DSP5600x фирмы Motorola

В дальнейшем, говоря о целых числах, будем подразумевать целые со знаком, а беззнаковые числа оговаривать особо.

3.7.3. Представление вещественных чисел

Представление дробных чисел в форме с ФТ в форматах "слово" и "двойное слово" предполагает следующее функциональное распределение битов:

- старший бит MSB используется, как *знаковый*; значение MSB = 0 соответствует положительному знаку, а MSB = 1 — отрицательному знаку; число ноль считается положительным; остальные биты являются значащими;
- все биты, кроме знакового, считаются значащими; они выравниваются по левому краю формата, т. е. старший бит MSB соответствует старшему разряду дробного двоичного числа; "лишние" младшие биты обнуляются;

- после старшего, знакового, бита логически фиксируется точка (запятая), отделяющая целую часть (равную 0) от дробной.

Символическое обозначение формата, в котором представлено дробное число, имеет вид Q*b*, где *b* — количество значащих битов дробного числа. На рис. 3.5 приведен пример представления дробного числа в формате Q7; указаны веса битов и дано правило перевода двоичного дробного числа в десятичное.

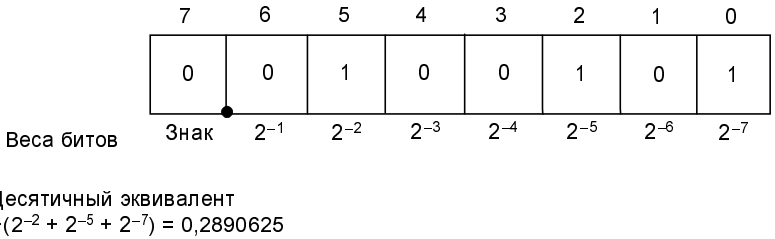


Рис. 3.5. Пример представления дробного числа в формате Q7

Представление вещественного числа в формате "расширенное слово" EXP:MSP:LSP зависит от того, является ли число дробным или смешанным (содержащим целую и дробную части).

Дробные числа размещаются в MSP:LSP части расширенного слова, при этом функциональное распределение битов таково:

- *знаковым* считается старший бит слова MSP:LSP;
- *точка* логически фиксируется после знакового бита;
- в расширении EXT происходит *расширение* знака дробного числа.

Функциональное распределение битов при размещении в расширенном слове EXP:MSP:LSP *смешанных* чисел следующее:

- *знаковый* бит перемещается из старшего бита слова MSB:LSB в старший бит расширения EXT;
- остальные биты расширения EXT *плюс старший бит* слова MSP:LSP отводятся для хранения *целой* части числа; значащие биты целой части смешанного числа выравниваются по *правому* краю; длина целой части равна длине расширения EXT; она может изменяться только в зависимости от режима масштабирования (сдвига вправо/влево) содержимого EXT:MSP:LSP; "лишние" старшие биты в EXT заполняются расширением знака;
- для *дробной* части смешанного числа отводится слово MSP:LSP без старшего бита; "лишние" младшие биты обнуляются.

Переход от дробного числа к смешанному фиксируется установкой специального бита в регистре состояния.

Функциональное распределение битов в слове аккумулятора процессоров DSP5600x фирмы Motorola показано на рис. 3.6. Отметим, что при представлении в формате расширенное слово дробного числа, значения всех битов расширения EXT одинаковы и равны значению старшего (47-го, знакового) бита слова MSP:LSP; количество одинаковых битов равно длине EXT плюс 1. При представлении смешанного числа в этой области размещается целая часть смешанного числа со знаком, и значения битов в ней неодинаковые. Рекомендуем вернуться к данному описанию после знакомства с дополнительным кодом чисел.

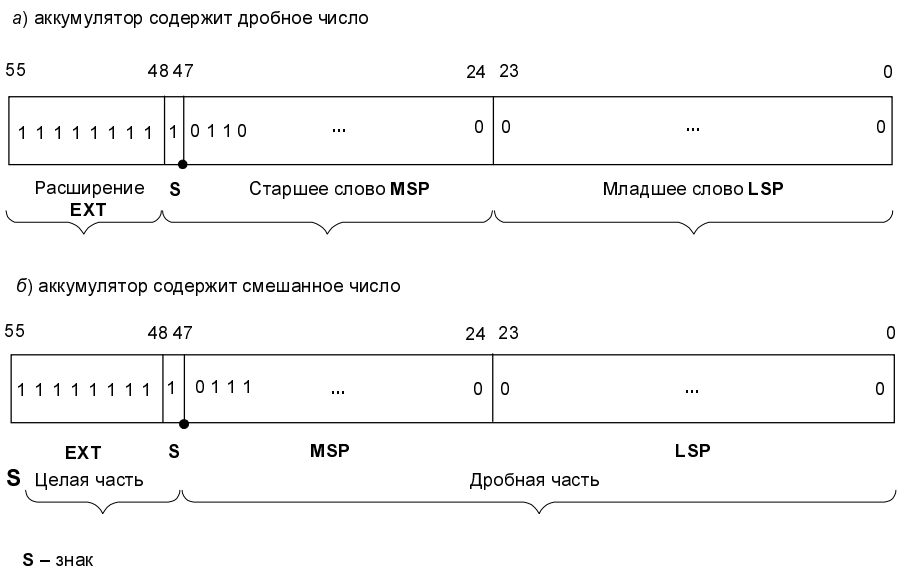


Рис. 3.6. Распределение битов в слове аккумулятора процессоров DSP5600x фирмы Motorola

В процессорах с ФТ, как правило, все исходные и конечные данные вещественного типа путем масштабирования приводятся к дробным числам (об этом см. ниже). Смешанные числа могут появляться во внутренних промежуточных вычислениях, что при необходимости можно проконтролировать по установке соответствующего бита в регистре состояния. Принципиально возможна программная организация обработки смешанных чисел, отдельно для целой и дробной частей, однако на практике это используется крайне редко. Поэтому в дальнейшем, говоря о вещественных числах в форме с ФТ, по умолчанию будем иметь в виду дробные числа, а смешанные оговаривать особо.

3.7.4. Шестнадцатеричный эквивалент представления данных

Для сокращения записи при представлении данных в ЦПОС используют их шестнадцатеричные эквиваленты, которые получают следующим образом:

- двоичное представление данных разбивается на тетрады слева направо, *не различая знаковых и значащих* битов;
- тетрады записываются шестнадцатеричными эквивалентами.

В различных ЦПОС применяют разные префиксы или суффиксы для записи шестнадцатеричных эквивалентов, например, в процессорах фирмы Motorola — префикс `$`, в процессорах фирмы Texas Instruments — суффикс `h` и т. д.

Приведем пример шестнадцатеричного эквивалента. Слово (длиной 16 битов) разбивается на тетрады:

0010 1011 1001 1101

Тетрады записываются шестнадцатеричными цифрами: 2B9D.

3.7.5. Целочисленная и дробная арифметики

Сравнивая рассмотренные выше представления в процессоре целых и дробных чисел, можно выделить общее и отличия в этих представлениях:

- общее — функциональное распределение битов на знаковые и значащие;
- отличия — выравнивание значащих битов; значащие биты целых чисел выравниваются по правому краю, а дробных — по левому, что обусловлено наличием у дробных чисел условной точки после знакового бита.

Пример двух различных представлений одной и той же последовательности битов 0101 в формате слово длиной 8 битов приведен на рис. 3.7. Функциональное распределение битов в обоих представлениях одинаковое, а именно: старший бит — знаковый, остальные — значащие. Далее решение неоднозначно: число может трактоваться, как *целое* и тогда значащие биты следует выровнять по *правому* краю (рис. 3.7, *а*), или как *дробное* и тогда значащие биты следует выровнять по *левому* краю (рис. 3.7, *б*).

Для того чтобы то или иное из этих двух возможных представлений реализовать, процессору необходима дополнительная информация о типе числа (целое или дробное), в соответствии с которой значащие биты будут выровнены по-разному. Если бы в процессорах с ФТ при представлении чисел указывался признак типа числа, то, кроме идентификации типа числа, потребовалась бы организация выполнения арифметических операций с числами различных типов, а именно: с целыми числами, с дробными числами, с их комбинацией. Понятно, что это привело бы к существенному усложне-

нию архитектуры процессора, поэтому разработчиками различных ЦПОС с ФТ было предложено условно привести все данные дробного и целого типов к одному из этих типов — целому или дробному. В первом случае потребуется аппаратная реализация арифметических операций только с целыми числами, во втором — только с дробными числами. Отсюда и появилась терминология "целочисленная и дробная арифметики".

Рис. 3.7. Пример представления последовательности 0101 как целого или дробного числа

Целочисленная арифметика в процессоре означает, что при выполнении арифметических операций все числа воспринимаются только как целые.

Дробная арифметика в процессоре означает, что при выполнении арифметических операций все числа воспринимаются только как дробные.

Соответственно двум типам реализуемых представлений чисел и арифметик различают:

- процессоры с целочисленной арифметикой;
- процессоры с дробной арифметикой.

К первой группе можно отнести, например, процессоры TMS320C2xxx фирмы Texas Instruments, ко второй — процессоры DSP56xxx фирмы Motorola. Как будет показано в дальнейшем, различия арифметик по существу проявляются только в реализации операции умножения, точнее, в трактовке полученного результата, поэтому во многих процессорах, в частности, в процессорах ADSP-21xx фирмы Analog Devices, TMS320C5xxx фирмы Texas Instruments и др., предусмотрена возможность выбора одного из типов арифметики путем установки соответствующего бита в регистре состояния.

Для представления с ФТ дробных чисел при целочисленной арифметике и, напротив, целых чисел при дробной арифметике используются соответствующие эквиваленты, а именно:

- *целочисленные эквиваленты* дробных чисел — при целочисленной арифметике;
- *дробные эквиваленты* целых чисел — при дробной арифметике.

Представление эквивалентов рассматривается в *разд. 3.7.7* и *3.7.8*.

Программная реализация обработки данных различного типа возлагается на пользователя, к функциям которого относятся:

- трактовка типа исходных данных;
- организация вычислений с данными различных типов;
- трактовка типа промежуточных и конечных результатов.

Более подробно отличия целочисленной и дробной арифметик обсуждаются в *разд. 3.7.7* и *3.7.8*.

Для того чтобы продолжить изучение особенностей представления и обработки чисел с ФТ, необходимо познакомиться еще с одной характеристикой — *кодом представления чисел*.

3.7.6. Коды чисел

Следующей (после формата и формы) характеристикой представления чисел является *код* их представления. В цифровой технике для представления двоичных чисел используют два основных кода:

- прямой;
- дополнительный.

Прямой код

Прямой код двоичных целых и дробных чисел совпадает с представлениями этих чисел, описанными в *разд. 3.7.2* и *3.7.3*.

Дополнительный код

Все процессоры с ФТ оперируют с числами, представленными в дополнительном коде, что позволяет существенно упростить выполнение арифметических операций.

Положительные целые числа представляются в дополнительном коде точно так же, как в прямом коде. Формирование дополнительного кода *отрицательного* числа производится по следующему правилу:

- значение знакового бита не меняется (равно 1);
- все b значащих битов инвертируются (0 заменяются на 1, а 1 — на 0);

- к младшему биту полученного числа прибавляется 1 с соблюдением правил сложения двоичных чисел.

Результат представляет собой число в дополнительном коде. Правило перевода в дополнительный код *не зависит от типа чисел* — целые или дробные.

Примеры перевода из прямого кода в дополнительный представлены на рис. 3.8 и 3.9.

а) положительное целое

$173_{(10)} = +1101101_{(2)} = 01101101_{пр} = 01101101_{доп}$

7	6	5	4	3	2	1	0
0	1	1	0	1	1	0	1

б) отрицательное целое

$-173_{(10)} = -1101101_{(2)} = 11101101_{пр} = 10010011_{доп}$
Формирование дополнительного кода:
инверсия всех битов, кроме знакового: 10010010
добавление 1 к младшему биту: 10010011

7	6	5	4	3	2	1	0
1	0	0	1	0	0	1	1

Рис. 3.8. Примеры целых чисел в дополнительном коде

Обратный перевод числа из дополнительного кода в прямой осуществляется точно по тому же правилу. Приведем пример обратного перевода из дополнительного кода в прямой. Пусть дано число в дополнительном коде: $\mathbf{1}0010011_{доп}$ (жирным шрифтом выделен знаковый бит). Формирование прямого кода:

- инверсия всех битов, кроме знакового → $\mathbf{1}1101100$;
- добавление 1 к младшему биту → $\mathbf{1}1101101_{пр}$.

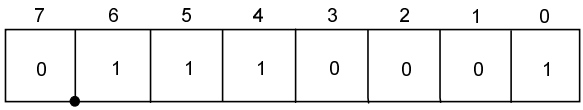
Код называется *дополнительным*, поскольку отрицательное дробное число можно представить как разность между числом 2 (беззнаковым целым) и модулем этого числа, т. е. как дополнение к 2. В качестве примера, получим дополнительный код отрицательного числа −0,875, как его дополнения к 2. Модуль этого числа в формате Q3 равен $0,111_{(2)}$. Вычитая $0,111_{(2)}$ из числа $2_{(10)} = 10,000_{(2)}$ имеем результат $1,001_{(2)}$. Определим дополнительный код числа −0,875 по приведенному выше правилу:

$-0,875 \rightarrow -0,111_{(2)} \rightarrow \mathbf{1}111_{пр}$ → инверсия всех битов, кроме знакового → $\mathbf{1}000$ добавление 1 в младшему биту → $\mathbf{1}001_{доп}$.

Результаты совпадают.

а) положительное дробное

$0{,}8828125_{(10)} = +0{,}1110001_{(2)} = 01110001_{пр} = 01110001_{доп}$



б) отрицательное дробное

$-0{,}8828125_{(10)} = -0{,}1110001_{(2)} = 11110001_{пр} = 10001111_{доп}$
Формирование дополнительного кода:
инверсия всех битов, кроме знакового: 10001110
добавление 1 к младшему биту: 10001111

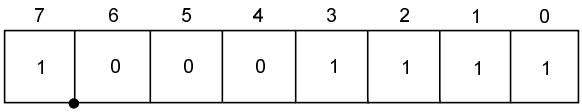


Рис. 3.9. Примеры дробных чисел в дополнительном коде

На рис. 3.10 указаны веса битов и приведено правило вычисления десятичного эквивалента, соответствующего двоичному *целому* числу в дополнительном коде.

На рис. 3.11 указаны веса битов и приведено правило вычисления десятичного эквивалента, соответствующего двоичному *дробному* числу в дополнительном коде.

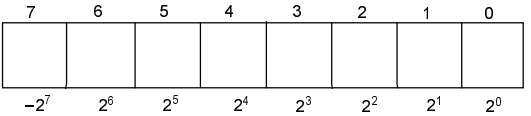
Сравнивая рис. 3.10 и 3.11, еще раз подчеркнем, что одинаковое представление чисел с ФТ может иметь *разную* трактовку типа чисел, а следовательно, соответствовать разным числам.

Приводимые правила нетрудно распространить на любой формат.

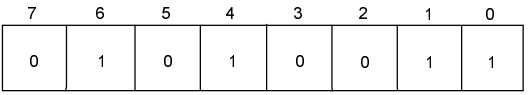
Наряду с простотой выполнения арифметических операций (см. далее), в дополнительном коде устраняется неоднозначность представления нуля. Ноль в прямом коде может быть представлен и как 0000…0 и как 1000…0, а в дополнительном коде только как 0000…0. Это легко проверить, вычислив для чисел, представленных в дополнительном коде, их десятичные эквиваленты:

- $0000\ldots = 0_{(10)}$;
- $1000\ldots = -1_{(10)}$.

а) веса битов

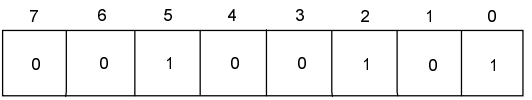


б) дополнительный код
положительного целого числа



Десятичный эквивалент
$(2^6 + 2^4 + 2^2 + 2^0) = 83$

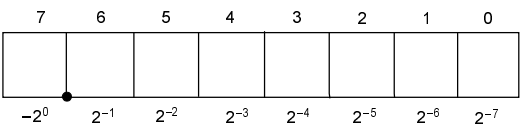
в) дополнительный код
отрицательного целого числа



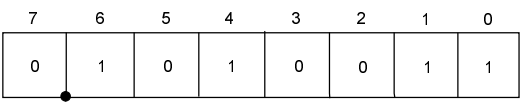
Десятичный эквивалент
$-2^7 + 2^5 + 2^3 + 2^2 = -84$

Рис. 3.10. Десятичный эквивалент двоичного *целого* числа в дополнительном коде

а) веса битов

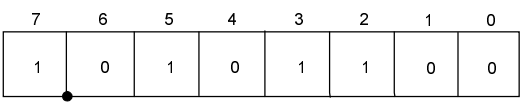


б) дополнительный код
положительного дробного числа



Десятичный эквивалент
$2^{-1} + 2^{-3} + 23^{-6} + 2^{-7} = 0{,}6484375$

в) дополнительный код
отрицательного дробного числа



Десятичный эквивалент
$-2^0 + 2^{-2} + 2^{-4} + 2^{-5} = -0{,}65625$

Рис. 3.11. Десятичный эквивалент двоичного *дробного* числа в дополнительном коде

На рис. 3.12 приведены примеры максимальных и минимальных по модулю значений для положительного и отрицательного целых чисел в дополнительном коде в формате "слово" длиной 8 битов и их десятичные эквиваленты. Десятичные эквиваленты целых чисел для формата "слово" длины n битов (где $n = b + 1$, b — количество значащих битов) равны:

- $2^b - 1$ — максимальное положительное;
- 1 — минимальное положительное;
- -2^b — максимальное по модулю отрицательное;
- -1 — минимальное по модулю отрицательное.

Рис. 3.12. Максимальные и минимальные целые числа в дополнительном коде

На рис. 3.13 приведены примеры максимальных и минимальных значений для положительного и отрицательного дробных чисел в дополнительном коде в формате Q7 и их десятичные эквиваленты. Десятичные эквиваленты *дробных* чисел в формате Q*b* длины равны:

- $1 - 2^{-b}$ — максимальное положительное;
- 2^{-b} — минимальное положительное;
- -1 — максимальное по модулю отрицательное;
- -2^{-b} — минимальное по модулю отрицательное.

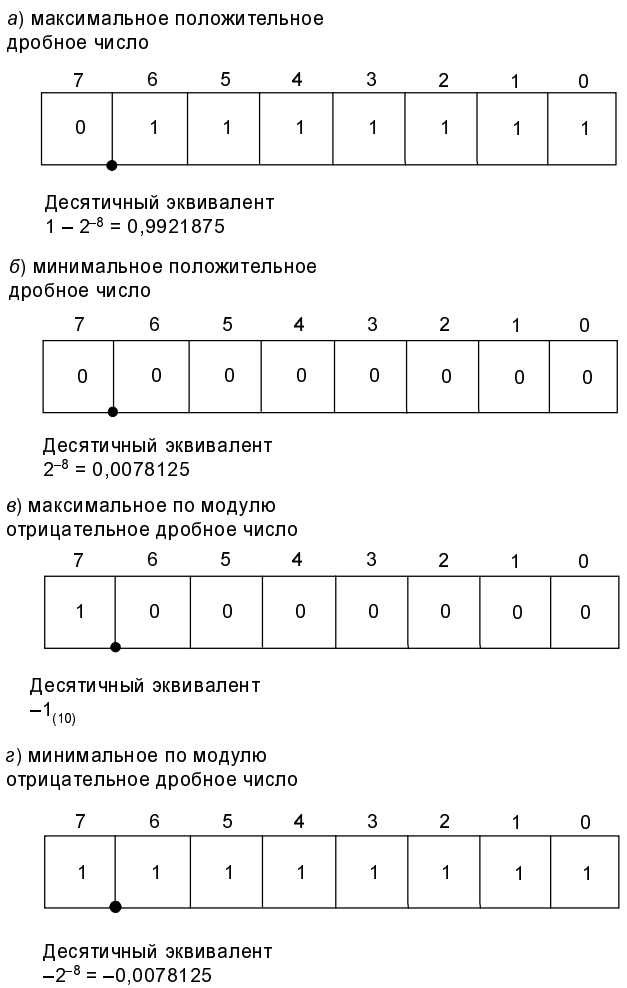


Рис. 3.13. Максимальные и минимальные дробные числа в дополнительном коде

Для чисел, представленных в дополнительном коде, старший разряд остается знаковым.

Трактовка типа чисел, представленных в дополнительном коде в формате "слово", весьма важна при сохранении этих чисел в форматах "двойное слово" и "расширенное слово". Рассмотрим два примера.

Пример 1. Число, представленное в дополнительном коде в формате "слово" длиной 8 битов ($n = 8$), сохраняется в формате "двойное слово" длиной $2n$ (рис. 3.14).

В зависимости от типа данных выполняются следующие действия:

- для *целых* чисел (рис. 3.14, *а*):
  - число (со знаком) в дополнительном коде сохраняется в младшем слове LSP с выравниванием по правому краю;
  - в старшем слове MSP и в "лишних" старших битах младшего слова LSP происходит *расширение знака*;
  - *знаковым* становится старший бит MSB слова MSP:LSP;
- для *дробных* чисел (рис. 3.14, *б*):
  - число (со знаком) в дополнительном коде сохраняется в старшем слове MSP с выравниванием по левому краю;
  - *знаковым* становится старший бит MSB слова MSP:LSP;
  - биты младшего слова LSP и "лишние" младшие биты старшего слова MSP заполняются нулями.

Пример 2. Число, представленное в дополнительном коде в формате "слово" длиной 8 битов ($n = 8$) сохраняется в формате "расширенное слово" длиной $k = 20$. Независимо от типа данных выполняются следующие действия (рис. 3.15, *а*, *б*):

- число (со знаком) в дополнительном коде размещается в слове MSP:LSP (см. выше);
- в расширении EXT происходит *расширение знака*;
- *знаковым* становится старший бит MSB слова EXT:MSP:LSP.

Из приведенных примеров видно, что расширение знака числа, представленного в дополнительном коде, не меняет его значения (см. десятичные эквиваленты на рис. 3.14 и 3.15). Это преимущество дополнительного кода используется, в частности, в операциях пересылок данных, когда изменяется формат их представления *(см. разд. 3.7.10)*.

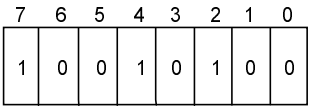
В заключение отметим, что непосредственно в процессоре преобразование кодов из прямого в дополнительный и обратно не производится. Оно выполняется на языке ассемблера на этапе трансляции программы.

Замечание

Современные АЦП выдают значения отсчетов в дополнительном коде.

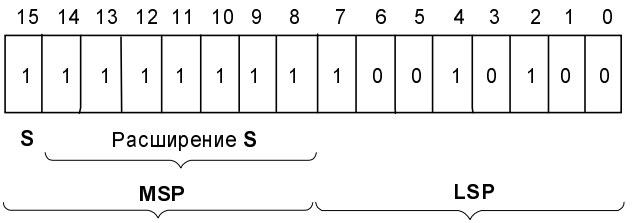
а) целое число

Слово



Десятичный эквивалент
$-2^7 + 2^4 + 2^2 = -128 + 16 + 4 = -108$

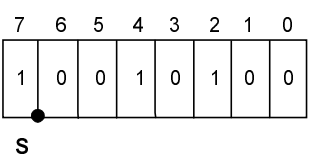
Двойное слово



Десятичный эквивалент
$-2^{15} + 2^{14} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^9 + 2^8 + 2^7 + 2^4 + 2^2 = -108$

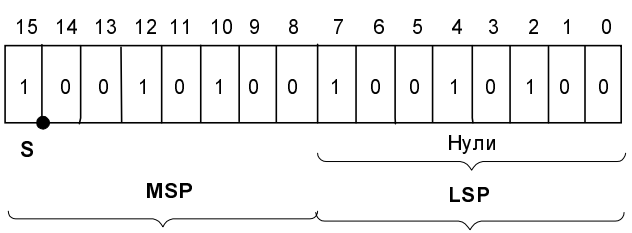
б) дробное число

Слово



Десятичный эквивалент
$-2^0 + 2^{-3} + 2^{-5} = -1 + 0{,}125 + 0{,}03125 = -0{,}84375$

Двойное слово



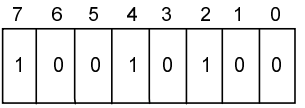
Десятичный эквивалент
$-2^0 + 2^{-3} + 2^{-5} = -0{,}84375$

S – знак

Рис. 3.14. Сохранение числа в формате "двойное слово" (иллюстрация примера 1)

а) целое число

Слово



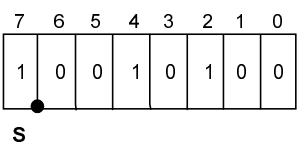
Десятичный эквивалент
$-2^7 + 2^4\,2^2 + 2^2 = -108$

Расширенное слово

Десятичный эквивалент
$-2^{19} + 2^{18} + 2^{17} + 2^{16} + 2^{15} + 2^{14} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^9 + 2^8 + 2^7 + 2^4 + 2^2 = -108$

б) дробное число

Слово



Десятичный эквивалент
$-2^0 + 2^{-3} + 2^{-5} = -1 + 0{,}125 + 0{,}03125 = -0{,}84375$

Расширенное слово

Десятичный эквивалент
$-2^3 + 2^2 + 2^1 + 2^0 + 2^{-3} + 2^{-5} = -0{,}84375$

S – знак

Рис. 3.15. Сохранение числа в формате "расширенное слово" (иллюстрация примера 2)

3.7.7. Представление данных при целочисленной арифметике

В процессорах с ФТ и *целочисленной* арифметикой *(см. разд. 3.7.5)* для представления исходных данных вещественного типа (вещественных чисел) используют их *целочисленные эквиваленты*. Они определяются в результате масштабирования, которое выполняется пользователем перед составлением программы. Масштабирование можно выполнить одним из двух способов:

- первый способ — из всей совокупности исходных данных вещественного типа выбирается максимальное по модулю число и приравнивается к максимально допустимому по модулю *целому* числу в формате "слово" (машинной единице); значение остальных исходных данных вещественного типа находится из соответствующей пропорции; дробная часть отбрасывается или округляется пользователем;
- второй способ — производится предварительное масштабирование данных вещественного типа так, чтобы они не превосходили 1 по модулю; затем осуществляется их масштабирование к машинной единице по первому способу.

На практике чаще масштабируют по второму способу, для которого в табл. 3.5 приведен пример соответствия чисел при длине слова 16 битов. Масштабировать следует к максимальному по модулю значению, в данном случае машинная единица равна 32 768.

Таблица 3.5. *Целочисленные эквиваленты при длине слова 16 битов*

Исходные данные алгоритма	Представление данных в программе	Шестнадцатеричный эквивалент в дополнительном коде
-2^{-31}	32 767	7FFF
0,5	16 384	4000
0	0	0000
−0,5	−16 384	C000
−1	−32 768	8000

При целочисленной арифметике исходные данные целого типа не масштабируются, если они не превосходят машинной единицы. Иначе масштабирование предусматривается пользователем.

При целочисленной арифметике результаты обработки (конечные данные) получаются также в виде целочисленных эквивалентов, поэтому, если необходимо знать истинные значения результатов, требуется выполнить процедуру обратного масштабирования — деление целочисленных эквивалентов на значение машинной единицы.

Пример 1. В процессоре (с ФТ, целочисленной арифметикой, формат представления исходных данных и результатов — слово длиной 16 битов) выполняется программа преобразования массива из 100 ($N = 100$) коэффициентов a_i (a_i — дробные числа) в другие, вычисленные по заданному алгоритму, 100 коэффициентов b_i. Дополнительно среди коэффициентов b_i рассчитывается количество L коэффициентов, значение которых удовлетворяет некоторому условию алгоритма. Рассмотрим:

1. Представление в программе (с помощью соответствующих директив языка ассемблера) исходных данных — коэффициентов a_i и их количества N.
2. Определение истинных значений коэффициентов b_i, а также количества L по рассчитанным в процессоре значениям.

Решение задачи иллюстрируется табл. 3.6. Итак:

1. В процессорах с целочисленной арифметикой при выполнении арифметических операций все числа воспринимаются как *целые*, поэтому дробные числа необходимо заменить их *целочисленными эквивалентами*, в примере

   $$a_i \to A_i,$$

   где A_i — целочисленный эквивалент a_i.

   Целочисленные эквиваленты A_i коэффициентов a_i (все коэффициенты по модулю меньше 1) находятся из пропорции

   $$1 — 32\,768$$

   $$a_i — A_i,$$

   откуда $A_i = 32\,768 \cdot a_i$, например, $A_0 = 32\,768 \cdot 0{,}57 = 18\,677{,}76$ или, после округления, $a_0 = 18\,678$ и т. д. (см. табл. 3.6). В программе истинные значения коэффициентов a_i заменяются их целочисленными эквивалентами A_i.

   Количество N — целое число, поэтому его целочисленный эквивалент определять не требуется.

2. В результате выполнения программы вычислены целочисленные эквиваленты B_i истинных коэффициентов b_i , а также значение L — целое число.

   Истинные значения коэффициентов b_i находятся по их целочисленным эквивалентам B_i из пропорции

   $$1 — 32768$$

   $$b_i — B_i,$$

   откуда $b_i = B_i / 32\,768$, например, $b_0 = B_0 / 32\,768 = 2457 / 32768 = 0{,}0749816$ (с точностью до 7 значащих цифр после запятой) и т. д. (см. табл. 3.6).

   Количество L — целое число, поэтому в результате вычислений выдается его истинное значение.

Таблица 3.6. *Истинные значения и целочисленные эквиваленты*

Исходные данные			Результаты вычислений		
Имя	**Истинное значение**	**Целочисленный эквивалент**	**Имя**	**Целочисленный эквивалент**	**Истинное значение**
a_0	0,57	18 678	b_0	2457	0,0749816
a_1	–0,13895	–4553	b_1	31 099	0,9490661
a_2	0,3	9830	b_2	–123	–0,0037536
…	…	…	…	…	…
a_{99}	0,85701	28 083	b_{99}	15 348	0,4683837
N	100	Не вычисляется	L	55	55

3.7.8. Представление данных при дробной арифметике

При *дробной* арифметике *(см. разд. 3.7.5)* значения исходных данных вещественного типа не масштабируются, если они не превосходят единицы по модулю, т. к. машинная единица, в отличие от ЦПОС с целочисленной арифметикой, равна 0,999... ≈ 1. Это основное преимущество дробной арифметики. Если хотя бы одно из значений вещественных данных превосходит 1 по модулю, пользователь должен предусмотреть предварительное масштабирование.

При дробной арифметике следует масштабировать значения исходных данных *целого* типа для получения их *дробных эквивалентов*. В этом случае максимальное по модулю целое число приравнивается к 1, а значения остальных данных целого типа находятся из пропорции.

Для конечных данных (результатов) целого типа, если необходимо знать истинные значения результатов, требуется выполнить процедуру обратного масштабирования. Следует иметь в виду, что в алгоритмах ЦОС объем вычислений с данными целого типа, как правило, невелик.

Пример 2. Задача, поставленная в примере 1 из *разд. 3.7.7*, решается в процессоре с *дробной* арифметикой; формат представления исходных данных и результатов — слово длиной 16 битов.

Решение задачи иллюстрируется табл. 3.7.

1. В процессорах с дробной арифметикой при выполнении арифметических операций все числа воспринимаются как дробные, поэтому, если все исходные данные вещественного типа по модулю меньше 1 (как в нашем случае) и их не надо заменять дробными эквивалентами (табл. 3.7).

Количество N — целое число, поэтому необходимо вычислить его дробный эквивалент, который в данном случае равен

$$N_{\text{экв}} = 0{,}999\ldots \approx 1,$$

поскольку целое число одно, его дробный эквивалент соответствует машинной единице $0{,}999\ldots \approx 1$.

Полученный дробный эквивалент $N_{\text{экв}}$ используется в программе вместо соответствующего целого числа N.

2. В результате выполнения программы вычислены истинные значения коэффициентов b_i (дробные числа, вопрос о точности вычислений не рассматривается) и *дробный эквивалент* целого числа $L_{\text{экв}} = 0{,}549$ (табл. 3.7); истинное значение L определяется из пропорции

$$1 — 100\ (N)$$

$$L_{\text{экв}} — L,$$

откуда $L = L_{\text{экв}} \cdot 100 = 0{,}5499 \cdot 100 = 54{,}99 \approx 55$ (см. табл. 3.7).

Таблица 3.7. *Истинные значения и дробные эквиваленты*

Исходные данные			Результаты вычислений		
Имя	**Истинное значение**	**Дробный эквивалент**	**Имя**	**Дробный эквивалент**	**Истинное значение**
a_0	0,57	Не вычисляется	b_0	Вычисляются сразу истинные значения коэффициентов	0,0749
a_1	–0,13895		b_1		0,9490
a_2	0,3		b_2		–0,0037
…	…	…	…	…	…
a_{99}	0,85701		b_{99}		0,4683
N	100	$0{,}999\ldots \approx 1$	L	0,549	55

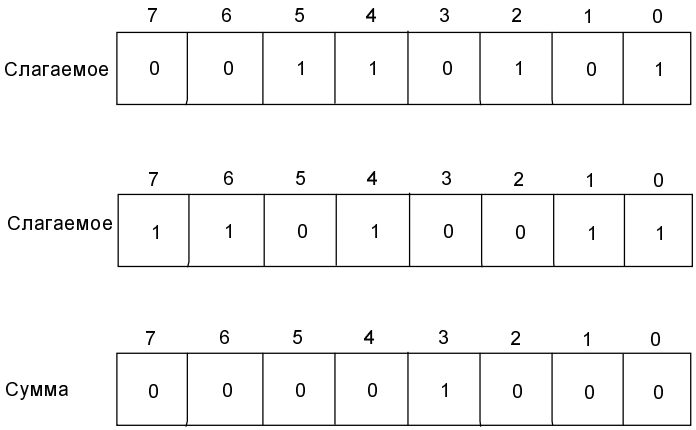
3.7.9. Арифметические операции в дополнительном коде

Основным преимуществом дополнительного кода является то, что все арифметические операции с числами независимо от их типа, целые или дробные, выполняются одинаково — как с *беззнаковыми числами*, когда знаковый разряд воспринимается как старший значащий. Результат арифметической операции представлен также в дополнительном коде.

Рассмотрим особенности выполнения операций сложения, вычитания и умножения при использовании целочисленной и дробной арифметик.

Сложение и вычитание

На рис. 3.16 приведен пример сложения чисел в дополнительном коде как беззнаковых чисел по правилам обычной двоичной арифметики. На этом же рисунке дается различная трактовка слагаемых и результата. Все числа представлены в формате "слово" длиной 8 битов.



Трактовка типа данных и типа арифметики

а) целые числа

$(2^5 + 2^4 + 2^2 + 2^0) + (-2^7 + 2^6 + 2^4 + 2^1 + 2^0) = 2^3 = 8$

б) дробные числа; дробная арифметика

$(2^{-2} + 2^{-3} + 2^{-5} + 2^{-7}) + (-2^0 + 2^{-1} + 2^{-3} + 2^{-6} + 2^{-7}) = 2^3 = 0{,}0625$

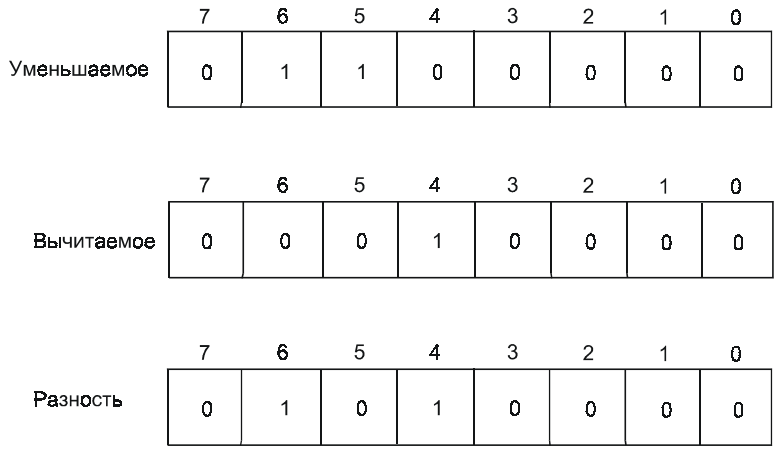
в) дробные числа; целочисленная арифметика; слагаемые и сумма – целые числа, которые масштабируются к машинной единице равной 2^{-7} (см. варианта)

$53/128 + (-45)/128 = 8/128 = 0{,}0625$

Рис. 3.16. Сложение в дополнительном коде

Как видно из рис. 3.16, операция сложения в дополнительном коде выполняется одинаково для целых и дробных чисел при целочисленной и дробной арифметиках.

На рис. 3.17 приведен пример вычитания чисел в дополнительном коде как беззнаковых чисел по правилам обычной двоичной арифметики. Трактовка исходных данных и результата аналогична рис. 3.16. Все числа представлены в формате "слово" длиной 8 битов.



Трактовка типа данных и типа арифметики

а) целые числа

$(2^6 + 2^5) - 2^4 = 96 - 16 = 80$

б) дробные числа; дробная арифметика

$(2^{-1} + 2^{-2}) - 2^{-3} = 0{,}625$

в) дробные числа; целочисленная арифметика;
уменьшаемое, вычитаемое и разность – целые числа,
которые масштабируются к машинной единице равной 2^7

$96/128 - 16/128 = 80/128 = 0{,}625$

Рис. 3.17. Вычитание в дополнительном коде

Проблемы переполнения при сложении и вычитании рассматриваются в *главе 4*.

Умножение

Как видно из рис. 3.16 и 3.17, при сложении и вычитании формат представления исходных данных и результата — одинаковый. На рис. 3.18 приведен пример умножения двух беззнаковых чисел, представленных в формате "слово" длиной 4 бита в дополнительном коде. Произведение этих чисел должно иметь длину $4 \cdot 2 - 1 = 7$ битов без потери точности.

Алгоритмы умножения, реализованные в различных процессорах, отличаются от общепринятого в двоичной арифметике; один из возможных алгоритмов будет рассмотрен далее. Сначала обсудим *результат умножения*.

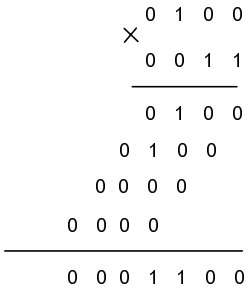


Рис. 3.18. Умножение в дополнительном коде

Без потери точности произведение двух беззнаковых чисел, представленных в формате "слово" длиной n битов, должно иметь длину $(2n - 1)$ битов. В процессорах с ФТ результат умножения размещается либо в формате "двойное слово" MSP:LSP длиной $2n$ битов, либо в MSP:LSP части расширенного слова также длиной $2n$. Отсюда возникает проблема "лишнего" бита. Кроме того, появляется вопрос о сохранении вычисленного произведения в формате "слово" длиной n битов — какую из частей произведения MSP или LSP следует сохранять? Выясним эти моменты на одном примере умножения (рис. 3.18) при различных трактовках типа данных и различных арифметиках в процессоре. Рассмотрим следующие четыре ситуации:

1. Умножение *целых* чисел при *целочисленной* арифметике (рис. 3.19):
   - "лишний" бит используется как *старший бит* MSB в слове MSP:LSP для хранения *расширения знака* результата;

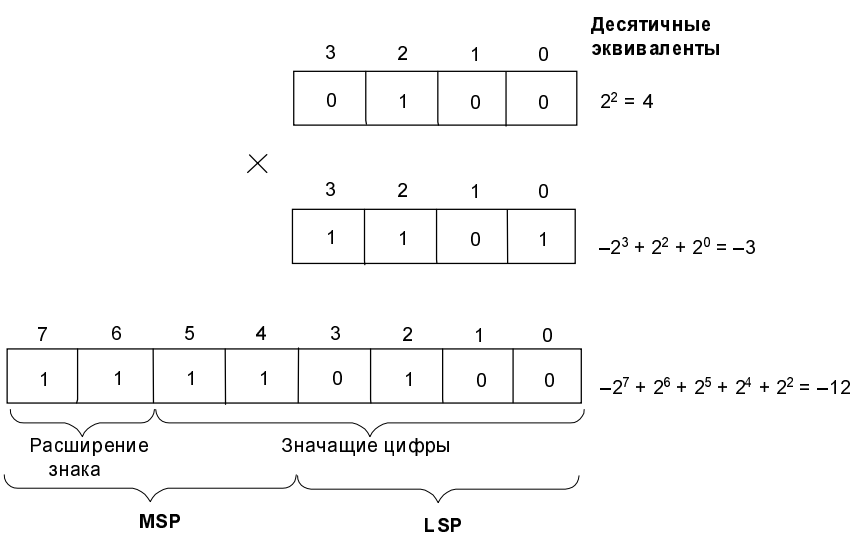


Рис. 3.19. Умножение целых чисел при целочисленной арифметике

- *знаковыми* становятся *два* старших бита, *значащими* — остальные биты;
- если длина каждого из сомножителей не превышает $n/2$, результат размещается в младшем слове LSP, и его можно сохранить в формате "слово" *без потерь*; старшее слово MSP служит для расширения знака;
- если длина сомножителей превышает $n/2$, для хранения результата потребуется *два* слова и специальная организация дальнейшей обработки.

2. Умножение *дробных* чисел при *целочисленной* арифметике; в этом случае дробные числа заменяются их *целочисленными* эквивалентами (рис. 3.20):
   - "лишний" бит используется, как *старший бит* MSB в слове MSP:LSP для хранения *расширения знака* результата;
   - *знаковыми* становятся *два* старших бита, *значащими* — остальные биты;
   - в формате "слово" сохраняется *старшее* слово MSP результата, младшее слово LSP отбрасывается; перед выполнением дальнейших операций, для устранения лишнего знакового бита, содержимое двойного слова MSP:LSP следует сдвинуть на один бит влево.
3. Умножение *дробных* чисел при *дробной* арифметике (рис. 3.21):
   - "лишний" бит используется, как *младший бит* LSB в слове MSP:LSP для хранения нуля; LSB = 0;
   - знаковым остается *один* старший бит, *значащими* — остальные биты;
   - для хранения результата в формате "слово" выбирается *старшее* слово MSP, младшее слово LSP отбрасывается.
4. Умножение *целых* чисел при *дробной* арифметике; в этом случае целые числа заменяются их *дробными* эквивалентами (рис. 3.22):
   - "лишний" бит используется, как *младший бит* LSB в слове MSP:LSP для хранения нуля; LSB = 0;
   - знаковым остается *один* старший бит, *значащими* — остальные биты *без младшего бита LSB*; после сдвига вправо на 1 бит получается результат точно такой же, как целое число при целочисленной арифметике — с расширением знака в двух старших битах слова MSP:LSP (сравните с рис. 3.19).

Отметим, что отсутствие "лишнего" знакового бита в представлении результата умножения дробных чисел при дробной арифметике, наряду с отсутствием необходимости масштабирования данных, считается преимуществом дробной арифметики.

Теперь рассмотрим один из наиболее распространенных алгоритмов умножения, реализованный в большинстве процессоров с ФТ.

❒ *Алгоритм умножения* двух беззнаковых двоичных чисел x и y, представленных в дополнительном коде в формате "слово" длиной n.

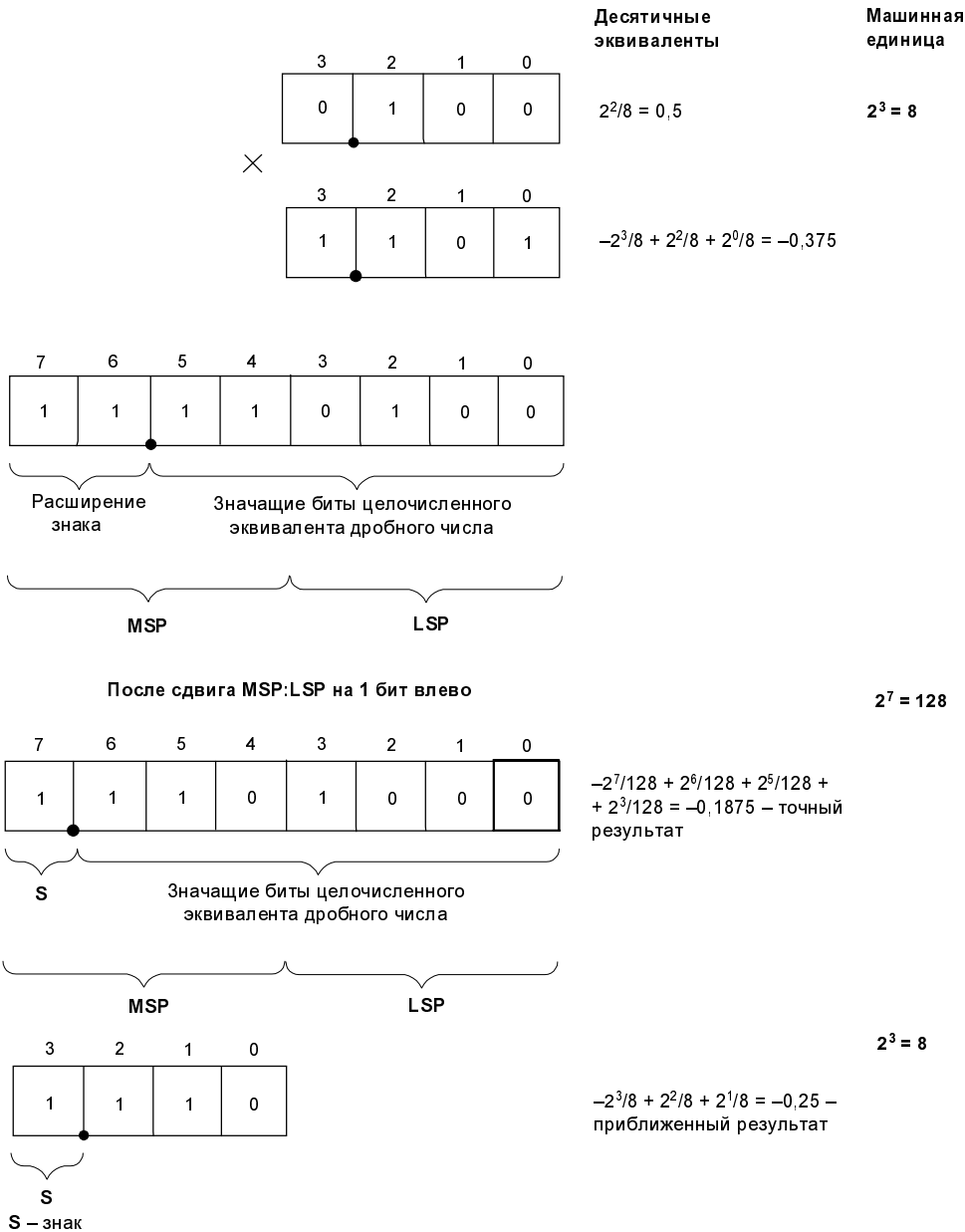


Рис. 3.20. Умножение дробных чисел при целочисленной арифметике

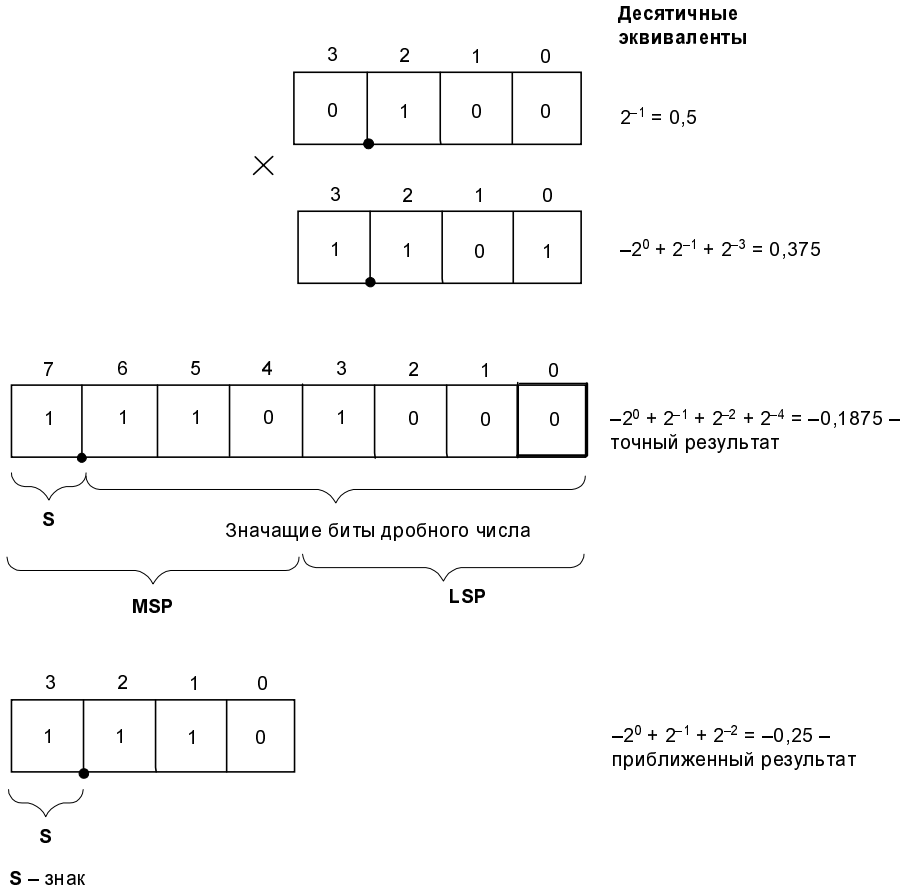


Рис. 3.21. Умножение дробных чисел при дробной арифметике

$$x = x_0\, x_1 \ldots x_i \ldots x_{n-1};$$
$$y = y_0\, y_1 \ldots y_i \ldots y_{n-1},$$

где x_i и y_i — значения i-х битов,

включает следующие шаги:

1. *Подготовка к циклу*:
   - $i = n - 1$;
   - начальная сумма локальных произведений $s_{i+1} = 0 \ldots 0$ (формат "слово" длиной n).
2. *Тело цикла*:
   - вычисляется сумма локальных произведений $s_i = s_{i+1} + y \cdot x_i$:

     если $x_i = 0$, значение s_i не меняется;

если $x_i = 1$, к значению s_i прибавляется y; при этом оба слагаемых выровнены по *левому* краю;

- выполняется расширение знака суммы s_i на один бит;
- присваивается $i = i - 1$.

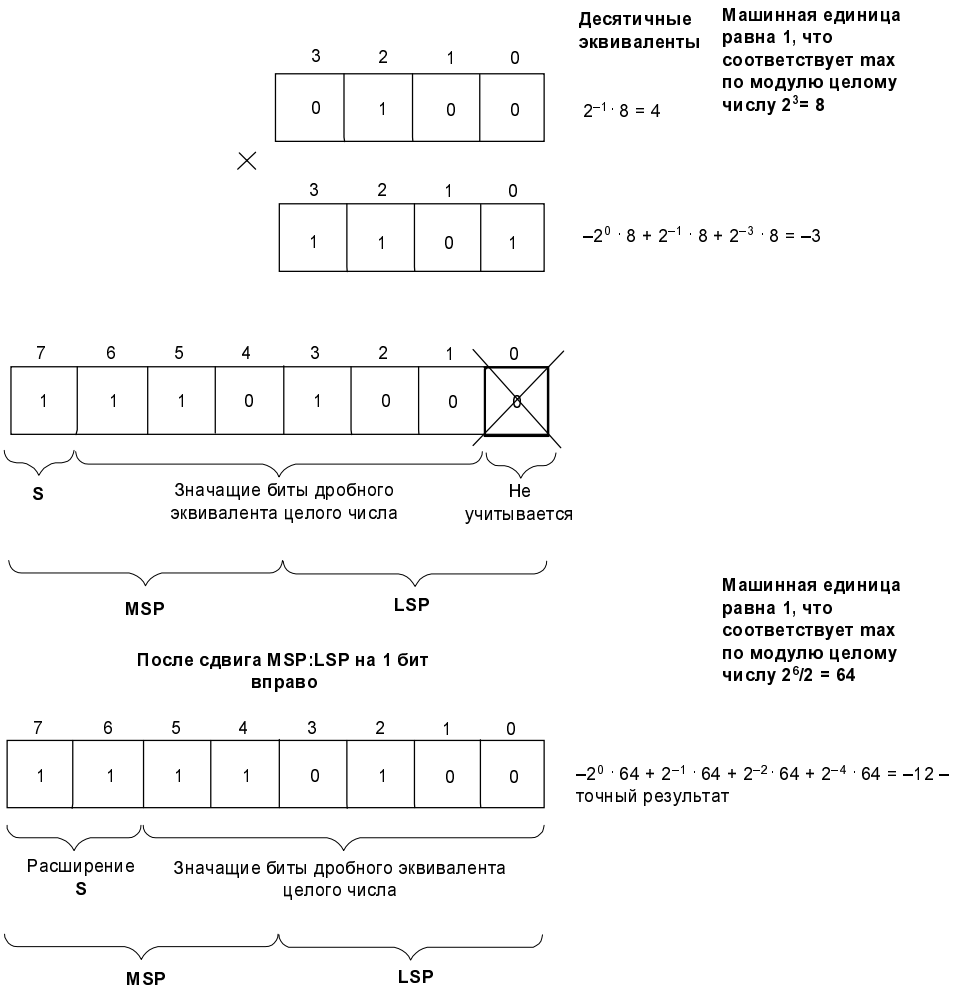


Рис. 3.22. Умножение целых чисел при дробной арифметике

3. *Проверка окончания цикла*:

- цикл повторяется до $i = 1$ включительно.

4. *Выход из цикла*:

- при $i = 0$ вычисляется разность $s_0 = s_1 - y \cdot x_0$:

  если $x_0 = 0$, значение s_0 не меняется;

  если $x_0 = 1$, из значения s_0 вычитается y; при этом вычитаемое и уменьшаемое выровнены по левому краю;

- разность s_0 равна произведению беззнаковых чисел x и y в дополнительном коде и имеет длину $(2n - 1)$ битов, где старший бит — знаковый;

- *конечный* результат формируется в формате "двойное слово" длиной $2n$; при этом "лишний" бит используется по-разному в зависимости от типа арифметики, а именно:

  "лишний" бит применяется, как старший бит MSB для расширения знака, при целочисленной арифметике;

  "лишний" бит играет роль младшего бита LSB (LSB = 0) при дробной арифметике.

Замечание

При вычислении в цикле суммы s_i локальных произведений переносы влево от старшего разряда игнорируются.

Приведем пример умножения в дополнительном коде двух чисел x и y длиной $n = 4$ бит:

$$x = x_0\, x_1\, x_2\, x_3 = 1100;$$

$$y = y_0\, y_1\, y_2\, y_3 = 1101.$$

Циклический алгоритм умножения показан в табл. 3.8.

Таблица 3.8. *Циклический алгоритм умножения*

Этап	Результат
Подготовка к циклу	
$i = n - 1 = 4 - 1 = 3$;	$s_4 = 0000$
Тело цикла	
$s_3 = s_4 + y \cdot x_3$	
Так как $x_3 = 0$	$s_3 = s_4 = 0000$
Расширение знака s_3	$s_3 = \mathbf{0}0000$
$i = i - 1 = 3 - 1 = 2$	
$s_2 = s_3 + y \cdot x_2$	
Так как $x_2 = 0$	$s_2 = s_3 = 00000$
Расширение знака s_2	$s_3 = \mathbf{0}00000$

Таблица 3.8 *(окончание)*

Этап	Результат
$i = i - 1 = 2 - 1 = 1$	
$s_1 = s_2 + y \cdot x_1$	
Так как $x_1 = 1$	$s_1 = 000000$
	$+ 1101$
	$s_1 = 110100$
Расширение знака s_1	$s_1 = \mathbf{1}110100$
$i = i - 1 = 1 - 1 = 0$	
Выход из цикла	
$s_0 = s_1 - y \cdot x_0$	
Так как $x_0 = 1$	$s_0 = 1110100$
	$- 1101$
	$s_0 = 0001100$
Результат при целочисленной арифметике	**0**0001100
Результат при дробной арифметике	0001100**0**

Из приведенного примера видно, что *операция умножения* выполняется *одинаково — как с беззнаковыми числами*, независимо от типа данных и арифметики, различие проявляется только на этапе сохранения конечного результата в формате "двойное слово" длиной $2n$. Трактовка типа результата — целые или дробные числа — возлагается на пользователя.

3.7.10. Преобразование форматов в ЦПОС с фиксированной точкой

Преобразование форматов происходит при различного рода пересылках данных (исходных, промежуточных и конечных) из ячеек памяти в регистры и наоборот, когда изменяется формат ("слово", "длинное слово", "расширенное слово"). Наиболее общие правила преобразований форматов показаны на рис. 3.23—3.25. В качестве примеров выбраны форматы: слово длиной 4 бита, двойное слово — 8 битов, расширенное слово — 10 битов. Приведенные правила легко распространяются на любую длину слов.

Рис. 3.23. Преобразование форматов представления целых чисел при целочисленной арифметике

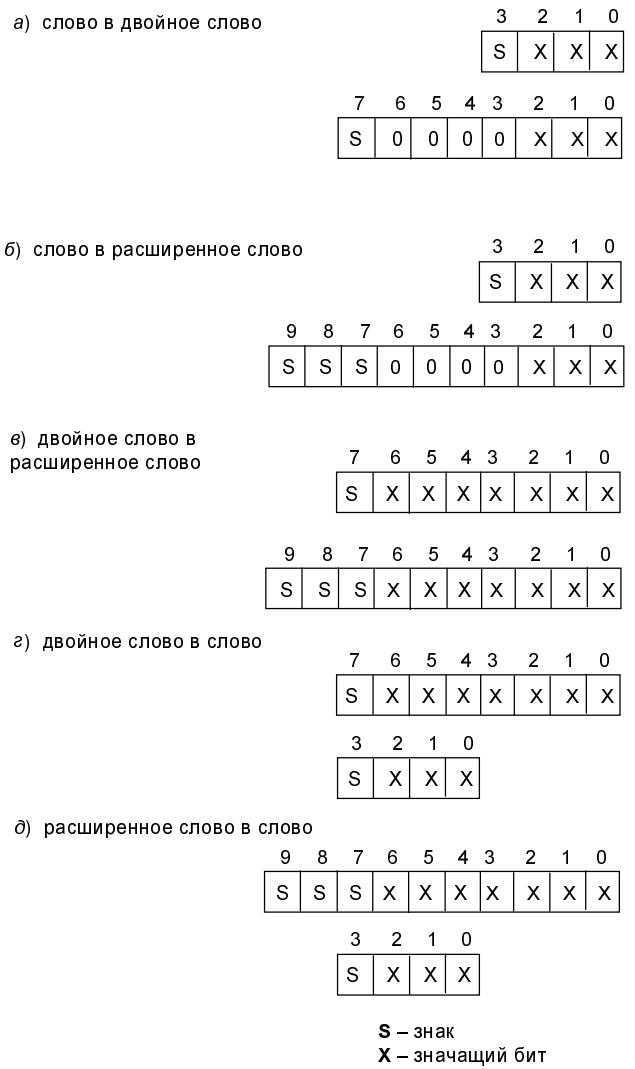


Рис. 3.24. Преобразование форматов представления дробных чисел при целочисленной арифметике

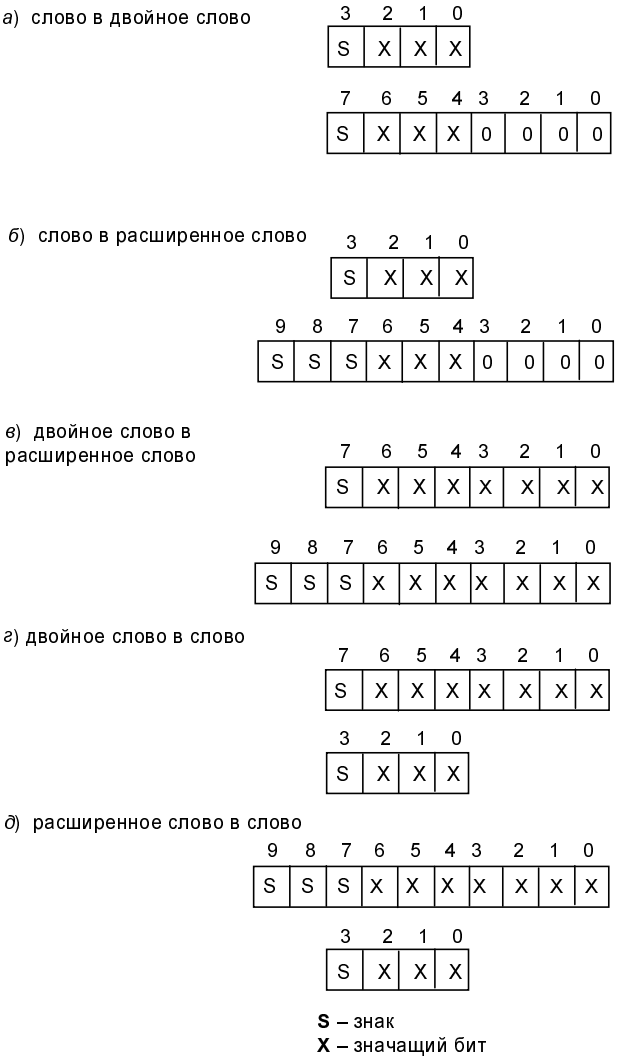


Рис. 3.25. Преобразование форматов представления дробных чисел при дробной арифметике

3.7.11. Диапазон, динамический диапазон и точность представления чисел с фиксированной точкой

Диапазон представления чисел устанавливает границы между минимально и максимально допустимыми значениями, представляемыми в заданном формате и коде.

Динамический диапазон ДД определяют как:

$$\text{ДД} = |\text{max значение}| \;/\; |\text{ min значение} \neq 0|,$$

где |max значение| и |min значение ≠ 0| представлены в заданном формате и коде.

Динамический диапазон ДД в децибелах равен

$$\text{ДД (дБ)} = 20 \lg (\text{ДД}).$$

Точность представления чисел определяет *максимально допустимую точность* представления *дробной* части вещественных чисел.

Напомним, что в двоичной системе счисления при заданном формате *целые* числа (в том числе, целые части вещественных чисел) представляются *точно*, а *дробные — приближенно*. Максимально допустимая *ошибка* при представлении дробных чисел равна:

- $2^{-b-1} = 2^{-(b+1)} = 2^{-m}$ — половине младшего значащего бита (половине шага квантования) при *округлении*;
- 2^{-b} — младшему значащему биту (шагу квантования) при *усечении*,

где b — количество значащих битов; $n = b - 1$; m — длина слова, в котором представляется число, например, для слова $m = n$, для двойного слова $m = 2n$.

Точность представления измеряется в битах и определяется как:

$$\log_2(|\text{max значение}| \;/\; |\text{ max ошибка при округлении}|),$$

где |max значение| соответствует:

- для *дробных* чисел — максимальному (по модулю) значению, представленному в заданном формате и коде;
- для *смешанных* чисел — максимальному (по модулю) значению дробной части числа, представленной в заданном формате и коде.

Определим диапазон, динамический диапазон и точность представления данных различных типов.

- Данные *целого* типа.
  - *Целые числа со знаком.*

Диапазон представления целых чисел со знаком в формате "слово" длиной n в дополнительном коде равен

$$-2^b \le C \le 2^b - 1,$$

где C — значение целого числа.

Динамический диапазон ДД равен

$$\text{ДД} = |-2^b| \,/\, |1| = 2^b.$$

Например, при длине слова 16 битов *диапазон* равен

$$-2^{15} \le C \le 2^{15} - 1,$$

а динамический диапазон

$$\text{ДД} = 2^{15};$$

или в децибелах

$$\text{ДД (дБ)} = 20 \lg (2^{15}) \approx 90{,}3 \text{ дБ}.$$

Аналогично можно определить диапазон представления и динамический диапазон для чисел в формате "двойное слово" длиной $2n$. Например, при длине двойного слова 32 бита *диапазон* представления равен

$$-2^{31} \le C \le 2^{31} - 1,$$

а динамический диапазон

$$\text{ДД} = 2^{31}$$

или в децибелах

$$\text{ДД (дБ)} = 20 \lg (2^{31}) \approx 186{,}6 \text{ дБ}.$$

- *Беззнаковые числа.*

  Диапазон представления целых беззнаковых чисел в формате слово длиной n в дополнительном коде *вдвое* больший, чем для целых со знаком (знаковый бит включен в число значащих) и равен:

$$0 \le C \le 2^n - 1.$$

❒ Данные *вещественного* типа.

- *Дробные числа.*

  Диапазон представления дробных чисел не зависит от типа арифметики (целочисленная/дробная). В формате "слово" длиной n (формат Qb) в дополнительном коде диапазон равен

$$-1 \le C \le 1 - 2^{-b},$$

где C — значение дробного числа.

Модуль дробного числа не превышает 1 независимо от длины формата.

Замечание

При целочисленной арифметике не следует путать диапазоны представляемых дробных чисел и их целочисленных эквивалентов. Диапазон представления целочисленных эквивалентов такой же, как у целых чисел.

Динамический диапазон ДД, равный

$$\text{ДД} = |-1| / |2^{-b}| = 2^b,$$

у дробных чисел точно такой же, как у целых чисел при одинаковом формате и коде. Например, при длине слова 16 битов, *диапазон* равен

$$-1 \le C \le 1 - 2^{-15},$$

а динамический диапазон

$$\text{ДД} = 2^{15}$$

или в децибелах

$$\text{ДД (дБ)} \approx 90{,}3 \text{ дБ}.$$

Аналогично можно определить диапазон представления и динамический диапазон для дробных чисел в формате "двойное слово" длиной $2n$. Например, при длине двойного слова 32 бита *диапазон* представления равен

$$-1 \le C \le 1 - 2^{-31},$$

а динамический диапазон

$$\text{ДД} = 2^{31}$$

или в децибелах

$$\text{ДД (дБ)} \approx 186{,}6 \text{ дБ}.$$

Согласно определению, *точность* представления дробного числа, например, при длине слова 16 битов, равна

$$\log_2(|-1| / |2^{-n}|) = 16 \text{ бит}$$

при длине слова 32 бита — 32 битам и т. д.

Подчеркнем, что определяемая таким образом точность является *максимально допустимой*. Числа, меньшие максимального значения (меньшие 1 по модулю), представляются с большей точностью.

- *Смешанные числа.*

Наличие расширения EXT в слове позволяет при *внутренних* вычислениях хранить *смешанные* числа. *Диапазон* их представления в дополнительном коде равен

$$-2^{\text{EXT}} \le C \le 2^{\text{EXT}} - 2^{-(k - \text{EXT} - 1)},$$

где C — значение смешанного числа; k — длина расширенного слова, равная длине смешанного числа со знаком; EXT — длина расширения

EXT, равная длине целой части числа; $(k - EXT - 1)$ — длина дробной части числа.

Динамический диапазон равен

$$ДД = | -2^{EXT} | / | 2^{-(k - EXT - 1)}| = 2^{(k - 1)}.$$

Точность представления дробной части смешанного числа в соответствии с определением равна

$$\log_2(| -1 | / | 2^{-(k - EXT)} |) = (k - EXT) \text{ бит},$$

где $2^{-(k - EXT)} = 2^{-(k - EXT - 1) - 1}$ — максимально допустимая ошибка при округлении дробной части числа, равная половине младшего значащего бита.

Например, для расширенного слова длиной $k = 56$ битов (слово аккумулятора в процессорах семейств DSP560xx фирмы Motorola) и длине расширения EXT 8 битов *диапазон* представления равен

$$-2^8 \le C \le 2^8 - 2^{-47};$$

динамический диапазон

$$ДД = 2^{55}$$

или в децибелах

$$ДД \text{ (дБ)} \approx 331 \text{ дБ};$$

точность представления дробной части числа равна $(56 - 8) = 48$ бит.

3.7.12. Увеличение динамического диапазона и точности представления данных в ЦПОС с фиксированной точкой

В ЦПОС с ФТ, кроме представления данных с удвоенной точностью в формате "двойное слово", существует дополнительная возможность увеличения динамического диапазона и точности. Для этого пользователь *моделирует* увеличенный формат данных, например, для хранения *одного* значения отводит *два* двойных слова. При выполнении арифметических операций с такими данными нельзя использовать обычные команды процессора, требуется специальная программная организация выполнения соответствующих операций, что, разумеется, снижает скорость их выполнения. Подобное представление данных предполагает отдельное хранение младшей и старшей частей числа. Для программной организации выполнения арифметических операций используется специальный бит в регистре состояния — бит переноса C (Carry bit). В нем сохраняется 1 — при *переносе* старшего бита младшей части слова в младший бит старшей части слова (в операции сложения), либо генерируется 1 — при *заеме* из младшего бита старшей части слова в старший бит младшей части слова.

Другой способ увеличения динамического диапазона и точности представления данных основан на программной организации представления данных в форме с ПТ в ЦПОС с ФТ *(см. разд. 3.10)*.

3.7.13. Упакованные данные

В табл. 3.4 символом ** отмечены процессоры TMS320C62xx/64xx фирмы Texas Instruments, поддерживающие обработку упакованных данных.

Упакованные данные представляют собой группу данных формата "полуслово" или "байт" (все данные в группе имеют одинаковый формат), сохраняемых в формате "слово". В процессорах TMS320C62xx/64xx слово имеет длину 32 бита, что позволяет последовательно расположить в этом формате (упаковать) пару данных формата "полуслово" (2×16) или 2 пары данных формата байт (4×8, только в TMS320C64xx). Упакованные данные являются локально замкнутыми, взаимные переносы или заемы битов между ними запрещены. Упаковка данных позволяет с помощью одной команды одновременно выполнить одинаковую операцию (например, сложение) отдельно со всеми данными, упакованными в формате "слово". Примеры упакованных данных и выполнение над ними операции сложения *см. в разд. 6.3.2.*

3.8. Представление данных с плавающей точкой

Во всех процессорах с ПТ поддерживается представление данных с ФТ и ПТ. При этом, в форме с ФТ, как правило, представляются целые числа, а в форме с ПТ — только вещественные. Соответственно, все команды обработки данных разделены на оперирующие с числами с ФТ или ПТ. Представление чисел с ФТ рассмотрено выше и в этом разделе не обсуждается.

Данные *вещественного типа* (вещественные числа) представляются в форме с ПТ и отображают алгебраическую показательную форму представления числа — с умножением на $10^{\pm n}$, где n — *целое*. Например, число 15,17593, имеющее однозначное алгебраическое представление в обычной форме, в показательной форме представляется неоднозначно:

- $1,517593 \cdot 10^{+1}$;
- $1517,593 \cdot 10^{-2}$;
- $15,17593 \cdot 10^{0}$;
- $0,1517593 \cdot 10^{+2}$;
- $151,7593 \cdot 10^{-1}$;
- $0,01517593 \cdot 10^{+3}$

и т. д. Количество вариантов бесконечно. Число 10 называют *основанием* системы счисления.

Аналогично, в двоичной системе счисления вещественные числа, представленные в форме с ПТ, отображают алгебраическую показательную форму

представления двоичного числа — с умножением на $2^{\pm n}$, где n — *целое*. Например, двоичное число 101,01101 можно представить как:

- $10{,}101101 \cdot 2^{+1}$;
- $101{,}01101 \cdot 2^{0}$;
- $1010{,}1101 \cdot 2^{-1}$;
- $10101{,}101 \cdot 2^{-2}$;
- $1{,}0101101 \cdot 2^{+2}$;
- $0{,}10101101 \cdot 2^{+3}$

и т. д.

В общем случае алгебраическая форма представления двоичного вещественного числа с ПТ имеет вид:

$$C = \mu \cdot 2^{E}, \tag{3.1}$$

где C — вещественное двоичное число, представленное в форме с ПТ (в дальнейшем для краткости — число с ПТ); μ — мантисса — вещественное двоичное число со знаком, представленное в форме с ФТ; E — порядок — целое двоичное число со знаком; 2 — основание двоичной системы счисления.

Для устранения неоднозначности и упрощения арифметики чисел с ПТ из всех вариантов возможных представлений выбирают один, который называется *нормализованной формой* числа с ПТ. В ЦПОС нормализованная форма соответствует такому представлению двоичного числа с ПТ, мантисса которого всегда (за исключением числа 0) содержит единицу в целой части.

В цифровой технике часто встречается другая нормализованная форма числа с ПТ, когда целая часть мантиссы равна нулю, а первая значащая цифра дробной части отлична от нуля.

Принятая в ЦПОС нормализация чисел с единицей в целой части мантиссы позволяет при заданном формате увеличить количество значащих цифр вещественного числа на одну, т. к. бит в целой части мантиссы является неявным и физически не хранится (см. далее).

С учетом сказанного, в ЦПОС форма представления чисел с ПТ принимает вид:

$$C = (-1)^{S} \cdot 2^{E} \cdot 1, f, \tag{3.2}$$

где C — двоичное число с ПТ; S — знак числа (0 — плюс, 1 — минус); $1, f$ — мантисса — вещественное двоичное число без знака, представленное в форме с ФТ: 1 — целая часть мантиссы (неявно присутствующая), f — дробная часть мантиссы; E — порядок — целое двоичное число со знаком; 2 — основание двоичной системы счисления.

3.8.1. Стандарт IEEE 754 представления данных с плавающей точкой

На представление данных с ПТ существует единый промышленный стандарт IEEE 754, разработанный в Институте инженеров по электротехнике и

электронике (Institute of Electrical and Electronics Engineers) в США в 1985 г. Он, в частности, регламентирует:

- форму представления чисел;
- форматы данных;
- представление нормализованных чисел;
- представление специальных данных;
- особые случаи.

Стандарт IEEE 754 полностью поддерживается сигнальными процессорами DSP9600x фирмы Motorola, процессорами TMS320C67xx фирмы Texas Instruments и практически полностью — процессорами ADSP-21xxx фирмы Analog Devices. Процессорами TMS320C3x стандарт IEEE 754 не поддерживается; обеспечить стандартное внешнее представление данных для совместимости этого процессора с другими устройствами (использующими стандарт) можно двумя способами: программно — с помощью специальной весьма простой подпрограммы, или аппаратно — с помощью приставки в виде простейшей интегральной схемы.

3.8.2. Форма представления данных с плавающей точкой

Согласно стандарту IEEE 754 форма представления чисел имеет вид:

$$C = (-1)^S \cdot 2^e \cdot 1, f, \qquad (3.3)$$

где C — двоичное число с ПТ; S — знак числа; $1, f$ — мантисса: 1 — целая часть мантиссы (неявно присутствующая), f — дробная часть мантиссы; $e = (E + bias)$ — *смещенный порядок* — целое положительное число; *bias* — смещение — целая положительная константа; 2 — основание двоичной системы счисления.

Отличие стандартной формы представления чисел (3.3) от (3.2) состоит в представлении порядка со смещением. Значение смещения *bias* выбирается так (см. ниже), чтобы смещенный порядок e всегда имел положительное значение. Это позволяет быстро сравнивать два вещественных числа одинакового формата и знака. При побитовом сравнении порядков слева направо (как беззнаковых целых) первое же отличие битов определяет соотношение чисел, которое используется в арифметике с ПТ.

Замечание

Нестандартная форма записи (3.2) применяется, например, в процессорах TMS320C3x фирмы Texas Instruments.

3.8.3. Форматы данных с плавающей точкой

Предназначение трех разновидностей форматов данных ("слово", "двойное слово", "расширенное слово"), приведенное в *разд. 3.5*, справедливо для ЦПОС с ПТ. Эти форматы имеют следующие наименования:

- SP (Single Precision floating-point format) — формат для представления данных с одинарной точностью — соответствует *слову*;
- DP (Double Precision floating-point format) — формат для представления данных с двойной точностью — соответствует *двойному слову*;
- SEP (Single Extended Precision floating-point format) — формат для представления результатов промежуточных и конечных вычислений с *расширенной одинарной* точностью — соответствует *расширенному слову* с длиной $n < k < 2n$; формат SEP расширяет диапазон и точность представления данных.

Форматы SP и DP называют *базовыми*; их длины и структуры стандартизированы, т. к. они используются для внешнего представления данных в памяти и должны быть совместимы с устройствами, поддерживающими стандарт IEEE 754. К расширенному формату SEP, используемому для внутреннего представления данных, предъявляются менее жесткие требования.

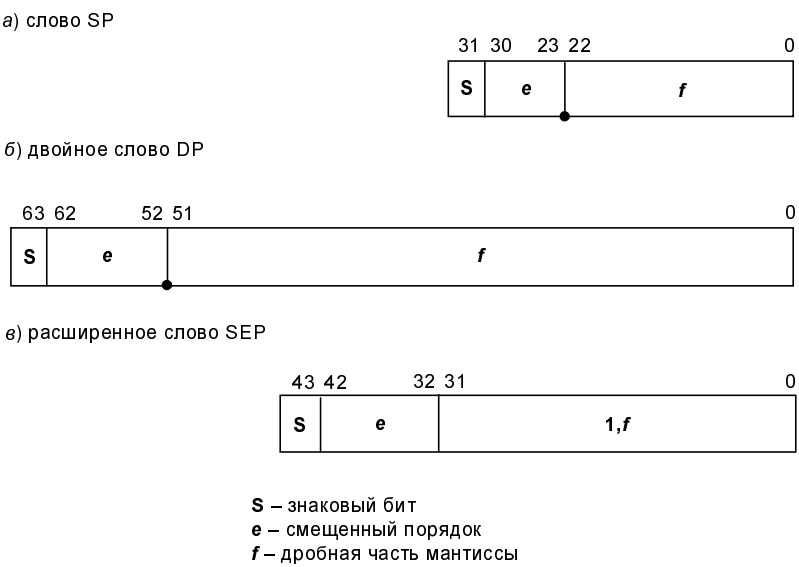


Рис. 3.26. Структура форматов SP, DP и SEP

Структура форматов SP, DP (рис. 3.26, *а*, *б*) одинакова и обусловлена формой представления чисел с ПТ в виде (3.3). В ней выделено три поля, размещенных согласно стандарту IEEE 754, в строго фиксированной последовательности:

- поле знака — для представления знака S числа (1 бит);
- поле смещенного порядка — для представления значения смещенного порядка e;
- поле мантиссы — для представления дробной части мантиссы $0, f$; целая часть мантиссы, равная 1, присутствует *неявно*.

В структуре формата SEP (рис. 3.26, *в*) целая часть мантиссы существует *явно*, что объясняется удобством выполнения операций.

В табл. 3.9 приведены параметры для форматов SP, DP (стандарт IEEE 754) и SEP (для DSP9600x фирмы Motorola).

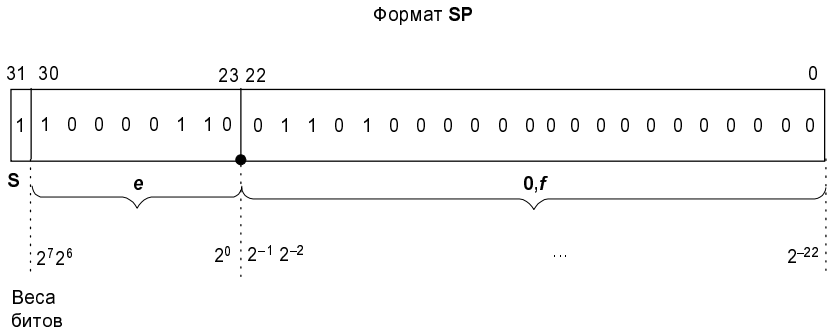
Таблица 3.9. *Параметры форматов SP, DP и SEP*

Параметр	Формат		
	SP	**DP**	**SEP**
Длина	32	64	44
Длина мантиссы 1, f	24: Дробная часть — 23 Неявная 1	53: Дробная часть — 52 Неявная 1	32 Дробная часть — 31 Явная 1
Длина смещенного порядка e	8	11	11
Смещение *bias*	127	1023	1023
E_{min}	−126	−1022	−1022
E_{max}	+127	+1023	+1023
e_{min}	+1	+1	+1
e_{max}	+254	+2046	+2046

Стандарт IEEE 754 *не поддерживает* представление чисел с ПТ (3.3) в *дополнительном коде*.

На рис. 3.27 дается пример вычисления десятичного эквивалента числа, представленного в форме (3.3) в прямом коде, по формуле

$$C_{(10)} = (-1)^S \cdot 1, f_{(10)} \cdot 2^{E(10)}. \qquad (3.4)$$



$$C_{(10)} = (-1)^S \cdot 1,f_{(10)} \cdot 2^{E(10)}$$

$bias = 127_{(10)} = 1111111_{(2)}$

$E_{(2)} = e - bias = 10000110 - 01111111 = 00000111$

$E_{(10)} = 7$

$1,f_{(2)} = 1.01101000000000000000000$

$1,f_{(10)} = 2^0 + 2^{-2} + 2^{-3} + 2^{-5} = 1,40625$

$C_{(10)} = (-1)^1 \cdot 1,40625 \cdot 2^7 = -180 = -1,8 \cdot 10^2$

Рис. 3.27. Пример определения десятичного эквивалента числа с ПТ

3.8.4. Преобразование форматов в ЦПОС с плавающей точкой

Преобразования форматов SP, SEP и DP выполняется в специальном устройстве — преобразователе форматов, предусмотренном в аппаратном обеспечении ЦПОС. При этом соблюдаются следующие общие правила:

- целые значения смещенного порядка выравниваются по правому краю; "лишние" старшие разряды обнуляются;
- значения дробной части мантиссы выравниваются по левому краю (в слове SEP после явной 1); "лишние" младшие разряды обнуляются.

Так как ЦПОС с ПТ поддерживают целочисленную арифметику с ФТ, то, кроме преобразований форматов вещественных данных, представленных в форме с ПТ, существует возможность взаимного преобразования форматов данных, представленных в формах с ФТ и ПТ. В зависимости от архитектуры преобразователя форматов, вариации таких преобразований разнообразны, однако их принцип будет ясен из следующих двух примеров.

Пример 1. Преобразование вещественного числа с ПТ в целое число с ФТ. Числа с ПТ и ФТ имеют одинаковый формат SP.

Сначала в преобразователе форматов производится проверка на особые случаи *(см. разд. 3.8.6)*. Если число с ПТ является нормализованным и не фиксируется особый случай, преобразование форматов выполняется в расши-

ренном формате SEP, где вещественное число с ПТ округляется до целого в соответствии с установленным режимом округления *(см. главу 4)* и затем представляется в формате SP в форме с ФТ с выравниванием по правому краю в дополнительном коде. Если имеет место особый случай, он отображается состоянием соответствующих битов в регистре состояния и обрабатывается в установленном порядке. В частности, если значение числа с ПТ выходит за границы диапазона представления целых чисел в форме с ФТ, т. е. имеет место особый случай переполнения, целое число округляется по правилам арифметики насыщения чисел с ФТ *(см. главу 4)*.

Пример 2. Преобразование целых чисел в форме с ФТ к форме с ПТ. Данные с ФТ имеют формат SP, а с ПТ — формат SEP.

В преобразователе форматов выполняется аналогичная проверка на особые случаи, после чего целое число в форме с ФТ переводится из дополнительного кода в прямой и сохраняется в форме с ПТ в формате SEP. В отличие от предыдущего примера, особого случая "переполнение" здесь произойти не может, т. к. диапазон представления чисел с ПТ существенно шире, чем с ФТ *(см. разд. 3.8.8)*.

3.8.5. Нормализованные числа

Используя параметры табл. 3.9, запишем максимальные и минимальные числа в форме с ПТ (3.3) для формата "слово" SP (табл. 3.10). Аналогично можно получить значения для форматов "двойное слово" DP и "расширенное слово" SEP. Нормализованные числа (кроме нуля) находятся в пределах между значениями max и min.

Таблица 3.10. *Максимальные и минимальные числа с ПТ*

Число с ПТ	Двоичное число в формате SP					Десятичный эквивалент
	Знак	Порядок			Мантисса	
	S	$E_{(10)}$	$e_{(10)}$	$e_{(2)}$	$1, f_{(2)}$	
Положительные числа						
Max	0	127	254	11…10	1,11…1	$(2-2^{-23})\cdot 2^{127}\approx 3{,}4\times\times 10^{38}$
Min	0	−126	1	00…01	1,00…0	$1\cdot 2^{-126}\approx 1{,}18\times 10^{-38}$
Отрицательные числа						
Min	1	127	254	111…10	1,11…1	$-(2-2^{-23})\cdot 2^{127}\approx --3{,}4\cdot 10^{38}$
Max	1	−126	1	00…01	1,00…0	$-1\cdot 2^{-126}\approx -1{,}18\times\times 10^{-38}$

Таблица 3.10 *(окончание)*

Число с ПТ	Двоичное число в формате SP					Десятичный эквивалент
	Знак	Порядок			Мантисса	
	S	$E_{(10)}$	$e_{(10)}$	$e_{(2)}$	$1, f_{(2)}$	
Отрицательные числа						
Длина	1	—	—	8	24: Дробная — 23 Целая — 1 (неявная)	—

3.8.6. Специальне данные

Кроме рассмотренных выше представлений нормализованных чисел стандарт IEEE 754 дополнительно поддерживает следующие *специальные*, свойственные только форме с ПТ, представления данных:

- ненормализованные числа;
- нули;
- бесконечности;
- нечисла.

Поясним отдельно особенности каждого из представлений на примере формата "слово" SP (на форматы DP и SEP их распространить несложно).

Ненормализованные числа соответствуют значению смещенного порядка ($e_{\min} - 1$), мантиссе $0, f \neq 0{,}000\ldots0$ и знаку $S = 0$ или 1 (табл. 3.11). Денормализация мантиссы используется для представления очень маленьких значений с ПТ, называемых "бесконечно малыми". Область их представления для формата SP показана на рис. 3.28.

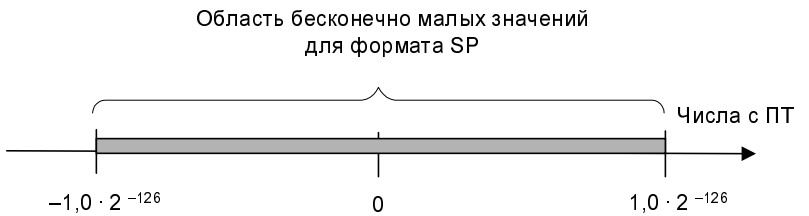


Рис. 3.28. Область бесконечно малых значений для формата SP

Нули соответствуют значению смещенного порядка ($e_{\min} - 1$), мантиссе $0, f = 0{,}000\ldots0$ и знаку $S = 0$ или 1 (табл. 3.11), т. е. имеют *двоякое* представ-

ление со знаком. Следует иметь в виду, что при *внешних* представлениях данных ноль указывается только со знаком плюс. Ноль со знаком минус получается в промежуточных вычислениях и указывает на переход к особому случаю "потеря значимости" *(см. главу 4)*.

Бесконечности соответствуют значению смещенного порядка ($e_{max} + 1$), мантиссе $1, f = 1{,}000\ldots0$, знаку $S = 0$ или 1 (табл. 3.11) и кодируют представления $+\infty$ и $-\infty$, в частности, при делении на 0 или при переходе к особому случаю "переполнение" *(см. главу 4)*.

Нечисла (NaN, Not a Number) соответствуют значению смещенного порядка ($e_{max} + 1$), мантиссе $1, f \neq 1{,}000\ldots0$ и знаку $S = 0$ или 1 (табл. 3.10). Различают два класса нечисел: сигнальные SNaN (Signaling NaN) и спокойные — QNaN (Quiet NaN).

Сигнальные нечисла SNaN никогда не получаются в результате вычислений, но распознаются в качестве операндов. Попытка выполнить арифметическую операцию с таким нечислом воспринимается процессором, как недопустимая операция, что может использоваться разработчиком для программного прерывания процессора и выполнения необходимой подпрограммы обслуживания прерывания.

Спокойные нечисла QNaN распознаются процессором и не прерывают его работу. Они генерируются процессором как реакция на особый случай недопустимых операций или создаются программистом для кодирования результатов различных недопустимых операций на этапе отладки программ.

Расширенный формат данных SEP допускает много двоичных представлений, не соответствующих ни одному из перечисленных. Все они образуют группу *неподдерживаемых форматов*.

Таблица 3.11. *Особые случаи при представлении данных с ПТ*

Особый случай	Представление с ПТ в формате SP					Десятичный эквивалент
	Знак	Порядок			Мантисса	
	S	$E_{(10)}$	$e_{(10)}$	$e_{(2)}$	$A, f_{(2)}$	
Ненормализованные положительные	0	−127	0	00…00	0,00…01 — min	$2^{-23} \cdot 2^{127} \approx \approx 0{,}7 \cdot 10^{-45}$
	0	−127	0	00…00	0,11…11 — max	$(1 - 2^{-23}) \cdot 2^{-127} \approx \approx 0{,}59 \cdot 10^{-38}$
Ненормализованные отрицательные	1	−127	0	00…00	0,11…11 — min	$-(1 - 2^{-23}) \cdot 2^{-127} \approx \approx -0{,}59 \cdot 10^{-38}$
	1	−127	0	00…00	0,00…01 — max	$\approx -0{,}7 \cdot 10^{-45}$

Таблица 3.11 *(окончание)*

Особый случай	Представление с ПТ в формате SP					Десятичный эквивалент
	Знак	Порядок			Мантисса	
	S	$E_{(10)}$	$e_{(10)}$	$e_{(2)}$	$A, f_{(2)}$	
Нули: + 0	0	–127	0	00…00	0,00…00	
– 0	1	–127	0	00…00	0,00…00	
Бесконечности: $+\infty$	0	128	255	11…11	1,00…00	
$-\infty$	1	128	255	11…11	1,00…00	
Нечисла: QNaN	0	128	255	11…11	1,1x…xx	
SNaN	0	128	255	11…11	1,0x…x1	
SNaN	1	128	255	11…11	1,0x…x1	
QNaN	1	128	255	11…11	1,1x…x1	

В табл. 3.11 символ "*A*" используется для обозначения целой части мантиссы, которая может равняться 0 или 1.

3.8.7. Арифметические операции над данными с плавающей точкой

Выполнение арифметических операций над данными, представленными с ПТ, в различных ЦПОС реализуется по-разному. Так, в ЦПОС, не поддерживающих стандарт IEEE 754, например, в TMS320C3x, применяется представление данных в дополнительном коде, и выполняются отдельно арифметические операции над значениями порядков и мантисс, как над беззнаковыми целыми числами.

Замечание

Процессор TMS320C3x с ПТ хорошо известен, в том числе, на российском рынке. Учитывая огромный накопившийся для этого процессора объем программного обеспечения, разработчики фирмы Texas Instruments воспроизвели это семейство на современной технологии (0,18 мкм вместо 0,65 мкм), существенно улучшив его качественные показатели и одновременно снизив цену.

В ЦПОС, поддерживающих стандарт IEEE 754, дополнительный код не используется, и для выполнения арифметических операций применяются иные алгоритмы. Пользователю нет необходимости подробно изучать достаточно сложные алгоритмы выполнения арифметических операций, т. к. при

обработке данных с ПТ (в отличие от ФТ) исключена возможность неоднозначной трактовки типа данных.

3.8.8. Диапазон, динамический диапазон и точность представления чисел с плавающей точкой

Используя введенные *в разд. 3.7.11* определения для представления вещественных чисел, запишем соответствующие выражения для *нормализованных* чисел, представленных с ПТ в формате "слово" SP.

Диапазон представления чисел равен (см. табл. 3.9)

$$2^{-126} < C < (2 - 2^{-23}) \cdot 2^{127}.$$

Динамический диапазон ДД

$$\text{ДД} = |\text{max значение}| \; / \; |\text{ min значение} \neq 0|$$

равен

$$\text{ДД} = |(2 - 2^{-23}) \cdot 2^{127}| \; / \; |2^{-126}| \approx 2^{254} \approx 2{,}9 \cdot 10^{76}.$$

Динамический диапазон ДД в децибелах равен

$$20 \lg (\text{ДД}) \approx 1530 \text{ дБ}.$$

Сравнивая значения *диапазонов* и *динамических диапазонов* для представлений вещественных чисел в форме с ПТ и ФТ при одинаковой длине слова равной 32 битам, видим, что для чисел с ПТ они существенно выше.

Точность представления вещественного числа с ПТ определяется *точностью представления мантиссы*, длина которой с учетом неявной единицы составляет 24 бита. Максимальное значение мантиссы равно $(2 - 2^{-23})$, а максимально допустимая *ошибка* при округлении ее дробной части равна половине младшего значащего бита, т. е. $2^{-23-1} = 2^{-24}$. Поэтому *точность* представления мантиссы, а, следовательно, и числа с ПТ составляет

$$\log_2(|\text{max значение}| \; / \; |\text{ max ошибка при округлении}|) =$$

$$= \log_2(|(2 - 2^{-23})| \; / \; |2^{-24}|) \approx 25 \text{ бит}.$$

Точность представления данных с ФТ равна длине слова, в котором они представляются, поэтому она будет выше только, если длина слова превышает 25 битов (например, в процессорах TMS320C62xxx/64xxx, см. табл. 3.4).

Говоря о точности, нельзя забывать о точности *внутреннего* представления чисел (результатов промежуточных вычислений). Выполняя аналогичные расчеты для конкретных ЦПОС (см. табл. 3.4), можно легко произвести сравнительный анализ.

3.9. Сравнение ЦПОС с фиксированной и плавающей точками

Кроме преимуществ по динамическому диапазону и точности представления данных с ПТ, одним из важнейших достоинств ЦПОС с ПТ является *естественность* представления данных. Все это существенно облегчает работу программиста, позволяя ему в большинстве случаев не задумываться о проблемах:

- масштабирования данных;
- их неоднозначной трактовки;
- переполнения.

Вместе с тем, архитектура ЦПОС с ПТ значительно сложнее, вследствие чего они, как правило, имеют большие размеры кристалла и стоимость. При выборе ЦПОС с точки зрения представления данных пользователь должен принимать во внимание:

- область его конкретного применения, в частности, требования к динамическому диапазону *сигналов* и точности их обработки;
- специфику программирования конкретных алгоритмов ЦОС, в частности, наличие большого объема вычислительных процедур с вероятностью выхода за границы возможного диапазона представления чисел (переполнения) и необходимостью масштабирования промежуточных данных во многих точках программы;
- ограничения на габариты кристалла.

При выборе типа процессора (с ФП или с ПТ) не следует забывать о способах повышения точности в процессорах с ФТ, если снижение скорости обработки останется в рамках заданных требований (дополнительно *см. главу 9*).

3.10. Организация обработки данных с плавающей точкой в ЦПОС с ФТ

Кроме рассмотренного выше способа увеличения динамического диапазона и точности представления данных в ЦПОС с ФТ *(см. разд. 3.7.12)* можно *моделировать* представление чисел и арифметику с ПТ.

При моделировании формы представления с ПТ для хранения числа требуется два слова — одно для мантиссы, другое — для порядка. Как будет показано далее, форма с ПТ может повысить динамический диапазон и точность представления результатов операций, получаемых в формате "двойное слово" либо "расширенное слово", при их сохранении в памяти данных в формате "слово". Платой за это является увеличение вдвое требуемого объема памяти данных.

В архитектуре процессоров предусмотрены специальные операции, позволяющие моделировать форму представления чисел с ПТ:

- нормализация числа;
- выделение порядка для блока чисел.

Рассмотрим эффективность представления с ПТ при выполнении этих операций.

Нормализация применяется для представления вещественных чисел в форме с ПТ в *нормализованном* виде. В ЦПОС с ФТ независимо от типа арифметики (целочисленная или дробная) при выполнении *нормализации* число следует трактовать как вещественное. Дробное двоичное число с ФТ в прямом коде считается нормализованным, а первая значащая цифра после запятой отлична от 0. Так как ЦПОС с ФТ оперирует с числами в дополнительном коде, признаком их нормализованного вида будет первая цифра дробной части, равная:

- 1 — для *положительного* числа;
- 0 — для *отрицательного* числа.

Отсюда легко формулируется правило, по которому определяют, является ли двоичное число в дополнительном коде нормализованным: число нормализовано, если значения знакового и старшего значащего битов не совпадают.

В табл. 3.12 приведены: исходные числа в форме с ФТ в формате "двойное слово" (длиной 8 битов) и они же, сохраняемые в формате "слово" (длиной 4 бита) в двух вариантах: в первом — в форме с ФТ, во втором — с ПТ. Как видно из этих примеров, эффект увеличения динамического диапазона и точности при переходе к форме с ПТ достигается, когда исходное двоичное число не было уже нормализованным.

Таблица 3.12. *Нормализованные и ненормализованные двоичные числа*

Двоичное число с ФТ		**Нормализованное число с ПТ**	**Эффективность представления с ПТ (есть/нет)**
Формат			
Двойное слово (8 битов)	**Слово (4 бита)**	**Слово (4 бита) (мантисса + знак)**	
0,0000101	0,000	$0{,}101 \cdot 2^{-4}$	Есть
0,0010011	0,001	$0{,}100 \cdot 2^{-2}$	Есть
0,0011101	0,001	$0{,}111 \cdot 2^{-2}$	Есть
0,0001111	0,000	$0{,}111 \cdot 2^{-3}$	Есть
1,1100110	1,110	$1{,}001 \cdot 2^{-2}$	Есть

Таблица 3.12 *(окончание)*

Двоичное число с ФТ		Нормализованное число с ПТ	Эффективность представления с ПТ (есть/нет)
Формат			
Двойное слово (8 битов)	Слово (4 бита)	Слово (4 бита) (мантисса + знак)	
1,1111000	1,111	$1{,}000 \cdot 2^{-4}$	Есть
1,1110101	1,111	$1{,}010 \cdot 2^{-3}$	Есть
1,1101010	1,110	$1{,}010 \cdot 2^{-2}$	Есть
0,1010101	0,101	$0{,}101 \cdot 2^{0}$	Нет
0,1110011	0,111	$0{,}111 \cdot 2^{0}$	Нет
1,0011010	1,001	$1{,}001 \cdot 2^{0}$	Нет
1,0110011	1,011	$1{,}011 \cdot 2^{0}$	Нет

Для *нормализации* чисел в процессорах с ФТ предусмотрена команда `NORM`, в соответствии с которой производятся необходимые сдвиги числа влево с одновременным запоминанием количества сдвигов в специально отводимом регистре.

Возможность представления числа с порядком используется для организации так называемой *блочной плавающей точки*. Принцип представления чисел с блочной плавающей точкой состоит в следующем:

- числа объединяются в группы из нескольких чисел — *блоки*;
- *внутри блока* выделяется самое большое (по модулю) число;
- это число представляется в форме с ПТ; *порядок* числа (максимальный в блоке) сохраняется в специальном регистре;
- *остальные* числа блока представляются в форме с ПТ при *том же* значении порядка.

Операция определения единого порядка для блока данных поддерживается в процессорах фирмы Analog Devices. Вся процедура осуществляется с помощью аппаратно реализованного цикла по специальной команде `EXPADJ`, называемой "выделение порядка блока". Блоки чисел, имеющих в форме с ПТ одинаковый порядок, в последующем могут обрабатываться с помощью программно организованной блочной арифметики.

Блочная плавающая точка эффективна для представления данных с малыми (по модулю) значениями и с незначительным разбросом этих значений. В табл. 3.13 даны примеры эффективной и неэффективной организаций

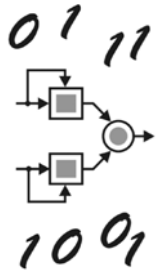
блочной плавающей точки в блоках размером 4 числа; жирным шрифтом выделено наибольшее (по модулю) число в блоке.

В заключение отметим, что моделировать представление данных и арифметику с ПТ имеет смысл только, если объем подобных вычислений невелик, в противном случае целесообразно выбрать процессор с ПТ.

Таблица 3.13. *Блочная плавающая точка*

Блок	Двоичное число	Блочная ПТ		Эффективность представления с блочной ПТ (есть/нет)
	Формат			
	Двойное слово (16 битов)	Слово (8 битов) (мантисса + знак)	Порядок	
1	0,000111011001010	0,0111011		Есть
	0,001011000100100	0,1011000	0110	
	0,000010100100011	0,0010100		
	0,001011010100110	**0,1011010**		
2	0,00111011001010	0,0011101		Нет
	0,101011000100100	0,1010110	0000	
	0,111010100100011	**0,1110101**		
	0,000101101010011	0,0001011		

Глава 4



Операции над данными

Совокупность основных устройств, формирующих конечный результат обработки данных, включает *(см. также главу 2)*:

- умножитель или устройство MAC;
- арифметико-логическое устройство (АЛУ);
- аккумулятор;
- сдвигатель;
- ограничитель.

Рассмотрим операции, выполняемые этими устройствами в процессе обработки данных.

4.1. Операции над данными в ЦПОС с фиксированной точкой

В следующих разделах рассматривается назначение вышеперечисленных устройств для обработки данных с ФТ.

4.1.1. Умножители и устройства MAC

Наличие одноциклового умножителя — один из ключевых признаков архитектуры ЦПОС, т. к. операции умножения и умножения с накоплением являются базовыми в алгоритмах ЦОС.

Все процессоры, предназначенные для цифровой обработки сигналов, в качестве одного из главных компонентов своей архитектуры содержат:

- умножитель;
- умножитель/сумматор, называемый *устройством MAC* (Multiplier/Accumulator).

Умножитель выполняет операцию умножения данных, представленных в формате "слово", результат умножения — произведение — сохраняется в специальном регистре в формате "двойное слово" или "расширенное слово". Если результат операции умножения сохраняется в памяти данных, проис-

ходит автоматическое преобразование форматов. Умножитель реализован, например, в процессорах TMS320C2xxx/5xxx/6xxx фирмы Texas Instruments.

Устройство MAC выполняет операцию умножения с накоплением (*операцию MAC*); сначала во внутреннем умножителе устройства MAC производится операция умножения данных, представленных в формате "слово", а затем сложение полученного произведения (выровненного по правому краю) с содержимым аккумулятора устройства MAC, который имеет формат "расширенное слово". Если результат операции MAC сохраняется в памяти данных, происходит автоматическое преобразование форматов. При выполнении в устройстве MAC операции *умножения* аккумулятор MAC предварительно обнуляется. Устройства MAC реализованы в процессорах фирм Motorola и Analog Devices.

Более подробно реализация операций умножения и MAC описана в *главе 2*, а алгоритм умножения, трактовка результатов и преобразование форматов — в *главе 3*.

4.1.2. Арифметико-логические устройства

АЛУ предназначено для выполнения всех арифметических и логических операций над данными (кроме операций умножения и MAC), а также операций бит-манипуляций.

Внутренняя архитектура АЛУ скрыта от пользователя, и действия этого устройства раскрываются через правила выполнения соответствующих команд. Выполнение основных арифметических операций рассмотрено в *главе 3*.

4.1.3. Аккумуляторы

Аккумуляторы представляют собой регистры, используемые для хранения внутренних данных — промежуточных и конечных результатов выполнения операций в АЛУ.

В архитектуре многих процессоров предусмотрено два аккумулятора. Наличие одного аккумулятора приводит к существенному снижению эффективности обработки данных из-за того, что аккумулятор используется и как источник, и как приемник данных. Следовательно, в процессе вычислений необходимо сохранять результаты промежуточных операций (содержимое аккумулятора) в ячейках памяти данных. В результате, с одной стороны, увеличивается время выполнения программы, с другой стороны, уменьшается точность и динамический диапазон представления промежуточных данных. Последнее происходит вследствие преобразования форматов из большего (слово аккумулятора) в меньший (слово).

Содержимое аккумулятора может быть представлено в формате "двойное слово", например, в процессорах TMS320C2xxx фирмы Texas Instruments, однако чаще всего оно представляется в формате "расширенное слово". Подобно структуре расширенного слова (сцеплению трех слов), аккумулятор состоит из трех последовательно соединенных регистров, для обозначения которых будем использовать те же имена, что и для слов, отображающих содержимое данных регистров, а именно:

- EXT — регистр расширения;
- MSP — регистр старшего слова;
- LSP — регистр младшего слова.

Наличие расширения EXT позволяет за счет добавления дополнительных битов (обычно 4 или 8) к формату "двойное слово" повысить порог, при котором происходит переполнение аккумулятора. По этой причине дополнительные биты расширения EXT обычно называют "сторожевыми" или "защитными" (guard bits).

Если длина слова EXT равна 4 или 8 битам, в аккумуляторе могут накапливаться значения промежуточных данных соответственно до 2^4 или 2^8 (по модулю) без риска переполнения. Если такие значения требуется сохранить в памяти данных, возможен один из двух вариантов:

- содержимое регистра EXT сохранить в отдельной ячейке;
- произвести *масштабирование* содержимого аккумулятора до его пересылки в память данных.

В противном случае при сохранении содержимого аккумулятора в памяти данных, оно автоматически округляется до амплитуды насыщения *(см. разд. 4.1.5)*.

Длины слов аккумуляторов в различных процессорах приведены в табл. 3.4.

4.1.4. Сдвигатели

Сдвигателем называют как устройство, выполняющее операцию сдвига данных, так и регистр, в котором хранится результат сдвига. Как правило, из контекста ясно, идет ли речь об устройстве или о регистре. При необходимости дается уточнение.

По выполняемым функциям сдвигатели можно разделить на:

- *пресдвигатели*, в которых сдвиг данных осуществляется перед выполнением или при выполнении операции;
- *постсдвигатели*, в которых сдвиг данных производится после выполнения операции.

В обоих случаях структура сдвигателя, хранящего результат операции сдвига, копирует структуру аккумулятора.

Основные функции *пресдвигателей* таковы.

- Предварительное масштабирование — сдвиг данных перед выполнением арифметических операций сложения и вычитания, например, перед выполнением в процессоре DSP5600x фирмы Motorola операции вычитания по команде

```
SUBL B, A
```

где `B`, `A` — аккумуляторы B и A в формате "расширенное слово" длиной 56 битов, в сдвигателе производится смещение содержимого A на 1 бит влево (буква `L` в команде); затем выполняется вычитание (A) × 2 − (B); результат сохраняется в A; в сдвигателе сохраняется (A) × 2 (рис. 4.1).

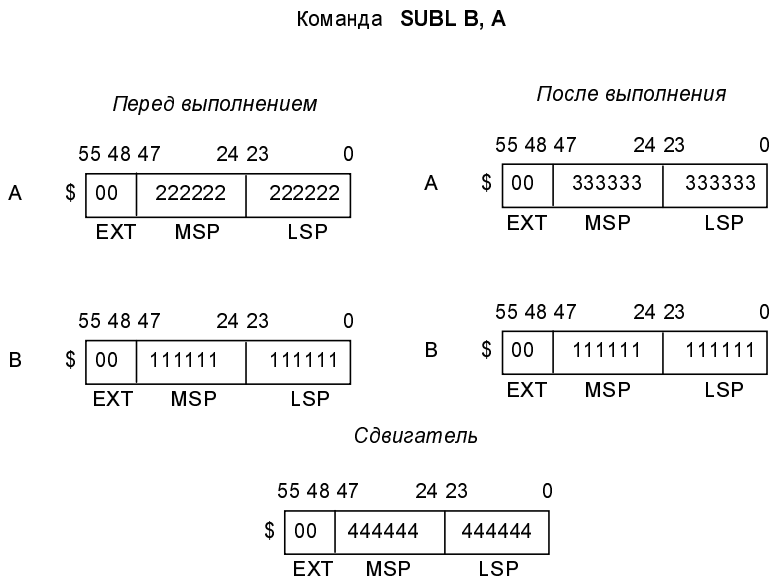


Рис. 4.1. Сдвиг влево и вычитание

- Предварительное масштабирование — сдвиг данных перед их загрузкой в аккумулятор, например, перед выполнением в процессоре TMS320C54xx фирмы Texas Instruments операции загрузки содержимого ячейки памяти данных в аккумулятор по команде

```
LD XN, 8, B
```

где `B` — аккумулятор B в формате "расширенное слово" длиной 40 битов; `XN` — ячейка памяти данных в формате "слово" длиной 16 бит, в сдвигателе происходит сдвиг содержимого XN на 8 битов влево (рис. 4.2); при этом, согласно правилам преобразования форматов данных с ФТ, в стар-

ших битах сдвигителя происходит расширение знака, а в младших — обнуление; результат копируется в аккумуляторе.

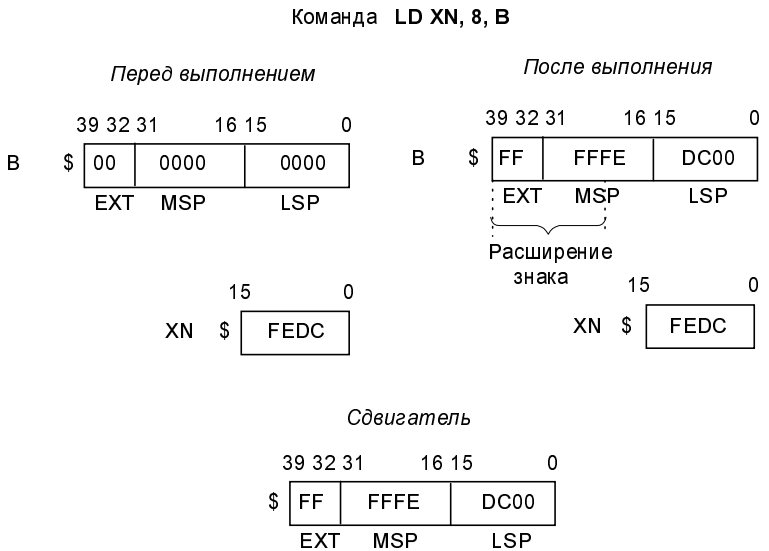


Рис. 4.2. Сдвиг влево и загрузка аккумулятора

- Сдвиг данных перед выполнением в АЛУ логических операций; например, перед выполнением в процессоре TMS320C2xxx фирмы Texas Instruments операции логического умножения

```
AND A, 3, B
```

где `A`, `B` — аккумуляторы A и B в формате "расширенное слово" длиной 40 бит, в сдвигателе происходит сдвиг содержимого A влево на 3 бита; затем выполняется операция логического умножения; результат сохраняется в B; содержимое A не меняется; в сдвигателе сохраняется результат сдвига (рис. 4.3).

- Сдвиг данных при выполнении в АЛУ операций арифметических сдвигов, например, при выполнении в процессоре DSP5600x фирмы Motorola операции арифметического сдвига по команде

```
ASR B
```

где `B` — аккумулятор B в формате "расширенное слово" длиной 56 битов, в сдвигателе происходит сдвиг содержимого `B` вправо на 1 бит; при сдвиге значение младшего бита LSB сохраняется в бите переноса C регистра состояния, а значение старшего бита MSB остается неизменным; результат копируется в аккумуляторе `B` (рис. 4.4).

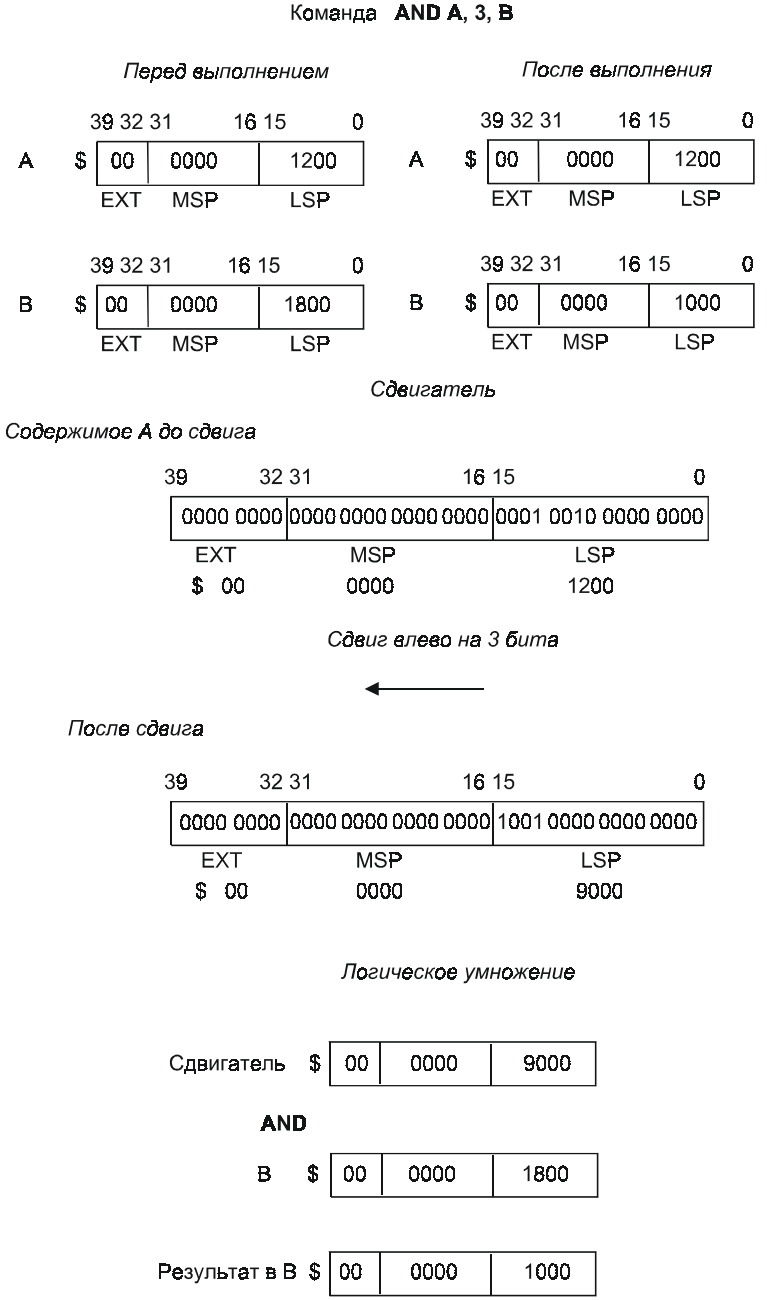


Рис. 4.3. Логическое умножение со сдвигом

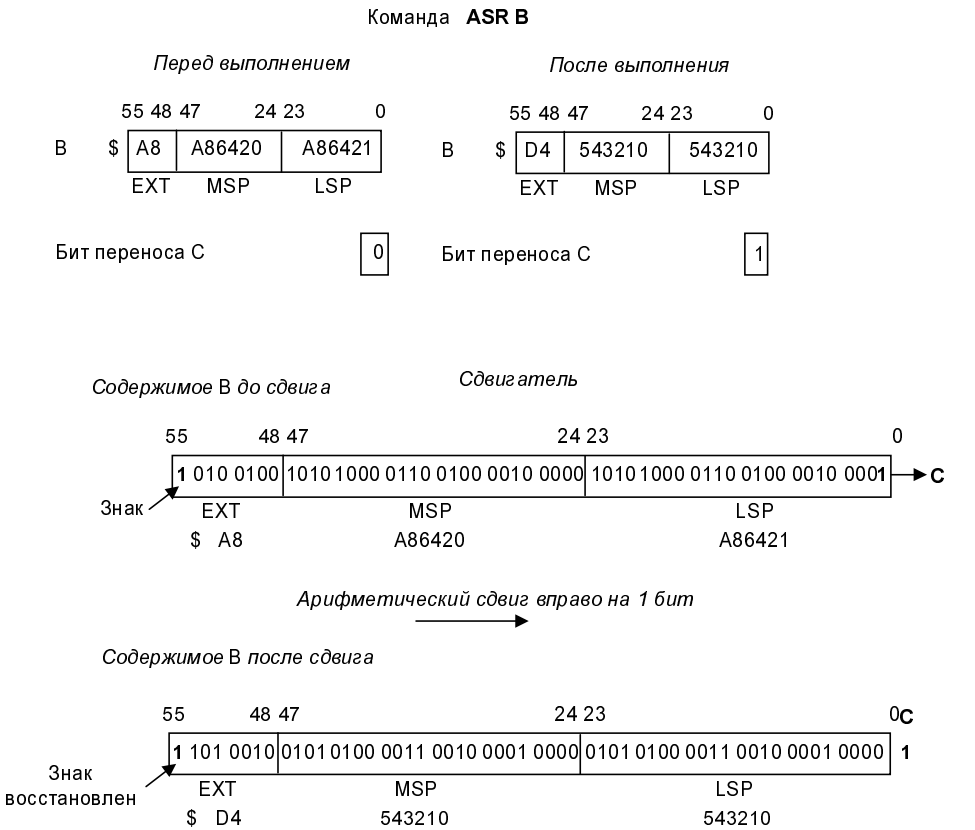


Рис. 4.4. Арифметический сдвиг содержимого аккумулятора

- Сдвиг данных при выполнении в АЛУ операций логических и циклических сдвигов, например, при выполнении в процессоре DSP5600x фирмы Motorola операции логического сдвига по команде

  `LSR B`

  где `B` — аккумулятор B в формате "расширенное слово" длиной 56 битов. В отличие от приведенного выше арифметического сдвига, при выполнении операции логического сдвига в сдвигателе выполняется сдвиг не всего содержимого аккумулятора, а только его старшего MSP-слова; состояния младшего слова LSP и расширения EXT не меняются; результат копируется в аккумуляторе (рис. 4.5).

Основные функции *постсдвигателей*:

- масштабирование результатов при их сохранении в памяти данных;
- устранение битов расширения знака;

- нормализация результатов;
- масштабирование результатов при выделении одинакового порядка в блоке.

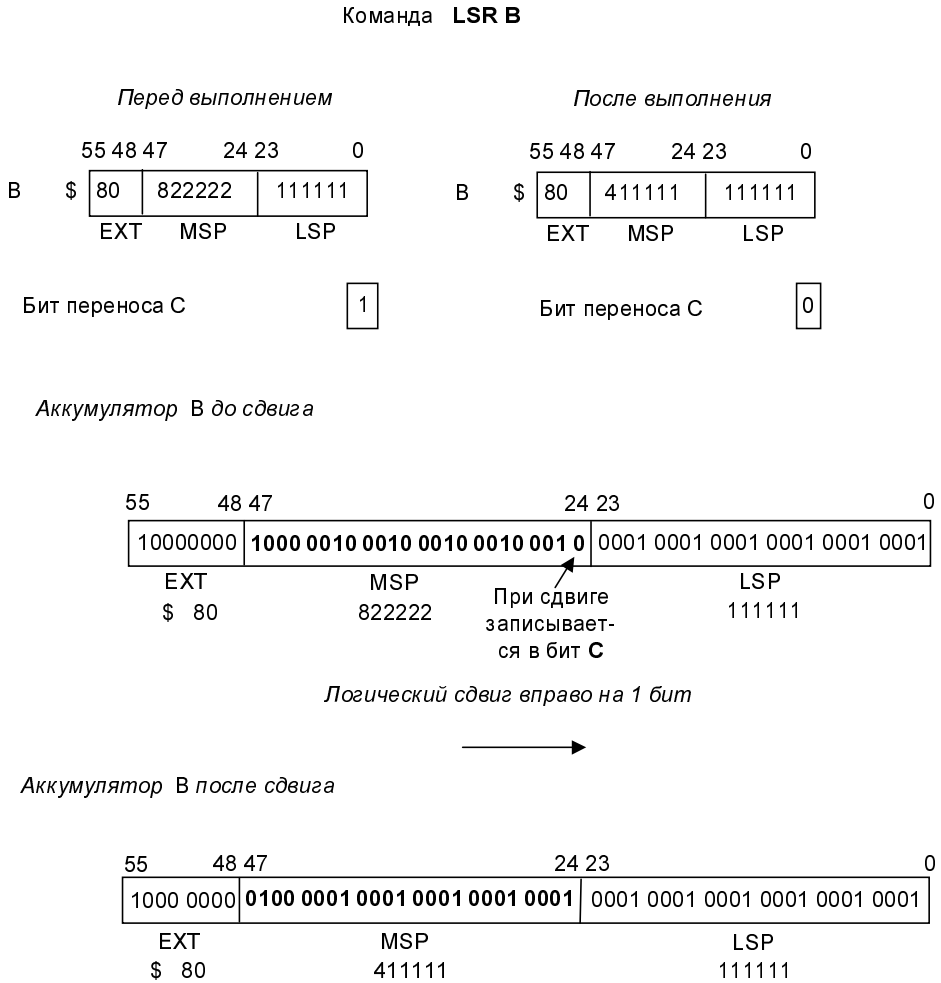


Рис. 4.5. Логический сдвиг содержимого аккумулятора

Масштабирование осуществляется при сохранении результатов операции в памяти данных. Операция сдвига содержимого аккумулятора производится в сдвигателе (копирующем содержимое аккумулятора), содержимое аккумулятора не меняется. В ячейках памяти сохраняется старшее MSP или младшее LSP слово сдвигателя (возможно, оба слова в соседних ячейках) с соблюдением правил преобразования форматов.

Постсдвигатели могут использоваться для устранения бита (или битов) расширения знака, автоматически генерируемого (генерируемых) в регистре, который хранит результат умножения данных при целочисленной арифметике.

Указанием на выполнение операции постсдвига может быть:

- состояние битов масштабирования в регистре состояния. Их комбинация определяет направление и величину сдвига. Например, в процессорах DSP56xxx фирмы Motorola состояние битов масштабирования S0 и S1 определяет три возможных режима в сдвигателе: отсутствие масштабирования, масштабирование вниз (сдвиг на 1 бит влево), масштабирование вверх (сдвиг на 1 бит вправо). Одновременно в сдвигателе меняется длина, отводимая на целую часть, соответственно, уменьшается или увеличивается на 1 бит. Содержимое аккумулятора не меняется;
- специальная команда. Например, в процессорах TMS320C2xxx фирмы Texas Instruments, слово аккумулятора которых имеет формат двойного слова, содержимое старшего MSP и младшего LSP слов аккумулятора сохраняется в памяти данных при выполнении команд `SACH` и `SACL`. В частности, по команде

  ```
  SACH XN, 1
  ```

  где XN — ячейка памяти данных в формате "слово" длиной 16 битов, в сдвигателе происходит сдвиг содержимого аккумулятора на 1 бит влево; после этого старшее слово MSP сдвигателя сохраняется в ячейке памяти данных XN; содержимое аккумулятора не меняется (рис. 4.6);
- непосредственное указание сдвига (количества битов и направления) в команде сохранения содержимого аккумулятора в памяти данных. В частности, в процессорах TMS32C54xx, слово аккумулятора которых имеет формат расширенного слова, при выполнении команды сохранения старшего MSP слова аккумулятора в памяти данных

  ```
  STH A,4,XN
  ```

  в сдвигателе происходит сдвиг содержимого аккумулятора A на 4 бита влево и сохранение его старшего MSP слова в ячейке XN; содержимое аккумулятора не меняется; команда выполняется аналогично предыдущей.

Операции нормализации и выделения одинакового порядка в блоке из нескольких чисел рассмотрены в *разд. 3.10*.

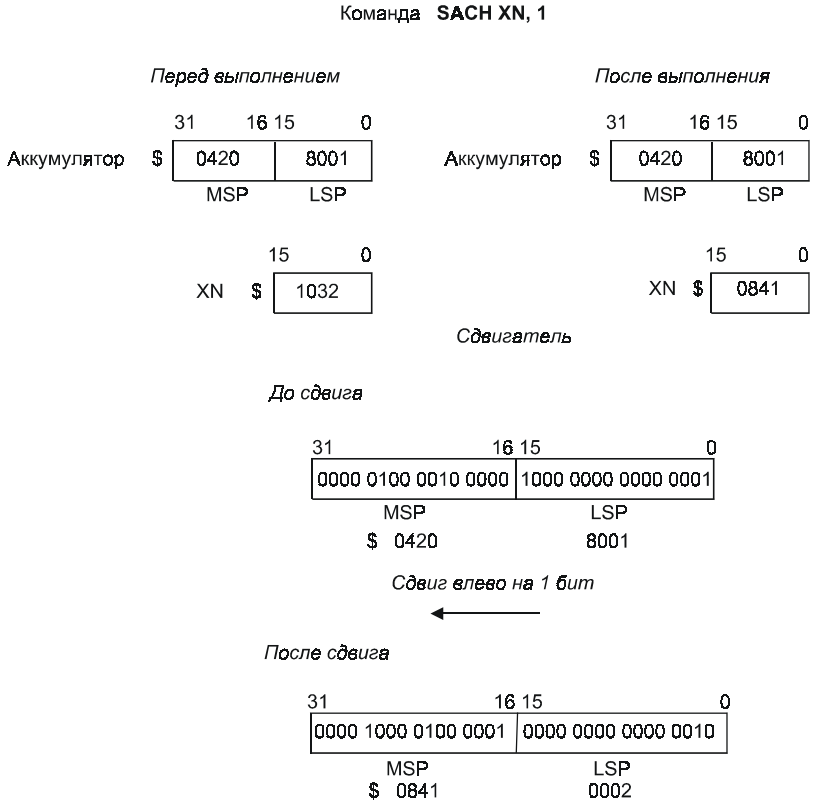


Рис. 4.6. Сдвиг при сохранении старшего слова аккумулятора

4.1.5. Ограничители

Ограничители — это условное названия устройств, которые реализуют арифметику насыщения при переполнении в аккумуляторе.

4.1.6. Переполнение в аккумуляторе

В процессорах с ФТ при выполнении операций:

- ❒ сложения, вычитания, MAC;
- ❒ сдвиге содержимого аккумулятора *влево*

и т.п. может произойти *переполнение.*

Различают *две* его разновидности, называемые:

- ❒ переполнение аккумулятора;
- ❒ переполнение при пересылках.

Переполнение аккумулятора означает, что результат арифметической операции не может быть представлен в аккумуляторе, т. к. он выходит за границы допустимого диапазона представления чисел в формате слова аккумулятора *(см. главу 3)*.

Если слово аккумулятора имеет формат "двойное слово" MSP:LSP, при представлении чисел со знаком старший бит двойного слова является знаковым; состояние переполнения аккумулятора возникает при попытке переноса из старшего значащего бита в знаковый бит.

Если слово аккумулятора имеет формат "расширенное слово" EXT:MSP:LSP, переполнение аккумулятора возникает при попытке переноса в старший (знаковый) разряд расширенного слова.

При представлении целых беззнаковых чисел аккумулятор переходит в состояние переполнения при попытке переноса из старшего бита слова аккумулятора, т. е. при выходе за левую границу формата.

Переполнение аккумулятора сопровождается установкой флага переполнения в регистре состояния.

Предотвратить переполнение аккумулятора возможно с помощью масштабирования исходных данных или/и результатов арифметических операций. В такой ситуации следует быть особо внимательным к масштабированию малых (по модулю) значений, т. к. в результате масштабирования эти значения могут оказаться меньше минимально допустимой величины (шага квантования).

В процессоре может быть установлен режим *запрещения* переполнения аккумулятора. В этом случае, независимо от формата слова аккумулятора ("двойное слово" или "расширенное слово") при возникновении переполнения результат автоматически заменяется максимальным (по модулю) числом, представленным в формате двойного слова, т. е. MSP:LSP части слова аккумулятора. Подобные операции ограничения соответствуют арифметике насыщения, а полученное в результате ограничения число называют *амплитудой насыщения*. С учетом знака, амплитуда насыщения положительного числа равна $7FF…FF, а амплитуда насыщения отрицательного числа равна $800…00, где количество цифр определяется длиной MSP:LSP части слова аккумулятора.

Переполнение при пересылках возникает при одновременном выполнении следующих условий:

- слово аккумулятора имеет формат "расширенное слово" EXT:MSP:LSP;
- содержимым аккумулятора является смешанное число;
- это содержимое аккумулятора сохраняется в памяти данных.

Замечание

Напомним, что в процессорах с ФТ и целочисленной арифметикой вещественным числам (в данном случае, смешанным числам) соответствуют их целочисленные эквиваленты *(см. главу 3)*.

Переполнение при пересылках возникает при переносе в старший (знаковый) бит MSP:LSP части слова аккумулятора. Это является признаком изменения типа результата — переходу от дробного числа к смешанному. Тип результата в аккумуляторе обычно отображается состоянием соответствующего бита (флага) в регистре состояния. Такой бит может называться *битом расширения* (в процессорах фирмы Motorola), битом *насыщения при пересылках* (в процессорах фирмы Texas Instruments) и т. п. Установленный флаг соответствует смешанному числу и является признаком переполнения аккумулятора при пересылках, поскольку смешанное число, представленное в формате "расширенное слово" EXT:MSP:LSP, невозможно сохранить в ячейке памяти данных в формате "слово" или "двойное слово" (в двух ячейках памяти данных).

Предотвратить переполнение при пересылках возможно так же, как и переполнение аккумулятора — с помощью масштабирования. Выполняя операцию сдвига вправо, смешанное число преобразуют в дробное число, которое можно сохранить в памяти данных.

Альтернативное решение — переход к *арифметике насыщения*. Подобная арифметика поддерживается всеми процессорами с ФТ, где аккумулятор имеет расширение EXT. Ее суть заключается в следующем. Как только обнаруживается переполнение при пересылках (установлен флаг), смешанное число автоматически заменяется максимально допустимым (по модулю) значением дробного числа — *амплитудой насыщения*. При этом *содержимое аккумулятора не меняется* и может быть использовано для дальнейших вычислений. Например, если содержимое 40-разрядного аккумулятора (рис. 4.7) содержит в расширении EXT целую часть, равную $2^0 = 1{,}0$, а значит, является смешанным числом, при сохранении старшего MSP слова аккумулятора в ячейке памяти данных без использования арифметики насыщения получим число (−1) — неверный результат, а с использованием арифметики насыщения — максимально допустимое положительное дробное число $0{,}11\ldots11 = 1 - 2^{-31}$ — амплитуду насыщения положительного числа.

Вычислим ошибку ε, допускаемую при сохранении содержимого аккумулятора (смешанного числа) в памяти данных:

$$\varepsilon = |(A) - (X)|,$$

где (A) — содержимое аккумулятора; (X) — содержимое ячейки памяти данных.

В примере (см. рис. 4.7) при сохранении положительного числа без использования арифметики насыщения ошибка равна

$$\varepsilon = |1 - (-1)| = 2,$$

а с использованием арифметики насыщения

$$\varepsilon = |1 - (1 - 2^{-31})| = 2^{-31}.$$

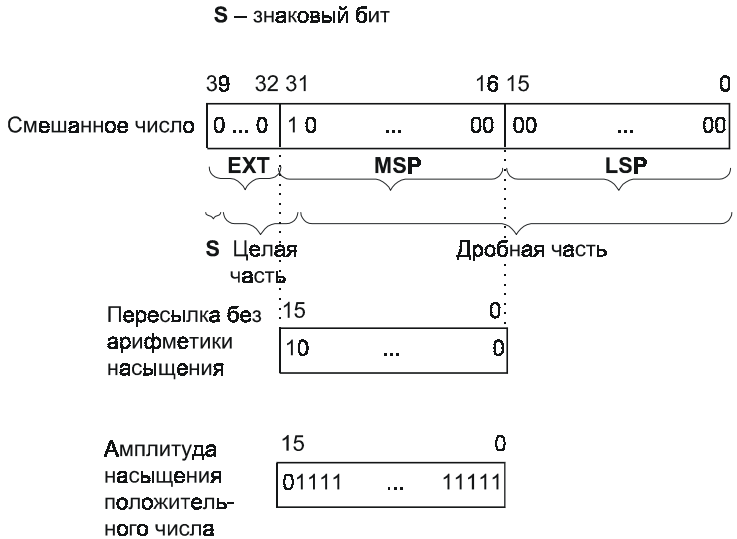


Рис. 4.7. Пример арифметики насыщения

Таким образом, арифметика насыщения при пересылке содержимого аккумулятора в ячейку памяти данных не устраняет ошибку, но делает ее минимальной.

В общем случае, амплитуда насыщения положительного числа равна $7FF... ...FF, а амплитуда насыщения отрицательного числа — $800...00, что не зависит от типа арифметики (целочисленная/дробная). Длина соответствующей двоичной константы равна длине слова или двойного слова при представлении данных с удвоенной точностью.

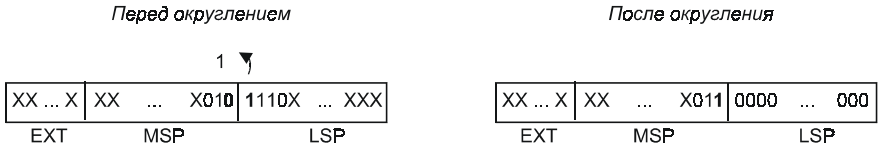
4.1.7. Округление результатов

Даже если в аккумуляторе нет переполнения при пересылках, сохранение его содержимого в памяти данных приводит к потере точности вследствие усечения младшего LSP слова аккумулятора при преобразовании формата "слово аккумулятора" в формат слова *(см. главу 3)*. Это означает, что в результирующих данных (например, в выходном сигнале) появляется искажение, которое часто называют *смещением результата*. Процедура округления результата при его сохранении в памяти данных позволяет уменьшить такое смещение.

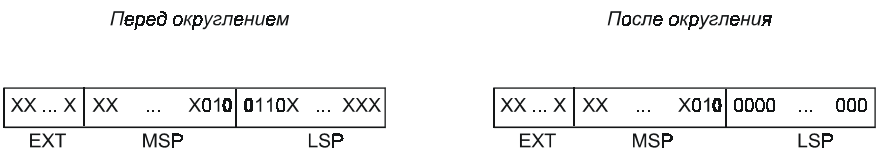
В некоторых процессорах округление, подобно ограничению в арифметике насыщения, происходит автоматически при сохранении результата в памяти данных. В других процессорах округление выполняется с помощью специальной команды.

Округление до ближайшего

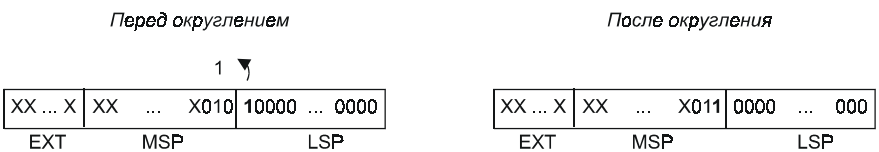
а) округление с избытком



б) округление с недостатком

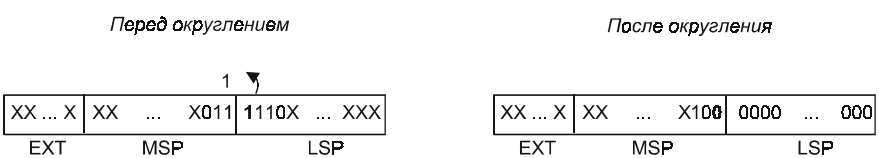


в) округление с избытком



Округление до ближайшего

г) округление с избытком



д) округление с недостатком

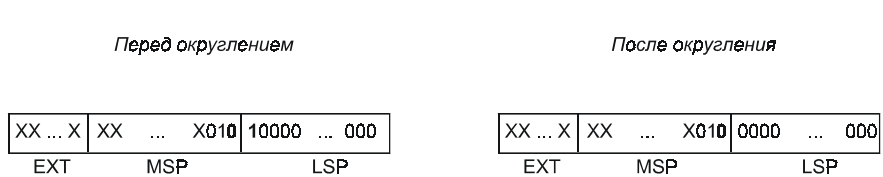


Рис. 4.8. Методы округления

В процессорах с ФТ, в основном, используют два метода округления:

- до ближайшего;
- до ближайшего четного.

Округление до ближайшего представляет собой обычное округление с избытком, применяемое в арифметике. Данные округляются до ближайшего значения, которое может быть представлено в формате слова. Если старший MSB-бит младшего LSP-слова аккумулятора равен 1, к младшему LSB-биту старшего MSP-слова аккумулятора (округляемой позиции) добавляется 1 — происходит *округление с избытком* (рис. 4.8, *а*). В противном случае значение LSB-бита не меняется, т. е. выполняется *округление с недостатком* (рис. 4.8, *б*). Подобный метод округления приводит к тому, что все числа, попадающие точно на границу между двумя ближайшими значениями (рис. 4.8, *в*), всегда округляются в сторону увеличения — с избытком. В алгоритмах ЦОС при обработке огромного количества данных это приводит к возникновению одностороннего смещения.

Округление до ближайшего четного позволяет устранить подобную систематическую ошибку. Его суть заключается в следующем. Когда значение старшего MSB-бита младшего LSP-слова аккумулятора равно 1 (находится точно на границе между двумя ближайшими), результат округления зависит от значения округляемой позиции. Если оно равно 1 (рис. 4.8, *г*), происходит округление с избытком, если же оно равно 0 (рис. 4.8, *д*) — округление с недостатком. Выполняемая процедура случайного округления приводит к равновероятному округлению с избытком и с недостатком. Метод называется округлением до ближайшего четного потому, что округленное число всегда будет четным (округляемая позиция всегда равна 0). Исключение систематической ошибки округления означает, что вычисления будут сходиться к правильному результату, поэтому метод округления до ближайшего четного часто называют сходящимся или *конвергентным*.

Этот метод округления применяется в процессорах фирм Analog Devices и Motorola.

4.2. Операции над данными в ЦПОС с плавающей точкой

Процессорами с ПТ поддерживается два типа численных данных — с ФТ и ПТ; операции над данными с ФТ рассмотрены выше, поэтому далее обсуждаются только особенности операций над данными с ПТ.

4.2.1. Умножители

В отличие от умножения данных с ФТ, где результат представлен точно, операция умножения данных с ПТ, как правило, приводит к потере точно-

сти результата. Это объясняется тем, что в формате "расширенное слово", в котором представляется результат умножения, на длину мантиссы отведено не $(2n - 1)$ битов, как положено для точного результата, а меньше. Например, в процессорах DSP9600x для вычисления точного результата требуется на мантиссу результата умножения отвести 47 битов, в то время как в действительности, на нее отведено только 32 бита *(см. главу 3)*. Тем не менее, обеспечиваемой точности вполне достаточно для реализации большинства алгоритмов ЦОС. В противном случае существует возможность обработки данных с удвоенной точностью.

4.2.2. Арифметико-логические устройства

Архитектура АЛУ в процессорах с ПТ значительно сложнее, чем в процессорах с ФТ. В АЛУ над данными с ПТ выполняются все арифметические операции (кроме умножения), операции преобразования данных, представленных в форме с ПТ, в форму с ФТ и наоборот, а также некоторые специальные операции (табл. 4.1).

Таблица 4.1. *Примеры некоторых специальных операций в процессорах с ПТ*

Операция	Описание
Оценка обратной величины	Оценка обратной величины $1/x$ — ее приближенное значение; может выбираться в качестве начальной точки при вычислении уточненного значения $1/x$ методом итераций
Оценка значения квадратного корня	Оценка значения квадратного корня $\sqrt{x}$ — его приближенное значение; может выбираться в качестве начальной точки при вычислении уточненного значения $\sqrt{x}$ методом итераций
Сложение с вычитанием	Одновременное вычисление суммы и разности двух величин с сохранением результатов в двух регистрах

АЛУ процессоров с ПТ не поддерживает выполнение логических операций и операций бит-манипуляций над данными с ПТ. Эти операции могут выполняться только с данными, представленными с ФТ.

4.2.3. Сдвигатели

В архитектуре процессоров с ПТ для обработки данных с ПТ также предусмотрены сдвигатели. Они выполняют сдвиги вправо/влево мантиссы результата для представления его в нормализованном виде *(см. главу 3)*. Работа подобных сдвигателей скрыта от пользователя.

4.2.4. Округление данных с плавающей точкой

Результаты арифметических операций в процессорах с ПТ автоматически округляются для представления в формате расширенного слова SEP или слова SP. Согласно стандарту IEEE 754 используются следующие четыре режима округления:

- до ближайшего четного;
- в направлении нуля;
- в направлении $-\infty$;
- в направлении $+\infty$.

Режим округления определяется состоянием битов режима округления, установка которых, в свою очередь, производится в соответствии с правилами выполнения конкретной команды независимо от пользователя. Рассмотрим округление чисел с ПТ в каждом из режимов, предполагая, что округляемое значение не выходит за границы диапазона представления в заданном формате, в противном случае имеет место *переполнение*, которое, вместе с другими *особыми случаями*, обсуждается ниже. Структура форматов слов SEP и SP, в частности, длины мантиссы для этих форматов приведены на рис. 3.26. Во всех четырех режимах производится *округление мантиссы*. Для того, чтобы понять механизм округления в каждом из режимов, достаточно его показать на простых примерах округления не двоичной, а десятичной мантиссы до одной значащей десятичной цифры после запятой.

В режиме *округления до ближайшего четного* мантисса преобразуется по правилам округления данных с ФТ, а именно, десятичные мантиссы 1,65 и 1,75 автоматически округлятся до 1,6 и 1,8 соответственно.

Режим округления в направлении нуля тождественен усечению мантиссы. Мантиссы 1,65, 1,75 и $-1,65$ будут усечены до 1,6, 1,7 и $-1,6$ соответственно. Округленные значения являются для этих чисел ближайшими к нулю.

В *режиме округления в направлении* $-\infty$ мантиссы положительных чисел усекаются, а отрицательных — округляются до ближайшего. Например, мантисса 1,65 округлится до 1,6, а мантисса $-1,65$ до $-1,7$. Округленные значения являются для этих чисел ближайшими к $-\infty$.

В *режиме округления в направлении* $+\infty$ наоборот, мантиссы отрицательных чисел усекаются, а положительных — округляются до ближайшего. Например, мантисса 1,65 округлится до 1,7, а мантисса $-1,65$ до $-1,6$. Округленные значения являются для этих чисел ближайшими к $+\infty$.

4.2.5. Особые случаи при обработке данных с плавающей точкой

В процессорах с ПТ обработка различного рода нетривиальных результатов операций регламентируется стандартом IEEE 754 и осуществляться автоматически.

Согласно этому стандарту, процессоры с ПТ должны распознавать следующие пять *особых случаев* (или исключений — exceptions):

- недопустимая операция;
- деление на ноль;
- переполнение (overflow);
- потеря значимости (underflow);
- потеря точности (inexact).

Переход к любому из особых случаев фиксируется установкой соответствующих битов в регистре состояния. Кратко определим каждое из исключений.

Недопустимая операция регистрируется, когда операцию по той или иной причине невозможно выполнить. Арифметическую операцию невозможно выполнить, если она производится над недопустимыми для нее данными. Примеры недопустимых арифметических операций: $0/0$; $0\times\infty$; $\infty/0$; $0/\infty$; $\infty-\infty$; $\infty+(-\infty)$. Кроме того, недопустимыми считаются все арифметические операции с сигнальными нечислами SNaN и неподдерживаемыми форматами *(см. главу 3)*. Недопустимая операция в качестве результата возвращает значение спокойного нечисла QNaN.

Замечание

Кроме недопустимых арифметических операций существуют и другие недопустимые операции, в частности, операции со стеком: загрузка стека, когда он находится в состоянии переполнения (overflow), или считывание из стека — в состоянии ниже пустого (underflow).

Особый случай *деления на ноль*, происходит, когда делимое — ненулевое число, а делитель — ноль. Результатом недопустимой операции деления на ноль будет бесконечность с соответствующим знаком *(см. главу 3)*.

Переполнение фиксируется, когда значение результата выходит за границы диапазона его представления *(см. главу 3)*. Несмотря на то, что диапазон представления данных с ПТ значительно шире, чем для данных с ФТ, в некоторых случаях возможно переполнение результатов, представляемых в формате расширенного слова SEP. Однако более вероятен особый случай переполнения при пересылках — сохранении результатов, представленных в формате расширенного слова SEP, в регистрах с форматом слова SP. В обоих случаях содержимое с переполнением автоматически заменяется значе-

нием, регламентируемым стандартом IEEE 754. Обработка переполнения зависит от режима округления. В табл. 4.2 представлены значения, на которые автоматически заменяются результаты операции при переполнении.

Таблица 4.2. *Автоматическая замена результата при переполнении*

Режим округления	Знак истинного результата	Замена результата при переполнении
До ближайшего четного	+	$+\infty$
	–	$-\infty$
В направлении нуля	+	Максимальное положительное число
	–	Максимальное (по модулю) отрицательное число
В направлении $-\infty$	+	Максимальное положительное число
	–	$-\infty$
В направлении $+\infty$	+	$+\infty$
	–	Максимальное (по модулю) отрицательное число

Потеря значимости результата — это особый случай, присущий обработке данных с ПТ и иногда называемый *антипереполнением*. Потеря значимости результата возникает, когда одновременно происходят два следующих события:

- результат становится бесконечно малым и представляется ненормализованным числом (см. рис. 3.35 для формата SP);
- результат (ненормализованное число) невозможно точно представить в заданном формате.

Появление этих взаимосвязанных событий фиксируется установкой в регистре состояния двух битов: бита потери значимости и бита потери точности (см. следующий особый случай "потеря точности").

Поясним на примерах представления результатов операции умножения в формате SP. Для упрощения будем выполнять расчеты с несмещенным значением порядка.

Пример 1. Результат умножения двоичных нормализованных чисел

$$(1{,}01 \cdot 2^{-126}) \cdot (1{,}00 \cdot 2^{-67}) = 1{,}01 \cdot 2^{-193}$$

является, во-первых, бесконечно малым и представляется как ненормализованное число, и, во-вторых, не может быть представлен в формате SP. Значению ненормализованного результата, равному

$$1{,}01 \cdot 2^{-193} = 0{,}101 \cdot 2^{-192} = (0{,}101 \cdot 2^{-66}) \cdot 2^{-126},$$

для точного представления требуется длина мантиссы значительно бóльшая, чем 24 бита в формате SP.

Результат округляется до нуля (независимо от режима округления).

В регистре состояния одновременно устанавливаются биты потери значимости и потери точности.

Пример 2. Результат умножения двоичных нормализованных чисел

$$(1{,}01 \cdot 2^{-126}) \cdot (1{,}00 \cdot 2^{-23}) = 1{,}01 \cdot 2^{-149} = (0{,}101 \cdot 2^{-23}) \cdot 2^{-126}$$

является также бесконечно малым и представляется как ненормализованное число. Его значение можно представить в формате SP, однако это значение будет неточным (округленным) из-за недостаточной длины мантиссы (24 бита) в формате SP. Неточное (округленное) значение ненормализованного числа равно

$$0{,}00000000000000000000001 \cdot 2^{-126}.$$

Так же, как в примере 1, в регистре состояния одновременно устанавливаются биты потери значимости и потери точности.

Пример 3. Результат умножения двоичных нормализованных чисел

$$(1{,}01 \cdot 2^{-126}) \cdot (1{,}00 \cdot 2^{-7}) = 1{,}01 \cdot 2^{-133} = (0{,}101 \cdot 2^{-6}) \cdot 2^{-126}$$

является бесконечно малым и представляется как ненормализованное число. Однако его значение, равное

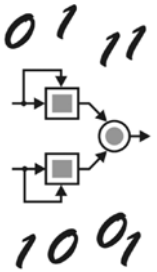
$$0{,}000000101 \cdot 2^{-126},$$

можно представить в формате SP точно, как ненормализованное число. В этом случае нет выполнения двух указанных выше условий, поэтому биты потери значимости и потери точности не устанавливаются.

Особый случай *потеря точности* отображается установкой соответствующего бита в регистре состояния, если выдается неточный (округленный) результат.

Результаты операций в рассмотренных примерах могут быть представлены точно в формате SEP, если он поддерживается соответствующей командой.

Глава 5



Адресация

Команда, предназначенная для выполнения операции над данными, содержит код операции и указание на операнды (исходные данные и результат).

Адресацией называется обращение к операнду, указание на который содержится в команде. Операнды, в зависимости от места своего хранения, могут указываться разными способами, которым соответствуют разные типы адресации, или, коротко, разные адресации. В данной главе рассматриваются основные типы адресаций.

Предварительно определим понятие формата команды.

Команды размещаются в n-разрядных ячейках памяти программ, их содержимое отображается словами длины n.

Различные источники трактуют термин "формат команды" по-разному. В частности, в руководствах пользователя по процессорам фирмы Analog Devices под этим подразумевают структуру слова команд, а в руководствах по процессорам фирмы Motorola форматом команды считают ее ассемблерный синтаксис. При описании команд процессоров фирмы Texas Instruments термин "формат команды", как правило, не используется.

Определим формат команды аналогично формату данных. *Формат команды* связан с разрядностью ячеек памяти программ, в которых хранятся команды; он определяет возможную длину команды в процессоре. Формат команды (рис. 5.1) в общем случае представляет собой два слова одинаковой длины n, соответствующие двум соседним n-разрядным ячейкам памяти программ.

Первое слово формата будем называть *словом команды* (его также называют словом операции), второе — *словом расширения*.

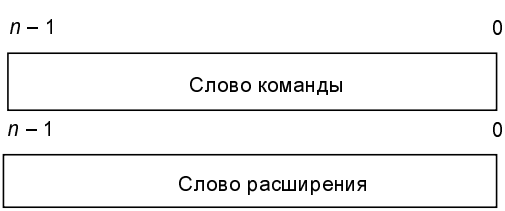


Рис. 5.1. Формат команды

Слово команды хранит код операции, указания на операнды и другую информацию о команде.

Слово расширения может хранить указание на тот операнд, для которого при определенной адресации недостаточно места в слове команды.

Различными процессорами поддерживаются однословные и двусловные форматы команд. В процессорах с однословным форматом слово расширения отсутствует, и указания на все операнды, независимо от типа адресации, размещаются в слове команды. В процессорах с двусловным форматом слово расширения может использоваться по назначению, либо отсутствовать, в этом случае команда становится однословной.

Замечание

Иногда применяется трехсловный формат команды (например, в процессорах MSC810x фирмы Motorola), который отнесем к числу исключений.

Структурой слова команды будем называть условное разделение слова команды на части (поля), содержащие различного рода информацию о команде. Структура слова команды рассматривается в *главе 6*.

Количество операндов в команде, а также возможные способы их указания определяются архитектурой процессора и спецификой конкретной команды. В настоящей главе обсуждаются основные из возможных способов указания операндов, безотносительно их использования в команде.

Согласно месту хранения операнды могут указываться:

- *адресами ячеек памяти данных*, если они хранятся в ячейках памяти данных; номер ячейки, в которой хранится операнд, называется его *адресом*;
- *именами регистров*, если они хранятся в регистрах;
- *константами*, если они хранятся в слове команды или в слове расширения.

В свою очередь, место хранения определяет возможные способы указания операнда в команде:

- для адресов ячеек памяти данных — *прямой* или *косвенный*;
- для имен регистров — только *прямой*;
- для констант — *непосредственный*.

Трем способам указаний на операнды соответствуют три типа адресации или три различные адресации:

- прямая;
- косвенная;
- непосредственная.

В командах *управления* выполнением операций (безусловных и условных переходах, вызовах подпрограмм, аппаратных циклах и т. п.) указывается адрес ячейки памяти программ — адрес другой команды, к которой осуществляется переход. Эту команду для краткости часто называют *переходом*, ее адрес — *адресом перехода*, а обращение к ней — *адресацией перехода* (в отличие от адресации операнда). Адрес перехода может указываться прямо или косвенно, что соответствует двум различным адресациям переходов; их особенности также рассматриваются в этой главе.

При описании различных адресаций операндов будем называть:

- *исполняемым адресом* — адрес операнда (исходных данных или результата), указываемый (прямо или косвенно) или вычисляемый по указанию в команде;
- *источником* — регистр, из которого считывают операнд (исходных данных);
- *приемником* — регистр, в который записывают операнд (результат).

Заметим, что исполняемый адрес, источник или приемник всегда связан с операндом, поэтому не требуется уточнений типа "исполняемый адрес операнда" и т. п.

5.1. Прямая адресация

Адресацию операнда называют *прямой*, если в команде непосредственно указывается:

- исполняемый адрес;
- адрес операнда на странице памяти данных, по которому автоматически вычисляется исполняемый адрес;
- имя источника или приемника.

Рассмотрим особенности каждой из разновидностей прямой адресации операндов.

5.1.1. Прямое указание адресов

Операнды могут прямо указываться исполняемыми адресами или адресами на странице памяти данных, если процессором поддерживается этот тип адресации. Различают процессоры, *поддерживающие* прямое указание адресов:

- в командах операций над данными и пересылок данных, например, процессоры TMS320C2xxx/5xxx фирмы Texas Instruments;
- только в командах пересылок данных, например, процессоры фирм Motorola и Analog Devices,

а также процессоры, *не поддерживающие* прямое указание адресов ни в одной команде, например, процессоры TMS320CC6xxx фирмы Texas Instruments.

Исполняемый адрес

Если в процессоре отсутствует условное разделение памяти данных на страницы, исполняемый адрес указывается полностью.

При двусловном формате короткие исполняемые адреса хранятся в слове команды, длинные — в слове расширения.

Приведем примеры прямого указания исполняемого адреса.

1. Команда пересылки

```
MOVE X:$5415, A0
```

процессора DSP5600x фирмы Motorola содержит операнд, указанный исполняемым 16-разрядным абсолютным адресом `$5415` в X-памяти данных (память данных в процессорах фирмы Motorola имеет два отдельно адресуемых пространства — X и Y); исполняемый адрес хранится в слове расширения.

Пример выполнения команды приведен на рис. 5.2. По команде `MOVE` происходит:

($5415) $_{\text{X-памяти}}$ → A0.

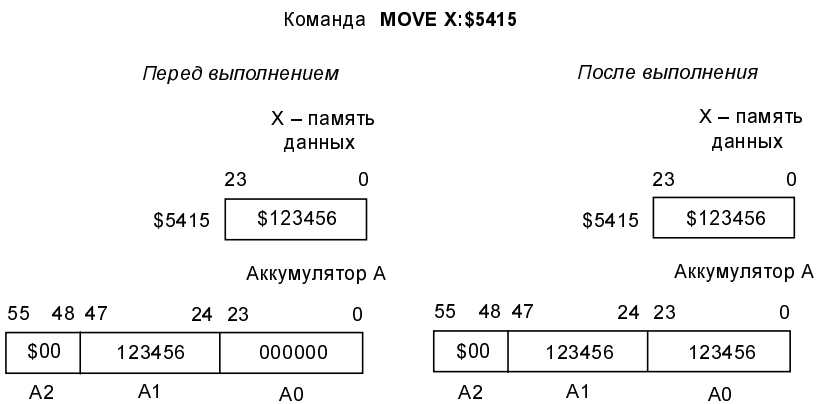


Рис. 5.2. Пример выполнения команды `MOVE X:$5415, A0`

2. Команда пересылки

```
AX0 = DM(0x1000);
```

процессора ADSP-21xx фирмы Analog Devices содержит операнд, указанный исполняемым 16-разрядным абсолютным адресом `0x1000`, который хранится в слове команды (формат всех команд в процессорах фирмы Analog Devices — однословный).

Пример выполнения команды приведен на рис. 5.3. По команде происходит:

(0x1000) → AX0.

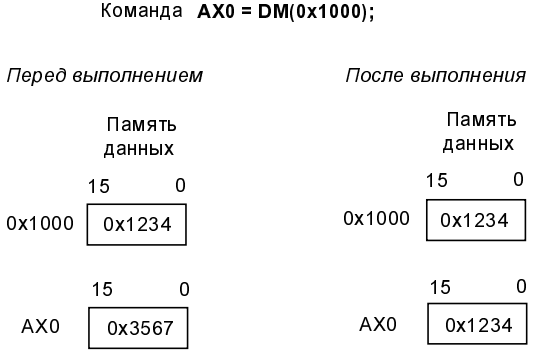


Рис. 5.3. Пример выполнения команды `AX0 = DM(0x1000);`

Адрес на странице памяти данных

Прямое указание адреса операнда на странице памяти данных используется в процессорах, память данных которых условно разделена на страницы, например, в процессорах TMS320C2xxx/5xxx/3x.

Исполняемый адрес вычисляется автоматически с учетом страницы, номер которой задается пользователем и хранится в указателе страницы DP (Data Pointer), размещаемом в регистре состояния.

Приведем примеры прямого указания адреса на странице памяти данных.

3. Команда сложения

   ```
   ADD X
   ```

   процессора TMS320C2xxx содержит операнд, указанный символическим именем `X` адреса на странице памяти данных.

   Абсолютный адрес a на странице p, соответствующий символическому имени X, хранится в слове команды, в специально отведенных 7 битах. Номер страницы p задается программно в 9-разрядном регистре указателя страницы DP, размещаемом в регистре состояния. Исполняемый 16-разрядный адрес вычисляется как $a + p \times 128$ (страница содержит 128 ячеек).

   Пример выполнения команды приведен на рис. 5.4. По команде `ADD` происходит:

   (A) + (X) → A.

4. Команда пересылки в памяти данных

   ```
   MVKD 1000h, X
   ```

   процессора TMS320C5xxx фирмы Texas Instruments содержит:

   - операнд, указанный символическим именем `X` адреса на странице памяти данных; исполняемый адрес вычисляется так же, как в примере 1;

- операнд, указанный исполняемым 16-разрядным абсолютным адресом 1000h, который хранится в слове расширения команды.

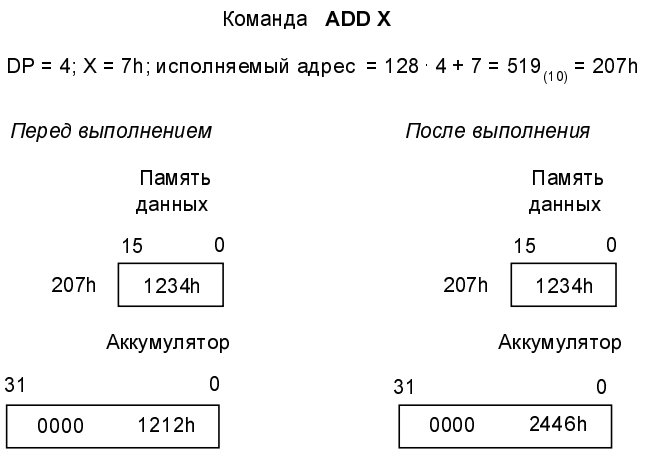


Рис. 5.4. Пример выполнения команды `ADD X`

Пример выполнения команды приведен на рис. 5.5. По команде `MVKD` происходит:

(1000h) → X.

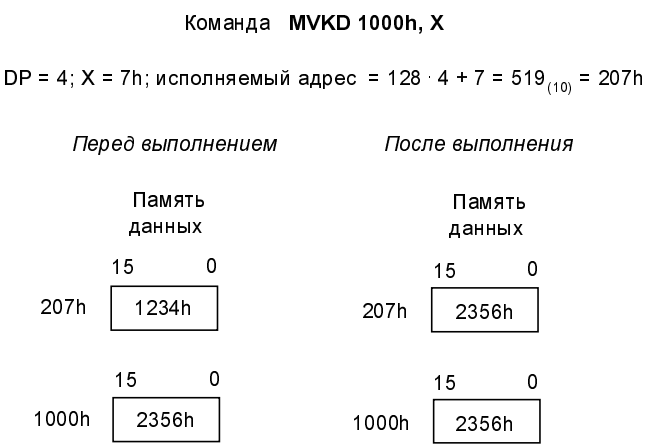


Рис. 5.5. Пример выполнения команды `MVKD 1000h, X`

5. Команда умножения с накоплением

   `MAC DAT1, DAT2, A, B`

   процессора TMS320C54xx содержит четыре операнда:

   - два операнда (сомножители), указанные символическими именами `DAT1` и `DAT2` адресов на странице памяти данных;
   - два операнда, указанные как `A` и `B` (см. пример 10).

   Согласно правилу выполнения команды, оба адреса `DAT1` и `DAT2` хранятся в слове команды (на каждый из них отведено по 4 бита) и находятся на одной странице памяти, указанной в 9-разрядном регистре указателя страницы DP. Исполняемые 13-разрядные (4 + 9) адреса операндов вычисляются так же, как в примере 1.

   Пример выполнения команды приведен на рис. 5.6. По команде `MAC` происходит:

   (DAT1) × (DAT2) + (A) → B;

   (DAT1) → регистр T.

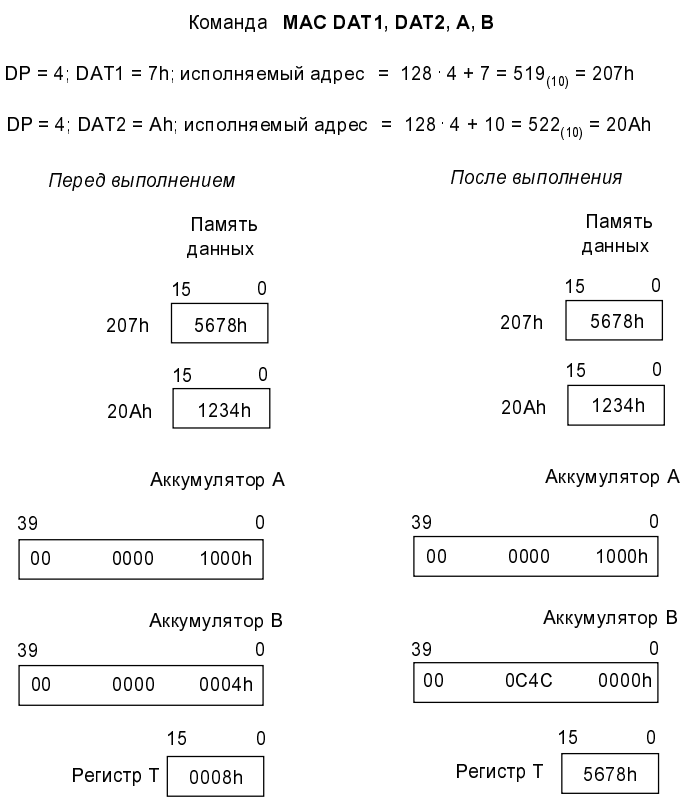


Рис. 5.6. Пример выполнения команды `MAC DAT1, DAT2, A, B`

6. Команда загрузки ячейки стека

```
PSHD SIMP
```

процессора TMS320C5xxx содержит операнд, указанный символическим именем `SIMP` адреса на текущей странице памяти данных. Исполняемый адрес вычисляется так же, как в примерах 1, 3.

Пример выполнения команды приведен на рис. 5.7. По команде `PSHD` происходит:

(SP) − 1 → SP (SP — указатель стека);

(SIMP) → ячейку, указанную SP (вершину стека).

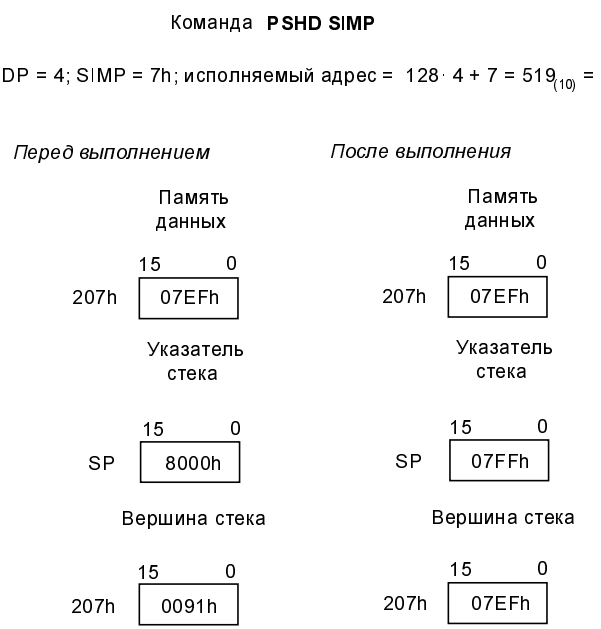


Рис. 5.7. Пример выполнения команды `PSHD SIMP`

7. Команда сложения целых чисел

```
ADDI @0BCDEh, R5
```

процессора TMS320C3x содержит операнд, указанный абсолютным адресом `BCDEh` на странице памяти данных. Исполняемый 24-разрядный адрес определяется как склейка 16-ти младших битов слова команды (младшие биты исполняемого адреса) и 8-ми младших битов указателя страницы DP (старшие биты исполняемого адреса). Например, если DP = 8Ah, исполняемый адрес равен 8ABCDEh.

Пример выполнения команды приведен на рис. 5.8. По команде `ADDI` происходит:

(8ABCDEh) + (R5) → R5.

Замечание

В этом примере и далее при записи в команде шестнадцатеричного эквивалента используются префиксы и суффиксы, принятые в версиях языка ассемблера соответствующего процессора.

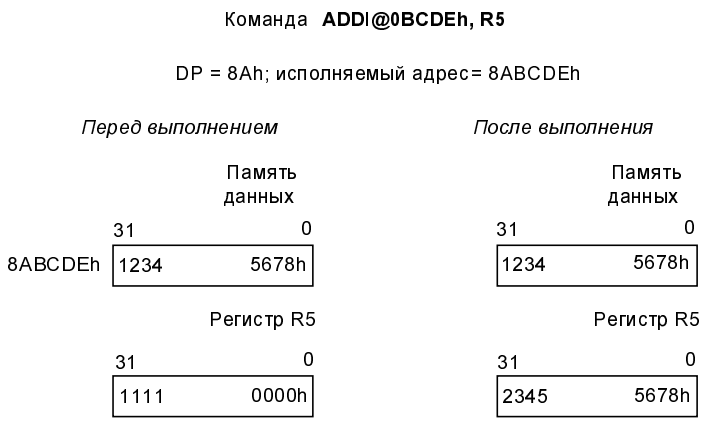


Рис. 5.8. Пример выполнения команды `ADDI @0BCDEh, R5`

5.1.2. Прямое указание источников и приемников

Если операнд хранится в регистре, то в команде он прямо указывается именем регистра — источника или приемника.

Такая разновидность прямой адресации поддерживается всеми процессорами, причем, процессоры, которые содержат специальные регистры операндов для хранения исходных данных или/и результатов, в командах операций над данными (кроме пересылок) поддерживают только эту разновидность прямой адресации. Сказанное справедливо для процессоров фирм Motorola, Analog Devices и TMS320C6xxx фирмы Texas Instruments.

Источник или приемник может указываться:

- явно, если существует возможность выбора;
- неявно, если возможности выбора нет.

Приведем примеры прямого указания источников и приемников.

8. Команда сложения

```
ADD X0, Y1, A
```

процессора DSP5600x фирмы Motorola содержит:

- два операнда, указанных явно именами источников — входных регистров `X0` и `Y1`;
- операнд, указанный явно именем приемника `A` — аккумулятора A.

Пример выполнения команды приведен на рис. 5.9. По команде происходит:

(X0) + (Y1) → A.

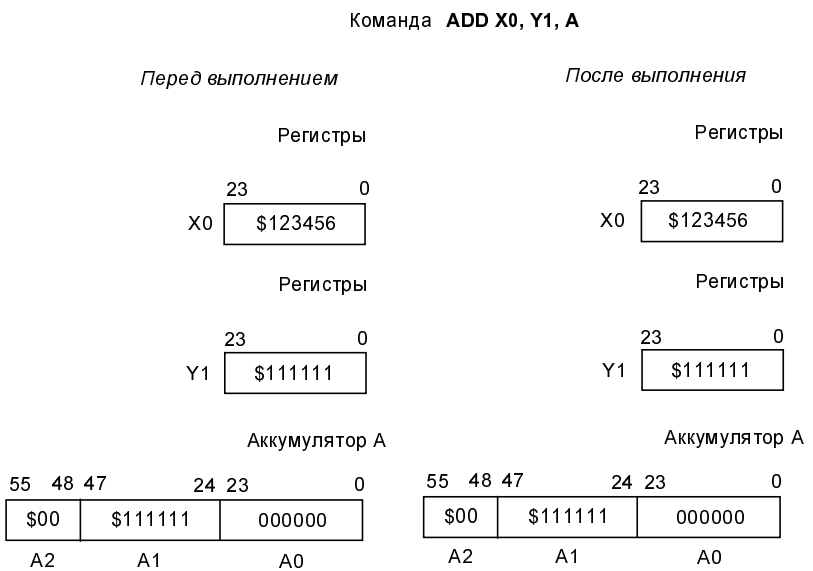


Рис. 5.9. Пример выполнения команды `ADD X0, Y1, A`

9. Команда сложения

```
AR=AX0+AX1;
```

процессора ADSP-21xx фирмы Analog Devices содержит:

- два операнда, указанные явно именами источников — входных регистров `AX0` и `AX1`;
- операнд, указанный явно именем приемника — выходного регистра `AR`.

Пример выполнения команды приведен на рис. 5.10. По команде происходит:

(AX0) + (AX1) → AR.

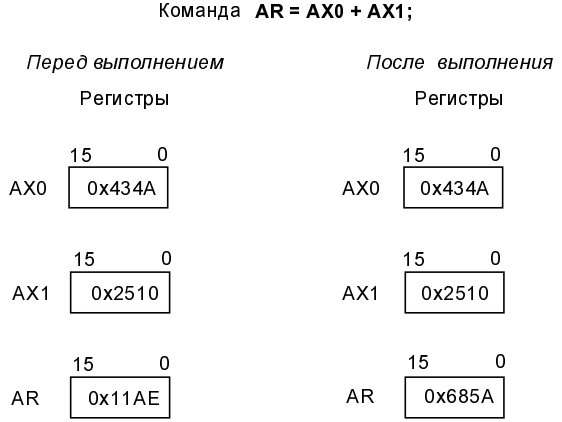


Рис. 5.10. Пример выполнения команды `AR=AX0+AX1;`

10. В примере 3 операнды (слагаемое и сумма) указаны неявно одним и тем же именем источника и приемника — аккумулятора `A`; возможности выбора аккумулятора не существует.

    В примере 5 два операнда указаны явно именами: источника — аккумулятора A и приемника — аккумулятора `B`.

    В примере 6 операнд указан неявно приемником — вершиной стека (верхней ячейкой стека, задаваемой указателем стека).

11. Команда пересылки

    ```
    MOVE R0, R4
    ```

    процессора DSP5600x фирмы Motorola содержит два операнда, указанных явно именами источника и приемника — регистров `R0` и `R4`.

    Пример выполнения команды приведен на рис. 5.11. По команде происходит:

    (R0) → (R4).

12. Команда сравнения

    ```
    CMPR 2, AR4
    ```

    процессора TMS320C54xx фирмы Texas Instruments содержит:

    - операнд, указанный явно именем источника `AR4` — регистра AR4;
    - операнд, указанный неявно именем источника `AR0` — регистра AR0, с содержимым которого в данной команде происходит сравнение по умолчанию.

    Число `2` в команде соответствует коду операции сравнения (>).

Пример выполнения команды приведен на рис. 5.12. По команде происходит следующее:

если (AR4) > (AR0) → бит TC = 1, иначе, бит TC = 0. Бит TC находится в регистре состояния ST0. Содержимое регистров AR4 и AR0 трактуется как беззнаковые целые числа.

Рис. 5.11. Пример выполнения команды `MOVE R0, R4`

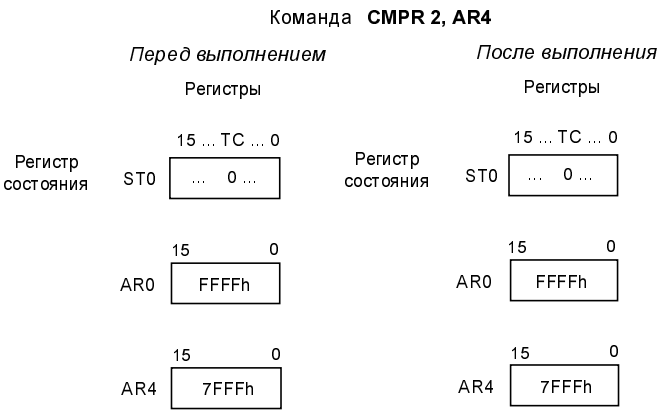


Рис. 5.12. Пример выполнения команды `CMPR 2, AR4`

5.1.3. Прямая адресация переходов

Прямая адресация переходов означает прямое указание адреса перехода, который копируется в счетчик команд, изменяя последовательность их выполнения.

Хранение адреса перехода зависит от длины формата команды; при двусловном формате короткие адреса хранятся в слове команды, длинные — в слове расширения.

Приведем примеры прямой адресации переходов.

13. Команда безусловного перехода

```
JMP $123
```

процессора DSP5600x фирмы Motorola содержит переход (безусловный), указанный прямо адресом перехода — абсолютным 12-разрядным коротким адресом в памяти программ.

14. Команда цикла

```
DO start UNTIL CE
```

процессора ADSP-21xx фирмы Analog Devices содержит переход (по условию `CE` на последнюю команду тела цикла), указанный прямо адресом перехода — меткой `start`.

15. Команда безусловного перехода

```
BR 8000h
```

процессора TMS320C3x содержит переход (безусловный), указанный прямо адресом перехода — абсолютным 16-разрядным длинным адресом `8000h` ячейки памяти программ.

5.2. Косвенная адресация

Адресацию операнда называют *косвенной*, если исполняемый адрес указывается именем регистра, в котором он хранится. Соответствующий регистр называют *регистром адреса*. При косвенной адресации слово расширения, как правило, не используется.

Косвенная адресация поддерживается всеми сигнальными процессорами.

Для независимого выполнения операций над данными и над адресами в архитектуре большинства процессоров предусмотрены *устройства генерации адреса* (УГА). В процессорах фирмы Motorola — это устройство генерации адреса AGU (Address Generation Unit), в процессорах фирмы Analog Devices — генераторы адресов данных DAG1 и DAG1 (Data Address Generator), в процессорах TMS320C2xxx/5xxx фирмы Texas Instruments — арифметические устройства вспомогательных регистров ARAU (Auxiliary Register Arithmetic Unit), в процессорах TMS320C6xxx — D-модули генерации адреса данных.

Вычисление адресов в УГА основано на одинаковых или сходных принципах. Оно предполагает наличие трех типов регистров (условное наименование которых соответствует выполняемым функциям):

- регистра адреса;
- регистра смещения;
- регистра типа арифметики.

Назначение регистров смещения и типа арифметики рассматривается далее.

Количество и разрядность регистров определяется конкретным процессором. Разрядность регистра адреса, обусловленная разрядностью шины адреса, определяет объем адресуемого пространства памяти данных.

Принципы косвенной адресации будут рассмотрены на примерах процессоров DSP5600x фирмы Motorola, TMS320C54xx фирмы Texas Instruments и ADSP-218x Analog Devices, архитектуру УГА которых можно считать базовой для процессоров данной фирмы. Другие УГА представляют собой соответствующую расширенную или сокращенную версии.

Символические имена и названия типовых регистров УГА для данных процессоров приведены в табл. 5.1.

В рассматриваемых процессорах используются 16-разрядные регистры адреса, что обеспечивает возможность адресации любой ячейки памяти данных в пределах 64 Кбайт. Регистры смещения и типа арифметики также 16-разрядные.

Таблица 5.1. *Регистры устройства генерации адреса (УГА)*

Условное название регистра	**Символические имена и названия регистров в процессоре фирмы**		
	Motorola DSP5600x	Analog Devices ADSP-218x	Texas Instruments TMS320C54xx
Регистр адреса	R*n* — регистр адреса *n* = 0,1, ..., 7	I*n* — регистр индекса *n* = 0, 1, ..., 7	Ar*n* — вспомогательный регистр общего назначения *n* = 1, ..., 7
Регистр смещения	N*n* — регистры смещения *n* = 0, 1, ..., 7	M*n* — регистры модификации *n* = 0, 1, ..., 7	AR0 — вспомогательный регистр общего назначения
Регистр типа арифметики	M*n* — регистры модификации *n* = 0, 1, ..., 7	L*n* — регистры длины *n* = 0, 1, ..., 7	BK — регистр размера циклического буфера

Использование регистров общего назначения AR*n* в процессорах фирмы Texas Instruments в качестве регистров адреса (табл. 5.1) отмечается префиксом * перед именем конкретного регистра.

Приведем примеры косвенной адресации. Во всех примерах содержимое регистров смещения и типа арифметики не влияет на исполняемый адрес, хранимый в регистре адреса. В командах имя регистра в круглых скобках соответствует содержимому регистра.

16. Команда пересылки

```
MOVE A1, X:(R0)
```

процессора DSP5600x фирмы Motorola содержит:

- операнд, исполняемый адрес которого указан косвенно (R0) — именем регистра адреса R0;
- операнд, указанный прямо именем источника A1 — аккумулятора A1.

Пример выполнения команды приведен на рис. 5.13. По команде происходит:

(A1) → в ячейку X-памяти.

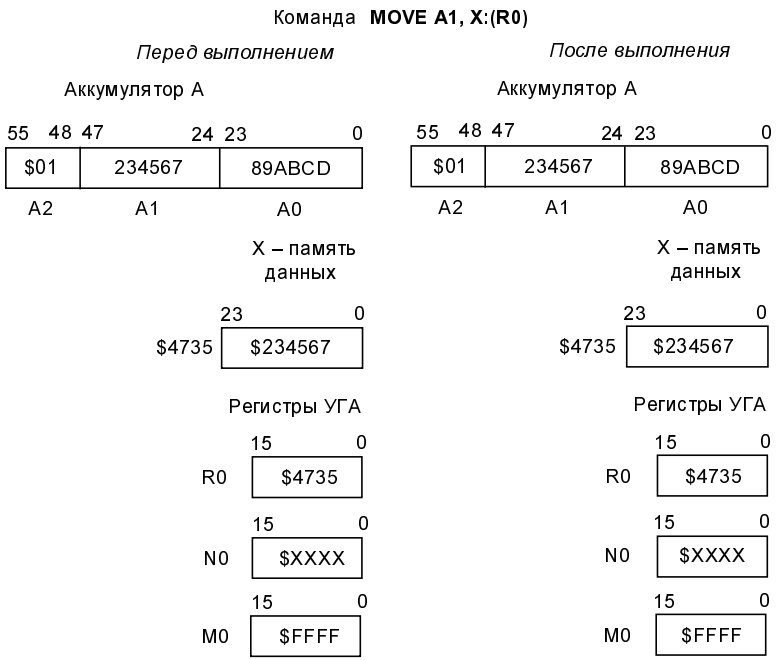


Рис. 5.13. Пример выполнения команды MOVE A1, X:(R0)

17. Команда пересылки

```
AX0=DM(I2,M2);
```

процессора ADSP-218x фирмы Analog Devices содержит:

- операнд, исполняемый адрес которого указывается косвенно (I2) — именем регистра адреса I2;
- операнд, указанный прямо именем приемника AX0 — входного регистра AX0.

Пример выполнения команды приведен на рис. 5.14. По команде содержимое ячейки памяти данных, адресом которой является (I2), пересылается во входной регистр AX0.

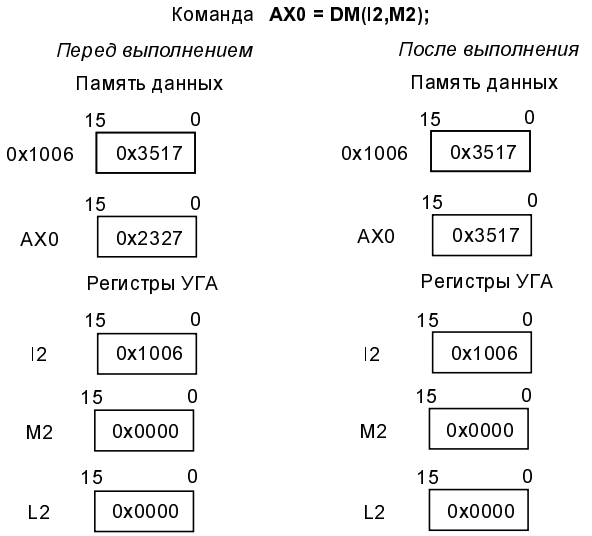


Рис. 5.14. Пример выполнения команды `AX0 = DM(I2,M2);`

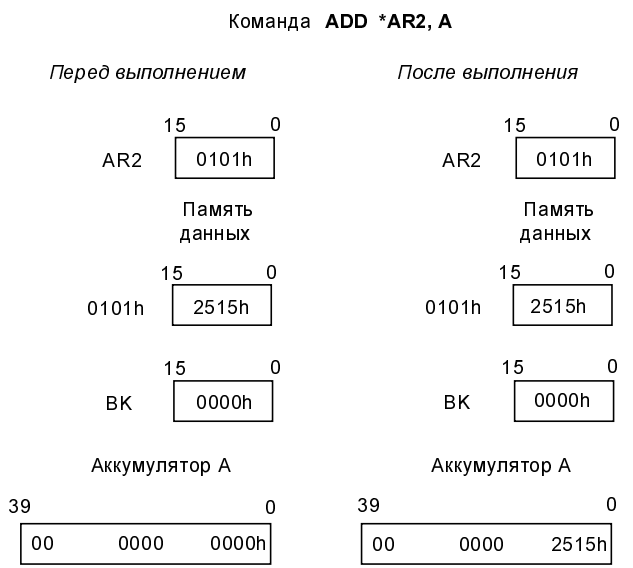


Рис. 5.15. Пример выполнения команды `ADD *AR2, A`

18. Команда сложения

    `ADD *AR2, A`

    процессора TMS320C54xx фирмы Texas Instruments содержит:

    - операнд, исполняемый адрес которого указывается косвенно — именем регистра адреса `AR2`;
    - операнд, указанный прямо именем приемника — аккумулятора `A`.

    Префикс `*` означает, что вспомогательный регистр используется как регистр адреса (сравните примеры 12 и 17).

    Пример выполнения команды приведен на рис. 5.15. По команде операнд, адресом которого является (AR2), складывается с содержимым аккумулятора A; результат сохраняется в аккумуляторе A.

5.2.1. Модификация адреса

Модификация (изменение) адреса означает автоматическое вычисление нового адреса, выполняемое в УГА независимо от основной операции команды. Модификация адреса может производиться как до, так и после выполнения основной операции.

В зависимости от типа модификации, указываемого в команде, может измениться содержимое регистра адреса и/или исполняемый адрес в данной команде, что, в отличие от рассмотренной выше косвенной адресации без модификации адреса, не одно и то же.

Различают следующие основные типы модификации:

- ❒ постдекремент на 1;
- ❒ постинкремент на 1;
- ❒ предекремент на 1;
- ❒ преинкремент на 1;
- ❒ постдекремент на N;
- ❒ постинкремент на N;
- ❒ индексация.

Рассмотрим подробнее каждую из модификаций.

Постдекремент/постинкремент и предекремент/преинкремент адреса на 1

Постдекремент/постинкремент адреса на 1 при косвенной адресации означает автоматическое уменьшение/увеличение содержимого регистра адреса на 1 *после* выполнения основной операции команды, следовательно, исполняемый адрес *в данной команде* не изменяется.

Предекремент/преинкремент адреса на 1 при косвенной адресации означает автоматическое уменьшение/увеличение содержимого регистра адреса на 1

перед выполнением команды, следовательно, *перед* выполнением команды вычисляется новый исполняемый адрес.

Большинство алгоритмов ЦОС связано с обработкой массивов данных (отчетов, коэффициентов и т. д.). Косвенная адресация с пост- и предекрементами/преинкрементами на 1 обеспечивает простую организацию программного обращения к соседним элементам массива в памяти данных.

Синтаксис косвенного указания адреса при пост- и предекременте/преинкременте на 1 приведен в табл. 5.2.

В процессорах DSP5600x фирмы Motorola при постдекременте/постинкременте адреса на 1 исполняемым адресом операнда в команде является содержимое регистра адреса R*n*, которое *после* выполнения операции декрементируется/инкрементируется на 1 и становится равным (R*n*) − 1 > R*n* или (R*n*) + 1 → → R*n* соответственно. При предекременте адреса на 1 содержимое регистра адреса R*n* уменьшается на 1 *перед* выполнением операции, R*n* → (R*n*) − 1 и становится исполняемым адресом.

В процессорах ADSP-218x фирмы Analog Devices для постдекремента/постинкремента адреса используются регистры смещения M*n*. Исполняемым адресом в команде является содержимое регистра адреса I*n*, которое после выполнения операции изменяется и становится равным (I*n*) + (M*n*) → I*n*. Содержимое регистров модификации трактуется как целое со знаком. При постдекременте на 1 содержимое регистра M*n* равно −1, (M*n*) = −1, а при постинкременте на 1 оно равно +1, (M*n*) = + 1.

В процессорах TMS320C54xx фирмы Texas Instruments при постдекременте/инкременте адреса на 1 исполняемым адресом является содержимое вспомогательного регистра AR*n*, которое *после* выполнения операции декрементируется/инкрементируется на 1 и становится равным (AR*n*) − 1 → AR*n* или (AR*n*) + 1 → AR*n* соответственно. При предекременте адреса на 1 содержимое регистра адреса AR*n* декрементируется на 1 *перед* выполнением операции, Ar*n* → (AR*n*) − 1, и становится исполняемым адресом.

Таблица 5.2. *Постдекремент/постинкремент и предекремент/преинкремент адреса на 1*

Модификация	Синтаксис косвенного указания адреса в процессорах фирмы		
	Motorola DSP5600x	Texas Instruments TMS320C54xx	Analog Devices ADSP-218x
Нет изменения	(R*n*)	*AR*n*	(I*n*, M*n*) при M*n* = 0
Постдекремент на 1	(R*n*) –	*Ar*n*–	(I*n*, M*n*) при M*n* = –1
Постинкремент на 1	(R*n*)+	*AR*n*+	(I*n*, M*n*) при M*n* = +1
Предекремент на 1	– (R*n*)	*–Ar*n*	Нет
Преинкремент на 1	Нет	*+AR*n*	Нет

Приведем примеры косвенной адресации с постдекрементом/постинкрементом адреса на 1. Во всех вариантах содержимое регистров смещения и типа арифметики не влияет на содержимое регистра адреса.

19. После выполнения команды пересылки

```
MOVE A1, X:(R0)-
```

в процессоре DSP5600x фирмы Motorola содержимое R0 декрементируется;

(R0) − 1 → R0 (сравните с примером 16).

Пример выполнения команды приведен на рис. 5.16 (сравните с рис. 5.13).

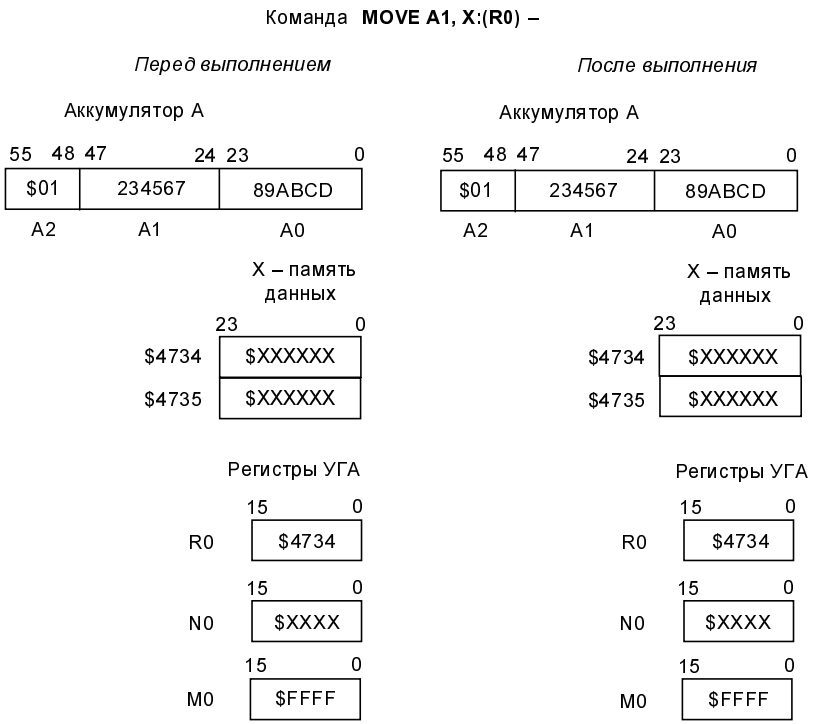


Рис. 5.16. Пример выполнения команды `MOVE A1, X:(R0)-`

20. После выполнения команды пересылки

```
AX0=DM(I2,M2);
```

в процессоре ADSP-218x фирмы Analog Devices содержимое I2 инкрементируется;

(I2) + 1 → I2 (сравните с примером 17).

Пример выполнения команды приведен на рис. 5.17 (сравните с рис. 5.14).

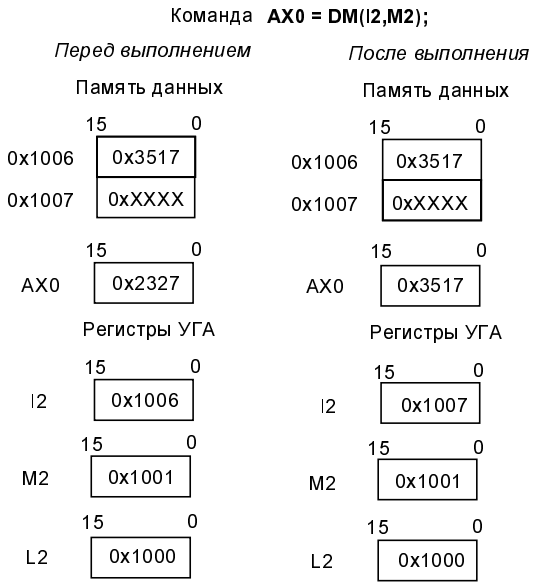


Рис. 5.17. Пример выполнения команды `AX0=DM(I2,M2);`

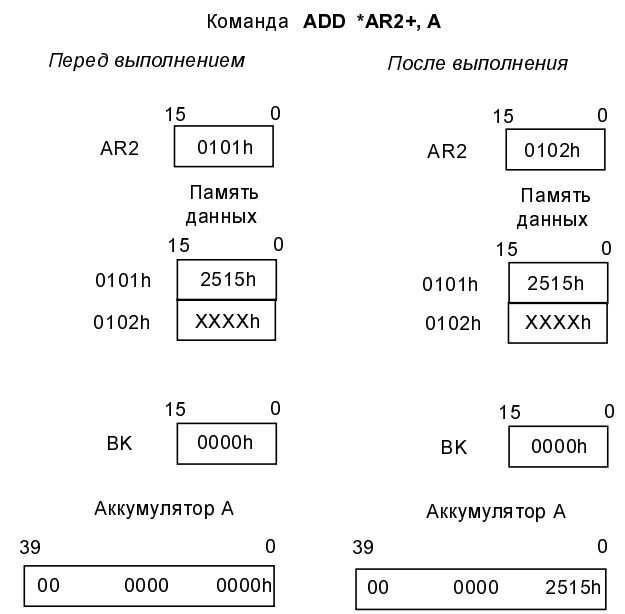


Рис. 5.18. Пример выполнения команды `ADD *AR2+, A`

21. После выполнения команды сложения

```
ADD *AR2+, A
```

в процессоре TMS320C54xx фирмы Texas Instruments содержимое AR2 инкрементируется;

(AR2) + 1 → AR2 (сравните с примером 18).

Пример выполнения команды приведен на рис. 5.18 (сравните с рис. 5.15).

Постдекремент/постинкремент и предекремент/преинкремент адреса на *N*

Постдекремент/постинкремент адреса на *N* при косвенной адресации означает автоматическое уменьшение/увеличение содержимого регистра адреса на целое число *N после* выполнения команды, следовательно, исполняемый адрес *в данной команде* не изменяется.

Подобная модификация обеспечивает простую организацию программного обращения к произвольным элементам массива в памяти данных.

Число *N*, называемое *смещением*, хранится в регистре смещения. Символические имена и названия регистров смещения в процессорах различных фирм приведены в табл. 5.1.

Синтаксис косвенного указания адреса при постдекременте/постинкременте на *N* приведен в табл. 5.3.

В процессорах DSP5600x фирмы Motorola при постдекременте/постинкременте адреса на *N* (где *N* — беззнаковое целое) исполняемым адресом является содержимое регистра адреса R*n*, которое после выполнения команды изменяется и становится равным (R*n*) + (N*n*) → R*n* при постинкременте и равным (R*n*) − (N*n*) → R*n* при постдекременте.

В процессорах ADSP-218x фирмы Analog Devices при постдекременте/постинкременте адреса на *N* (где *N* — целое со знаком) исполняемым адресом является содержимое регистра I*n*, которое после выполнения команды изменяется и становится равным (I*n*) + (M*n*) → I*n*. При постдекременте содержимое M*n* равно −*N*, (M*n*) = −*N*, а при постинкременте оно равно +*N*, (M*n*) = + *N*.

В процессорах TMS320C54xx фирмы Texas Instruments при постдекременте/постинкременте адреса на *N* (где *N* — беззнаковое целое) исполняемым адресом является содержимое регистра адреса AR*n* (*n* = 1, 2, ..., 7), которое после выполнения команды изменяется и становится равным (AR*n*) + + (AR0) → AR*n* при постинкременте и (AR*n*) − (AR0) → A при постдекременте.

Таблица 5.3. *Постдекремент/постинкремент адреса на N*

Модификация	Синтаксис косвенного указания адреса в процессорах фирмы		
	Motorola DSP5600x	Texas Instruments TMS320C54xx	Analog Devices ADSP-218x
Постдекремент на N	(R*n*) – N*n*	*AR*n* – 0	(I*n*, M*n*) при M*n* = – *N*
Постинкремент на N	(R*n*) + N*n*	*Ar*n* + 0	(I*n*, M*n*) при M*n* = + *N*

Приведем примеры косвенной адресации с постдекрементом/постинкрементом адреса на *N*. Во всех примерах содержимое регистра типа арифметики не влияет на содержимое регистра адреса.

22. После выполнения команды пересылки

```
MOVE A1, X:(R0)-N0
```

в процессоре DSP5600x фирмы Motorola содержимое `R0` декрементируется;

(R0) − (N0) = (R0) − 3 → R0 (сравните с примером 16).

Пример выполнения команды приведен на рис. 5.19 (сравните с рис. 5.13, 5.16).

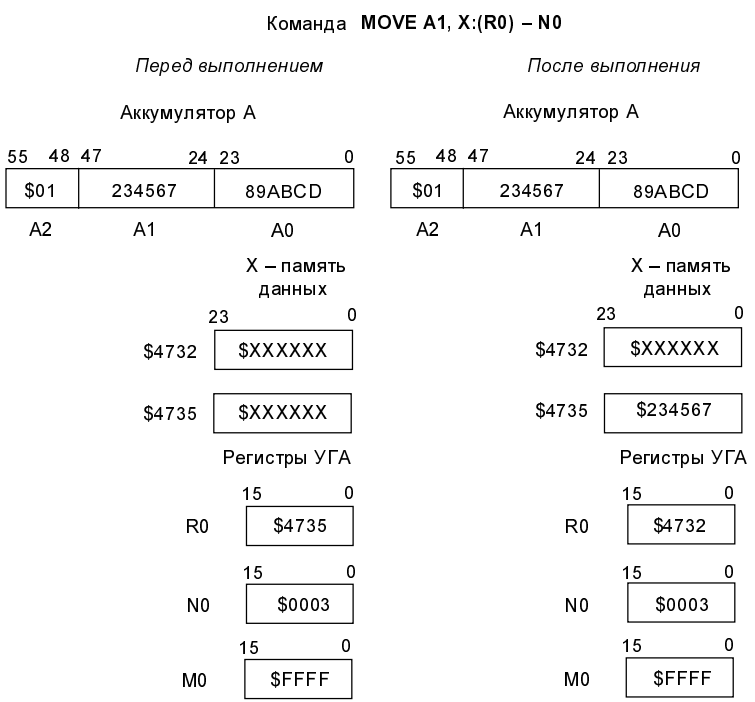


Рис. 5.19. Пример выполнения команды `MOVE A1, X:(R0)-N0`

23. После выполнения команды пересылки

```
AX0=DM(I2,M2);
```

в процессоре ADSP-218x фирмы Analog Devices содержимое I2 инкрементируется;

(I2) + (M2) = (I2) + 5 → I2 (сравните с примером 17).

Пример выполнения команды приведен на рис. 5.20 (сравните с рис. 5.14, 5.17).

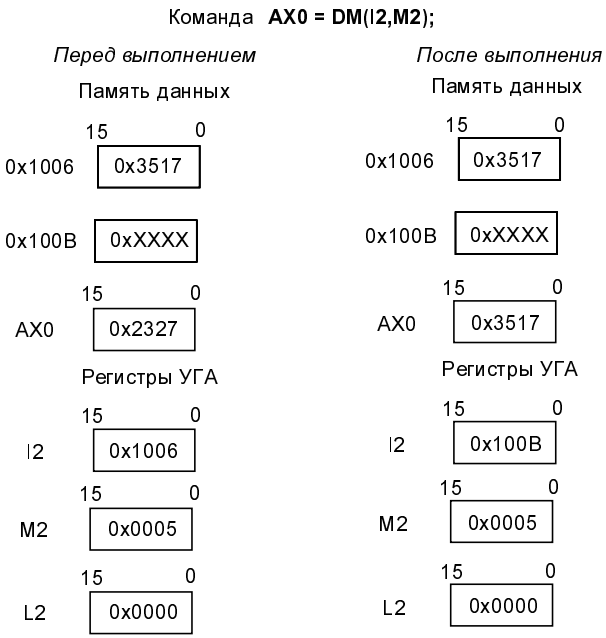


Рис. 5.20. Пример выполнения команды `AX0=DM(I2,M2);`

24. После выполнения команды сложения

```
ADD *AR2-0, A
```

в процессоре TMS320C54xx фирмы Texas Instruments содержимое AR2 декрементируется;

(AR2) − (AR0) = (AR2) − 3 → AR2 (сравните с примером 18).

Пример выполнения команды приведен на рис. 5.21 (сравните с рис. 5.15, 5.18).

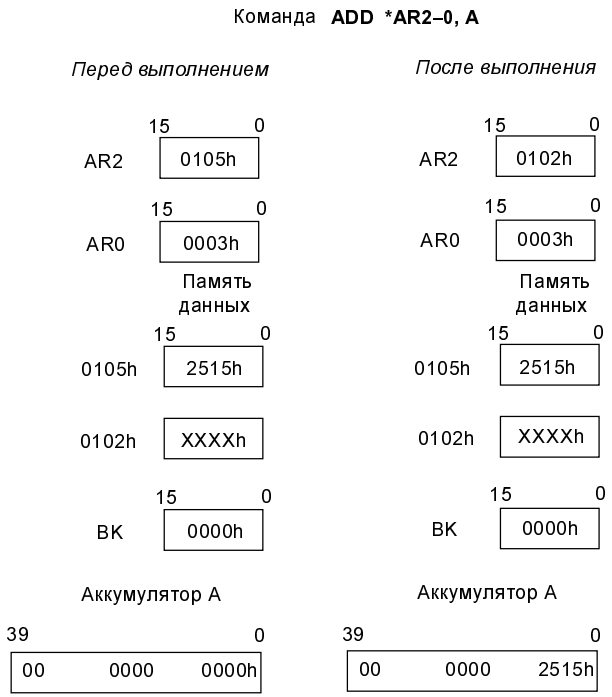


Рис. 5.21. Пример выполнения команды `ADD *AR2-0, A`

Индексация адреса

Индексация (указание) адреса означает автоматическое изменение исполняемого адреса без изменения содержимого регистра адреса; исполняемый адрес вычисляется как сумма содержимого регистра адреса и смещения *N*.

Синтаксис косвенного указания адреса при индексации приведен в табл. 5.4.

В процессорах DSP5600x фирмы Motorola при индексации исполняемый адрес вычисляется как (R*n*) + (N*n*). После выполнения команды содержимое регистров N*n* и R*n* не меняется.

В процессорах ADSP-218x фирмы Analog Devices индексация адреса должна быть организована программно; смещение *N* (где *N* — целое со знаком) записывается в регистр M*n*; исполняемый адрес, равный (I*i*) + (M*i*), сохраняется в новом регистре адреса I*j*, (I*i*) + (M*i*) → I*j* (*i*, *j* = 0, 1, 2, …, 7; $i \neq j$).

В процессорах TMS320C54xx фирмы Texas Instruments при индексации адреса смещение lk (где lk — целое со знаком) размещается в слове расширения команды (это исключение — слово расширения обычно не используется при

косвенной адресации). Исполняемый адрес вычисляется как (AR*n*) + lk. Содержимое AR*n* после выполнения команды не меняется.

Существуют модификации адреса, когда после его индексации выполняется постинкремент/декремент на величину смещения. Например, в процессорах TMS320C54xx фирмы Texas Instruments после индексации содержимое регистра адреса AR*n* может не меняться или инкрементироваться на 1 в зависимости от состояния 4-х битов модификации MOD в слове команды (табл. 5.4). В других процессорах подобная модификация адреса должна быть организована программно.

Таблица 5.4. *Индексация адреса*

Модификация	**Синтаксис косвенного указания адреса в процессорах фирмы**	
	Motorola DSP5600x	Texas Instruments TMS320C54xx
Индексация	(R*n* – N*n*)	*AR*n* (lk) (MOD = 1100)
Индексация с постинкрементом	Нет	*+AR*n* (lk) (MOD = 1101)

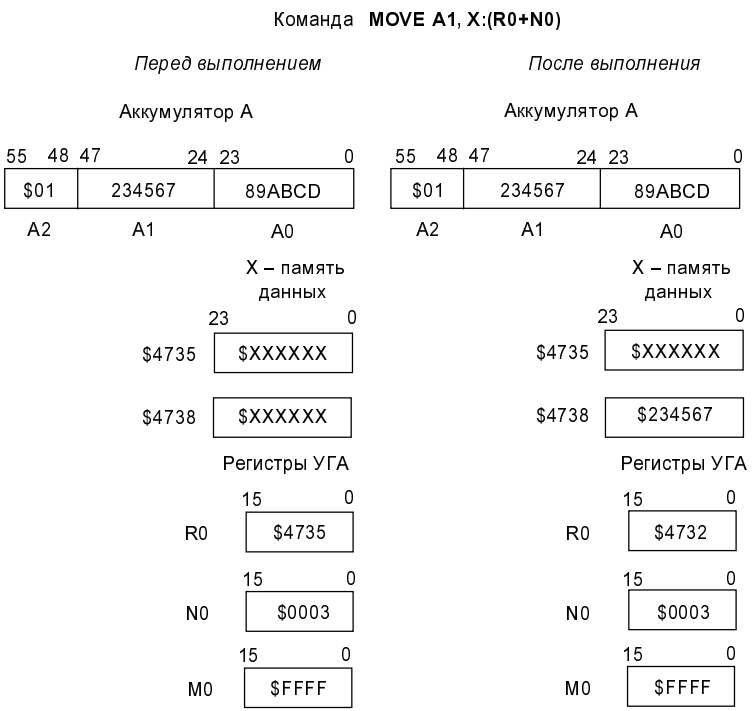


Рис. 5.22. Пример выполнения команды `MOVE A1, X:(R0+N0)`

Приведем примеры косвенной адресации с индексацией:

25. В команде пересылки

```
MOVE A1, X:(R0+N0)
```

процессора DSP5600x фирмы Motorola исполняемым адресом является (R0+N0); после завершения работы команды (R0) и (N0) не меняются (сравните с примером 16).

Пример выполнения команды приведен на рис. 5.22 (сравните с рис. 5.16, 5.19).

26. В команде сложения

```
ADD *AR2(2), A
```

процессора TMS320C54xx фирмы Texas Instruments исполняемым адресом является (AR2)+2; после завершения работы команды (AR2) не меняется (сравните с примером 18).

Пример выполнения команды приведен на рис. 5.23 (сравните с рис. 5.18, 5.21).

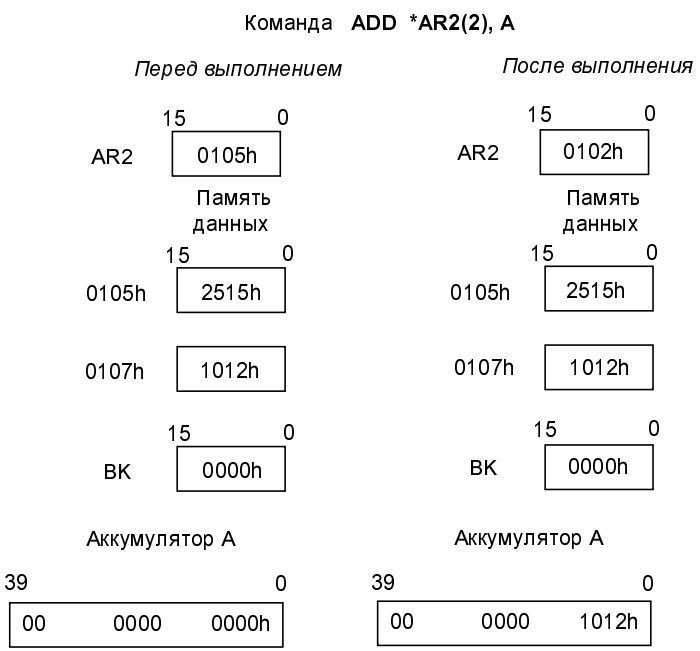


Рис. 5.23. Пример выполнения команды `ADD *AR2(2), A`

5.2.2. Типы арифметики

При вычислении модифицированного адреса может использоваться один из следующих типов арифметики:

- ❒ линейная;
- ❒ модульная;
- ❒ бит-реверсивная (с обратным переносом).

Тип арифметики определяется содержимым регистра типа арифметики. Символические имена и названия регистров типа арифметики приведены в табл. 5.1.

Рассмотрим подробнее каждый из типов арифметики.

Линейная арифметика

Линейная арифметика подразумевает обычную целочисленную арифметику. Все модификации адреса, рассмотренные выше, выполнялись с использованием линейной арифметики, что отображалось содержимым соответствующих регистров типа арифметики (табл. 5.5).

Таблица 5.5. *Линейная арифметика*

Условное название регистра	Содержимое регистра при линейной арифметике в процессоре фирмы		
	Motorola DSP5600x	Texas Instruments TMS320C54xx	Analog Devices ADSP-218x
Регистр типа арифметики	(M*n*) = $FFFF	(BK) = 0000h	(L*n*) = 0x0000

Модульная арифметика

Суть *модульной арифметики* состоит в следующем. Рассмотрим целое число A. Разделив его на другое целое положительное число M, получим остаток от деления a, которое называют значением A, вычисленным по модулю M и обозначают:

$$a = A \bmod M.$$

Рассмотрим два целых числа A и B. Разделив эти числа на третье целое положительное число M, получим остатки от деления a и b, которые называют значениями A и B, вычисленными по модулю M, и обозначают:

$$a = A \bmod M;$$

$$b = B \bmod M.$$

Числа A и B считаются равными (сравнимыми) между собой по mod M, если $a \equiv b$, где символ "≡", называемый сравнением, имеет тот же смысл, что и символ равенства в линейной арифметике.

Например, числа 373 и 693 по mod 20 равны (сравнимы) между собой

$$373 \equiv 693 \equiv 13 \bmod 20.$$

По mod 2 равны между собой все четные числа (0 mod 2) и нечетные числа (1 mod 2) и т. д.

5.2.3. Циклическая адресация

Многие алгоритмы ЦОС, в частности, вычисление разностных уравнений, сверток и т. п., представляют собой циклическую обработку блока данных фиксированной длины. Для хранения таких данных в памяти выделяется область заданного объема, которую называют *буфером*. Данные могут поступать в буфер извне или вычисляться в цикле. Организация последовательной записи данных в буфер или считывания из буфера представляет собой *циклический буфер* (*см. также главу 2).*

При обращении к ячейкам буфера исполняемые адреса вычисляются по правилам *модульной арифметики*, что позволяет автоматически выполнять операцию проверки достижения конца буфера и возврата в его начало, тем самым обеспечивая циркуляцию данных внутри буфера.

Адресацию операнда называют *циклической*, если исполняемый адрес вычисляется по правилам модульной арифметики.

Рассмотрим организацию циклического буфера. Укажем его размер — M ячеек памяти данных (рис. 5.24). Это значение (либо на 1 меньшее, $M - 1$, как в процессорах фирмы Motorola) записывается в регистр типа арифметики, который в процессорах различных фирм имеет собственные имена и обозначения (табл. 5.1).

Рассмотрим вычисление исполняемого адреса при циклической адресации, для чего введем понятие *относительного* адреса в циклическом буфере.

Будем называть *относительными* адреса ячеек буфера, не учитывающие его месторасположение в пространстве памяти данных (рис. 5.24). Все относительные адреса начинаются с 0 (*нижняя* относительная граница буфера) и заканчиваются ($M - 1$)-м (*верхняя* относительная граница буфера). Очевидно, что абсолютные адреса отличаются от соответствующих относительных адресов на некоторую целую положительную константу, равную абсолютной нижней границе буфера. Она называется *базой* буфера и обозначается Bs (Base).

Содержимое 16-разрядного регистра адреса (исполняемый адрес) при циклической адресации показано на рис. 5.25, *а*. Оно равно сумме двух 16-разрядных чисел: базы Bs (рис. 5.25, *б*) и относительного адреса (рис. 5.25, *в*), причем база Bs содержит k нулевых младших разрядов, а относительный адрес содержит ($16 - k$) нулевых старших разрядов.

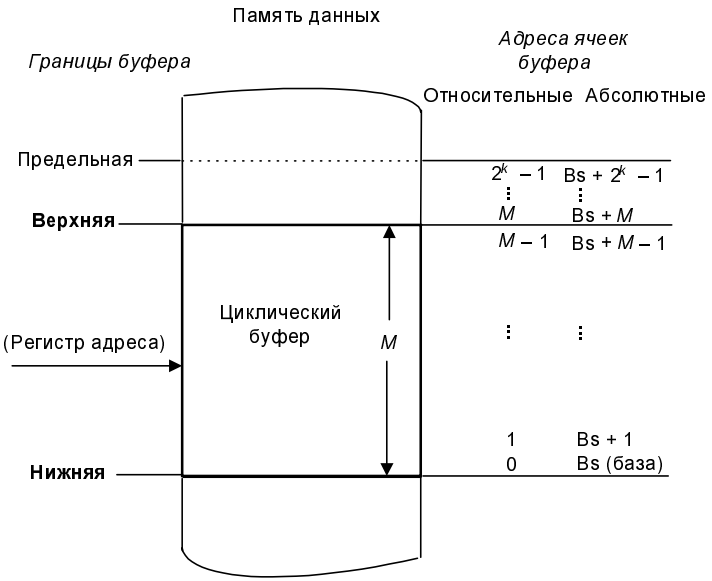


Рис. 5.24. Циклический буфер

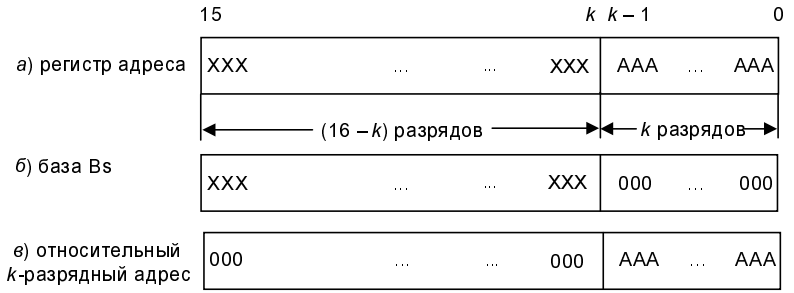


Рис. 5.25. Регистр адреса при циклической адресации

Количество k младших разрядов в регистре адреса, отводимое для хранения относительного адреса, определяется по заданному размеру M циклического буфера из условия

$$2^{k-1} < M \le 2^k.$$

Например:

- при $M = 5$, $k = 3$ ($2^2 < M < 2^3$);
- при $M = 32$, $k = 5$ ($2^4 < M = 2^5$);
- при $M = 90$, $k = 7$ ($2^6 < M < 2^7$)

и т. д.

Полученное значение k обеспечивает адресацию содержимого ячеек циклического буфера с относительными адресами с 0-го по $(2^k - 1)$-й, что соответствует абсолютным адресам с базового Bs по $(Bs + (2^k - 1))$-й. Адрес $(2^k - 1)$ называется *предельной* относительной границей буфера; абсолютная предельная граница равна $Bs + (2^k - 1)$. Пространство памяти буфера с относительными адресами с M-го по $(2^k - 1)$-й включительно свободно и может использоваться для других целей.

Значение базы Bs должно быть кратным 2^k

$$Bs = p \times 2^k,$$

где $p = 1, 2, \ldots$

Выбор p определяется свободной областью в памяти данных, где может размещаться циклический буфер.

Начальный исполняемый адрес может быть:

- равен нижней границе буфера — k младших разрядов содержат 0;
- равен верхней границе буфера — k младших разрядов содержат 1;
- произвольным в границах буфера — в k младших разрядах хранится соответствующий относительный исполняемый адрес.

Тогда следующий исполняемый адрес EA при постинкременте/постдекременте на N ($|N| < M$) согласно правилам модульной арифметики вычисляется по формуле

$$(EA - Bs + N) \bmod M + Bs,$$

если $(EA - Bs + N) \geq 0$

или по формуле

$$(EA - Bs + N) \bmod M + Bs + M,$$

если $(EA - Bs + N) < 0$.

Содержимое триплета: регистров адреса, смещения и типа арифметики при организации циклического буфера размером M приведено в табл. 5.6. Для процессоров фирм Motorola и Analog Devices при циклической адресации номер n в триплете должен быть одинаковым, например, M0, R0, N0 и т. п.

Таблица 5.6. *Содержимое регистров при циклической адресации*

Условное название регистра	Содержимое регистров при модульной арифметике в процессоре фирмы		
	Motorola DSP5600x	Texas Instruments TMS320C54xx	Analog Devices ADSP-218x
Регистр типа арифметики	(Mn) = M – 1	(Ln) = M	(BK) = M
Регистр адреса	(Rn) = абсолютный начальный адрес	(In) = абсолютный начальный адрес	(ARn) = абсолютный начальный адрес

Таблица 5.6 *(окончание)*

Условное название регистра	Содержимое регистров при модульной арифметике в процессоре фирмы		
	Motorola DSP5600x	Texas Instruments TMS320C54xx	Analog Devices ADSP-218x
Регистр смещения	(N*n*) = постинкремент/постдекремент	(M*n*) = постинкремент/постдекремент	(AR0) = постинкремент/постдекремент

Приведем примеры циклической адресации в процессорах DSP5600x фирм Motorola и ADSP-218x фирмы Analog Devices. Все адреса указаны в десятичной системе.

27. В команде пересылки из `X`-памяти данных в регистр аккумулятора `A1`

```
MOVE A1, X:(R0)+N0
```

процессора DSP5600x фирмы Motorola после выполнения команды следующий исполняемый адрес определяется как

((R0) − Bs + (N0)) mod M + Bs > R0 (сравните с примером 16).

Пример организации циклического буфера приведен на рис. 5.26.

Размер циклического буфера M = 90; значение (M − 1) хранится в регистре M0. Количество k младших разрядов, определенное из условия

$$2^{k-1} < M \le 2^k \ (2^6 < 90 < 2^7),$$

равно 7. База Bs (нижняя граница буфера), кратная 2^7, выбрана равной 128. Верхняя граница равна 217 (Bs + M − 1), а предельная граница — 255 (Bs + (2^7 − 1)). Ячейки с 218-й по 255-ю свободны и могут быть использованы для других целей. Начальный 16-разрядный исполняемый адрес равен 165 и хранится в регистре адреса R0. Относительный исполняемый 7-разрядный начальный адрес равен $37 = 0100101_{(2)}$. После первого выполнения команды следующий исполняемый адрес вычисляется как

((R0) − Bs + (N0)) mod 90 + Bs = (165 − 128 + 15) mod 90 + 128 = 180.

После второго и третьего выполнения команды исполняемые адреса последовательно становятся равными

(180 − 128 + 15) mod 90 + 128 = 195;

(195 − 128 + 15) mod 90 + 128 = 210.

Очередной инкремент при линейной арифметике привел бы к исполняемому адресу 225, который выходит за верхнюю границу буфера, однако модульная арифметика заставляет содержимое R0 оставаться внутри буфера, т. е. в действительности исполняемый адрес становится равным

(210 − 128 + 15) mod 90 + 128 = 135

и т. д.

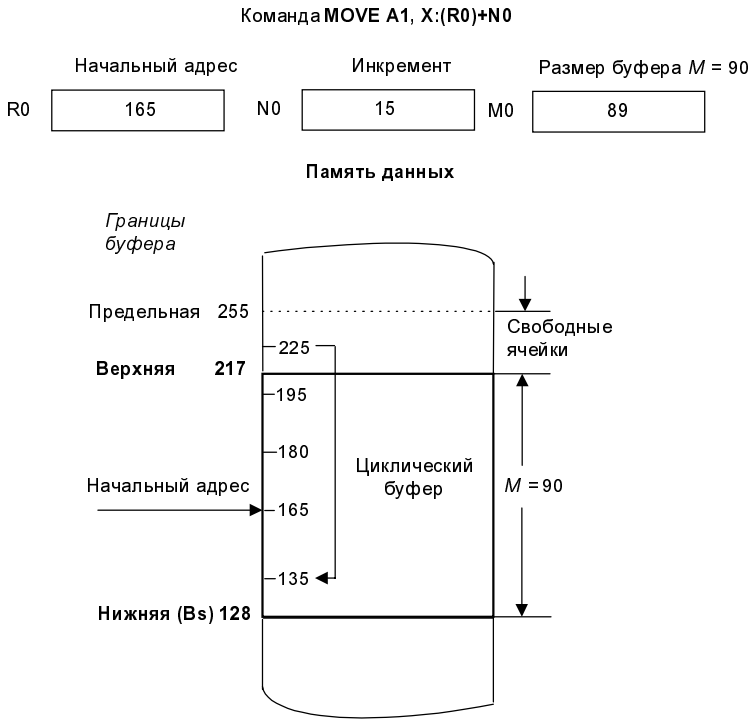


Рис. 5.26. Пример циклической адресации при выполнении команды `MOVE A1, X:(R0)+N0`

28. В команде пересылки из памяти данных во входной регистр `AX0`

```
AX0=DM(I0,M0);
```

процессора ADSP218x фирмы Analog Devices после выполнения команды следующий исполняемый адрес определяется как

$$((I0) - Bs + (M0)) \bmod M + Bs > I0,$$

если $((I0) - Bs + (M0)) \geq 0$,

или как

$$((I0) - Bs + (M0)) \bmod M + Bs + M > I0,$$

если $((I0) - Bs + (M0)) < 0$ (сравните с примером 17).

Пример организации циклического буфера приведен на рис. 5.27.

Размер циклического буфера $M = 3$ хранится в регистре L0. Количество k младших разрядов, определенное из условия

$$2^{k-1} < M \leq 2^k \ (2^1 < 3 < 2^2),$$

равно 2. База Bs (нижняя граница буфера), кратная 2^2, выбрана равной 4. Верхняя граница равна 6 (Bs + M − 1), а предельная граница 7 (Bs +

+ (2^2 − 1)). Ячейка 7 свободна и может быть использована для других целей. Исполняемый 16-разрядный начальный адрес равен 5 и хранится в регистре I0, (I0) = 5. Исполняемый относительный 2-разрядный начальный адрес равен $1 = 01_{(2)}$. Постдекремент равен −1 и хранится в регистре M0, (M0) = −1. После первого выполнения команды следующий исполняемый адрес вычисляется как

$$((I0) - Bs + (M0)) \bmod 3 + Bs = (5 - 4 - 1) \bmod 3 + 4 = 0 + 4 = 4.$$

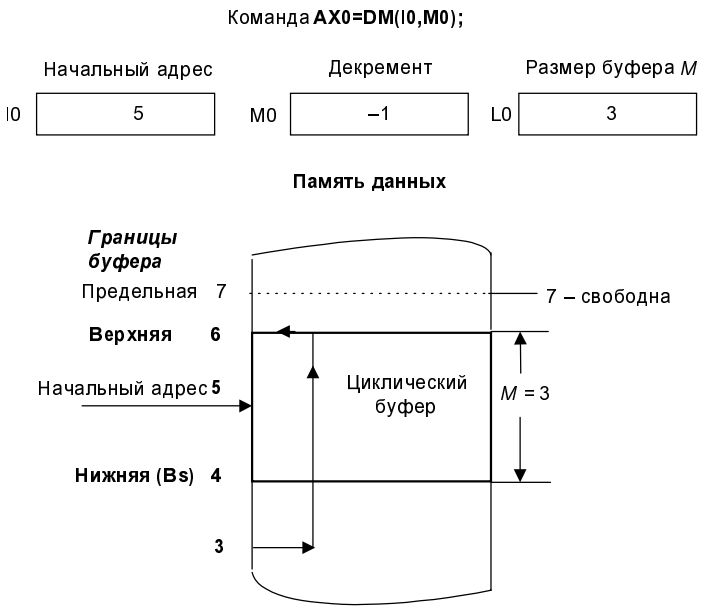


Рис. 5.27. Пример циклической адресации при выполнении команды `AX0=DM(I0,M0);`

Следующий декремент при линейной арифметике привел бы к исполняемому адресу 3, который выходит за нижнюю границу буфера (базу), однако модульная арифметика заставляет содержимое I0 перейти на верхнюю границу и исполняемый адрес становится равным

$$(4 - 4 - 1) \bmod 3 + 4 + 3 = -1 + 4 + 3 = 6.$$

Следующий исполняемый адрес определяется как

$$(6 - 4 - 1) \bmod 3 + 4 = 1 + 4 = 5.$$

Затем адреса циклически повторяются: 4, 6, 5, 4, 6, 5 и т. д.

Как видно из этих примеров, в процессорах фирм Motorola и Analog Devices указание операндов в команде при циклической адресации такое же, как

при постинкременте/постдекременте адреса. Признаком циклической адресации является соответствующее содержимое регистра типа арифметики.

Синтаксис косвенного указания адреса при циклической адресации в процессорах фирмы Texas Instruments приведен в табл. 5.7.

Таблица 5.7. *Циклическая адресация в процессорах TMS320C54xx фирмы TI*

Циклическая адресация	Синтаксис косвенного указания адреса
С постдекрементом на 1	*AR*n*–%
С постдекрементом на *N*	*AR*n*–0%
С постинкрементом на 1	*AR*n*+%
С постинкрементом на *N*	*AR*n*+0%

Приведем пример циклической адресации в процессорах фирмы Texas Instruments. Все адреса указаны в десятичной системе.

29. В команде сложения

```
ADD *AR2+%, A
```

процессора TMS320C54xx после выполнения команды следующий исполняемый адрес определяется как

((AR2) − Bs + (AR0)) mod M + Bs > AR2 (сравните с примером 18).

Пример организации циклического буфера приведен на рис. 5.28.

Размер циклического буфера $M = 10$; это значение хранится в регистре BK. Количество k младших разрядов, определенное из условия

$$2^{k-1} < M \le 2^k \ (2^3 < 10 < 2^4),$$

равно 4. База Bs (нижняя граница буфера), кратная 2^4, выбрана равной 64. Верхняя граница буфера равна 73 (Bs + M − 1), а предельная граница — 79 (Bs + (2^4 − 1)). Ячейки с 74-ю по 79-ю свободны и могут быть использованы для других целей. Исполняемый 16-разрядный начальный адрес равен 64 и хранится в регистре AR2. Исполняемый относительный 4-разрядный начальный адрес равен $0 = 0000_{(2)}$. После первого выполнения команды следующий исполняемый адрес вычисляется как

((AR2) − Bs + 1) mod 10 + Bs = (64 − 64 + 1) mod 10 + 64 = 65.

Дальнейшие вычисления дадут последовательно исполняемые адреса 66, 67, …, 73. Следующий исполняемый адрес

(73 − 64 + 1) mod 10 + 64 = 64

обеспечит возврат на нижнюю границу циклического буфера.

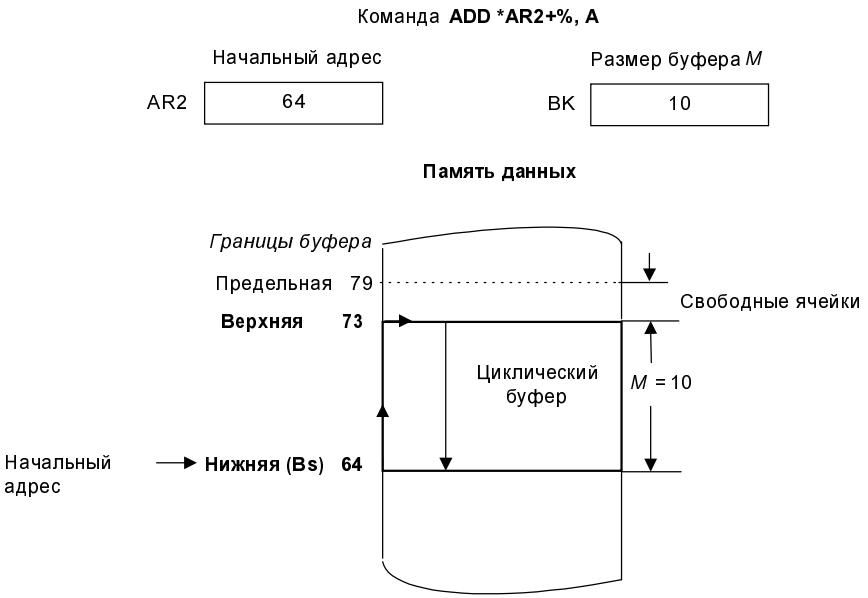


Рис. 5.28. Пример циклической адресации при выполнении команды `ADD *AR2+%, A`

5.2.4. Бит-реверсивная адресация

Адресацию операнда называют *бит-реверсивной*, если исполняемый адрес, вычисляется по правилам *бит-реверсивной арифметики.*

Рассмотрим особенности бит-реверсивной арифметики.

Бит-реверсивная арифметика

Бит-реверсивная арифметика (арифметика с обратным переносом) используется только при обработке положительных целых двоичных чисел. В результате выполнения операции бит-реверсии биты двоичного числа оказываются переставленными в обратном порядке. Например, двоичное число $0111 \rightarrow 7_{(10)}$ после операции бит-реверсии станет равным $1110 \rightarrow 14_{(10)}$.

Бит-реверсивная арифметика применяется в алгоритмах БПФ по основанию 2, когда количество исходных отсчетов L равно 2^k, $L = 2^k$. В частности, при реализации алгоритмов БПФ с прореживанием по времени последовательность исходных отсчетов должна быть расставлена в бит-реверсивном порядке. Пример такой расстановки для 8-точечного БПФ ($L = 8$) приведен в табл. 5.8.

Таблица 5.8. *Бит-реверсия исходных отсчетов в 8-точечном БПФ*

Порядок следования *n* исходных отсчетов *x* (*n*)			
Прямой		Бит-реверсивный	
Десятичная система	Двоичная система	Двоичная система	Десятичная система
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Для сокращения времени при вычислении БПФ в большинстве современных сигнальных процессоров предусмотрена аппаратная реализация операции бит-реверсии.

Генерация бит-реверсивных последовательностей в процессорах различных фирм может осуществляться по-разному. Например, в процессорах TMS320C64x фирмы Texas Instruments бит-реверсия выполняется по специальной команде `BITR`. В процессорах ADSP-218x фирмы Analog Devices предусмотрен режим бит-реверсии, задаваемый установкой специального бита в регистре режима. В этом режиме на шину адреса выставляется исполняемый адрес, в котором порядок следования битов изменен на обратный (выполнена операция бит-реверсии), при этом содержимое регистра адреса не меняется.

В процессорах фирмы Motorola, а также в процессорах TMS320C5xxx/62xx/67xx фирмы Texas Instruments для выполнения операции бит-реверсии используют бит-реверсивную адресацию.

Бит-реверсивная адресация

Признаком бит-реверсивной арифметики является:

- в процессорах DSP5600x фирмы Motorola — запись кода бит-реверсивной арифметики в регистре типа арифметики (см. табл. 5.1), (M*n*) = $0000;
- в процессорах TMS320C54xx фирмы Texas Instruments — установка 4-х битов модификации MOD в слове операции: состояние MOD = 0100 соответствует бит-реверсивной арифметике с постдекрементом адреса, а MOD = 0111 — с постинкрементом адреса.

Исполняемый адрес при бит-реверсивной адресации в процессорах фирмы Motorola указывается так же, как при постинкременте/постдекременте.

Синтаксис указания исполняемого адреса при бит-реверсивной адресации в процессорах фирмы Texas Instruments приведен в табл. 5.9.

Таблица 5.9. *Бит-реверсивная адресация в процессорах TMS320C54xx фирмы TI*

Бит-реверсивная адресация	Синтаксис указания исполняемого адреса
С постдекрементом на *N*	*AR*n*–0B (MOD=0100)
С постинкрементом на *N*	*AR*n*+0B (MOD=0111)

Рассмотрим процедуру расстановки последовательности исходных отсчетов в бит-реверсивном порядке при L-точечном БПФ ($L = 2^k$). Эта процедура одинакова для процессоров DSP5600x фирмы Motorola и TMS320C54xx фирмы Texas Instruments.

Действия пользователя должны быть следующими.

- ❒ Установить признак бит-реверсивной арифметики.
- ❒ Записать значение постинкремента $N = 2^{(k-1)}$ в регистр смещения (табл. 5.1).
- ❒ Сформировать буфер для последовательности L исходных отсчетов в прямом порядке: нижняя граница буфера (база Bs) выбирается кратной 2^k, верхняя граница равна Bs + (2^k − 1).
- ❒ Записать *первый* (начальный) исполняемый адрес (нижнюю границу Bs) в регистр адреса.

  В регистре адреса k младших битов хранят относительный исполняемый адрес в буфере. Если адресация буфера начинается с нижней границы (относительный исполняемый адрес равен 0), то этот адрес указывает на первый отсчет последовательности, как при прямом, так и при бит-реверсивном порядке отсчетов.
- ❒ Использовать косвенную адресацию ячеек буфера с постинкрементом на N.

  При косвенной адресации с постинкрементом на N и *бит-реверсивной арифметикой* в процессоре выполняются следующие действия:

  - бит-реверсия k младших битов в регистре адреса;
  - инкремент на 1 содержимого регистра адреса;
  - повторная бит-реверсия k младших битов в регистре адреса.
- ❒ Вычислить (L − 1) следующих исполняемых адресов (второй, третий, …, (L − 1)-й) с постинкрементом на N и бит-реверсивной арифметикой.

Приведем пример реализации описанной процедуры в процессоре DSP5600x фирмы Motorola для 8-точечного БПФ ($L = 2^k = 2^3 = 8$).

В качестве регистров адреса, смещения и типа арифметики выбран триплет R0, N0 и M0 (табл. 3.1).

Действия пользователя:

- (M0) = $0000;
- $N = 2^{(3-1)} = 2^2 = 4$;

  (N0) = 0000000000000100 = $4_{(10)}$.
- Нижняя граница буфера Bs выбрана равной $25 \times 2^3 = 200$.

  Верхняя граница — $200 + 2^3 - 1 = 207$.
- Начальный исполняемый адрес хранится в R0 и равен

  (R0) = 0000000011001**000** = $200_{(10)}$.

  Жирным шрифтом выделены младшие три реверсируемых бита.
- Вычисление исполняемого адреса с постинкрементом на 4 и бит-реверсивной арифметикой повторяется 7 раз:

  Вычисление *второго исполняемого адреса*: (R0) + N0:

  - бит-реверсия 3-х младших битов в регистре R0:

    (R0) = 0000000011001**000**;
  - инкремент на 1 содержимого R0:

    (R0) = 0000000011001**001**;
  - повторная бит-реверсия *k* младших битов в регистре R0:

    (R0) = 0000000011001**100** = $204_{(10)}$.

  Вычисление *третьего исполняемого адреса*: (R0) + N0:

  - (R0) = 0000000011001**001**;
  - (R0) = 0000000011001**010**;
  - (R0) = 0000000011001**010** = $202_{(10)}$.

  Аналогично вычисление исполняемого адреса производится еще пять раз, в результате чего исполняемые адреса (содержимое регистра R0) последовательно становятся равными:

  Четвертый: (R0) = 0000000011001**110** = $206_{(10)}$.

  Пятый: (R0) = 0000000011001**001** = $201_{(10)}$.

  Шестой: (R0) = 0000000011001**101** = $205_{(10)}$.

  Седьмой: (R0) = 0000000011001**010** = $203_{(10)}$.

  Восьмой: (R0) = 0000000011001**010** = $207_{(10)}$.

  Полученные в младших 3-х битах значения сравните с представленными в табл. 5.8.

В процессорах TMS320C54xx фирмы Texas Instruments исполняемые адреса при бит-реверсивной адресации вычисляются аналогично; в качестве регистра адреса используется AR*n*, а регистра смещения, хранящего величину постинкремента *N*, — регистр AR0 (см. табл. 3.1).

5.2.5. Косвенная адресация переходов

Косвенная адресация переходов предполагает косвенное указание адреса перехода именем регистра адреса. Она используется в процессорах фирм Motorola и Analog Devices. Все разновидности модификаций адреса и все типы арифметики могут использоваться при вычислении адресов перехода точно так же, как при вычислении адресов операндов.

Приведем пример косвенной адресации перехода.

30. Команда безусловного перехода

```
JMP (R1+N1)
```

    в процессоре фирмы Motorola содержит адрес перехода, указанный косвенно и равный сумме содержимого регистров `R1` и `N1`; при косвенном указании адреса перехода использована индексация адреса.

5.3. Непосредственная адресация

При *непосредственной адресации* операнд указывается в команде константой.

При двусловном формате короткие константы хранятся в слове команды, длинные — в слове расширения. При однословном формате и короткие и длинные константы хранятся в слове команды.

Непосредственная адресация может использоваться только для указания *исходных данных*. Признаком непосредственной адресации в командах процессоров фирм Motorola и процессоров TMS320C2xxx/5xxx фирмы Texas Instruments служит префикс `#` перед источником-константой. В процессорах фирмы Analog Devices и процессорах TMS320C3x префикс отсутствует.

Рассмотрим примеры непосредственной адресации константы.

31. Команда пересылки

```
MOVE #$123456, A0
```

    процессора DSP5600x содержит операнд, указанный непосредственно 24-разрядной длинной константой `#$123456`, которая размещается в слове расширения.

    По команде происходит запись константы в 24-разрядный регистр A0 аккумулятора (младшее слово).

32. Команда пересылки

```
AX0 = 0x7FFF
```

процессора ADSP21xx содержит операнд, указанный непосредственно 16-разрядной длинной константой `0x7FFF`, которая размещается в слове команды.

По команде происходит запись константы в 16-разрядный входной регистр AX0.

33. Команда сложения

```
ADD #1A
```

процессора TMS320C2xxx содержит операнд, указанный непосредственно 8-разрядной короткой константой `#1Ah`, которая размещается в слове команды и трактуется как беззнаковое целое.

По команде происходит сложение константы с содержимым аккумулятора, результат сохраняется в аккумуляторе.

34. Команда пересылки

```
MOVE #$1F, A1
```

процессора DSP5600x содержит операнд, указанный непосредственно 8-разрядной короткой константой `#$1F`, которая размещается в слове команды и при пересылке в старшее слово аккумулятора A1 трактуется как беззнаковое целое.

При выполнении команды происходит запись константы в 24- разрядный регистр A1 аккумулятора (старшее слово).

35. Команда пересылки

```
MOVE #$1F, Y1
```

процессора DSP5600x содержит операнд, указанный так же, как в примере 32, однако, при пересылке во входной регистр Y1 8-разрядная короткая константа трактуется как дробное число со знаком.

По команде происходит запись константы в 24-разрядный входной регистр Y1.

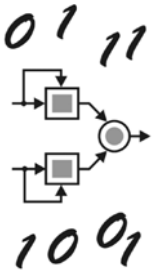
36. Команда сложения данных целого типа

```
ADDI 8000h, R7
```

процессора TMS320C3x содержит операнд, указанный непосредственно 16-разрядной длинной константой, которая хранится в слове команды.

При выполнении команды происходит сложение 16-разрядной константы (целого числа 8000h) с содержимым 32-разрядного регистра R7; слагаемые выравниваются по правому краю, результат сохраняется в регистре R7.

Глава 6



Система команд

Программа, составленная на языке ассемблера ЦПОС, представляет собой символическое описание последовательности действий, которые необходимо выполнить для получения результата по заданному алгоритму. Требуемые действия в зависимости от их характера могут указываться директивами, командами или макрокомандами. Директивы не транслируются в машинные команды, они служат указаниями ассемблеру и компоновщику, в частности, указаниями на размещение в процессоре команд и данных *(см. главу 9)*. Макрокоманды при трансляции автоматически заменяются соответствующими последовательностями команд.

Команды определяют следующие действия процессора:

- выполнение операций с данными;
- управление последовательностью выполнения операций, включая разветвления, циклы, обращение к подпрограммам и т. п.;
- общее управление работой процессора, в том числе его инициализацию, переход в различные состояния (прерывания, ожидания и т. д.), к различным режимам обработки данных (с удвоенной точностью, с автоматическим масштабированием, и т. д.)

и т. п.

Совокупность команд, в целом обеспечивающая выполнение указанных функций, образует *систему команд* процессора.

Процессоры различных фирм или различных семейств одной фирмы, отличающиеся концепциями, на основе которых разработана их архитектура, имеют различные системы команд. В этой главе рассматриваются наиболее общие, единые для всех ЦПОС принципы организации отдельных команд и формирования из них системы команд, а также основные особенности реализации команд в конкретных процессорах фирм Texas Instruments, Motorola и Analog Devices.

Организация команд в различных процессорах, определяемая их архитектурой, в частности, зависит от используемого *формата команд*, который в свою очередь влияет на *структуру слова команды* и связанный с ней *синтаксис команды*. Структура слова команды включает обязательные компоненты, обусловленные функциональным назначением команды, и потому общие для однотипных команд в различных процессорах, а также дополнительные компоненты, набор

которых (количество и способы указания в команде) характеризует индивидуальные особенности этих команд в различных процессорах.

Формирование системы команд предполагает наличие в различных процессорах типовых групп команд, объединенных соответственно выполняемым действиям, и в целом ориентированных на реализацию систем ЦОС общего или специального назначения.

Рассмотрим сформулированные понятия подробнее. Общие положения будем иллюстрировать примерами конкретных команд процессоров фирм Texas Instruments, Motorola и Analog Devices.

6.1. Форматы команд

Понятие формата команды введено *в главе 5*. Табл. 6.1 позволяет сравнить форматы команд (количество и длину слов) в семействах процессоров фирм Texas Instruments, Motorola и Analog Devices. Напомним, что при двусловном формате (см. рис. 5.1) второе слово команды (слово расширения) предназначено только для указания операнда — длинной константы, длинного исполняемого адреса или длинного адреса перехода, в противном случае команда является однословной.

Трехсловный формат команды в процессорах MSC810x фирмы Motorola можно отнести к исключению. При относительно небольшой длине слова команды (16 битов) иногда необходимы два слова расширения: для хранения длинной константы и длинного исполняемого адреса.

Два последних столбца табл. 6.1 указывают на возможность одновременного выполнения группы команд (об этом *см. разд. 6.2.2, 6.3.2*), соответствующие процессоры, отмечены символом *.

Таблица 6.1. *Форматы команд*

Фирма	Процессор	ФТ или ПТ	Формат команды		Длина пакета выборки	Количество команд в пакете выборки
			Длина слова в битах	Кол-во слов		
Texas Instruments TMS320	C24xx/28xx	ФТ	16	2	—	—
	C54xx	ФТ	16	2	—	—
	C55xx	ФТ	8—48	6—1	—	—
	C62xx*	ФТ	32	1	256	8
	C64xx*	ФТ	32	1	256	8

Таблица 6.1 *(окончание)*

Фирма	Процессор	ФТ или ПТ	Формат команды		Длина пакета выборки	Количество команд в пакете выборки
			Длина слова в битах	Кол-во слов		
	C67xx*	ПТ и ФТ	32	1	256	8
	C3x	ПТ	32	1	—	—
Motorola DSP	5600x/563xx/ 566xx	ФТ	24	2	—	—
	568xx	ФТ	16	2	—	—
	9600x	ПТ	32	2	—	—
	MSC810x*	ФТ	16	3	128	8
Analog Devices	218x/219x	ФТ	24	1	—	—
ADSP-	210xx/211xx	ПТ и ФТ	48	1	—	—

Длина слова команды в процессорах TMS320C55xx фирмы Texas Instruments (табл. 6.1) различна и может составлять от 1 до 6 байтов (8—48 битов). Поэтому каждая ячейка памяти программ, имеющая фиксированную длину 32 бита, логически разделена на 4 адресуемых байта. Конкретные команды, имеющие различные, но кратные байту, длины слова команды, записываются последовательно, побайтно в 32-разрядные ячейки и аналогично считываются. Различная длина слова команды позволяет существенно увеличить плотность программного кода (то есть размещать большее количество команд в пределах одинакового объема памяти программ).

Количество слов в формате команды влияет на время ее выполнения. Одно слово команды выполняется за один командный цикл. При двусловном и трехсловном форматах время выполнения команды зависит от типа адресации операндов, т. к. второе и третье слово используется только при непосредственной адресации длинных констант или прямой адресации ячеек памяти с длинными адресами.

6.2. Структура слова команды

Слово команды условно разделяют на *поля* (части), отводимые для хранения определенной информации о команде. Набор основных полей, из которых составлены слова всех команд в различных процессорах, ограничен. Совокупность таких полей образует *обобщенную структуру* слова команды. Слова конкретных команд представляют собой комбинацию определенных полей.

Структура слова команды зависит от длины слова (формата), однако определяющим фактором является архитектура процессора. Чтобы не усложнять обобщенную структуру "слово" команды, включая в нее все разнообразие полей, выделим для отдельного рассмотрения процессоры:

- со стандартной архитектурой *(см. главу 2)*;
- с улучшенной стандартной архитектурой и архитектурой VLIW *(см. главу 2)*; такие процессоры условно назовем *процессорами с одновременным выполнением группы команд*; особенности организации команд в подобных процессорах будем оговаривать особо, в противном случае, по умолчанию станем подразумевать процессоры со стандартной архитектурой.

6.2.1. Структура слова команды в процессорах со стандартной архитектурой

Слово команды в процессорах со стандартной архитектурой содержит следующие основные поля (рис. 6.1):

- условия;
- операции;
- операндов;
- параллельных пересылок.

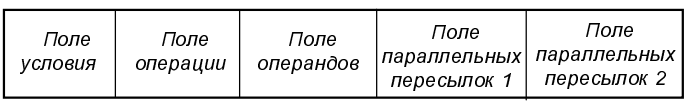


Рис. 6.1. Обобщенная структура слова команды

Количество, последовательность и длина полей зависят от архитектуры конкретного процессора и функционального назначения команды. Рассмотрим перечисленные поля.

Поле условия содержит код условия выполнения операции, указанной в поле операции.

Поле условия может содержаться в словах команд:

- управления;
- операций над данными.

Слово команды *управления* (перехода, вызова подпрограммы, цикла) имеет поле условия, если выполняется операция перехода (разветвления) к различным точкам программы по условию, *указываемому пользователем.*

Поле условия в слове команды *управления* отсутствует, если условие не зависит от пользователя, например, условием является равенство нулю содержимого конкретного регистра.

Слово команды *управления* имеет поле условия, если сама команда управления (как правило, безусловного перехода) выполняется в зависимости от *условия, указываемого пользователем.*

В различных процессорах реализованы различные варианты команд управления.

Команды *операций над данными* также могут выполняться в зависимости от условия, задаваемого пользователем. Существует два способа реализации условного выполнения таких команд:

- *программный* с использованием команды условного перехода; в этом случае слово самой команды операции над данными не имеет поля условия;
- *аппаратный*, когда условие выполнения операции указывается непосредственно в команде операции над данными, для чего в слове этой команды предусмотрено поле условия.

Сведения о наличии поля условия в командах операций над данными приводятся в табл. 6.2.

Таблица 6.2. *Поле условия выполнения операции над данными*

Фирма	Процессор	Поле условия
Texas Instruments	TMS320C2xxx	Отсутствует
	TMS320C5xxx	Имеется в слове команды пересылки по условию
	TMS320C3x	Имеется в слове команды пересылки по условию
Motorola	DSP56xxx/96xxx	Имеется в слове команды пересылки по условию
Analog Devices	ADSP-21xx/21xxx	Имеется в словах большинства команд

Как частный случай условного выполнения операций над данными можно рассматривать некоторые команды процессора TMS320C55xx фирмы Texas Instruments с внутренним разветвлением, когда выполняется одна из двух арифметических операций в зависимости от некоторого условия. Однако условие во всех подобных командах не выбирается пользователем, а определяется состоянием конкретного бита, поэтому в словах таких команд поле условия отсутствует. Например, по команде `ADDSUBCC` может выполняться операция сложения или вычитания в зависимости от состояния бита TCx и т. п.

Поле операции содержит код операции (КОП) и является обязательным для слова любой команды.

Поле операндов содержит:

- *указания на операнды* в командах операций над данными;
- адреса *переходов* в командах управления.

Кроме того, поле операндов может содержать дополнительную информацию об операции. Например, в процессорах TMS320C2xxx/5xxx фирмы Texas Instruments поле операндов может содержать информацию о сдвиге (количестве битов и направлении сдвига) одного из операндов, производимом до операции. В процессорах фирм Motorola и Analog Devices сдвиг выполняется по специальным командам, поэтому в поле операндов информация о сдвиге отсутствует.

Поле операндов является обязательным для команд операций над данными и команд управления последовательностью выполнения операций. Количество операндов определяется архитектурой конкретного процессора и спецификой команды. В большинстве команд общего управления работой процессора поле операндов отсутствует; примерами команд без операндов являются: `NOP` — пустая операция, `RESET` — программный сброс, `SWI` — программное прерывание и т. д.

Основные способы указания операндов и адресов переходов в командах рассмотрены *в главе 5*.

Поля параллельных пересылок содержат *указания* на операнды пересылок.

Пересылки, выполняемые одновременно с основной операцией команды и указываемые в этой команде, называют *параллельными*. В различных процессорах поддерживается от одной до двух операций параллельных пересылок. Возможность и типы параллельных пересылок, а также правила их организации определяются архитектурой конкретного процессора и спецификой команды. В частности, параллельные пересылки недопустимы в командах управления. Краткие сведения о возможных параллельных пересылках (безотносительно команд, в которых они допустимы) содержатся в табл. 6.3, где использованы следующие условные обозначения:

- EA1, EA2 — исполняемые адреса для первого и второго полей пересылок;
- EA — исполняемый адрес для одного поля пересылок;
- EAP — исполняемый адрес в памяти программ (в процессорах фирмы Analog Devices);
- EAX — исполняемый адрес в X-памяти данных (в процессорах фирмы Motorola);
- EAY — исполняемый адрес в Y-памяти данных (в процессорах фирмы Motorola);
- EAL — исполняемый адрес в L-памяти данных; ячейка L-памяти образуется из объединения двух ячеек X- и Y-памяти с одинаковыми адресами (в процессорах фирмы Motorola);
- S1, S2 — источники для первого и второго поля пересылок;
- D1, D2 — приемники для первого и второго поля пересылок;

- S, D — источник и приемник для одного поля пересылок;
- #C — константа.

Таблица 6.3. *Поля параллельных пересылок*

Фирма	Процессор	Кол-во полей пересылок	Допустимые типы пересылок	
			Первое поле пересылок	Второе поле пересылок
Texas Instruments	TMS320C2xxx/3x/5xxx	Нет	Нет	Нет
Motorola	DSP56xxx/96xxx	До 2-х	#C → D	
			S → D	
			S → EAX	
			EAX → D	
				S → EAY
				EAY → D
			S1 → EAX	S2 → D2
			EAX → D1	S2 → D2
			S1 → D1	S2 → EAY
			S1 → D1	EAY → D2
			S → EAL	
			EAL → D	
			EAX → D1	EAY → D2
			EAX → D1	S2 → EAY
			S1 → EAX	EAY → D2
			S1 → EAX	S2 → EAY
Analog Devices	ADSP-21xx/21xxx	До 2-х	EA → D	
			EAP → D	
			EA → D1	EAP → D2
			S → EA	
			S → D	

Выполнение операций параллельных пересылок в каждом процессоре имеет специфику, однако можно сформулировать следующие общие правила:

- дублирование *источников* в поле операндов и в поле пересылок разрешается, а дублирование *приемников* — запрещается;
- если *источником* в поле пересылок является *приемник* в поле операндов, то соответствующая пересылка выполняется *до* основной операции, заданной кодом операции (КОП);
- если *приемником* в поле пересылок является *источник* в поле операндов, то соответствующая пересылка выполняется *после* основной операции.

Параллельные пересылки позволяют одновременно с основной операцией сохранить результат, полученный при выполнении предыдущей команды, и загрузить исходные данные, необходимые для выполнения следующей команды.

Отметим, что слово команды, за исключением специальных команд пересылки по условию, *не может содержать одновременно и поле условия и поля параллельных пересылок*.

Параллельные пересылки не следует путать с комбинированными командами, имеющимися, например, в системе команд процессоров TMS320C5xxx фирмы Texas Instruments; при выполнении этих команд одновременно выполняются две операции, одной из которых может быть операция пересылки *(см. разд. 6.5.4)*.

6.2.2. Структура слова команды в процессорах с одновременным выполнением группы команд

Возможность одновременного выполнения группы команд обеспечивается за счет специальной архитектуры процессоров, при которой различные команды выполняются независимо функционирующими устройствами. Это позволило перейти к качественно новому, значительно более высокому, уровню производительности. Архитектуры процессоров TMS320C6xxx фирмы Texas Instruments и MSC810x фирмы Motorola (см. табл. 6.1) построены по этому принципу, однако существенно различны. В рамках рассматриваемых вопросов эти отличия относятся к форматам команд, структуре слов команд, способу объединения команд в группы для одновременного выполнения, синтаксису команд и т. д. В качестве примера рассмотрим структуру слова команд в процессорах TMS320C6xxx; по ходу обсуждения будем также отмечать принципиальные отличия организации команд в процессорах MSC810x.

Слово команды (длина 32 бита) в процессорах TMS320C6xxx содержит следующие поля (рис. 6.2):

- ❒ условия;
- ❒ операции;
- ❒ операндов;
- ❒ устройства;
- ❒ признака группировки.

Рис. 6.2. Обобщенная структура слова команды в процессорах с одновременным выполнением группы команд

Поле условия хранит код условия выполнения операции.

Поле условия в словах всех команд процессоров MSC810x отсутствует. Переход (разветвление) в командах управления выполняется только по фиксированному условию (состоянию определенного бита в регистре состояния).

В процессорах TMS320C6xxx, напротив, слова почти всех команд содержат поле условия. Характерной особенностью этих процессоров является единообразная структура слова большинства команд (см. рис. 6.2). Поле условия, как в командах операций над данными, так и в командах безусловных переходов, используется для указания условия выполнения самой команды. Команд условного перехода как таковых в этих процессорах нет. Передача управления различным точкам программы (разветвление) реализуется с помощью команд безусловного перехода, которые выполняются или не выполняются в зависимости от условия, указываемого пользователем.

Поле условия в словах всех команд процессоров TMS320C6xxx имеет одинаковую длину 4 бита, из которых:

- ❒ три бита отводятся для кода имени тестируемого регистра (одного из пяти регистров общего назначения в процессорах TMS320C62xx/67xx, или одного из шести регистров — в TMS320C64xx);
- ❒ один бит отводится непосредственно для условия, которым может быть равенство или неравенство нулю содержимого тестируемого регистра.

Поле операндов содержит указания на операнды.

Процессорами с параллельной организацией выполнения команд не поддерживается прямое или косвенное обращение к ячейкам памяти данных (кроме команд пересылок, где оно может быть только косвенным), поэтому операнды указываются именами регистров регистрового файла или непосредственно константой.

Команды переходов в поле операндов содержат адрес перехода, указанный прямо или косвенно (именем регистра, в котором он хранится).

Поле устройства содержит код устройства, выполняющего операцию.

В словах команд процессоров MSC810x фирмы Motorola поле устройства отсутствует; определение активного устройства (выполняющего соответствующую операцию) производится на этапе группировки команд с учетом последовательности команд в командной строке.

Поле признака группировки содержит код признака объединения данной команды со следующей для их совместного выполнения в группе.

В процессорах TMS320C6xxx поле признака группировки представляет собой один бит, состояние которого указывает на группировку или ее отсутствие.

В процессорах MSC810x фирмы Motorola поле признака группировки включает два бита; в этих процессорах может использоваться и другой способ указания группировки — специальное (префиксное) слово, предшествующее группируемым командам и содержащее информацию обо всей группе: количестве команд, условии выполнения всей группы команд и т. д.

Организация одновременного выполнения группы команд определяется архитектурой конкретного процессора. Рассмотрим принцип такой организации на примере процессоров TMS320C6xxx.

Пакеты команд

Процессоры TMS320C6xxx, использующие архитектуру VLIW (Very Long Instruction Word, очень длинное слово команд), имеют в своем составе два одинаковых набора устройств, каждый из которых включает один умножитель, два АЛУ и одно устройство генерации адреса (УГА), что позволяет объединить в одном слове длиной 256 битов до восьми 32-разрядных команд.

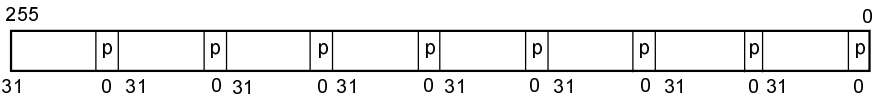
Слово длиной 256 битов называют *пакетом выборки* (рис. 6.3, *а*), т. к. на каждом такте ГТИ в процессоре осуществляется выборка такого пакета (выборка — первая стадия конвейерной обработки).

Объединение данной команды со следующей для одновременного выполнения различными устройствами задается установкой бита p в слове команды; бит p и представляет собой поле признака группировки. Состояние $p = 1$ в слове i-й команды означает, что она должна быть объединена с $(i + 1)$-й командой для одновременного выполнения. Слово последней (восьмой) команды пакета всегда должно содержать $p = 0$, поскольку она не может быть объединена в группу с командами из следующего пакета выборки.

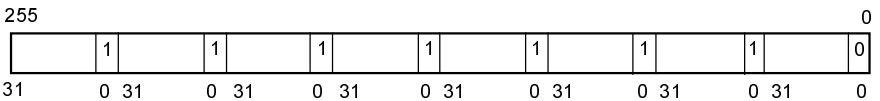
Пакет выборки не обязательно содержит группу из восьми одновременно выполняемых команд. В зависимости от значений бита p в словах различных команд одного пакета выборки возможны следующие ситуации:

- одновременное выполнение группы из 8-ми команд (рис. 6.3, *б*);
- последовательное выполнение всех 8-ми команд (рис. 6.3, *в*);
- n групп команд, выполняемых одновременно и m команд, выполняемых последовательно (рис. 6.3, *г*).

а) пакет выборки

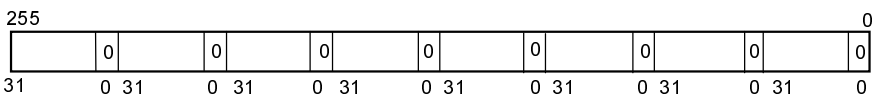


б) одновременное выполнение всех 8-ми команд в пакете выборки



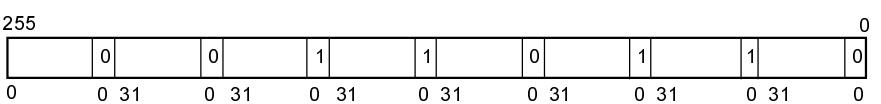
Один исполняемый пакет из 8-ми команд
C1:C2:C3:C4:C5:C6:C7:C8

в) последовательное выполнение всех 8-ми команд в пакете выборки



Восемь исполняемых пакетов
C1
C2
C3
C4
C5
C6
C7
C8

г) частично одновременное и частично параллельное выполнение команд в пакете выборки



Четыре исполняемых пакета
C1
C2
C3:C4:C5
C6:C7:C8

Рис. 6.3. Группировка команд

Команды пакета выборки, выполняемые одновременно, образуют *исполняемый пакет*. Соответственно рассмотренным выше ситуациям имеем (рис. 6.3):

- один исполняемый пакет;
- восемь исполняемых пакетов;
- (n + m) исполняемых пакетов.

6.3. Синтаксис команд

Известны два варианта ассемблерного синтаксиса команд:

- мнемонический;
- алгебраический.

Мнемонический синтаксис, напрямую связанный со структурой слова команды, по существу, представляет собой символическую запись информации, содержащейся в каждом из полей. *Алгебраический* синтаксис отличается от мнемонического более привычной и потому легче читаемой записью вычислительных операций. Покажем отличие на примере команды сложения:

- мнемонический синтаксис

```
ADD X, A
```

- алгебраический синтаксис

```
A=A+X
```

В дальнейшем для сравнения будем использовать обе разновидности синтаксиса при записи одинаковых команд процессора TMS320C54xx фирмы Texas Instruments.

Поскольку синтаксис команды связан со структурой слова команды, рассмотрим отдельно синтаксис команд в процессорах со стандартной архитектурой и особенности синтаксиса в процессорах с одновременным выполнением группы команд; такие процессоры будем оговаривать особо, в противном случае, по умолчанию будем подразумевать процессоры со стандартной архитектурой.

Для краткости указания на операнды будем называть просто *операндами* (что часто используется), понимая разницу между ними.

6.3.1. Синтаксис команд в процессорах со стандартной архитектурой

Слово команды, предназначенной для выполнения *операций над данными*, в общем случае содержит поля: условия, операции, операндов и параллельных

пересылок (см. рис. 6.1). Этой структуре соответствует следующий синтаксис команд в различных процессорах:

- *Мнемонический* синтаксис команд операций над данными:
  - в процессорах DSP56xxx/9600x фирмы Motorola:

    ```
    КОП [Список операндов] [Параллельная пересылка 1] [Параллельная пересылка 2]
    ```

    в том числе, отдельно для команд пересылок:

    ```
    КОП [Условие] [Пересылка 1] [Пересылка 2]
    ```

    где синтаксис `Пересылки` в командах пересылок всех процессоров при загрузке (чтении) данных из памяти в регистр имеет вид:

    ```
    Операнд, Место пересылки
    ```

    или при сохранении (записи) данных из регистра в памяти:

    ```
    Место пересылки, Операнд
    ```

  - в процессорах TMS320C2xxx фирмы Texas Instruments:

    ```
    КОП [Список операндов, Shift]
    ```

    где `Shift` указывает длину (в битах) и направление сдвига операнда.

    В этих процессорах команды пересылок по условию отсутствуют;

  - в процессорах TMS320C5xxx фирмы Texas Instruments:

    ```
    [КОП пересылки Пересылка]
        [ || ]КОП [Список операндов,Shift]
    ```

    где символ `||` используется только в комбинированных командах *(см. разд. 6.5.4)*,

    в том числе, отдельно для команд пересылок:

    ```
    КОП Пересылка[, Условие]
    ```

- *Алгебраический* синтаксис команд выполнения операций над данными:
  - в процессорах TMS320C5xxx фирмы Texas Instruments:

    ```
    [Операция пересылки]
    [ || ] Операция
    ```

    в том числе, отдельно для команд пересылок:

    ```
    [if (Условие)] Операция пересылки
    ```

  - в процессорах фирмы Analog Devices с параллельными пересылками:

    ```
    Операция [Операция параллельной пересылки 1] [Операция параллельной пересылки 2];
    ```

    в том числе, отдельно для команд пересылок:

    ```
    [if Условие] Операция;
    ```

Во всех командах содержимое, заключенное в квадратные скобки, может отсутствовать.

Синтаксис параллельных пересылок в конкретных командах различных процессоров определяется типом разрешенных пересылок (см. табл. 6.3) и адресацией операндов. В табл. 6.4 приведены примеры параллельных пересылок с использованием мнемонического и алгебраического синтаксисов.

При описании типов пересылок в табл. 6.4 использованы следующие условные обозначения:

- # — префикс константы;
- $ — префикс шестнадцатеричного эквивалента константы в процессорах фирмы Motorola;
- (·) — содержимое;
- D:(регистр адреса), X:(регистр адреса), Y:(регистр), P:(регистр адреса) — указанный косвенно адрес ячейки памяти данных, X-памяти данных, Y-памяти данных или памяти программ соответственно.

Особенности адресации операндов *см. в главе 5.*

Таблица 6.4. *Примеры синтаксиса параллельных пересылок*

Процессор	Пример синтаксиса параллельных пересылок		Операция	
	Поле 1	**Поле 2**	**Поле 1**	**Поле 2**

Motorola DSP56xxx
Мнемонический синтаксис
	`#$45,B0`
	`B,X1`
	`X:(R6)+,A`
	
	`X:(R3),A`
	`X:(R0)-,X1`
		
		`A,Y1`
		`Y0,Y:(R7)+`
			$45 → B0
			(B) → X1
			(X:(R6)) → A
			(R6)+1 → R6
			(X:(R3)) → A
			(X:(R0)) → X1
			(R0)–1 → R0
				
				(A) → Y1
				(Y0) → Y:(R7)
				(R7)+1 → R7

Analog Devices ADSP-21xx
Алгебраический синтаксис
	`MX0 =`
	`= DM(I0,M0)`
	
	`DM(I1,M1) =`
	`= AR`
	
	`AX0 = MR2`
		`MY0 =`
		`= PM(I4,M5)`
		
		`AY0 =`
		`= PM(I4,M5)`
		
		`AY0 =`
		`= DM(I0,M3)`
			(D:(I0)) → MX0
			При (M0) = 0
			
			AR → D:(I1)
			При (M1) = 0
			(MR2) → AX0
				(P:(I4)) → MY0
				При (M5) = 0
				(P:(I4)) → AY0
				При (M5) = 0
				(D:(I0)) → AY0
				При (M3) = 0

В дальнейшем будем рассматривать команды *без операций параллельных пересылок*.

Синтаксис команд операций с данными в различных процессорах показан на примерах одной и той же команды сложения `ADD` (табл. 6.5), которая отличается количеством и адресацией операндов, наличием сдвига, условия и т. д.

Для краткости описания операций в табл. 6.5 введены следующие условные обозначения:

- Shift — сдвиг при мнемоническом синтаксисе (−Shift — соответствует сдвигу вправо);
- R_n — n-разрядный регистр R;
- A, B — аккумуляторы A и B;
- << — символ сдвига вправо при алгебраическом синтаксисе.

Особенности адресации операндов *см. в главе 5*.

Таблица 6.5. *Примеры синтаксиса команд без параллельных пересылок*

Процессор	Примеры синтаксиса команды сложения		Операция
	Поле условия	Поля операции и операндов	
Motorola DSP56xxx Мнемонический синтаксис		`ADD X0,A` `ADD A,B`	$(X0)_{24} + (A)_{56} \rightarrow A_{56}$ $(A)_{56} + (B)_{56} \rightarrow B_{56}$
Texas Instruments TMS320C2xxx Мнемонический синтаксис		`ADD XN, Shift` `ADD #1`	$(A)_{32} + (XN)_{16} \times$ $\times 2^{Shift} \rightarrow A_{32}$ $(A)_{32} + 1 \rightarrow A_{32}$
Texas Instruments TMS320C5xxx Мнемонический синтаксис		`ADD *AR2,*AR3,A` `ADD A,-8,B`	$(D:(AR2))_{16} +$ $+ (D:(AR3))_{16} \rightarrow A_{40}$ $(B)_{40} + (A)_{40} \times 2^{-8} +$ $+ \rightarrow B_{40}$
Алгебраический синтаксис		`A = *AR2 + *AR3` `B = B + A << 8`	$(D:(AR2))_{16} +$ $+ (D:(AR3))_{16} \rightarrow A_{40}$ $(B)_{40} + (A)_{40} \times 2^{-8} +$ $+ \rightarrow B_{40}$
Analog Devices ADSP-21xx Алгебраический синтаксис	`if NE`	`AR = AX0 + AY0;` `AR = AR + 2;`	Если регистр, хранящий результат операции АЛУ $\neq 0$, $(AX0)_{16} +$ $+ (AY0)_{16} \rightarrow AR_{40}$ $(AR)_{40} + 2 \rightarrow AR_{40}$

Слово *команды управления* (перехода, вызова подпрограмм, цикла и т. д.) в общем случае содержит поля: условия, операции и операндов; при этом в поле операндов может размещаться: адрес перехода, адрес последней команды цикла и др. Параллельные пересылки в командах управления *недопустимы*. Этой структуре соответствует следующий синтаксис команд в различных процессорах:

❒ *Мнемонический* синтаксис команд управления:

- в процессорах DSP56xxx/9600x фирмы Motorola:
  - ◊ команд переходов

    ```
    КОП [Условие] [Адрес перехода]
    ```

  - ◊ команд повторения

    ```
    КОП Количество повторений
    ```

  - ◊ команды цикла

    ```
    DO Количество циклов Адрес последней команды цикла
    ```

- в процессорах TMS320C2xxx/5xxx фирмы Texas Instruments:
  - ◊ команд переходов

    ```
    КОП [Адрес перехода] [Условие][,Условие[,Условие]
    ```

  - ◊ команд повторения

    ```
    КОП Количество повторений
    ```

  - ◊ повторения блока команд

    ```
    КОП Адрес последней команды повторяемого блока
    ```

  - ◊ команд условного выполнения следующих (одного или двух) слов команды

    ```
    XC k, Условие [,Условие[,Условие]]
    ```

    где `k` = 1, 2, что соответствует выполнению одного или двух слов команды, в зависимости от `Условия`.

❒ *Алгебраический* синтаксис команд управления:

- в процессорах TMS320C5xxx фирмы Texas Instruments:

  ```
  if (Условие[, Условие [,Условие]]) Операция Адрес перехода
  ```

  команды условного выполнения следующих (одного или двух) слов команды:

  ```
  if (Условие) [, Условие [,Условие]]) Операция (k)
  ```

- команд управления в процессорах фирмы Analog Devices:

  ```
  [if Условие] Операция [Адрес перехода];
  ```

команды цикла:

```
DO Адрес последней команды [Условие];
```

Адреса в командах могут указываться прямо (меткой или абсолютным значением) или косвенно. Примеры использования мнемонического и алгебраического синтаксиса конкретных команд управления в различных процессорах приведены в табл. 6.6. Для краткости описания операций в табл. 6.6 введены следующие условные обозначения:

- PM:ADR — адрес перехода в памяти программ, указанный прямо;
- PM:(Регистр адреса) — адрес перехода в памяти программ, указанный косвенно;
- A, B — аккумуляторы A и B;
- h — суффикс шестнадцатеричного адреса ячейки памяти программ в процессорах фирмы Texas Instruments.

Особенности адресации переходов *см. в главе 5.*

Таблица 6.6. *Примеры синтаксиса команд управления*

Процессор	Примеры синтаксиса команд управления	Операция
Motorola DSP56xxx Мнемонический синтаксис	`JMP (R1)`	Безусловный переход к PM:(R1)
	`JSLT $125`	Если результат предыдущей операции < 0, переход к PM:125
	`REP (X0)`	Следующая команда повторяется (X0) раз
	`DO #5 LABEL`	5 циклов, `LABEL` — метка последней команды цикла
Texas Instruments TMS320C2xxx TMS320C5xxx Мнемонический синтаксис	`B MET1`	Безусловный переход к PM:MET1
	`CC 1240h, AGT`	Если (A) > 0, переход к PM:1240h
	`RPT #1103`	Следующая команда повторяется 1103h раз
	`XC 1, ALEQ`	Если (A) ≤ 0, выполняется следующая однословная команда
Алгебраический синтаксис (TMS320C5xxx)	`Goto MET1`	Безусловный переход к PM:MET1
	`if (AGT)call 1240h`	Если (A) > 0, переход к PM:1240h
	`Repeat(#1103h)`	Следующая команда повторяется 1103h раз
	`if (ALEQ)execute (1)`	Если (A) ≤ 0, выполняется следующая однословная команда

Таблица 6.6 *(окончание)*

Процессор	Примеры синтаксиса команд управления	Операция
Analog Devices ADSP-21xx Алгебраический синтаксис	`if EQ JUMP (I4);`	Если результат операции АЛУ = 0, выполняется переход к PM:(I4), иначе — к следующей команде
	`if LT RTS;`	Если результат операции АЛУ < 0, возврат из подпрограммы
	`DO LAST UNTIL EQ;`	`LAST` — последняя команда цикла; цикл продолжается пока результат операции АЛУ = 0

6.3.2. Синтаксис команд в процессорах с одновременным выполнением группы команд

Слова команд операций с данными и команд управления в процессорах TMS320C6xxx фирмы Texas Instruments (с одновременным выполнением группы команд) имеют одинаковую структуру и содержат поля: условия, операции, операндов, устройства и признака группировки (см. рис. 6.2). Согласно этой структуре мнемонический синтаксис команд имеет вид:

```
[ || ] [Условие] КОП .Устройство [Список операндов]
```

где `||` — признак объединения команд в группу для одновременного выполнения различными устройствами процессора; `Условие`, если оно есть, должно указываться в квадратных скобках.

Мнемонический синтаксис команд в процессорах MSC810x фирмы Motorola представляет собой строку команд:

```
КОП 1 [1-й Список операндов] ... [КОП 8] [8-й Список операндов]
```

Приведем пример мнемонического синтаксиса группы из 4-х одновременно выполняемых команд (исполняемого пакета) в процессорах TMS320C6xxx:

```
     [B0]   ADD  .L2  B0,1,B0   ; инкремент (B0), если (B0)=0
||   [B0!]  ADD  .S2  B8,B7,B7  ; (B8)+(B7) → B7, если (B0) ≠ 0
||   [A1]   ADD  .L1  A5,A4,A5  ; (A5)+(A4) → A5, если (B0) = 0
||          B    .S1  LOOP      ; безусловный переход к PM:LOOP
```

Каждая из этих команд выполняется независимо в одном из четырех АЛУ с символическими именами L1, L2, S1 и S2.

6.3.3. Операции над упакованными данными

В процессорах TMS320C62xx/64xx фирмы Texas Instruments (с одновременным выполнением группы команд) имеются команды, предназначенные для обработки упакованных данных *(см. главу 3)*. Напомним, что упакованные данные представляют собой группу данных формата "полуслово" или "байт" (все данные в группе имеют одинаковый формат), сохраняемых в формате слово. В процессорах TMS320C62xx/64xx слово имеет длину 32 бита, что позволяет последовательно расположить в этом формате (упаковать) пару данных формата "полуслово" (2×16) или 2 пары данных формата "байт" (4×8, только в TMS320C64xx).

В процессорах TMS320C62xx возможно с помощью одной команды выполнить одновременно одинаковую операцию с данными формата "полуслово", упакованными в формате "слово" длиной 32 бита, а в процессорах TMS320C64xx — дополнительно операцию с данными формата "байт", упакованными в формате "слово". При выполнении соответствующих операций упакованные данные рассматриваются как локально замкнутые, поэтому переносы и заемы битов между ними недопустимы.

Рассмотрим пример выполнения операции сложения упакованных данных.

По команде

```
ADD4 .L1 A0, A1, A2
```

процессора TMS320C64xx в одном из АЛУ (с символическим именем L1) выполняется одновременно 4 операции сложения над данными формата "байт", упакованными в формате "слово" длиной 32 бита (рис. 6.4).

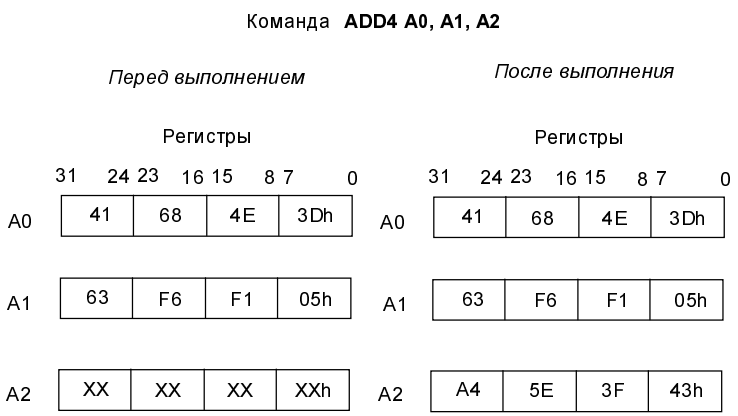


Рис. 6.4. Сложение упакованных данных

Слагаемые (данные, упакованные по 4 байта) размещаются в регистрах A0 и A1. Байты данных представлены в дополнительном коде; операция сложения выполняется побайтно как с беззнаковыми целыми, результат операции — данные, упакованные побайтно в дополнительном коде — сохраняется в регистре A2. При побайтном выполнении сложения данные считаются локально замкнутыми, переносы битов между ними запрещены. Например, при сложении байтов данных 4E и F1 (см. рис. 6.4), результат без переноса в старший разряд равен 3F, в то время как в действительности он равен 13F.

Для упаковки и распаковки данных в процессорах TMS320C6xxx используются специальные команды.

Формирование системы команд в процессорах с одновременным выполнением групп команд не имеет принципиальных отличий по сравнению с остальными процессорами, поэтому при дальнейшем обсуждении команды подобных процессоров особо не выделяются.

6.4. Формирование системы команд

Система команд включает комплект команд, обеспечивающих:

- выполнение операций над данными;
- управление последовательностью этих операций;
- общее управление работой процессора

и позволяющих в целом реализовать системы ЦОС различного назначения. С этой целью система команд любого ЦПОС формируется из однотипных групп команд, которые обсуждаются *в разд. 6.5*.

Сравнение систем команд различных процессоров производится по совокупности следующих основных показателей:

- полнота системы (достаточность команд);
- унификация (единообразие) структуры слова и синтаксиса различных команд;
- разнообразие типов адресаций;
- количество указаний на операнды в командах арифметических операций;
- возможность параллельных пересылок;
- возможность условного выполнения команды.

Произвести сравнительную оценку систем команд при большом количестве показателей нелегко, поэтому подобные оценки носят в определенной мере субъективный характер. Вместе с тем, понятно, что эти показатели связаны с длиной слова команды. При большой длине слова команды для каждого поля, а также для каждого объекта внутри поля, можно выделить свои, неза-

висимые друг от друга, участки (количество битов), тем самым унифицируя структуру слова команды и ее синтаксис. Программисту не потребуется помнить о множестве индивидуальных особенностей различных команд.

Однако, при большом количестве команд в системе, указании трех операндов (в арифметических операциях), разнообразии типов адресаций, наличии параллельных пересылок, условия выполнения операции и т. д. длина слова команды окажется слишком большой, а, следовательно, потребуются бóльшая разрядность соответствующих шин и регистров, бóльший объем памяти программ, увеличатся габариты и возрастет стоимость процессора.

В то же время, для многих приложений требуются простые и дешевые сигнальные процессоры, поэтому разработчики систем команд учитывают область применения процессоров. При относительно небольшой длине слова команды (16—24 бита) по перечисленным показателям принимаются компромиссные решения, в результате системы команд различных процессоров имеют свои сильные и слабые стороны.

Например, показатель унификации синтаксиса различных команд наиболее высок для процессоров TMS320C6xxx фирмы Texas Instruments (что достигается за счет достаточно большой длины слова команды — 32 бита), возможность параллельных пересылок наиболее полно обеспечивается в процессорах фирмы Motorola, а возможность условного выполнения отдельных команд — в процессорах фирмы Analog Devices и т. д.

6.5. Группы команд

Согласно выполняемым действиям команды можно разделить на следующие основные группы:

- ❒ пересылки;
- ❒ арифметические;
- ❒ логические;
- ❒ бит-манипуляций;
- ❒ управления.

Рассмотрим команды по группам.

6.5.1. Команды пересылок

Команды пересылок в основном предназначены для:

- ❒ чтения (загрузки) операндов (исходных данных) из памяти в регистр;
- ❒ записи (сохранения) операндов (результатов) из регистра в память;
- ❒ организации ввода/вывода данных — пересылки в ячейки памяти, которые предназначены для связи с внешней и внутренней периферией.

Различают следующие основные типы пересылок:

- ❒ в регистр:
  - из ячейки памяти данных;
  - из другого регистра;
  - из ячейки памяти программ;
  - константы;
- ❒ в ячейку памяти данных:
  - из другой ячейки памяти данных;
  - из регистра;
  - из ячейки памяти программ;
  - константы;
- ❒ в ячейку памяти программ:
  - из ячейки памяти данных;
  - из регистра.

Соответственно перечисленным типам пересылок место, куда пересылается операнд, указывается:

- ❒ именем приемника;
- ❒ исполняемым адресом;
- ❒ адресом ячейки памяти программ.

Пересылаемый операнд указывается:

- ❒ именем источника;
- ❒ исполняемым адресом;
- ❒ константой;
- ❒ адресом ячейки памяти программ.

Команды пересылок могут отличаться:

- ❒ наличием *условия* пересылки;
- ❒ указанием *сдвига* данных перед их пересылкой;
- ❒ *типом* пересылаемых *данных* (байт, старшее или младшее полуслово, слово, длинное слово).

Синтаксис команд пересылок приведен выше, а примеры команд пересылок — в табл. 6.7. Особенности адресации операндов *см. в главе 5.*

Таблица 6.7. *Примеры команд пересылки*

Процессор	Примеры команд пересылок	Операция

Motorola DSP56xxx Мнемонический синтаксис	`MOVE #75,A0`
	`MOVE X1,B`
	`MOVEC A,LC`
	`MOVEM R3,P:(R2)`
	`MOVEP X: << $FFFF,A`
		#75 → $A0_{24}$
		$(X1)_{24}$ → B_{56}
		$(A)_{56}$ → LC_{16}
		$(R3)_{16}$ → P:(R2)
		(X:$FFFF) → A_{56} для организации ввода данных

Texas Instruments TMS320C6xxx Мнемонический синтаксис	`LDW .D1 *A10,B1` `STB .D1 A1,*A10`	(D:(A10)) → B1 $(A1)_{32}$ → D:(A10)

Texas Instruments TMS320C2xxx/ C5xxx Мнемонический синтаксис	`LD A,7,B`
	`LD #251,A`
	`LD *AR1,A`
	`STH B,-8,*AR7`
	`MVDD *AR3,*AR5`
		$(A)_{40} \times 2^{7}$ → B_{40}
		#251→ A_{40}
		(D:(AR1)) → A_{40}
		$(B)_{\text{старшее слово}} \times 2^{-8}$ → D:(AR7)
		(D:(AR3)) → D:(AR5)

Алгебраический синтаксис (TMS320C5xxx)	`B = A << 7`
	`A = #251`
	`A = *AR1`
	`*AR7 = hi(B) << (-8)`
	`AR5* = AR3*`
		$(A)_{40} \times 2^{7}$ → B_{40}
		#251 → A_{40}
		$(D:(AR1))_{16}$ → A_{40}
		$(B)_{\text{старшее слово}} \times 2^{-8}$ → D:(AR7)
		(D:(AR3)) → D:(AR5)

Analog Devices DSP-21xx Алгебраический синтаксис	`I1 = I4;`
	`MX0 = DM(I0,M0);`
	`DM(I1,M1) = AR;`
	`MY0 = PM(I4,M5);`
		$(I1)_{16}$ → $I4_{16}$
		(D:(I0)) → $MX0_{16}$ при M0 = 0
		AR_{40} → (D:(I1)) при M1 = 0
		(P:(I4)) → $MY0_{16}$

6.5.2. Команды арифметических операций

Широкий набор арифметических команд обусловлен спецификой алгоритмов ЦОС, в которых основными операциями являются сложение (вычитание), умножение и их комбинация — умножение с накоплением (MAC).

Сложение и вычитание

Варианты команд сложения (вычитания) в одном процессоре или в процессорах различных фирм в основном отличаются:

- количеством указаний на операнды;
- адресацией операндов;
- форматом операндов (слово, двойное слово, упакованные данные);
- типом данных (целые со знаком, беззнаковые);
- возможностью сдвига одного из операндов перед выполнением операции;
- использованием бита переноса C для переноса (заема) при программной организации сложения (вычитания) операндов увеличенного формата (двойное слово, учетверенное слово и т. п.);
- возможностью выполнения по одной команде операции сложения или вычитания в зависимости от состояния определенного бита в регистре состояния (только в процессорах TMS320C55xx);
- наличием в команде условия выполнения операции.

Основные правила выполнения операций сложения (вычитания) обсуждались в *главе 3*, а обработка переполнения, которое может возникнуть при выполнении этих операций, в *главе 4*. Примеры команд сложения приведены в табл. 6.5.

Умножение и умножение с накоплением

Команда *умножения* поддерживается всеми ЦПОС, а команда *умножения с накоплением* — большинством ЦПОС (не поддерживается, например, процессорами TMS320C6xxx фирмы Texas Instruments).

Варианты команд умножения и умножения с накоплением в одном процессоре или в процессорах различных фирм в основном отличаются:

- количеством указаний на операнды;
- адресацией операндов;
- возможностью округления результата;
- типом данных (целые со знаком, беззнаковые);
- форматом данных (слово, двойное слово, упакованные данные);
- наличием в команде условия выполнения операции.

Основные правила выполнения операции умножения обсуждались в *главе 3*, а обработка переполнения, которое может возникнуть при выполнении операции умножения с накоплением (`MAC`), в *главе 4*. Примеры команд для процессоров различных фирм приведены в табл. 6.8. Особенности адресации операндов *см. в главе 5*.

Таблица 6.8. *Примеры команд умножения и умножения с накоплением*

Процессор	Примеры команд умножения и умножения с накоплением	Операция
Motorola DSP56xxx Мнемонический синтаксис	`MPY Y1,Y0,B` `MAC X0,X1,A`	$(Y1)_{24} \times (Y0)_{24} \rightarrow B_{56}$ $(A)_{56} + (X0)_{24} \times (X1)_{24} \rightarrow A_{56}$
Texas Instruments TMS320C6xxx Мнемонический синтаксис	`MPY .M A1,A2,A3`	В умножителе M1: $(A1)_{32} \times (A2)_{32} \rightarrow A3_{32}$
Texas Instruments TMS320C5xxx Мнемонический синтаксис	`MPY AR2*,AR4*,B` `MAC #345,A`	$(AR2)_{16} \rightarrow T_{16}$; $(T) \times (D:(AR4)) \rightarrow B_{40}$ $(A)_{40} + \#345 \times (T)_{16} \rightarrow A_{40}$ Регистр T хранит один из сомножителей
Алгебраический синтаксис	`B = *AR2 * AR4*, T = AR2*` `A = A + #345*T`	$(AR2)_{16} \rightarrow T_{16}$; $(T) \times (D:(AR4)) \rightarrow B_{40}$ $(A)_{40} + \#345 \times (T)_{16} \rightarrow A_{40}$ Регистр T хранит один из сомножителей
Analog Devices ADSP-21xx Алгебраический синтаксис	`if NOT MV MR = MR + + M X0*MY0;`	Если регистр, хранящий результат операции устройства MAC, не переполнен, выполняется операция `MAC`

Арифметический сдвиг

При *арифметическом сдвиге* содержимого регистра *влево* младшие, "освободившиеся", биты заполняются нулями, а старшие, "выдвигаемые", теряются (рис. 6.5, *а*). При арифметическом сдвиге *вправо* младшие, "выдвигаемые" биты теряются, а старшие, "освободившиеся", заполняются расширением знака (рис. 6.5, *б*).

В различных процессорах выполнение команд арифметического сдвига реализовано по-разному. Команды отличаются:

- возможностью сдвига по умолчанию на один бит или на произвольное количество битов;
- сохранением "выдвигаемого" бита (если он один) или последнего из "выдвигаемых" битов в бите переноса C;

- возможностью сдвига относительно старшего или младшего слова регистра-сдвигателя (в процессорах фирмы Analog Devices)

и т. д.

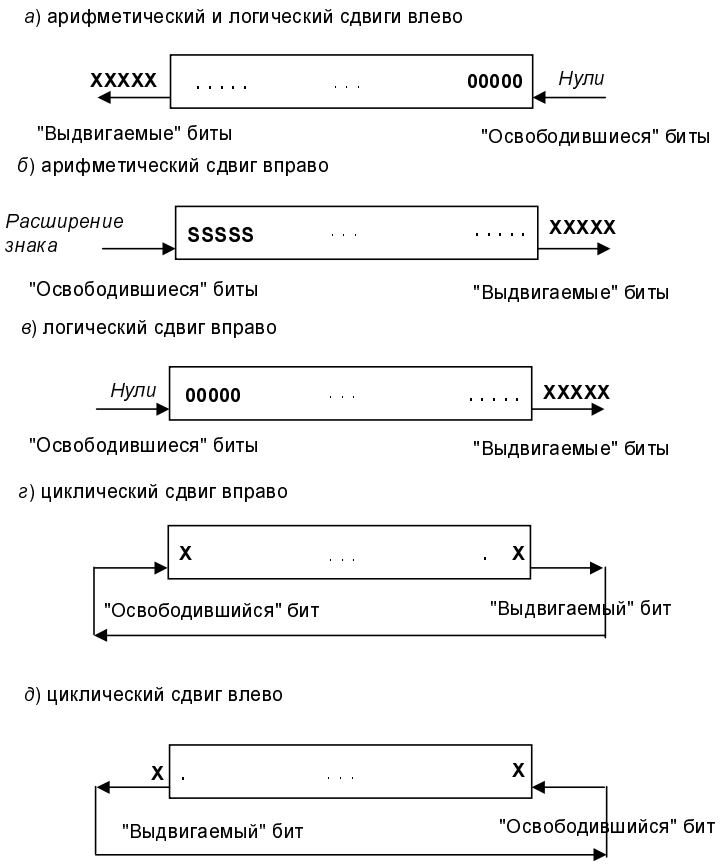


Рис. 6.5. Операции арифметического, логического и циклического сдвигов

Команды арифметического сдвига используют, в частности, для масштабирования данных в процессорах с фиксированной точкой *(см. главу 4)*.

Другие арифметические команды

Кроме основных арифметических команд, различными процессорами поддерживается дополнительный набор команд, из которых наиболее распространенными являются:

- вычисление модуля числа (`ABS`);
- декремент, инкремент (`DEC`, `INC`);

- изменение знака содержимого аккумулятора (NEG);
- округление содержимого аккумулятора (RND);
- нормализация (NORM, *см. главу 3*);
- сравнение (CMP — с содержимым аккумулятора, вспомогательного регистра, константой и т. п.);
- деление (DIV) — итерационная процедура.

6.5.3. Команды логических операций

Группа команд логических операций включает следующие основные команды:

- логических операций;
- логического и циклического сдвигов.

Логические операции

Команды логических (булевых) операций предназначены для выполнения операций AND (умножение), OR (сложение), исключающее OR, NOT (отрицание); в различных процессорах они в основном отличаются:

- количеством указаний на операнды;
- адресацией операндов;
- возможностью сдвига одного из операндов перед выполнением операции;
- выполнением операции со старшим или младшим словом, если операнды имеют разные форматы;
- наличием в команде условия выполнения операции.

Примеры выполнения булевых операций приведены в табл. 6.9.

Таблица 6.9. *Примеры команд логических операций*

Процессор	Примеры команд логических операций	Операция
Motorola DSP56xxx Мнемонический синтаксис	`AND X0,A`	Операции выполняются со старшим словом аккумулятора *До операции:* X0 = $**FF0000** A = $00 **123456** 789ABC *После операции:* X0 = $FF0000 A =$00 **120000** 789ABC
	`NOT A1`	*До операции:* A = $00 **123456** 789ABC *После операции* A = $00 **EDCBA9** 789ABC

Таблица 6.9 *(окончание)*

Процессор	Примеры команд логических операций	Операция
Texas Instruments TMS320C5xxx Мнемонический синтаксис	`OR A,+4,B`	*До операции:* A = 00 0000 1200h B = 00 0000 1800h 1. Сдвиг (A) на 4 бита влево; (A) не меняется: 00 0001 2000h 2. Операция ИЛИ с операндами **00 0001 2000**h **00 0000 1800**h *После операции:* A = 00 0000 1200h B = **00 0001 3800**h
Texas Instruments TMS320C5xx Алгебраический синтаксис	`B = B|A << +4`	См. выше
Analog Devices ADSP-21xx Алгебраический синтаксис	`if NE AR = AX0 XOR AY0`	Если результат операции АЛУ ≠ 0 *До операции:* AX0 = 0x**0003** AY0 = 0x**0005** *После операции:* AR = 0x**0006**

Логический и циклический сдвиг

Логический сдвиг отличается от арифметического тем, что при сдвиге *вправо* старшие, "освободившиеся" биты заполняются не расширением знака, как при арифметическом сдвиге, а *нулями*. При сдвиге влево арифметический и логический сдвиги выполняются одинаково (рис. 6.5, *а*, *в*). Отличия команд логического сдвига в различных процессорах те же, что и для команд арифметического сдвига.

Команды *циклического* (или кольцевого) *сдвига* отличаются от арифметического и логического сдвигов тем, что бит, "выдвигаемый" с одной стороны, автоматически перемещается на место "освободившегося" бита с противоположной стороны, тем самым образуя замкнутое кольцо (рис. 6.5, *г*, *д*).

6.5.4. Комбинированные команды

Комбинированные команды могут быть явными или неявными.

Неявная комбинированная команда синтаксически представляется как одна команда, однако, ей соответствует выполнение двух или более операций. Типичными неявными комбинированными командами являются: команда `MAC`

(умножение со сложением) и ее различные модификации; команды сложения или вычитания и одновременной пересылкой результата (например, команды APAC и SPAC в TMS320C2xxx); команды с округлением результата; команды одновременного выполнения двух однотипных арифметических операций (например, двух умножений), когда один из операндов в обеих операциях — одинаковый (набор подобных комбинированных команд поддерживается в процессорах TMS320C55xx) и т. д. Неявные комбинированные команды различных вариаций имеются в системах команд практически всех ЦПОС.

Явная комбинированная команда представляет собой объединение *двух* команд предназначенных для одновременного выполнения. Такого типа команды поддерживаются в процессорах TMS320C5xxx/3x. Синтаксическим признаком комбинации двух команд служит символ параллельности || между командами *(см. разд. 6.3)*. Явная комбинация команд в этом случае строго регламентирована, разрешены только те комбинации, которые указаны в списке команд соответствующего процессора. Так, в процессорах TMS320C5xxx с фиксированной точкой допускаются только комбинации одной арифметической команды (ограниченного набора из них) плюс команды пересылки, либо двух команд пересылок. В процессорах TMS320C3x с плавающей точкой варианты комбинаций двух команд расширяются, к вышеотмеченным добавляются комбинации: команда пересылки с логической командой, команда пересылки с командой преобразования формы представления данных ФТ ↔ ПТ и др.

Примеры явных комбинированных команд даются в табл. 6.10. При описании операций использованы введенные ранее краткие обозначения; сокращением hi обозначено старшее слово аккумулятора. Особенности адресации операндов *см. в главе 5*.

Таблица 6.10. *Примеры явных комбинированных команд*

Процессор	Примеры синтаксиса команды сложения Поля операции и операндов	Операция
Texas Instruments TMS320C54xx Мнемонический синтаксис	`ST A,*AR2` `||ADD *AR3,B`	$hi(A)_{40} \rightarrow D:(AR2)_{16}$ $(D:(AR3))_{16} + (B)_{40} \rightarrow B_{40}$
Алгебраический синтаксис	`A = *AR2 + *AR3` `B = B + A << 8`	$hi(A) \rightarrow D:(AR2)$ $(D:(AR3))_{16} + (B)_{40} \rightarrow B_{40}$
Texas Instruments TMS320C3xxx Мнемонический синтаксис	`MPYF3 *AR2,*AR3,R0` `|| ADDF3 R5,R7,R3`	$(D:(AR2))_{32} \times (D:(AR3))_{32} \rightarrow R0_{40}$ $(R5)_{40} + (R7)_{40} \rightarrow D:(R3)_{40}$ Данные представлены с ПТ

6.5.5. Команды бит-манипуляций

Бит-манипуляция означает выполнение операции над отдельным, указываемым в команде, битом. В ячейке памяти данных или в регистре возможны следующие операции бит-манипуляции:

- обнуление бита;
- установка бита;
- изменение состояния бита;
- проверка бита.

При выполнении соответствующих команд значение бита, бывшее до его изменения, сохраняется в специальном бите регистра состояния.

Команды бит-манипуляций необходимы, в частности, для *управления* режимами обработки данных и общего управления работой процессора в тех случаях, когда оно определятся состоянием одного бита или группы битов в управляющих регистрах. Например, в процессорах DSP56xxx фирмы Motorola для перехода к обработке с удвоенной точностью необходимо установить бит DM в регистре режима MR, или для запрещения всех прерываний по сигналам от внутренних периферийных устройств в процессоре TMS320C54xx необходимо установить бит INTM в регистре состояния ST1 и т. д.

Приведем пример команды проверки и изменения состояния бита в процессорах DSP5600x фирмы Motorola:

```
BCHG #15, Y:(R5)
```

По команде проверяется состояние 15-го бита в 24-разрядной ячейке Y-памяти данных; это состояние сохраняется в бите переноса C регистра состояния, а затем изменяется на противоположное.

6.5.6. Команды управления

К группе команд управления можно отнести следующие типы команд:

- перехода;
- цикла;
- повторения;
- обращения к подпрограммам;
- возврата из подпрограмм;
- общего управления.

Команды управления в целом обеспечивают программное управление обработкой данных и работой процессора. Рассмотрим особенности перечисленных типов команд.

Команды перехода

Команды перехода предназначены для изменения последовательности выполнения команд, среди них различают команды:

- безусловного перехода;
- условного перехода.

Команды безусловного перехода передают управление ячейке памяти программ, адрес которой указывается в команде. В процессорах TMS320C6xxx фирмы Texas Instruments и процессорах фирмы Analog Devices сами эти команды могут выполняться в зависимости от условия, указанного в команде.

Команды условного перехода передают управление в разные точки программы в зависимости от условия, которое в команде может указываться:

- непосредственно (символически);
- именем регистра, содержимое которого проверяется на выполнение фиксированного условия (например, на равенство 0);
- одновременно именем регистра и номером бита, состояние которого проверяется на выполнение фиксированного условия (например, на равенство 1).

Примеры команд перехода приведены в табл. 6.11. Особенности адресации переходов *см. в главе 5.*

Таблица 6.11. *Примеры команд условного и безусловного переходов*

Процессор	Примеры команд условного и безусловного переходов	Операция
Motorola DSP56xxx Мнемонический синтаксис	`JMP (R2)`	Безусловный переход к ячейке памяти программ, адрес которой хранится в регистре R2
	`JLT $123`	Условный переход к ячейке памяти программ с адресом $123, если результат операции меньше 0, иначе — к следующей команде
	`JSET #12,X0,$4321`	Условный переход к ячейке памяти программ с адресом $4321, если 12-й бит в 24-разрядном регистре X0 установлен, иначе — к следующей команде
Texas Instruments TMS320C2xxx/5xxx Мнемонический синтаксис	`B 2000h`	Безусловный переход к ячейке памяти программ с адресом 2000h
	`BANZ 2000h,*AR3`	Условный переход к ячейке памяти программ с адресом 2000h, если (AR3) = 0, иначе — к следующей команде

Таблица 6.11 *(окончание)*

Процессор	Примеры команд условного и безусловного переходов	Операция
Texas Instruments TMS320C5xx Алгебраический синтаксис	`goto 2000h` `if (*AR3 = 0)goto2000h`	См. выше
Analog Devices ADSP-21xx Алгебраический синтаксис	`if LT JUMP (I1);`	Если результат операции АЛУ меньше 0, выполняется команда, адрес ячейки памяти которой указан в регистре I1, иначе команда воспринимается как `NOP`

Команды цикла

Команды цикла обеспечивают повторное выполнение группы команд при изменяющихся значениях данных.

Программная реализация цикла с фиксированным числом повторений группы команд требует организации счетчика цикла (регистра с инкрементом или декрементом содержимого) и условного перехода по результатам проверки этого счетчика: к продолжению цикла (возврату к первой из повторяемых команд) или выходу из цикла.

Для сокращения времени выполнения цикла (уменьшении количества команд, необходимых для программной организации цикла) и упрощения программирования разработаны команды цикла, *реализуемые аппаратно*, что означает автоматическую организацию счетчика цикла (в специальном регистре) и проверку выхода из цикла.

Синтаксис команд цикла для процессоров фирм Motorola и Analog Devices, приведен *в разд. 6. 3.1.*

Количество повторений в команде цикла может указываться непосредственно константой (статический цикл), адресом ячейки памяти данных или именем регистра, в котором оно хранится (динамический цикл).

Приведем примеры команд цикла для процессоров DSP56xxx фирмы Motorola:

- статический цикл

```
DO #37 MET
```

- динамический цикл

```
DO Y0 MET
```

В системе команд процессоров фирмы Motorola имеется также аппаратно реализованная команда принудительного выхода из цикла до его завершения

```
ENDDO
```

Количество повторений цикла в процессорах фирмы Analog Devices предварительно загружается в специальный регистр CNTR — счетчик цикла.

Рассмотрим пример команды цикла для процессоров ADSP-21xx фирмы Analog Devices:

```
CNTR = 10;
DO MET UNTIL MV;
```

Условие `MV` определяет выход из цикла — при переполнении регистра, хранящего результат операции устройства MAC.

Количество вложенных циклов определяется архитектурой конкретного процессора.

Команды повторения

Команды повторения позволяют организовать повторение:

- *одной* команды;
- *блока* команд.

Системы команд, включающие аппаратно реализованную команду цикла `DO`, как правило, не имеют дублирующей ее команды повторения блока. Повторение одной команды также можно выполнить с помощью `DO`, как это делается в процессорах фирмы Analog Devices, которые не имеют команд повторения.

Процессоры фирмы Motorola, поддерживающие аппаратно команду цикла `DO`, имеют дополнительно команду `REP` повторения одной, следующей за `REP` команды. Количество повторений, подобно команде `DO`, может быть фиксированным и указываться непосредственно константой (статический цикл), либо переменным и указываться адресом ячейки памяти данных или именем регистра, в котором оно хранится (динамический цикл).

Система команд процессоров TMS320C2xxx/5xxx фирмы Texas Instruments не содержит аппаратно реализованной команды `DO`, но включает различные команды, поддерживающие цикл повторения одной команды и блока команд. Количество повторений блока команд предварительно загружается в специальный регистр — счетчик цикла.

Синтаксис команд повторения приведен *в разд. 6.3.1.*

Рассмотрим примеры команд повторения.

- Повторение одной команды в процессорах фирмы Motorola:
  - статический цикл повторения команды `ADD`:

```
REP #3
```

```
ADD X0,A
```

команда ADD повторяется 3 раза;

- динамический цикл повторения команды ADD; количество повторений предварительно загружено в регистр Y1:

```
MOVE #3,Y1
REP Y1
ADD X0,A
```

❐ Повторение блока команд в процессоре TMS320C54xx:

```
    ST #54,BRC
    RPTB end_block – 1
    ADD *AR3,A
    MPY *AR2,*AR4,B
    NEG B
end_block
```

Выполнение блока из трех команд повторяется 55 раз; количество циклов минус 1 предварительно по команде ST загружается в специальный регистр-счетчик BRC.

Команды обращения к подпрограммам

Команды обращения к подпрограммам CALL, в том числе подпрограммам обслуживания прерывания, подобно командам переходов, изменяют последовательный порядок выполнения команд в программе, однако, в отличие от них, при выполнении команд CALL в стеке сохраняется адрес следующей команды для возврата к ней после выполнения подпрограммы. Команды CALL также могут быть безусловными или условными; синтаксис команд CALL и типы возможных условий такие же, как в командах перехода.

Команды возврата

После выполнения подпрограммы, в том числе подпрограммы обслуживания прерывания, как правило, следует продолжить выполнение прерванной программы. Для этого предусмотрены команды возврата, которые восстанавливают содержимое программного счетчика, сохранившееся в стеке, и передают управление команде, следующей за командой вызова подпрограммы CALL.

Команды возврата не содержат операндов, но могут быть *безусловными* или *условными*.

Приведем примеры безусловных команд возврата в процессорах DSP56xxx:

❐ RTS — возврат из подпрограммы;

❐ RTI — возврат из подпрограммы обслуживания прерывания.

Условный возврат из подпрограммы означает, что возможно разветвление: возврат к продолжению прерванной программы, если выполняется условие, указанное в команде возврата, или выполнение команды, следующей за командой возврата.

Приведем примеры условных команд возврата:

- в процессорах TMS320C5xxx фирмы Texas Instruments

  ```
  RC AGT
  ```

  возврат к продолжению прерванной программы происходит, если содержимое аккумулятора A > 0 (условие `AGT`); в этой команде возможно одновременное указание до трех условий;

- в процессорах ADSP-21xx фирмы Analog Devices

  ```
  if NE RTS;
  ```

  возврат к подпрограмме происходит, если результат выполнения операции АЛУ ≠ 0 (условие `NE`);

  ```
  if NE RTI;
  ```

  возврат из подпрограммы обслуживания прерывания происходит при том же условии.

Команды общего управления

Система команд всех процессоров содержит набор команд общего управления, предназначенных для программного управления работой процессора; этот набор обычно включает следующие основные разновидности команд:

- команды перехода процессора в различные состояния, например:
  - `STOP` — останов работы генератора тактовых импульсов в процессорах фирмы Motorola;
  - `IDLE` — переход в режим ожидания в процессорах фирм Texas Instruments и Analog Devices;
  - `WAIT` — переход в режим ожидания в процессорах фирмы Motorola;
  - `RESET` — программный сброс в процессорах фирмы Motorola и Texas Instruments;
  - `SWI`, `III` — команды программного прерывания в процессорах фирмы Motorola;
  - `INTR`, `TRAP` — команды программного прерывания в процессорах фирмы Texas Instruments;
- команды операций со стеком: загрузки вершины стека (верхней ячейки) или сохранения ее содержимого;
- `NOP` — пустая операция

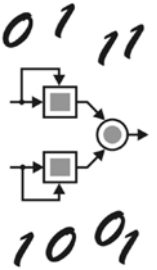
и др.

6.5.7. Особенности команд с плавающей точкой

Арифметические команды над данными с ПТ имеют следующие основные особенности:

- выполнение всех команд включает автоматическую проверку на особые случаи;
- наличие команд преобразования формы представления данных из ФТ в ПТ и наоборот;
- использование в командах (кроме команд пересылок) только прямой адресации операндов с указанием имен источников и приемника — регистров регистрового файла;
- наличие специальных команд обработки данных с ПТ *(см. главу 3)*.

Глава 7



Прерывания

При использовании процессоров в реальных системах ЦОС часто возникает ситуация, когда необходимо прервать выполнение текущей программы по сигналу, поступившему от некоторого источника. В этом случае (рис. 7.1) выполнение текущей программы приостанавливается, запоминается состояние на момент прерывания, выполняется другая, специальная, заранее загруженная программа, после чего восстанавливается сохраненное до прерывания состояние процессора, и продолжается выполнение прерванной программы (или не продолжается, в зависимости от характера прерывания или условия возврата, указанного пользователем).

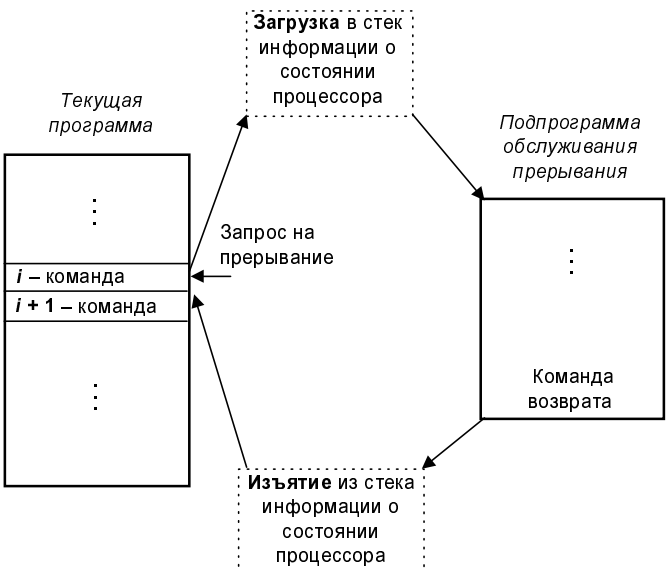


Рис. 7.1. Прерывание процессора

Описанная процедура называется *прерыванием* процессора; сигнал, вызвавший прерывание, — *запросом на прерывание*; источник данного сигнала — *источником прерывания*; последовательность действий, выполняемая по за-

просу на прерывание, — *обслуживанием прерывания*, а выполняемая по прерыванию программа — *подпрограммой обслуживания прерывания*.

Прерывания поддерживаются всеми процессорами; по существу, это способ организации "диалога" процессора с другими объектами — *источниками прерывания*. В настоящей главе рассматриваются общие принципы организации обслуживания прерывания, единые для всех сигнальных процессоров.

7.1. Источники прерывания

Различают два типа источников прерывания:

- аппаратные;
- программные.

Источниками а*ппаратного прерывания* служат внешние и внутренние периферийные устройства.

Описание типовых внутренних и внешних периферийных устройств дается в *главе 8*. Особенности обслуживания прерывания от внешних устройств рассматриваются в этой главе.

Запросом на прерывание от внешнего аппаратного источника является активный сигнал на соответствующем выводе; источник *идентифицируется* по выводу, на котором появляется такой сигнал.

К источникам внешнего прерывания также относится *аппаратный сброс* процессора (установка в начальное состояние), выполняемый по сигналу на выводе $\overline{\text{RESET}}$, который имеется во всех процессорах.

К источникам *программного прерывания* относятся:

- специальные *команды* для *управляемого* программного прерывания процессора; эти команды предусматриваются пользователем в одной или нескольких точках программы (например, команда `INTR` в процессорах TMS320C54xx фирмы Texas Instruments или `SWI` в процессорах DSP5600x фирмы Motorola);
- *особые условия* (exceptions — исключения) — *неуправляемые* программные прерывания, которые представляют собой реакцию процессора на исключительную ситуацию, возникшую при выполнении какой-либо команды (переполнение, деление на ноль, ошибка стека, недопустимая операция и т. д.); прерывания по особым условиям могут обслуживаться:
  - автоматически, например, в процессорах фирмы Motorola автоматически обслуживаются прерывания по ошибке стека, когда адрес в указателе стека становится "выше" или "ниже" допустимого;
  - по команде прерывания, предусмотренной пользователем в тех точках программы, где существует вероятность возникновения особых усло-

вий; прерывание обслуживается при возникновении этих условий (например, по команде `TRAP` в процессорах TMS320C54xx или по команде `III` в процессорах DSP5600x);

- команда программного сброса RESET (если она предусмотрена в процессоре, например, как в процессорах фирм Motorola или Texas Instruments).

Запросом на прерывание от *программного* источника является непосредственно команда прерывания, или установка бита (возможно, битов), фиксирующая возникновение особого условия и вызывающая автоматическое прерывание по соответствующему условию.

Общее количество источников аппаратного и программного прерывания в процессоре может быть различным, например, в процессорах DSP5600x фирмы Motorola и в процессорах TMS320C54xxx фирмы Texas Instruments — 32 источника, в процессорах TMS320C6xxx — 16 источников, в процессорах ADSP-21xx — от 5 до 12 источников в зависимости от модификации и т. д.

7.2. Средства управления прерываниями

Процедура обслуживания прерываний по запросам от нескольких источников в различных процессорах реализуется по-разному. Управление этой процедурой осуществляется специальными устройствами в составе аппаратного обеспечения процессора, например, в процессорах фирм Motorola и Analog Devices — контроллерами прерываний, в процессорах фирмы Texas Instruments — логическими схемами соответствующего функционального назначения и т. д. Вместе с тем, независимо от конкретной реализации, управление прерываниями по существу организуется одинаково, а основными средствами управления являются:

- векторы прерываний;
- приоритеты прерываний;
- маскирование прерываний;
- режимы запуска прерываний;
- флаги прерываний.

Прежде чем рассматривать процедуру обслуживания прерываний, необходимо познакомиться с этими средствами.

Векторы прерываний

Для управления прерываниями по запросам от N источников в пространстве памяти программ выделяется фиксированная область, которая условно делится на N блоков (по количеству источников) объемом m ячеек каждая. В каждом их этих блоков размещаются команды, которые необходимо выполнить по запросу на прерывание от соответствующего источника. Такие блоки назы-

вают *векторами прерывания* (или, коротко, векторами), а адрес первой ячейки каждого блока — *адресом* векторов прерывания (адресом векторов). Таким образом, каждому *источнику* прерывания ставится в соответствие свой адрес вектора прерывания. Совокупность из N векторов образует *таблицу* векторов прерывания, которая обычно располагается, начиная с нулевого адреса в памяти программ (или с нулевого адреса на странице памяти программ в процессорах фирмы Texas Instruments, в этом случае специальные биты в регистре состояния используются для указания страницы).

Количество ячеек m может быть различным; например, $m = 2$ — в процессорах DSP5600x фирмы Motorola, $m = 4$ — в процессорах TMS320C5xxx фирмы Texas Instruments и ADSP-21xx фирмы Analog Devices и т. д. Располагаются векторы не произвольно, а упорядоченно, в соответствии со *структурой приоритетов*, которая рассматривается в следующем разделе. Пример размещения таблицы векторов в памяти программ при $m = 2$ приведен на рис. 7.2.

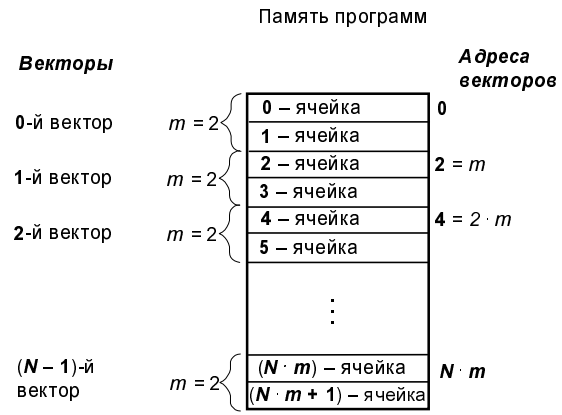


Рис. 7.2. Пример размещения таблицы векторов в памяти программ при $m = 2$

Векторы хранят команды, количество и назначение которых определяет *тип прерывания (см. разд. 7.3)*.

В процессорах с улучшенной стандартной архитектурой и архитектурой VLIW, где имеется возможность одновременного выполнения группы команд, например, в процессорах TMS320C6xxx фирмы Texas Instruments, каждому источнику прерывания также назначается вектор прерывания; при этом адреса векторов следуют с интервалом 32 байта (0020h — в шестнадцатеричной системе), т. к. каждый вектор отводится для хранения пакета выборки из восьми команд длиной 32 бита каждая. Совокупность таких пакетов образует таблицу векторов прерывания. За исключением того, что команды заменяются пакетами выборки, средства управления и процедура обслуживания прерываний в подобных процессорах не имеют принципиальных отличий, поэтому они не рассматриваются отдельно.

Приоритеты прерываний

Возможность одновременного (в течение одного периода ГТИ) поступления запросов на прерывание от различных источников или поступления нового запроса в период обслуживания прерывания по ранее поступившему запросу требует организации *очередности* обслуживания прерываний. С этой целью каждому источнику ставится в соответствие определенный код очередности на обслуживание прерывания, называемый *приоритетом прерывания*, и очередность устанавливается согласно *ранжированию по приоритетам*. Структура приоритетов, определяемая архитектурой процессора, может быть:

- *одноуровневой*, когда каждому источнику ставится в соответствие свой приоритет;
- *двухуровневой*, когда на первом уровне источники разделяются на группы, каждая со своим приоритетом, а на втором — источники ранжируются по индивидуальным приоритетам внутри группы.

Одноуровневая система приоритетов используется в процессорах фирм Texas Instruments и Analog Devices, двухуровневая — в процессорах фирмы Motorola.

Приоритеты символически обозначаются целыми положительными числами; в различных процессорах высший приоритет может кодироваться как наибольшим, так и наименьшим числом. Например, в процессорах TMS320C5xxx фирмы Texas Instruments высший приоритет обозначен наименьшим числом 1 (остальные приоритеты обозначены числами от 2 и выше), а в процессорах DSP56xxx — наибольшим числом 3 (остальные приоритеты обозначены числами 2, 1, 0).

Приоритеты одних источников строго фиксированы, других (но только из числа аппаратных источников) — могут устанавливаться пользователем (в рамках заданных границ) и присваиваться на этапе инициализации процессора. Например, в процессорах фирм Motorola и Analog Devices имеется специальный регистр конфигурации аппаратных прерываний, который содержит пары битов, отображающих приоритет каждого источника; состояние этих битов устанавливается пользователем программно на этапе инициализации процессора.

Векторы в таблице векторов выстраиваются согласно ранжированию приоритетов источников, от высшего (первый вектор) к низшему. Высший приоритет во всех процессорах имеет аппаратный сброс на выводе $\overline{\text{RESET}}$. Далее обычно (но не всегда) последовательно располагаются векторы: внешнего аппаратного прерывания на выводе $\overline{\text{NMI}}$ (Non Maskable Interrupt), программных источников, остальных внешних аппаратных источников, внутренних аппаратных источников. Пример таблицы векторов с одноуровневой структурой приоритетов дается в табл. 7.1.

Таблица 7.1. *Пример таблицы векторов прерывания*

Адрес вектора прерывания	Источник прерывания	Приоритет
0	$\overline{\text{RESET}}$	Высший
m	$\overline{\text{NMI}}$	
2*m*	Источники программного прерывания	
...	...	
...	Внешние источники аппаратного прерывания	
...	...	
...	Внутренние источники аппаратного прерывания	
...	...	
N×*m*	...	Низший

Маскирование прерываний

Поступивший запрос на прерывание не обязательно будет обслуживаться. Запрещение (блокирование) нежелательных прерываний называется *маскированием*.

В зависимости от возможности маскирования все источники прерывания делятся на *маскируемые* (прерывания от которых могут запрещаться или разрешаться) и *немаскируемые* (прерывания от которых не могут запрещаться). К последним относятся:

- аппаратный сброс $\overline{\text{RESET}}$;
- внешнее аппаратное прерывание на выводе $\overline{\text{NMI}}$;
- программные прерывания.

Приоритет немаскируемых источников прерывания всегда выше, чем у маскируемых.

Для маскируемых источников различают общее и индивидуальное маскирование прерываний. *Общее* маскирование означает, что все прерывания (естественно, кроме немаскируемых) запрещены, независимо от их индивидуального маскирования. *Индивидуальное* маскирование говорит о том, что запрещены прерывания конкретных источников (одного или нескольких), указываемых пользователем.

Как общее, так и индивидуальное маскирование реализуются в различных процессорах по-разному. Например, *общее* маскирование в процессорах

фирм Texas Instruments и Analog Devices осуществляется с помощью специальных команд (`SSBX` и `DIS INT` соответственно) и отображается установкой бита общего маскирования; снимается общее маскирование также по командам (`SSBX` и `DIS INT` соответственно), что отображается сбросом бита общего маскирования. В процессорах фирмы Motorola для общего и индивидуального маскирования используются два бита маски прерывания в регистре состояния, их комбинация задает так называемый *текущий приоритет*, означающий, что все прерывания с приоритетом ниже текущего маскируются. Для *общего* маскирования устанавливается *высший текущий приоритет* (соответствующий немаскируемым прерываниям), который автоматически запрещает все маскируемые прерывания. Содержимое табл. 7.2 иллюстрирует принцип общего маскирования; высший приоритет 3 соответствует группе немаскируемых прерываний.

Таблица 7.2. *Маскирование в процессорах фирмы Motorola*

Биты маски прерывания		Приоритет		
I0	I1	Текущий	Разрешенных прерываний	Маскируемых прерываний
0	0	0	0, 1, 2, 3	Маскируемых нет
0	1	1	1, 2, 3	0
1	0	2	2, 3	0, 1
1	1	3	3	0, 1, 2
				Общее маскирование

Для *индивидуального* маскирования в процессорах фирм Texas Instruments и Analog Devices используется регистр масок прерывания, в котором каждому маскируемому источнику соответствует свой бит, и установка этого бита фиксирует маскирование (запрещение) прерывания данного источника. В процессорах фирмы Motorola для этого служат два бита маски прерывания, их комбинация задает текущий приоритет, означающий, что все прерывания с приоритетом ниже текущего маскируются (табл. 7.2).

Режимы запуска прерываний

Режим запуска определяет формирование активного сигнала (запроса на прерывание) для внешних аппаратных источников. Различают два режима запуска:

- по уровню;
- по отрицательному фронту.

Режим запуска может задаваться по умолчанию, либо выбираться пользователем с помощью программной установки битов режима запуска соответствующих аппаратных источников.

Аппаратному сбросу $\overline{\text{RESET}}$ всегда соответствует режим запуска по низкому уровню.

Флаги прерываний

Флагом прерывания называют бит, который устанавливается при поступлении запроса на прерывание от внешнего аппаратного источника. Флаг прерывания имеет каждый аппаратный источник, кроме источников немаскируемых прерываний $\overline{\text{RESET}}$ и $\overline{\text{NMI}}$. В некоторых процессорах, например, процессорах фирмы Texas Instruments, флаги прерывания хранятся в специальном регистре флагов прерывания.

7.3. Типы прерываний

При обслуживании прерывания от конкретного источника происходит автоматическое обращение к вектору прерывания данного источника — ячейкам памяти данных, содержимое которых определяет тип прерывания:

- быстрый, если подпрограмма обслуживания прерывания размещается непосредственно в векторе;
- долгий, если вектор содержит обращение к более сложной подпрограмме обслуживания прерывания.

Быстрые прерывания могут использоваться для организации обмена данными между памятью и устройствами ввода/вывода; они позволяют сократить время обслуживания прерывания, поскольку не требуют команд обращения к подпрограмме и возврата из нее; быстрые прерывания предполагают сохранение ("защелкивание") состояния программного счетчика и *автоматический* возврат в основную программу. Примером быстрого прерывания в процессорах DSP5600x фирмы Motorola (вектор прерывания состоит из двух ячеек памяти программ) служит двусловная команда ввода данных

```
MOVEP X:<<$FFFF,X1
```

согласно которой во входной регистр X1 из ячейки X-памяти данных с адресом $FFFF (соответствующей области связи с внутренней и внешней периферией) поступают данные. После выполнения двусловной команды `MOVEP` происходит автоматический возврат в основную программу и ее продолжение.

Долгие прерывания используются для более сложной обработки по запросу на прерывание. При обращении к соответствующим подпрограммам необходимо сохранять не только значение программного счетчика (адрес команды, к которой следует вернуться после обслуживания прерывания), но также

состояние процессора на момент прерывания; обычно это содержимое регистров состояния и/или других регистров. Содержимое счетчика и необходимых регистров загружается в стек, сохраняется в нем на время выполнения подпрограммы обслуживания прерывания и изымается после возврата из нее (по команде возврата) перед продолжением прерванной программы. Регистры, содержимое которых сохраняется в стеке, а также программный или аппаратный (автоматический) способ загрузки в стек и изъятия из стека определяются архитектурой процессора.

Вложенные прерывания

Обслуживаемое прерывание может быть прервано по запросам от источников, имеющих более высокий приоритет. Такие прерывания называются *вложенными*; они могут быть только долгими, и процедура их обслуживания аналогична обслуживанию обычных прерываний, с той разницей, что прерывается выполнение не основной программы, а подпрограммы обслуживания прерывания от источника с более низким приоритетом. Глубина вложенных прерываний определяется объемом стека и спецификой программы обработки, т. к. стек используется и для других целей, в частности, для реализации аппаратных циклов (в процессорах фирм Motorola и Analog Devices) и для повторения блока команд (в процессорах фирмы Texas Instruments).

В некоторых процессорах, например, фирмы Analog Devices, возможно запрещение вложенных прерываний установкой соответствующего бита. Быстрые прерывания не могут прерываться никакими источниками, т. е. не допускают вложенных прерываний.

7.4. Инициализация процессора для работы в состоянии прерывания

Инициализация (начальная установка) процессора для возможности работы в состоянии прерывания предполагает:

- установку приоритетов и режимов запуска для аппаратных прерываний (не заданных по умолчанию);
- формирование таблицы векторов согласно ранжированию приоритетов источников;
- размещение таблицы векторов в памяти программ (выбор страницы при страничной организации памяти);
- определение содержимого конкретных векторов;
- конфигурацию общего и индивидуального маскирования;
- разрешение или запрещение вложенных прерываний (в тех процессорах, где допустимы оба варианта).

7.5. Обслуживание прерываний

Процедура обслуживания прерывания, несмотря на различные технические решения в конкретных процессорах, одинакова для всех процессоров и условно разделяется на следующие основные фазы:

- прием запросов на прерывание;
- арбитраж прерываний;
- подтверждение прерывания;
- выполнение прерывания.

Остановимся коротко на каждом из этих этапов.

Прием запросов на прерывание

Запрос на прерывание от *немаскируемого* источника фиксируется и после этого сразу начинается следующая фаза его обслуживания — арбитраж.

Запрос на прерывание от *маскируемого* источника также идентифицируется, однако, в отличие от немаскируемых прерываний, после этого автоматически выполняются дополнительные действия в следующей последовательности:

- устанавливается *флаг* прерывания соответствующего источника;
- проверяется состояние бита *общего маскирования*; в зависимости от его состояния возможны две ситуации:
  - *первая ситуация*: бит общего маскирования прерывания *установлен*; следовательно, все маскируемые прерывания *запрещены*, тогда текущая программа продолжается, игнорируя запросы на прерывания от всех маскируемых источников; программа может быть прервана только по запросу немаскируемого программного или аппаратного источника; состояние бита общего маскирования может измениться в ходе выполнения программы с помощью соответствующей команды, тогда происходит автоматический переход ко второй ситуации;
  - *вторая ситуация*: бит общего маскирования прерывания *сброшен*; следовательно, *запрещение* или *разрешение* прерывания данного источника определяется только состоянием бита *индивидуального* маскирования;
- проверяется состояние бита *индивидуального* маскирования данного источника и также возможны две ситуации:
  - *первая ситуация*: бит *установлен*, следовательно, прерывания по запросам от данного источника *запрещены*, и программа продолжается, игнорируя запросы от этого источника;
  - *вторая ситуация*: бит *сброшен*, следовательно, прерывания по запросам от данного источника *разрешены*, поэтому для данного источника начинается следующая фаза его обслуживания прерывания — *арбитраж*.

Описанная процедура принципиально не меняется, если общее и индивидуальное маскирование определяется по состоянию двух битов, например, как в процессорах фирмы Motorola.

Арбитраж прерываний

Арбитраж прерываний необходим для разрешения конфликтной ситуации, когда одно из прерываний выполняется (последняя фаза обслуживания прерывания), несколько прерываний стоят в очереди на выполнение и поступают новые запросы на прерывания.

Арбитраж всех имеющихся на данный момент запросов происходит каждый раз заново *после поступления сигнала подтверждения прерывания*. В этот момент все поступившие запросы на прерывания ранжируются по приоритетам источников. Для прерывания с высшим приоритетом начинается следующая фаза — выполнение прерывания, остальные прерывания, называемые *отложенными* (или ожидающими), ставятся в очередь на выполнение в порядке убывания своих приоритетов. Очередность выполнения может измениться в результате нового арбитража, если к моменту поступления сигнала подтверждения появились новые запросы на прерывание.

Подтверждение прерывания

Немаскируемое прерывание подтверждается немедленно после его идентификации или, если одновременно поступило несколько запросов на немаскируемые прерывания, сразу после арбитража этих запросов.

Для маскируемого *разрешенного* прерывания автоматически выполняются следующие действия:

- устанавливается флаг прерывания обслуживаемого источника;
- производится *общее* маскирование (запрещение всех прерываний, кроме немаскируемых);
- происходит обращение к вектору прерывания обслуживаемого источника;
- после выборки первого или второго слова команды (в зависимости от архитектуры процессора) генерируется *сигнал подтверждения*;
- сбрасывается флаг обслуживаемого прерывания;
- продолжается выполнение прерывания и одновременно возобновляется процесс арбитража.

Выполнение прерывания

Выполнение прерывания соответствует переходу к подпрограмме обслуживания прерывания, ее выполнению и возврату (или невозврату) к продолжению текущей программы. Выполнение прерывания начинается с обращения к вектору прерывания обслуживаемого источника, содержимое которого оп-

ределяет тип прерывания — быстрое или долгое. В различных процессорах выполнение прерываний в основном отличается сохраняемой в стеке информацией, способом загрузки в стек и изъятия из стека (программным или аппаратным), организацией стека (специальный или общего назначения).

При выполнении прерываний по некоторым *особым условиям* (например, по ошибке стека или недопустимой команде) возможно невозвращение к продолжению прерванной программы, поскольку ошибка может оказаться неустранимой, и попытка возврата в прерванную программу приведет к зацикливанию; в этих случаях подпрограмма обслуживания прерывания должна заканчиваться не командой возврата, а командой останова.

Обслуживание прерывания по сигналам аппаратного сброса $\overline{\text{RESET}}$ принципиально отличается от прерываний всех остальных источников. Аппаратный сброс $\overline{\text{RESET}}$ имеет наивысший приоритет и в любой момент может прервать работу процессора для установки его в начальное состояние. Конкретные действия в этом случае определяются архитектурой процессора.

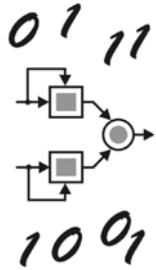
Задержка обслуживания прерывания возникает, если необходимо завершить выполнение команд, прерывание которых привело бы к необратимому нарушению выполнения текущей программы, например, задержка происходит, если выполняется команда повторения `REP` (или ее модификации) и следующая за ней до их полного завершения.

7.6. Состояние ожидания

Ожиданием называется состояние низкого потребления энергии процессором, когда блокируются все схемы процессора за исключением внутрикристальной периферии. Процессор переходит в состояние ожидания по команде "ожидание до прерывания" (`WAIT` — в процессорах фирмы Motorola, `IDLE` — в процессорах фирм Analog Devices и Texas Instruments).

Выход из состояния ожидания происходит по запросам от немаскируемых аппаратных источников или разрешенных маскируемых аппаратных источников. Обслуживание прерывания выполняется в установленном порядке, после чего выполняется команда, следующая за командой "ожидание до прерывания". При этом, если в процессоре установлен бит общего маскирования, например, как в процессорах фирм Texas Instruments и Analog Devices, процессор все равно выходит из состояния прерывания, однако в этом случае сразу начинает выполняться команда, следующая за `IDLE`. Различные модификации команды `IDLE K` (где `K` принимает несколько возможных значений) в процессорах данных фирм соответствуют разным уровням экономичности энергопотребления и сопутствующим режимам работы периферийных устройств.

Глава 8



Периферийные устройства

8.1. Основные понятия и определения

Процессор, кроме вычислительного блока, содержит разнообразные устройства, обеспечивающие его работу; эти устройства называют *внутренней периферией*. Подключаемые к процессору устройства называются *внешней периферией*.

Внутреннюю периферию составляют:

- аналого-цифровые и цифро-аналоговые преобразователи, если они расположены в одном с вычислительным блоком кристалле;
- порты, обеспечивающие сопряжение с внешней периферией;
- таймеры;
- компандеры;
- генераторное оборудование;
- внутрикристальный эмулятор

и т. д.

К внешней периферии относятся:

- разнообразная память;
- контроллеры, обеспечивающие управление системой;
- аналого-цифровые и цифро-аналоговые преобразователи, если они не входят в состав кристалла, а представляют собой отдельные, самостоятельные микросхемы;
- системы сопряжения с каналом связи;
- устройства отладки

и т. д.

Элементы внешней периферии, здесь не рассматриваемые, отличаются принципами построения, видами сигналов, скоростью передачи сигналов,

способами обмена информацией, конструктивными особенностями и другими параметрами. В связи с этим должна обеспечиваться возможность связи и обмена информацией между любыми устройствами, входящими в систему ЦОС; иначе говоря, необходимо обеспечить *совместимость* ЦПОС с периферией так, чтобы система работала в реальном времени. Кроме того, в процессе отладки системы к ней могут подключаться специальные устройства — *средства отладки*, работающие в диалоговом с инженером режиме. Отсюда вытекает необходимость организации специального *сопряжения* процессора с внешней периферией.

Совокупность методов и средств, обеспечивающая сопряжение процессора и периферии, называется *интерфейсом* (от англ. *interface* — сопряжение, согласование). Интерфейс включает в себя средства обеспечения функциональной, электрической и механической совместимости. В соответствии с этим часто говорят о функциональном, электрическом и механическом интерфейсах. Различают внутренний и внешний интерфейс. Внутренний интерфейс обеспечивает сопряжение между центральным процессорным устройством (ЦПУ) и внутренней периферией; внешний — между процессором и внешней периферией.

8.1.1. Функциональный интерфейс

Функциональный интерфейс предназначен для обмена информацией и данными. Он включает в себя:

- систему согласования по характеристикам управляющих сигналов обмена;
- правила обмена управляющими сигналами;
- протокол — правила обмена информацией и данными.

Согласование по характеристикам управляющих сигналов обмена выявляет две проблемы:

- выработки управляющих сигналов для обеспечения своевременного обмена информацией (данными);
- преобразования внешних сигналов, которые могут быть как цифровыми, так и аналоговыми, в сигналы, совместимые с сигналами на шинах процессора.

Эти проблемы решаются устройствами ввода/вывода, которые называются *портами* (рис. 8.1). Общепринято направление потоков информации рассматривать относительно процессора. Порты и соответствующие им интерфейсы различаются не только по назначению, но и по ряду других функциональных признаков, приводимых в табл. 8.1.

Таблица 8.1. *Общие характеристики портов*

Признак	Общая характеристика
Назначение	Ввод/вывод, управление, связи с другими ЦПОС и процессорами общего назначения, коммуникационные (для организации многопроцессорных систем)
Последовательный и параллельный	Различаются по количеству битов, участвующих в обмене информацией в единицу времени
Тип синхронизации	Синхронный и асинхронный; асинхронным может быть только последовательный порт
Битовый	Управляет процессором по условию своего состояния BIO
Хост-порты	Специализированные параллельные порты для обмена информацией между ведущим (хост) и ведомым процессорами

Портом ввода называют устройство, через которое информация от периферии поступает в процессор. *Порт вывода* — это устройство, через которое информация от процессора передается на периферию. Задача порта ввода/вывода состоит в размещении информации на шине данных, дальнейший же путь информации определяется программой, записанной в памяти процессора.

С точки зрения способа обмена данными различают последовательные и параллельные порты (интерфейсы).

Последовательный интерфейс имеет формат в один бит (символ), т. е. передает и принимает данные по принципу "один бит за другим"; *параллельный интерфейс* — имеет формат в несколько битов (8, 16 или 32 бита), которые принимаются или передаются одновременно.

Для адресации портов используется *адресная шина*. Зачастую адреса портов ввода совпадают с адресами портов вывода (например, и тот и другой адрес может быть равен 5, т. е. выбран пятый порт ввода и пятый порт вывода). Выбор порта ввода или вывода осуществляется с помощью сигналов на соответствующих управляющих линиях. В случаях, когда для ввода и вывода служит одна и та же шина, называемая *мультиплексированной*, предусматривается дополнительная, управляющая линия, сигнал на которой указывает направление движения данных: например, при установке на ней "1" задается вывод, а при установке "0" — ввод.

Момент передачи/приема (вывода/ввода) информации определяется коротким импульсом либо на линии синхронизации, либо на специально выделяемой линии. Этот сигнал называется *стробом* (англ. *strobe* — селекторный импульс). Длительность строба обычно составляет от половины до нескольких периодов тактовой частоты (частоты синхронизации). Таким образом, моменты ввода и вывода можно описать в виде логических выражений:

$$\text{Ввод} = (\text{ввод данных}) \ \& \ \text{строб};$$

$$\text{Вывод} = (\text{вывод данных}) \ \& \ \text{строб}.$$

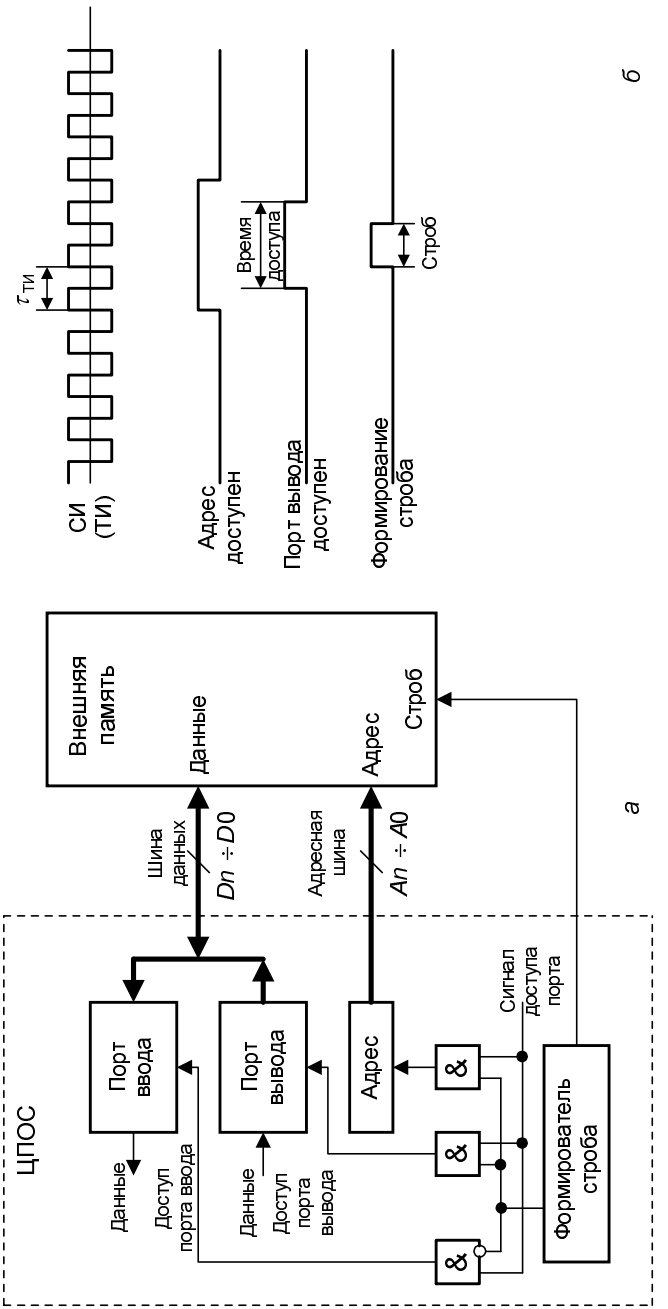


Рис. 8.1. Структурная схема (*а*) и диаграмма вывода данных (*б*)

На рис. 8.1, *а* представлена структурная схема обмена данными между процессором и внешней памятью, а на рис. 8.1, *б* — временная диаграмма процесса вывода данных. По команде пересылки данные из ячейки памяти процессора поступают на порт вывода и формируется сигнал доступа порта вывода; одновременно на шине адреса устанавливается адрес ячейки внешней памяти и формируется сигнал доступа шины адреса. Формирователь строба выдает короткий импульс, который разрешает выставить на шине данных передаваемую информацию и активизирует ячейку внешней памяти согласно адресу, выставленному на адресной шине. Данные записываются в выбранную ячейку.

При организации системы ЦОС возникает проблема полного согласования работы процессора и внешней периферии. Если такого согласования нет, вполне возможна ситуация, когда будет осуществляться ввод данных в процессор, хотя предыдущие данные еще не переданы из порта ввода в ЦПУ. Аналогичный сбой может произойти (и обязательно происходит!) при выводе данных из процессора. Во избежание подобного рода неприятностей вводят правила обмена информацией.

Совокупность правил, устанавливающих единый порядок обмена информацией (данными) и управляющими сигналами, называют *протоколом*.

Используются три способа организации обмена информацией (протокола):

- программный обмен;
- обмен по прерываниям согласно запросам от внешних источников (устройств);
- обмен в режиме прямого доступа к памяти (ПДП).

Программный обмен

Программный обмен осуществляется под управлением программы с одновременной проверкой готовности внешнего устройства к обмену. Для опроса состояния порта ввода/вывода используется регистр состояний порта, в котором выделяется бит готовности порта вывода и бит готовности порта ввода, устанавливаемые в "1" в процессе обмена и сбрасываемые в "0" по окончании обмена. Бит готовности тестируется (проверяется): если выясняется, что порт не готов, операция опроса готовности повторяется. Переход порта в состояние готовности обычно называют *активизацией порта*.

Обмен по прерываниям

Напомним, что прерывание — это переход на выполнение подпрограммы, который вызывается сигналом, поступающим в процессор от внешних устройств по специально выделяемым для этого управляющим линиям. Поступающий сигнал называют *запросом на прерывание*.

Часто возникает необходимость в организации больших временных задержек и систематических с заданным периодом прерываний. Программный

способ организации таких задержек и прерываний неэффективен в связи с тем, что существенно увеличивается время выполнения основной программы. Для реализации таких событий процессоры снабжаются специальными устройствами — *таймерами*.

Обнаружив запрос на прерывание, процессор откладывает выполнение текущей программы и начинает выполнять программу обработки прерывания, предназначенную для обработки события, вызвавшего прерывание. Программа прерывания (если она не соответствует фатальной ошибке или сбросу) обычно заканчивается командой возврата, по которой продолжается выполнение прерванной программы. Переход на программу обработки прерывания с последующим ее выполнением называют *обслуживанием прерывания*. Действия, происходящие при обслуживании прерывания, и их последовательность подробно изложены в *главе 7*. Здесь только напомним, что при поступлении запросов от нескольких источников первым обслуживается прерывание, имеющее более высокий уровень приоритета, остальные запросы обрабатываются по цепочке снижения уровня приоритета. Задача распределения маскируемых приоритетов решается пользователем на этапе проектирования системы ЦОС и инициализации процессора.

Обмен в режиме прямого доступа к памяти (ПДП)

Рассмотренные выше два способа организации обмена информацией с внешней периферией требуют большой затраты временных ресурсов — и это при том, что никаких дополнительных операций (например, преобразование формата данных) не осуществляется. Поэтому описанные способы оказываются неприемлемыми для организации обмена информацией с внешней памятью, где хранятся большие массивы данных, и к которой происходит частое обращение.

Обмен в режиме ПДП означает, что ввод/вывод данных осуществляется с помощью аппаратных средств независимо от программы, которая в данном случае лишь инициирует процесс ввода/вывода, определяет адрес, формат слова данных и устанавливает соответствующий флаг в регистре управления прямым доступом.

Для организации канала ПДП создаются линии управления ПДП, а также используются имеющиеся шины данных и шины адреса периферии, к которым в режиме временного разделения подключается внешняя память.

Внешняя память и другие устройства внешней периферии, к которым должен обращаться процессор, имеют различные технические характеристики; в частности, они отличаются по длительности *цикла обмена*: чем короче цикл, тем быстрее происходит обмен. В связи с этим различают низкоскоростные и высокоскоростные устройства, с которыми необходимо обеспечить согласование скорости работы процессора. Эту функцию возлагают на *генератор задержек доступа к памяти*.

При каждом обращении к памяти происходит обмен одним словом данных или его частью между ячейкой памяти и процессором, поэтому каждый раз должен быть указан адрес ячейки памяти, участвующей в обмене. Желательно, чтобы при обмене большими блоками данных адрес изменялся автоматически с использованием инкремента или декремента — для этого организуют специальные области памяти, называемые *циклическими буферами (см. главу 6)*. Кроме адреса должно указываться направление обмена, для чего предусматриваются линии управляющих сигналов типа "Чтение/Запись" ("Read/Write").

Из всего сказанного следует, что функциональный интерфейс процессора должен обеспечить возможность реализации как способов обмена данными, так и обслуживание прерываний согласно таблице уровней прерываний. Кроме того, процессор должен работать в многоканальном режиме, когда осуществляется обмен данными с разнообразными внешними устройствами с временным или адресным разделением каналов. Временное разделение каналов часто называют *мультиплексированием*. При этом каналы приема и передачи должны работать независимо друг от друга (как увидим, не все процессоры обеспечивают такую важную функцию).

Наконец, к процессору должно подключаться устройство отладки. В настоящее время этот интерфейс стандартизирован, и им обладают все процессоры. Этот интерфейс имеет наименование *JTAG (Joint Test Automation Group)*.

Таким образом, функциональный интерфейс должен содержать (рис. 8.2):

- ❒ систему портов, количество и типы которых зависят от назначения процессора:
  - последовательные;
  - параллельные;
  - многоканальные;
  - хост-порт;
  - отладки

  и другие;
- ❒ шины:
  - данных;
  - адресов;
- ❒ линии:
  - передачи разнообразных управляющих сигналов;
  - запросов и подтверждений прерываний;
- ❒ контроллер прерываний;
- ❒ генератор задержек доступа к памяти;
- ❒ таймеры.

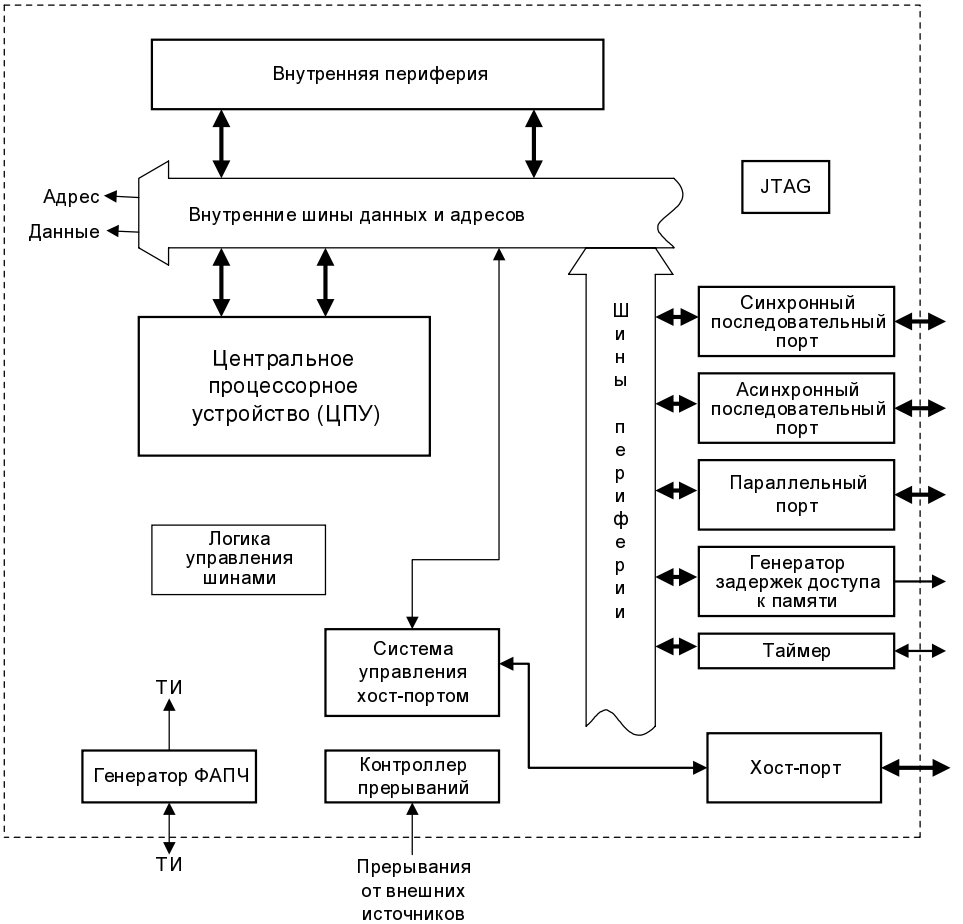


Рис. 8.2. Функциональный интерфейс

8.2. Компандеры

Одной из важных задач является сжатие информации *(см. главу 1)*, при условии несущественных ее потерь, с целью сокращения скорости передачи данных по каналам связи. Особенно важно снижение скорости передачи в КВ-каналах, где принципиально невозможна скорость выше 2400 бит/с, а также в многоканальных системах с временным разделением каналов.

Большинство реальных сигналов имеют динамический диапазон 70 дБ, а типовые каналы и линии связи характеризуются диапазоном 40 дБ (рис. 8.3, *а*). Известно, что один бит соответствует ≈6 дБ, поэтому для обеспечения динамического диапазона сигнала необходимо его линейно кодировать с раз-

рядностью не меньше 14 битов при постоянном шаге квантования $Q = 2^{-m}$ (m — разрядность регистра).

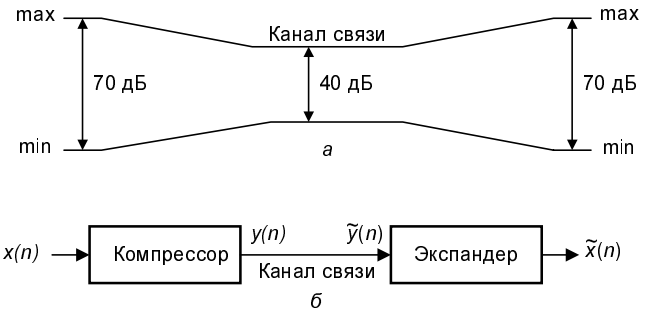


Рис. 8.3. Структурная схема компандера

Линейное квантование, наиболее простое и удобное, не позволяет полностью охарактеризовать сигнал, когда при переходе от кадра к кадру амплитуда сигнала резко меняется. Кроме того, относительная точность представления сильных и слабых сигналов существенно отличается. В результате отношение сигнал/шум оказывается непостоянным, и качество воспроизведения сигнала снижается. Этого можно избежать, если добиться независимости отношения сигнал/шум от уровня сигнала, использовав переменный шаг квантования $Q = var$. Тогда вместо постоянной, не зависящей от уровня сигнала ошибки округления $\varepsilon = Q/2$, что имеет место при линейном квантовании, получим постоянную относительную ошибку.

Поставленная задача решается с помощью специальных устройств, получивших название компандеров (рис. 8.3, *б*), состоящих из двух блоков: *компрессора* (устройства сжатия) на передаче и *экспандера* (устройства расширения) на приеме.

Компрессор усиливает слабые сигналы, поднимая их над уровнем помех в канале, и ослабляет сильные сигналы так, что динамический диапазон передаваемого сигнала уменьшается с 70 дБ (требуется не менее 14 разрядов при равномерном квантовании) до 40 дБ, соответствующих 8-разрядному квантованию.

Экспандер производит обратные действия и восстанавливает динамический диапазон сигнала.

Данные на входе и выходе компандера представляются в 16-разрядном дополнительном коде. Компрессия может осуществляться по линейному закону (рис. 8.4), а также по двум логарифмическим: A-закону и μ-закону (рис. 8.5), т. е. компандер допускает *три формата* 8-разрядных компрессированных данных: *беззнаковое слово* в линейном коде; слово A-закона; слово μ-закона.

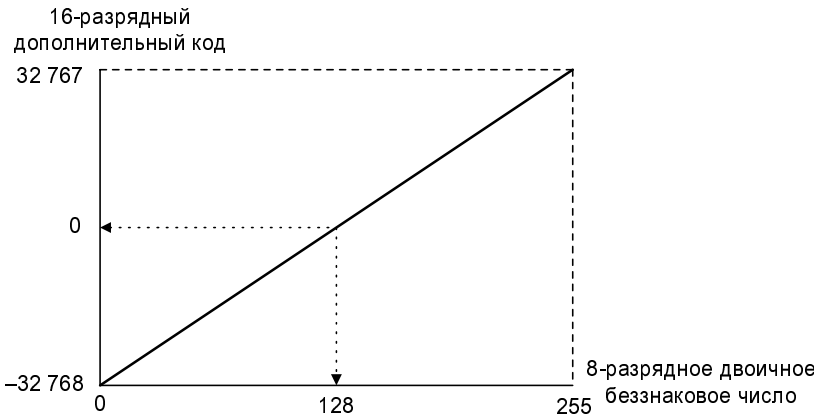


Рис. 8.4. Линейный закон компандирования

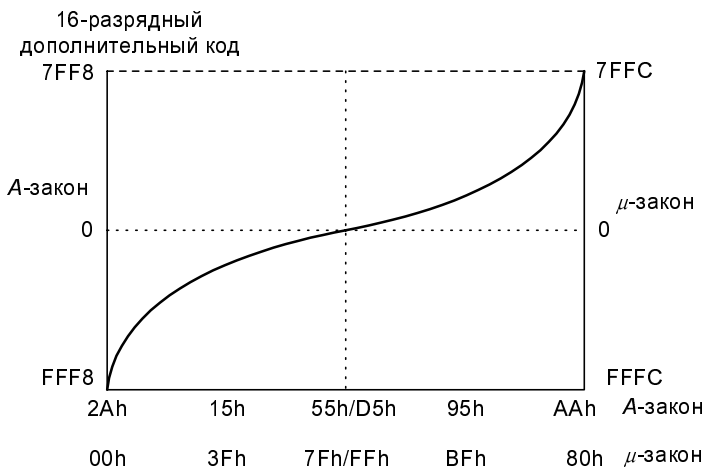


Рис. 8.5. Логарифмические законы компандирования

Компрессия по линейному закону применяется в компандере фирмы Motorola, входящем в состав кодека CS4215. Суть линейного закона компрессии состоит в том, что в 8-разрядном беззнаковом слове, принимающем значения от 0 до 255, отрицательное минимально допустимое 16-разрядное число −32 768 передается нулем; максимально допустимое 16-разрядное положительное число 32 767 передается величиной 255, а ноль компрессируемого числа передается величиной 128 (80h).

Компрессия по A- и μ-закону осуществляется согласно стандарту CCITT G.711, причем A-закон используется в Европе, а μ-закон — в США и

Японии. Согласно этому стандарту указанные законы определяются соотношениями:

A-закон компандирования:

$$y(n) = F[x(n)] = \begin{cases} sign[x(n)]\dfrac{A|x(n)|}{1 + \lg|x(n)|}; & 0 \le |x(n)| < \dfrac{1}{A}; \\ sign[x(n)]\dfrac{1 + \lg A|x(n)|}{1 + \lg A}; & \dfrac{1}{A} \le |x(n)| \le 1, \end{cases} \quad (8.1)$$

где $A = 87{,}6$;

μ-закон компандирования:

$$y(n) = F[x(n)] = sign[x(n)]\frac{\lg\left[1 + \mu\dfrac{|x(n)|}{X}\right]}{\lg(1 + \mu)}X, \quad (8.2)$$

где $X = \max|x(n)|$, $\mu = 255$.

Параметры A и μ определяют степень сжатия: с их ростом отношение сигнал/шум остается постоянным во все более широком динамическом диапазоне значений компрессируемого сигнала, но величина самого отношения при этом уменьшается. Указанные значения параметров являются оптимальным компромиссом между этими противоречиями.

Логарифмическая нелинейность проявляется на больших значениях сигнала и скрадывается на малых (см. рис. 8.5). Оба закона обеспечивают более точную передачу малых амплитуд сигнала и менее точную — больших амплитуд.

Принципиальная особенность логарифмических законов компандирования, кроме нелинейности, состоит в том, что компрессии подвергается не все 16-разрядное слово, а лишь часть его.

Стандарт μ-закона допускает компрессирование 14-разрядных (с учетом знака), а стандарт A-закона — 13-разрядных (с учетом знака) данных, что соответствует динамическому диапазону 70 дБ. Поэтому любое положительное значение 16-разрядного числа вне этих границ заменяется числами 7FFCh для μ-закона и 7FF8h для A-закона, а любое отрицательное значение 16-разрядного числа вне этих границ заменяется числами FFFCh для μ-закона и FFF8h для A-закона; ноль в стандарте μ-закона передается чередованием чисел 7Fh/FFh, а в стандарте A-закона чередованием чисел 55h/D5h, что отражено на рис. 8.5 и в табл. 8.2. При экспандировании 8-разрядное слово перекодируется соответственно в 14- или 13-разрядные данные, с которыми должен работать процессор. Форматы компандируемых данных представлены на рис. 8.6, откуда видно, что компандируемые данные размещаются в старших разрядах 16-разрядных слов.

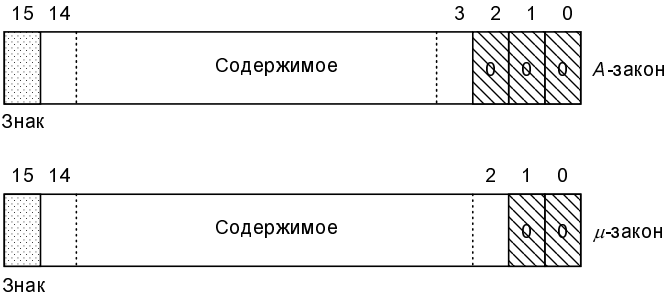


Рис. 8.6. Форматы компандируемых данных

Таблица 8.2. *Законы компандирования*

Законы	Параметры компандирования			
	Длина слова (бит)	Предельно допустимые значения		Передача нуля
		min	max	
Линейный	16	FFFFh	7FFFh	80h
μ-закон	14	FFFCh	7FFCh	7Fh/FFh
A-закон	13	FFF8h	7FF8h	55h/D5h

Законы компандирования в процессорах реализованы по-разному. В процессорах фирмы Motorola они представлены в виде таблиц, помещенных во внутреннюю X-память, поэтому пользователю необходимо составлять специальные программы для обращения к таблицам. В процессорах фирм Texas Instruments и Analog Devices компандирование реализовано аппаратно, что позволяет выбирать закон компандирования для каждого последовательного порта независимо установкой соответствующих битов в регистрах управления портами с помощью команд бит-манипуляций.

8.3. Генератор задержек доступа к памяти

Генератор задержек доступа к памяти предназначен для временного согласования процессора с низкоскоростной и высокоскоростной внешней периферией, включая память. Каждое из внешних устройств может иметь свою длительность цикла обмена. Для высокоскоростной периферии цикл обмена составляет один машинный такт; если цикл обмена превышает это значение, процессору необходимо предоставить некоторое *время ожидания*,

зависящее от скорости обмена конкретного устройства. Генератор задержек (рис. 8.7) формирует сигнал ожидания обменом данных.

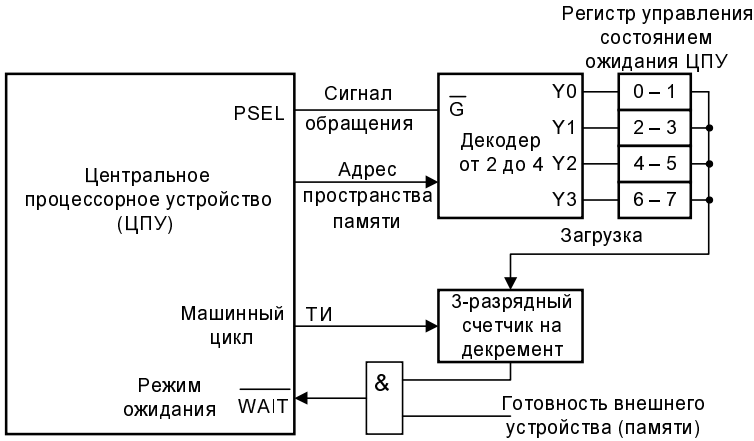


Рис. 8.7. Генератор задержек

Генератор задержек является программируемым и состоит из следующих элементов:

- декодера;
- регистра управления состоянием ожидания ЦПУ;
- счетчика на инкремент;
- схемы "И".

При обращении процессора к внешней периферии из ЦПУ на декодер подается *сигнал обращения* и адрес группы устройств с одинаковым циклом обмена (на рисунке таких групп четыре). Декодер по указанному адресу выбирает соответствующее поле в регистре управления ожиданием ЦПУ. Записанное в это поле на этапе инициализации число загружается в 3-разрядный счетчик, работающий на декремент. Если загруженное содержимое не равно нулю и от внешнего устройства не поступил сигнал готовности, с выхода схемы "И" на ЦПУ подается сигнал ожидания (WAIT) и одновременно запускается счетчик подачей на него ТИ. Состояние ожидания (неготовности) поддерживается до тех пор, пока счетчик не достигнет нуля и от внешнего устройства не поступит сигнал готовности, который проверяется только при достижении счетчиком нуля. Поскольку разрядность счетчика в данном примере равна 3 (для процессоров TMS320C5x), время ожидания может составлять до 7 машинных циклов (ТИ).

Если вся периферия конфигурирована под нулевое состояние счетчика, ТИ отключаются от генератора задержек.

Если по достижении счетчиком нуля сигнал готовности не обнаруживается, счетчик перезагружается и начинает новый счет. Так удается организовать более длительное время ожидания, чем 7 машинных циклов. Например, загрузив в счетчик число 4, можно получить время ожидания в 8 машинных циклов.

Разрядность счетчика и потому длительность цикла ожидания может превосходить 3; например, в процессорах TMS320C54xx разрядность счетчика равна 14, поэтому длительность цикла ожидания может превышать 14 машинных циклов.

8.4. Таймеры

Прерывания и разнообразные временные задержки в процессоре можно организовывать программно, для чего необходимо создавать специальные подпрограммы прерывания, которые отсчитывали бы время и по окончании счета выдавали бы требуемый сигнал прерывания. Программный способ организации задержек и прерываний при своей несложности имеет два существенных недостатка:

- выполнение программы прерывания запрещает параллельную работу процессора по какой-либо другой программе; процессор начинает работать по другим программам лишь по истечении времени задержки, определяемого программой прерывания;
- обеспечение большого времени задержки вынуждает пользователя создавать большие буферы, т. е. уменьшать резерв памяти.

Поскольку необходимость в организации прерываний и временных задержек не является редким событием, программная их реализация приводит к увеличению времени выполнения основной программы. Во избежание этих неприятностей современные процессоры снабжены таймером — программируемым аппаратным внутрикристальным устройством.

Если не углубляться в детали, *таймер* — это внутрикристальный счетчик на вычитание (декремент) или приращение (инкремент), который по окончании счета выдает импульс, называемый сигналом "прерывания по таймеру", или просто прерыванием таймера. Начальное состояние таймера (инициализация) устанавливается программно.

8.4.1. Таймеры на декремент

Наиболее распространенными являются таймеры, работающие на декремент. Таймер декрементируется на единицу один раз за один цикл заданной частоты декремента $f_{\text{дек}}$, которая может составлять половину или четверть тактовой частоты процессора. Как только содержимое счетчика становится равным нулю, таймер вырабатывает импульсный сигнал, воспринимаемый

как запрос на выполнение процессором функций, определенных программистом. К таким функциям могут относиться:

- счет событий;
- организация прерываний центрального процессорного устройства по прошествии времени, задаваемого пользователем при инициализации таймера;
- генерирование периодических прямоугольных импульсов заданной длительности и скважности (рис. 8.8);
- прерывание контроллера DMA для организации передачи данных;
- автоматическая перезагрузка счетчика для продолжения процесса, если это предусмотрено пользователем.

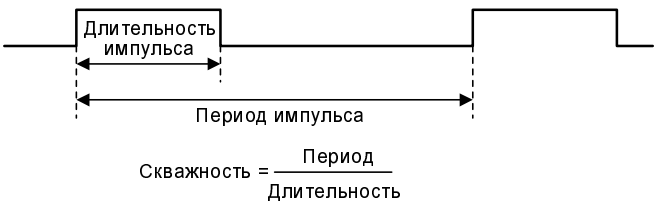


Рис. 8.8. Периодические прямоугольные импульсы таймера

Процессор может содержать один или более таймеров. Их архитектура отличается от семейства к семейству, но тем не менее, они обладают некоторыми основными чертами и элементами, отображенными на обобщенной структурной схеме таймера (рис. 8.9).

Таймер включает в себя три программно доступных регистра и два программно недоступных счетчика:

- регистр управления таймером;
- регистр счетчика;
- регистр коэффициента деления;
- счетчик коэффициента деления;
- счетчик таймера.

Регистры и счетчики располагаются в X-памяти.

Регистр управления таймером обеспечивает:

- останов и перезапуск таймера;
- выбор генератора тактовых импульсов (внутренний или внешний);
- задание типа сигнала прерывания (одно- или биполярный сигнал);
- определение режима работы (вид сигнала на выводе: одиночный, периодический, меандр).

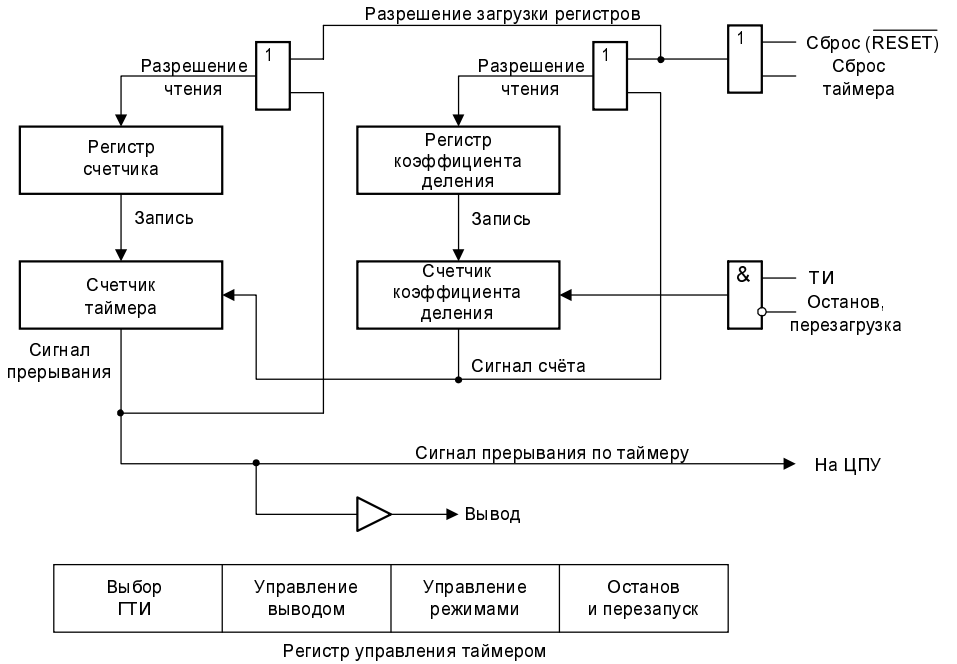


Рис. 8.9. Обобщенная структурная схема таймера на декремент

Назначение и функции остальных регистров и счетчиков будут ясны из дальнейшего. Разрядность регистров и счетчиков определяется конкретной модификацией процессора.

Перед запуском таймера, который может осуществляться как от внешнего сигнала "сброса", так и битами останова и перезапуска регистра управления таймером, необходимо произвести инициализацию (установку начального состояния) таймера, для чего:

- ❒ установить биты регистра управления согласно руководству;
- ❒ загрузить в регистр счетчика и регистр коэффициента деления константы $N_1 - 1$ и $N_2 - 1$ соответственно.

Коэффициенты N_1 и N_2 называются коэффициентами счета и определяют длительность периода, с которым таймер будет выдавать сигнал прерывания.

Рассмотрим работу таймера по рис. 8.9 и 8.10, полагая $N_1 = 4$ и $N_2 = 3$. При инициализации в регистре счетчика должна быть размещена константа 3, а в регистре коэффициента деления — константа 2, поскольку счет при декременте идет "вниз", включая 0.

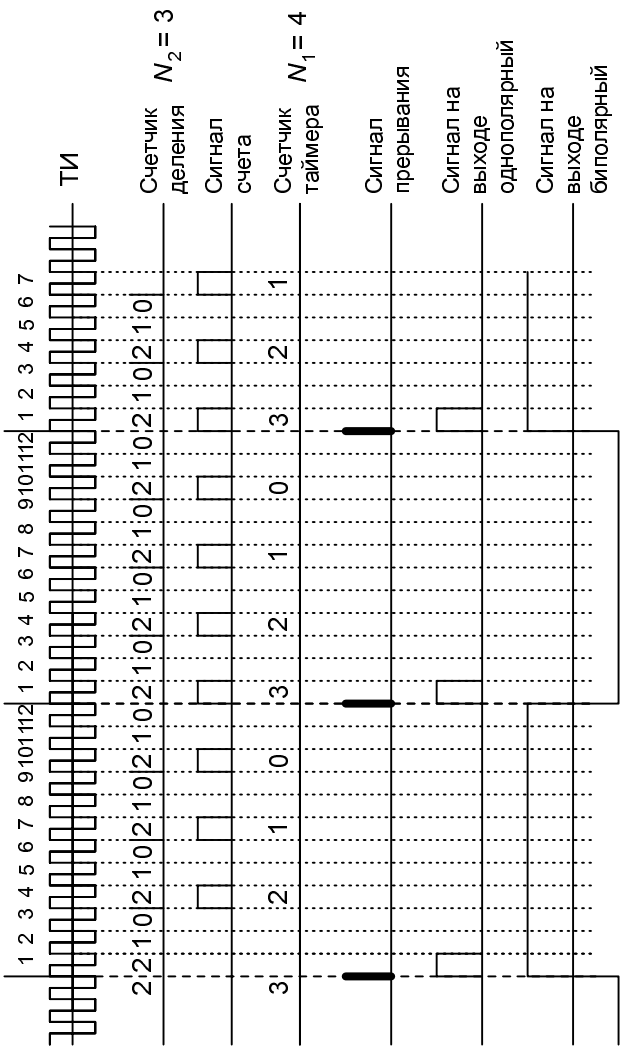


Рис. 8.10. Временная диаграмма работы таймера

При запуске таймера содержимое регистров записывается в соответствующие счетчики. Счетчик коэффициента деления управляется генератором ТИ и би-

том запуска, который остается равным нулю, если не требуется останова. С момента загрузки начинается декрементирование содержимого счетчика деления на каждом такте ТИ; как только его содержимое становится равным нулю, формируется сигнал счета, по которому счетчик коэффициента деления перезагружается из регистра коэффициента деления, а счетчик таймера декрементируется на единицу. Таким образом, для достижения счетчиком таймера 0 потребуется четыре цикла счетчика деления, и коэффициент периода N оказывается равным

$$N = N_1 \cdot N_2 = 3 \cdot 4 = 12,$$

что хорошо видно на рис. 8.10. Это означает, что через каждые N циклов ТИ на ЦПУ будет выдаваться сигнал прерывания таймера, который может служить для реализации перечисленных выше функций.

Сигнал прерывания через усилитель может подаваться на вывод для управления внешней периферией, например, в качестве сигнала дискретизации АЦП.

Важно отметить, что таймер работает параллельно с основной программой, не замедляя ее выполнение.

Подобные таймеры с небольшими отличиями используются в процессорах TMS320C54x и ADSP-21xx; в процессорах фирмы Motorola отсутствуют регистр коэффициента деления и счетчик коэффициента деления, поэтому коэффициент периода определятся только величиной, записанной в счетчик таймера.

8.4.2. Таймеры на инкремент

В перспективных процессорах платформы TMS320C6000 таймеры построены на принципе инкремента (рис. 8.11).

Регистр управления таймером имеет то же назначение, что и в уже описанном варианте таймера. Регистр периода таймера (32-разрядный) предназначен для записи коэффициента счета N циклов источника тактовой частоты; этот коэффициент управляет частотой сигнала прерывания по таймеру, равной $f_{ТИ}/N$. Регистр счетчика таймера (32-разрядный) является инкрементируемым счетчиком от 0 до N. Для инициализации таймера необходимо:

1. Установить таймер в состояние сброса или блокировки (после сброса процессора регистры таймера всегда находятся в состоянии блокировки).
2. Записать требуемую величину N в регистр периода.
3. Определить источник ТИ; он может быть внешним или внутренним. Тактовые импульсы от внешнего источника поступают через контакт "вход"; внутренним источником ТИ являются импульсы, поступающие от ЦПУ с частотой, равной четверти частоты ТИ центрального процессорного устройства.

4. Задать режим работы таймера: импульсный или генератора ТИ.
5. Установить длительность импульса, если работа осуществляется в импульсном режиме.
6. Запустить таймер сигналами "старт" и "снятие блокировки".

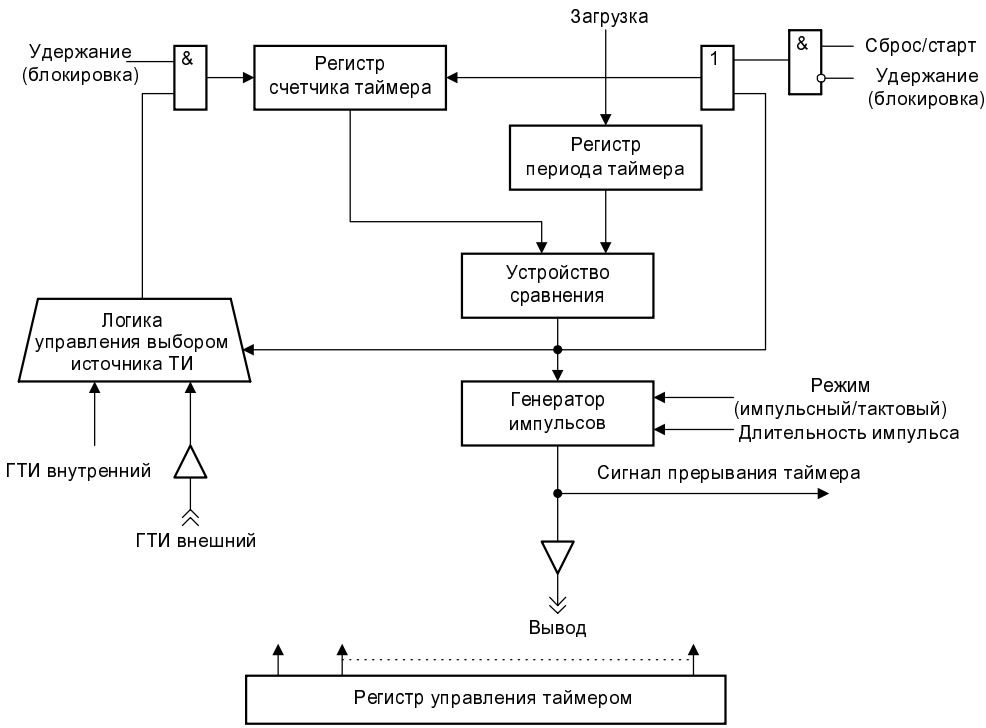


Рис. 8.11. Обобщенная структурная схема таймера на инкремент

8.4.3. Работа таймера

Импульсы тактовой частоты, пройдя логику управления ТИ, через схему "И", на второй вход которой подается сигнал "удержания" ("снятия блокировки") от регистра управления, поступают на находящийся в нулевом состоянии регистр счетчика, который начинает инкрементировать. Как только его содержимое станет равным содержимому регистра периода N, устройство сравнения выдает сигнал, по которому:

1. Регистр счетчика сбрасывается в ноль на очередном тактовом импульсе.
2. Генератор импульсов формирует сигнал прерывания таймера.
3. Начинается новый цикл счета.

Следует заметить, что нулевая комбинация в регистре счетчика является начальным его состоянием, а N — конечным, поэтому, несмотря на то что счетчик считает от 0 до N, его период равен N. Например, пусть в регистр периода записано число 2 и тактовые импульсы следуют с частотой $f_{ТИ} = f_{ЦПУ} / 4$, равной четверти частоты ТИ центрального процессорного устройства. Тогда после старта регистр счетчика будет выдавать последовательность, отображенную на рис. 8.12. Таким образом, на выходе устройства сравнения сигнал прерывания будет формироваться через каждые 8 импульсов, подаваемых из логики управления ТИ.

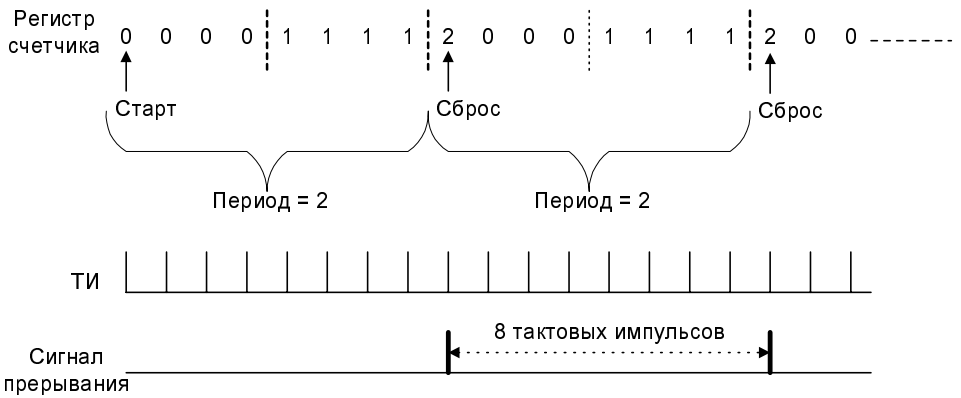


Рис. 8.12. Формирование таймером сигналов прерывания

Импульсный режим

В импульсном режиме можно устанавливать длительность импульса, равную одному или двум периодам ТИ (рис. 8.13). Это производится с целью обеспечения минимальной длительности импульса, когда предусматривается управление со стороны таймера внешним устройством (например, АЦП) через контакт "выход".

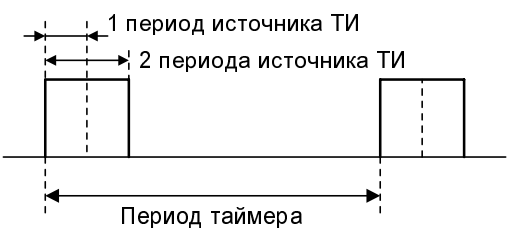


Рис. 8.13. Работа таймера в импульсном режиме

Режим генератора ТИ

В этом случае формируется последовательность импульсов высокого и низкого уровней (рис. 8.14) одинаковой длительности τ (типа "меандр"), равной периоду таймера

$$\tau = \tau_{\text{ТИ}} \cdot N. \tag{8.3}$$

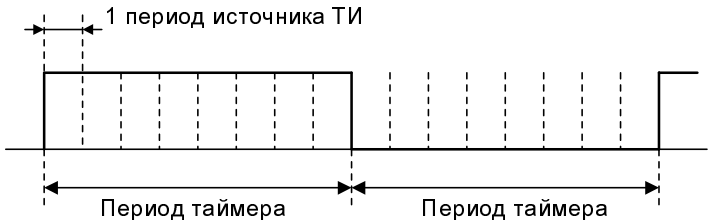


Рис. 8.14. Работа таймера в режиме генератора ТИ

8.5. Синхронизация портов

Под *синхронизацией* понимают *процесс*, обеспечивающий временное согласование работы передающего и принимающего устройств с целью сохранения временных соотношений между элементами передаваемых и принимаемых данных. С точки зрения синхронизации порты разделяют на синхронные и асинхронные.

Синхронными называют порты, в которых длительности всех элементов сигнала строго фиксированы и одинаковы для всех символов (либо находятся в простых кратных соотношениях), а также известен момент начала приема данного элемента. Элементом сигнала в нашем случае является бит, поэтому можно говорить о длительности бита, что часто встречается в описаниях и руководствах. Символами являются логические "1" и "0".

Асинхронными называют порты, в которых данные передаются/принимаются с различной скоростью и момент начала приема конкретного элемента (слова) не определен. Длина элемента (слова) зависит от его конфигурации. Асинхронные порты применяются в асинхронных системах телекоммуникации, где множество источников данных независимо друг от друга и в неопределенное время передают индивидуальные сообщения одному или нескольким получателям. Индивидуальные сообщения источников объединяются в групповой сигнал, который передается по каналу связи и затем разделяется по соответствующим приемникам. Подобные системы называют *разделимыми*. В разделимых асинхронных системах индивидуальные данные могут быть синхронными, но в суммарном групповом сигнале начала элементов индивидуальных сигналов не совпадают. Асинхронные адресные

системы представляют собой разделимые системы, в которых сообщение, передаваемое конкретному адресату (приемнику), характеризуется своей реализацией сигнала.

Наглядным примером асинхронной передачи данных является работа оператора на клавиатуре PC.

Далее отдельно рассматривается синхронизация последовательных синхронных и асинхронных портов.

8.5.1. Генераторы тактовых частот

Синхронизация работы отдельных устройств процессора, включая устройства ввода/вывода, обеспечивается синхронизирующими сигналами (СИ). Синхросигнал (рис. 8.15) представляет собой последовательность прямоугольных импульсов длительностью $\tau_{СИ}$ с частотой следования $f_{СИ}$ (периодом $T_{СИ} = 1/f_{СИ}$). Синхронизация может осуществляться как по возрастающему (переднему), так и падающему (заднему) фронту синхроимпульса.

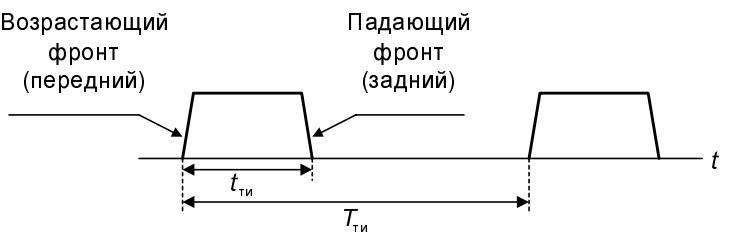


Рис. 8.15. Синхронизирующие (тактовые) импульсы

В процессоре действует система синхросигналов разной частоты, формируемая *генератором тактовых частот* (ГТЧ), или *тактовым генератором*, и синтезатором частот (рис. 8.16). Наивысшая частота синхронизации, вырабатываемая ГТЧ, называется *тактовой* $f_{ТИ}$, а импульсы — *тактовыми импульсами* (ТИ). Период ТИ определяет время, за которое процессор выполняет простейшую операцию *(см. разд. 1.6.1)*. Все остальные частоты являются производными от тактовой частоты. Современные процессоры характеризуются широким диапазоном тактовых частот *(см. разд. 1.7)*, зависящих от назначения процессора, областей его применения и технологии производства. Напомним, что тактовая частота совместно с архитектурой процессора определяет *производительность*, выражаемую в MIPS для процессоров с ФТ и в MFLOPS для процессоров с ПТ *(см. разд. 1.7)*.

Количество MIPS (FLOPS) может как совпадать, так и не совпадать с тактовой частотой (МГц). Если производительность равна тактовой частоте, то говорят, что процессор имеет 1X-генератор; когда тактовая частота в K раз превышает производительность, говорят, что процессор имеет KX-генератор.

Примерами процессоров, имеющих 1X-генератор, является семейство DSP5630x с тактовыми частотами 66/80/100 МГц и производительностью 66/80/100 MIPS соответственно, процессоры TMS320C24x с тактовой частой 20 МГц и производительностью 20 MIPS.

Принцип работы ГТЧ с ФАПЧ

Тактовый генератор (рис. 8.16) строится на основе *фазовой автоподстройки частоты (ФАПЧ)*, именуемой в английской литературе как PLL (Phase-Locked Loop). Задающим генератором может быть как внутренний, так и внешний источник — кварцевый генератор.

Генератор тактовой частоты включает в себя:

- систему ФАПЧ;
- делитель частоты;
- таймер;
- регистр управления и режимов.

Тактовый генератор может работать в одном из двух режимов:

- в режиме ФАПЧ;
- в режиме пониженного потребления мощности;

Режим ФАПЧ

Система ФАПЧ состоит из:

- фазового детектора (ФД);
- генератора, управляемого напряжением (ГУН);
- умножителя частоты.

В системе ФАПЧ используется интегральная зависимость между фазой и частотой колебаний. ФАПЧ устраняет фазовый сдвиг между сигналами задающего генератора (ЗГ) и сигналами тактовой частоты ГТЧ. Область начальных расстроек, в пределах которой ФАПЧ входит в режим автоподстройки, называется *полосой захвата*; процесс вхождения в режим автоподстройки называется *захватом*. Захватив частоту задающего генератора, ФАПЧ удерживает ее с допустимой погрешностью в области частот, называемой *полосой удержания*.

Работой ФАПЧ управляет регистр режимов, в который программно загружаются:

- коэффициент умножения;
- коэффициент деления;
- коэффициент счета таймера.

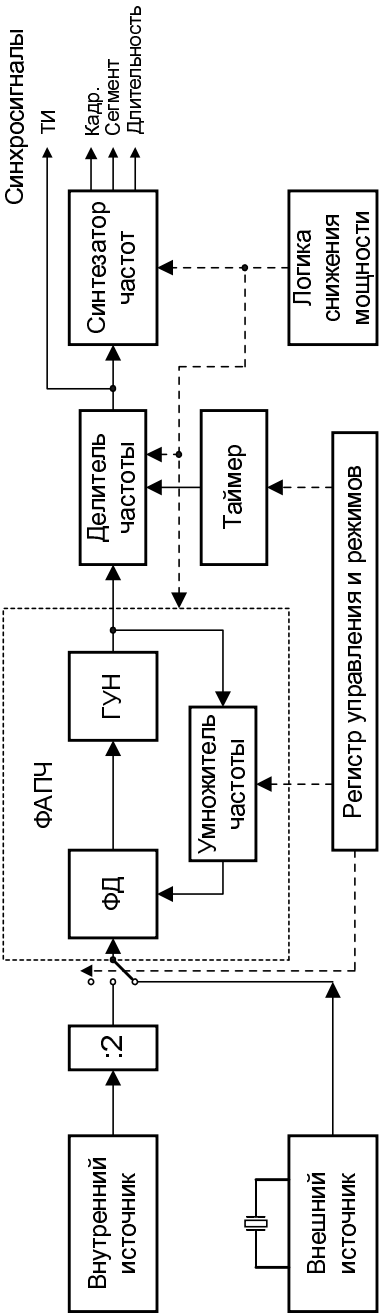


Рис. 8.16. Структурная схема генератора тактовых частот

ФАПЧ обеспечивает формирование высокой тактовой частоты при низкочастотном ЗГ. Например, процессоры DSP56K генерируют тактовую частоту 40 МГц от внешнего кварцевого ЗГ, имеющего частоту 10 кГц. ФАПЧ при использовании низкочастотного ЗГ позволяет:

- сократить электромагнитные помехи от высокочастотных переключений;
- понизить потребляемую процессором мощность;
- использовать практически любой доступный внешний системный ЗГ;
- сократить (вплоть до полного устранения) фазовый сдвиг между внешним источником и ГТЧ;
- программировать тактовую частоту с помощью собственного умножителя частоты.

Сигнал на ФД подается либо от внешнего источника непосредственно, либо от внутреннего источника через делитель на 2. Выбор источника определяется программно в регистре управления и режимов. В этом же регистре устанавливается коэффициент умножения частоты $K_{умн}$, который может быть как меньше 1 (то есть фактически осуществляется деление частоты источника), так и существенно превышать ее (например, в процессоре TMS320C549 коэффициент умножения может принимать значения от 0,25 до 15, а в процессорах фирмы Motorola он может достигать 4096). Следует иметь в виду, что выбор ЗГ зависит от требуемой тактовой частоты $f_{ТИ}$ и определяется расчетом его частоты $f_{ЗГ}$ по формуле

$$f_{ЗГ} = f_{ТИ}/K_{умн}. \quad (8.4)$$

Если $K_{умн} = 1$, тактовая частота и частота ЗГ совпадают.

Фазовый детектор выявляет разность фаз между сигналом источника и сигналом тактовой частоты. Если разность фаз превышает допустимую погрешность, ФД выдает на ГУН сигнал на увеличение (+) или уменьшение (−) фазы, т. е. ФАПЧ переходит в режим захвата, если перед этим произошла рассинхронизация с источником. Если захват частоты источника произошел, то ФАПЧ переходит в режим удержания частоты.

Частота на выходе ФАПЧ определяется соотношением

$$F_{ФАПЧ} = f_{ЗГ} \cdot K_{умн}, \quad (8.5)$$

где $f_{ЗГ}$ — частота ЗГ, $K_{умн}$ — коэффициента умножения. Делитель частоты $F_{ФАПЧ}$ формирует тактовую частоту $f_{ТИ}$

$$f_{ТИ} = F_{ФАПЧ}/K_{дел}. \quad (8.6)$$

Если $K = 1$, тактовая частота и частота ФАПЧ совпадают. Вся сетка частот синхронизации вырабатывается синтезатором частот.

Поскольку делитель частоты находится вне замкнутого контура ФАПЧ, изменения коэффициента деления не приведут к выходу ФАПЧ из режима удержания, что очень важно для обеспечения режимов пониженного потребления мощности.

Режимы пониженного потребления мощности

Бóльшая часть мощности в процессоре потребляется во время переключений цепи из одного логического состояния в другое и при высокой производительности расход энергии может стать препятствием как для использования одиночного процессора, так и для создания многопроцессорных устройств, когда несколько процессоров размещается в небольшом объеме, и требуется организация сложной системы теплоотвода. Особо критичны к потреблению мощности мобильные телекоммуникационные системы, имеющие батарейное питание.

Для снижения потребляемой мощности необходимо, с одной стороны, свести до минимума время активной работы процессора, и, с другой стороны, найти такой режим, в котором при активной работе не превышается допустимое потребление мощности.

Существенная экономия энергии достигается в трех режимах пониженного потребления мощности, часто называемых "спящими режимами". Эти режимы устанавливаются программно и управляются логикой снижения мощности и таймером генератора (см. рис. 8.16):

- *режим 1* — отключается подача СИ на ЦПУ, за исключением логики прерывания ЦПУ; выход из режима осуществляется по внутреннему или внешнему сигналу управления или Reset;
- *режим 2* — прекращается вывод всех СИ устройства процессора, происходит полная остановка кристалла; ФАПЧ остается в рабочем состоянии и удерживает частоту источника; содержимое всех регистров и памяти данных сохраняется; выход из режима только по Reset; работа процессора восстанавливается быстро, поскольку ФАПЧ не отключена;
- *режим 3* — отключается ФАПЧ и прекращается генерирование всех сигналов синхронизации; содержимое всех регистров и памяти данных сохраняется; выход из режима только по Reset; для восстановления работы процессора требуется время с целью перезахвата частоты источника, как при включении.

Таймер представляет собой счетчик на инкремент, работающий на частоте 1/16 частоты источника. Он исполняет роль блокирующего устройства, обеспечивая автоматическую задержку генерирования синхросигналов при переключении процессора из "спящего режима" в режим ФАПЧ. Время задержки зависит от величины коэффициента счета таймера $K_т = 0 \div 255$, загруженного в таймер из регистра режимов, и составляет от 0 до 255×16 периодов частоты источника.

Переключение из режима ФАПЧ в один из "спящих режимов" происходит после коротко переходного процесса, для чего таймер не требуется.

8.5.2. Синхронизация синхронных последовательных портов

В зависимости от временного отрезка, на котором осуществляется синхронизация, выделяют *тактовую* и *кадровую* синхронизации.

Тактовая синхронизация

Тактовая синхронизация (рис. 8.17) является характерным признаком синхронных систем и служит основой для кадровой синхронизации. Она осуществляется на временном отрезке, равном длительности одного бита (элемента), продолжительность которого, в свою очередь, равна одному периоду тактового импульса ТИ.

Цель тактовой синхронизации состоит в *определении момента начала приема элементарной посылки (бита)*. Обычно тактовая синхронизация осуществляется на возрастающем фронте ТИ (рис. 8.17, *а*). Если сигналы тактовой синхронизации ТИ для передачи и приема не разделены, синхронизация передачи осуществляется по возрастающему фронту, а приема — по падающему (рис. 8.17, *б*). Поскольку длительность фронта мала, в дальнейшем фронт тактового импульса будем изображать вертикальной линией.

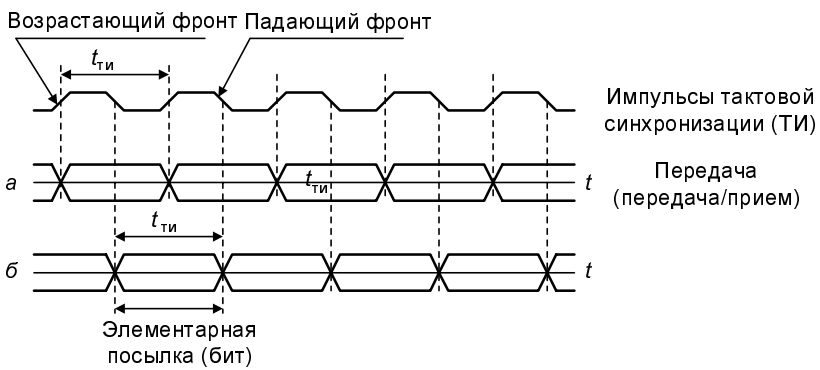


Рис. 8.17. Тактовая синхронизация: *а* — на возрастающем фронте ТИ, *б* — на падающем фронте ТИ

Кадровая синхронизация

Кадром называется кодовая комбинация заданной разрядности. Пример кадра был приведен при рассмотрении вокодеров *(см. главу 1)*. Длительность кадра $t_к$ определяется количеством битов N и длительностью каждого бита $t_{бит}$

$$t_к = N \cdot t_{бит}. \qquad (8.7)$$

Простейшим примером кадра является 8-разрядное слово (байт). Синхронизация, осуществляемая на временном отрезке, равном длительности кадра, называется *кадровой*.

Последовательные синхронные порты поддерживают разнообразную длину слов передаваемых данных, а потому и разнообразную длительность кадров, причем в общем случае длина слова данных и длина кадра не совпадают. Наиболее часто встречаются 8- и 16-разрядные слова, но большинство процессоров поддерживает и другие длины слов (табл. 8.3).

Таблица 8.3. *Слова данных, поддерживаемые синхронными портами*

Процессор	TMS320C				DSP56K	ADSP-21xx
	2000	24x	54x	5xxx, 6xxx		
Длина слова (бит)	8, 16	от 1 до 8	8, 10, 12, 16	8, 12, 16, 20, 24, 32	8, 12, 16, 24	от 3 до 16

Восьмиразрядные слова данных обычно применяются в телефонных системах, 16-разрядные слова — в цифровых стереоаудиосистемах (по одному байту на каждый из стереоканалов). В дальнейшем для удобства изложения используются кадры, соответствующие 8-разрядному слову, если не рассматривается другой вариант.

Цель кадровой синхронизации состоит в *определении первого элемента кадра*, для чего формируется сигнал кадровой синхронизации (СКС), который может иметь длительность одного элемента, или бита (рис. 8.18, *а*), либо охватывать весь синхронизируемый кадр (рис. 8.18, *б*). В первом случае СКС может предшествовать первому (то есть нулевому) элементу (биту) кадра, совпадая с последним элементом (битом) предыдущего кадра (это показано на рис. 8.18, *а* сплошными линиями) или передаваться одновременно с первым элементом кадра (это показано на рис. 8.18, *а* пунктирными линиями, поскольку такой вариант используется редко). Во втором случае (см. рис. 8.18, *б*) сигнал кадровой синхронизации имеет длительность одного кадра.

Как уже было сказано и как видно из табл. 8.3, кадр может состоять из нескольких слов. Простейший пример такого варианта изображен на рис. 8.19, где кадр состоит из двух 8-разрядных слов. В этом случае внутри одного кадра каждому слову отводится свое место, которое называется временным сегментом, или просто *сегментом*. Кадровая синхронизация осуществляется по первому сегменту и сочетается с внутрикадровой синхронизацией.

Все сегменты внутри кадра передаются без перерыва, в то время как между кадрами могут быть паузы. Пауза между соседними кадрами называется *интервалом неактивности* (см. рис. 8.18, *б*). Чем меньше интервал неактивности, тем чаще передаются кадры, в связи с чем вводится понятие кадровой

частоты $f_{\text{кадр}}$, которая определяется периодом между сигналами кадровой синхронизации

$$f_{\text{кадр}} = \frac{f_{\text{ТИ}}}{K_{\text{ТИ}}}, \tag{8.8}$$

где $f_{\text{ТИ}}$ — частота тактовой синхронизации; $K_{\text{ТИ}}$ — количество ТИ между сигналами кадровой синхронизации.

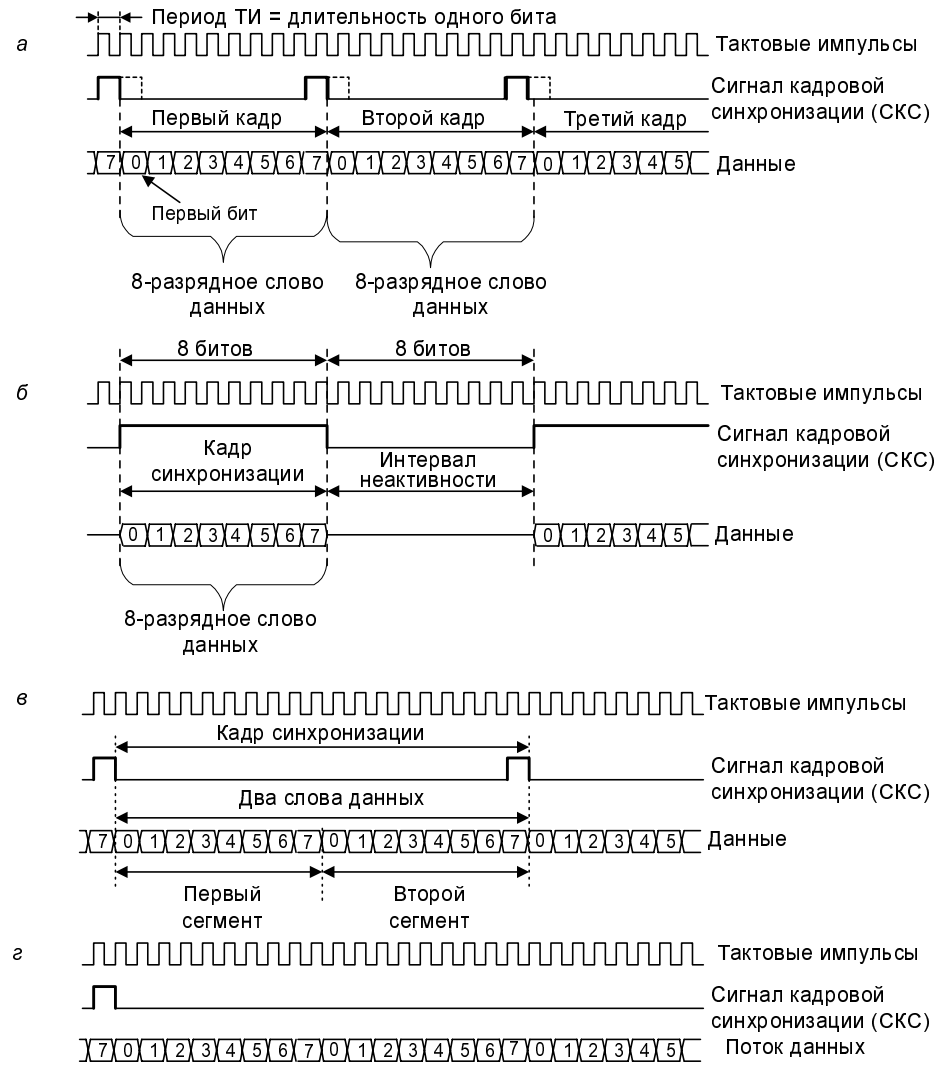


Рис. 8.18. Кадровая синхронизация

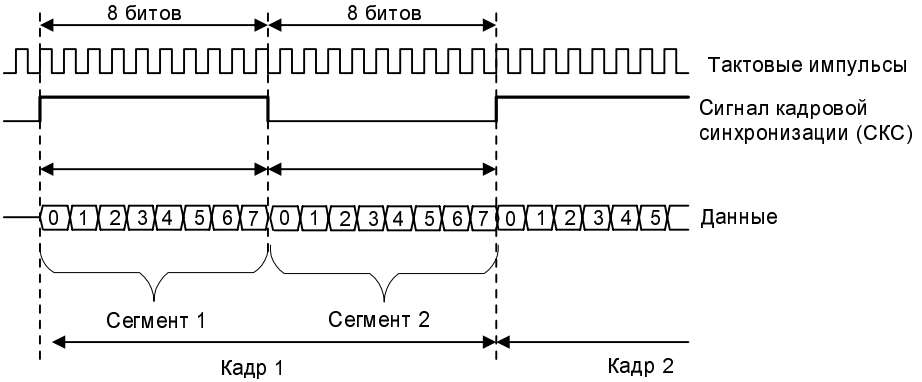


Рис. 8.19. Структура кадра

Кадровая частота может быть увеличена за счет сокращения интервала неактивности — при этом уменьшаются паузы между сигналами кадровой синхронизации. Наконец, когда интервал неактивности становится равным нулю, формируется *непрерывный поток данных*, в котором сигнал кадровой синхронизации перекрывает последний бит предыдущего кадра. В этом случае достигается максимальная кадровая частота

$$\max(f_{\text{кадр}}) = \frac{f_{\text{ти}}}{K_{\text{кадр}}}, \tag{8.9}$$

где $K_{\text{кадр}}$ — количество битов в одном кадре.

Передача кадров в виде непрерывного потока данных называется *непрерывным режимом*. В непрерывном режиме СКС генерируется следом за первой загрузкой регистра передаваемых данных (рис. 8.23) и в дальнейшем не генерируется, если не происходит каких-либо сбоев и перехода процессора в состояние останова. По завершении состояния останова процессор начинает работать в непрерывном режиме, и генерируется единственный сигнал кадровой синхронизации (рис. 8.18, *г*). Следовательно, принципиальное различие между работой процессора на максимальной кадровой частоте и непрерывным режимом состоит в том, что в первом случае (рис. 8.18, *а*, *в*) сигнал кадровой синхронизации формируется на каждом кадре, а во втором — передается один раз только в начале передачи.

Существуют и другие, более сложные варианты кадровой синхронизации; некоторые из них будут показаны при рассмотрении синхронных последовательных портов.

Обязательность тактовой и кадровой синхронизации приводит к необходимости строить последовательный интерфейс в общем случае из пяти линий (рис. 8.20):

- тактовой синхронизации (тактовых импульсов);
- приема данных;

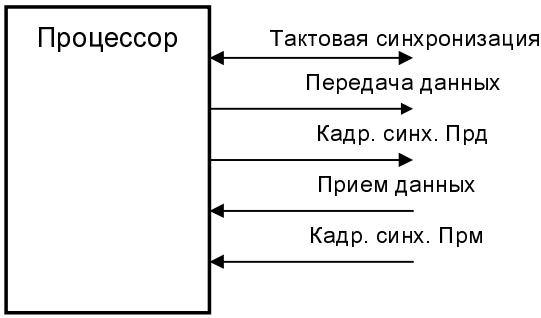


Рис. 8.20. Последовательный синхронный интерфейс

- кадровой синхронизации приема;
- передачи данных;
- кадровой синхронизации передачи.

8.5.3. Синхронизация асинхронных портов

Асинхронная передача/прием, как было отмечено, используется в многоадресных системах, когда момент поступления данных неизвестен. Кроме того, в подобных системах необходимо различать данные и адрес приемника, которому предназначены передаваемые данные, а также учесть их большую подверженность влиянию импульсных помех. Три этих фактора существенно усложняют процедуру временного согласования обмена данными. Тем не менее, выход найден в *особом построении передаваемого слова* таким образом, чтобы из его структуры можно было однозначно выделить слово данных. Это означает, что передаваемое слово должно отличаться от 8-разрядного слова данных, используемого в синхронных портах, введением дополнительных, служебных битов, которые охватывали бы слово данных и предъявляли его приемнику. Типичным представителем процессоров, содержащих асинхронные порты, является семейство DSP56K, на примере которого и рассмотрим принцип организации асинхронной работы.

Передаваемое слово (рис. 8.21) содержит 8-разрядное слово сообщения (данные или адрес) и служебные биты, назначение которых ясно из их наименования:

- старта (начало передаваемого слова);
- стопа (окончания передаваемого слова);
- проверки на четность/нечетность (защита от помех);
- типа сообщения (адрес или данные).

Все передаваемые *слова обязательно содержат биты старта и стопа*, располагающиеся в начале и конце слова соответственно. Возможны три формата асинхронных слов (см. рис. 8.21):

- десятиразрядное слово (рис. 8.21, *а*);
- одиннадцатиразрядное слово, содержащее бит "проверки четности/нечетности" (рис. 8.21, *б*);
- одиннадцатиразрядное слово, содержащее бит "типа сообщения" (рис. 8.21, *в*).

Во всех случаях состояние бита "старт" всегда равно нулю, а бита "стоп" — единице. Эти биты позволяют обнаружить ошибку кадровой синхронизации, поскольку асинхронный кадр состоит только из одного слова.

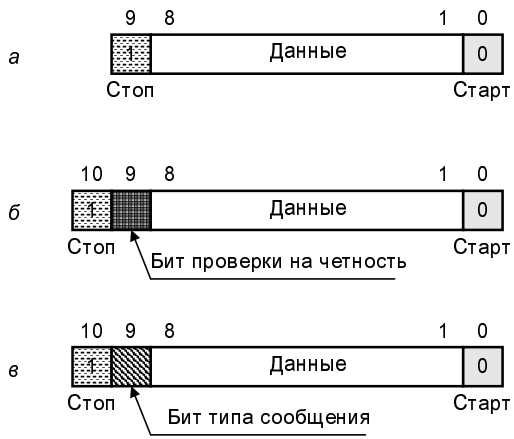


Рис. 8.21. Форматы асинхронных слов

Обмен данными только между двумя асинхронными портами не требует передачи адреса, поэтому используется формат (*а*) или формат (*б*), причем формат (*б*) применяется на линиях, подверженных импульсным помехам.

Бит проверки "четности/нечетности", или просто проверки на четность, позволяет обнаружить ошибку в принимаемых данных.

Обнаружение ошибки осуществляется с помощью *кода Вагнера* (кода проверки на четность), комбинация которого образуется введением дополнительного проверочного символа "1" или "0" в зависимости от того, каков *вес* кодовой комбинации данных. *Весом W комбинации* называется количество символов со значением "1". Если вес четный, проверочный символ принимает значение "0"; если вес нечетный, проверочный символ принимает значение "1". Таким образом, вес любой комбинации кода Вагнера всегда четен. Например, 8-разразрядная комбинация данных

10101100

имеет вес $W = 4$, поэтому проверочный — девятый — символ устанавливается в 0, и комбинация кода Вагнера принимает вид

010101100.

Комбинация данных

10101101

имеет вес $W = 5$, поэтому проверочный символ установится в "1", а комбинация кода Вагнера примет вид

110101101,

т. е. ее вес равен 6.

Эти примеры показывают, что значение проверочного символа равно весу комбинации данных по модулю 2

$$b_8 = W = \left(\sum_{i=0}^{7} b_i \right) \bmod 2. \qquad (8.10)$$

Такой код не способен исправлять ошибки, а может их только обнаруживать. Более того, код Вагнера обнаруживает ошибки лишь нечетной кратности.

Другой вариант кода Вагнера настраивается на нечетный вес комбинации данных: проверочный символ равен "0", если вес нечетный, и равен "1", если вес четный. В этом варианте комбинации кода Вагнера преобразуются в **1**10101100 для первого примера и в **0**10101101 для второго, т. е. они всегда имеют нечетный вес.

Замечание

При обнаружении ошибок кадровой синхронизации или ошибочно принятой комбинации процессор будет обслуживать прерывание приема данных согласно заложенной подпрограмме, адрес которой размещается в таблице векторов прерываний памяти программ.

Многоадресная система позволяет обеспечивать обмен данными между любой парой или группой процессоров, объединенных в систему. Все процессоры подключаются к одному проводу. Протокол обмена данными предусматривает (рис. 8.22):

- организацию соединения по принципу "ведущий-ведомый";
- запрещение возбуждения линии более чем одним процессором; сообщение передается одним из процессоров, в это время другие процессоры могут анализировать принимаемое сообщение: каждый процессор отзывается только на свой адрес (индивидуальный или групповой);
- кадр представляет собой 11-разрядное слово формата (см. рис. 8.21, *б* или рис. 8.21, *в*); установка бита "типа сообщения" в "1" означает, что слово содержит адрес, в противном случае — данные;

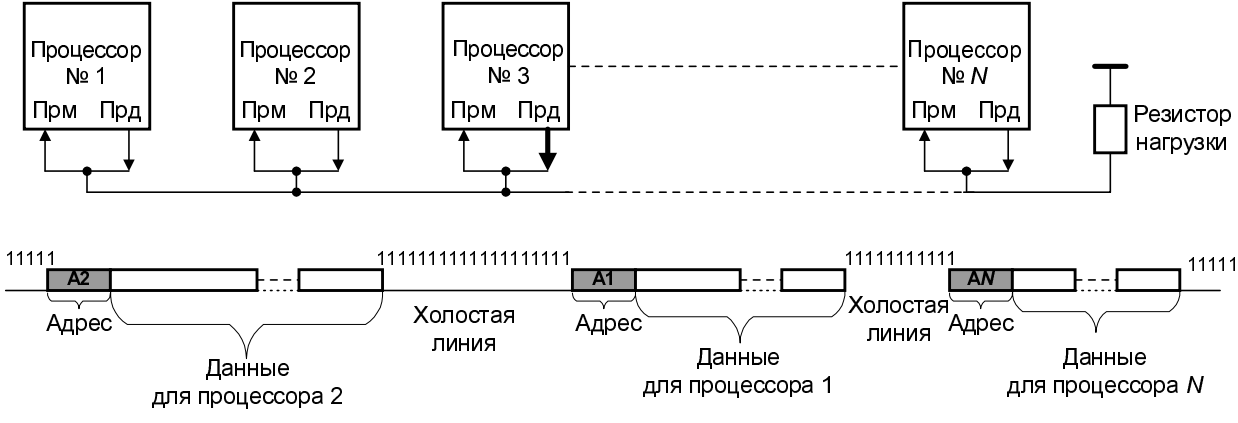


Рис. 8.22. Асинхронный обмен данными в многоадресной системе; данные передает процессор 3

- в промежутке между сообщениями устанавливается режим холостой линии, который поддерживается подачей на нее не менее 11 единиц и при обнаружении которого можно передавать сообщение; этот режим завершается прерыванием холостой линии.

8.6. Синхронные последовательные порты

Существуют два класса последовательных портов: *синхронные* и *асинхронные*. Главное отличие между ними, как было сказано в *разд. 8.4*, состоит в организации временного согласования между передатчиком и приемником. В остальном их структуры похожи, поэтому в дальнейшем рассматриваются только синхронные порты.

Абсолютное большинство ЦПОС снабжены одним или несколькими последовательными портами, которые отличаются способами передачи и приема данных. В связи с этим выделяют:

- базовый последовательный порт;
- буферизированные порты;
- многоканальные буферизированные порты;
- многоканальные порты с временным разделением каналов.

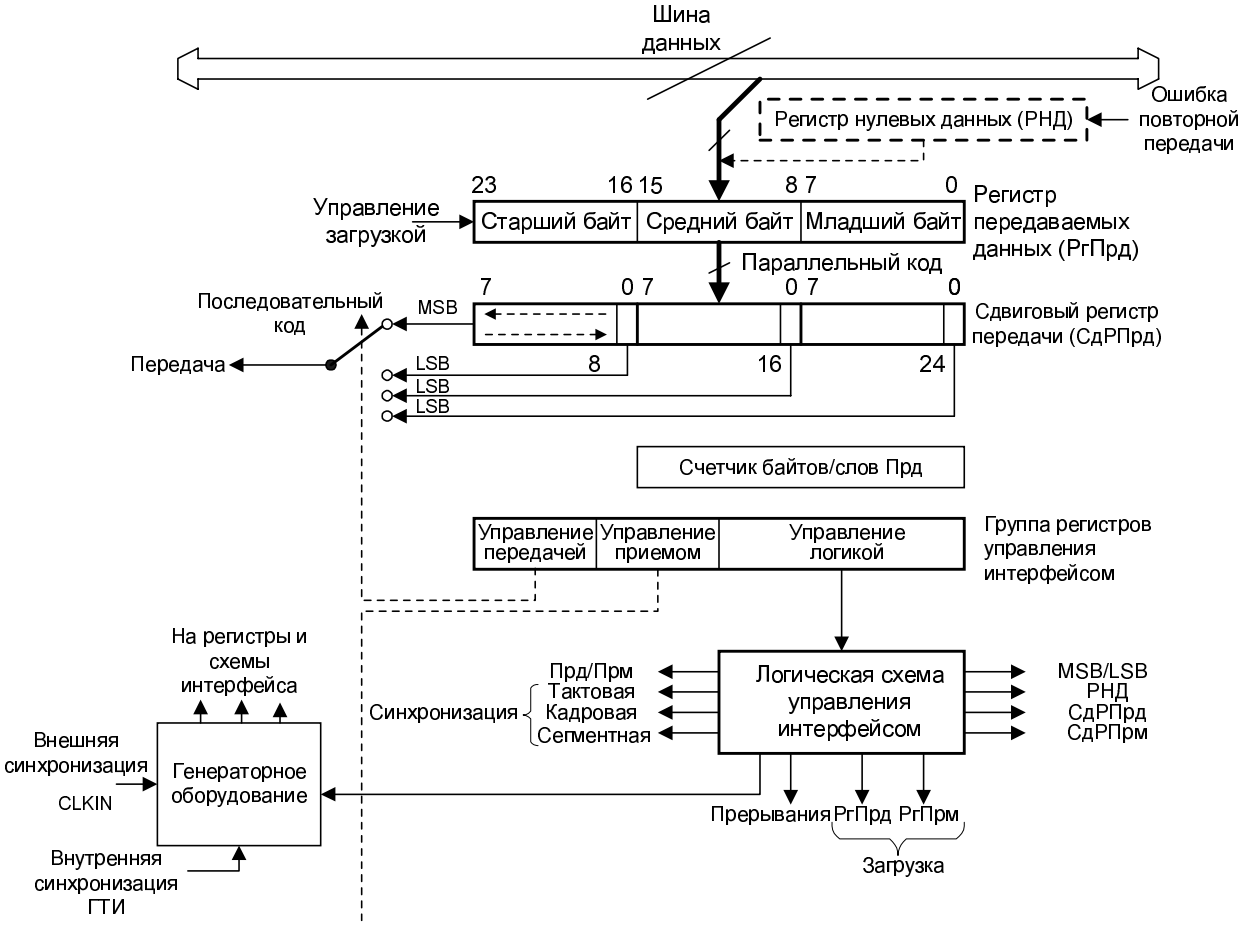
Все эти порты представляют собой модификации *базового порта* с разнообразными усложнениями с целью расширения возможностей процессора и увеличения его быстродействия. Поэтому сначала рассмотрим базовый синхронный порт.

8.6.1. Базовый синхронный порт

В состав базового синхронного порта (рис. 8.23) входят:

- регистр передаваемых данных (РгПрд);
- сдвиговый регистр передачи (СдРПрд);
- регистр принимаемых данных (РгПрм);
- сдвиговый регистр приема (СдРПрм);
- группа регистров управлением интерфейсом;
- логическая схема управления интерфейсом;
- счетчики байтов/слов передачи и приема;
- генераторное оборудование.

Интерфейс обеспечивает дуплексный обмен данными с различными устройствами по последовательному каналу, поэтому на передаче осуществляется преобразование параллельного кода в последовательный, а на приеме — последовательного в параллельный.


Шина данных
Регистр нулевых данных (РНД)
Ошибка повторной передачи
23
16 15
8 7
0
Управление загрузкой
Старший байт
Средний байт
Младший байт
Регистр передаваемых данных (РгПрд)
Параллельный код
7
0 7
0 7
0
Последовательный код
MSB
Передача
Сдвиговый регистр передачи (СдРПрд)
LSB
8
16
24
LSB
LSB
Счетчик байтов/слов Прд
Управление передачей
Управление приемом
Управление логикой
Группа регистров управления интерфейсом
На регистры и схемы интерфейса
Прд/Прм
Тактовая
Кадровая
Сегментная
Синхронизация
Логическая схема управления интерфейсом
MSB/LSB РНД
СдРПрд
СдРПрм
Внешняя синхронизация
CLKIN
Генераторное оборудование
Внутренняя синхронизация ГТИ
Прерывания
РгПрд
РгПрм
Загрузка

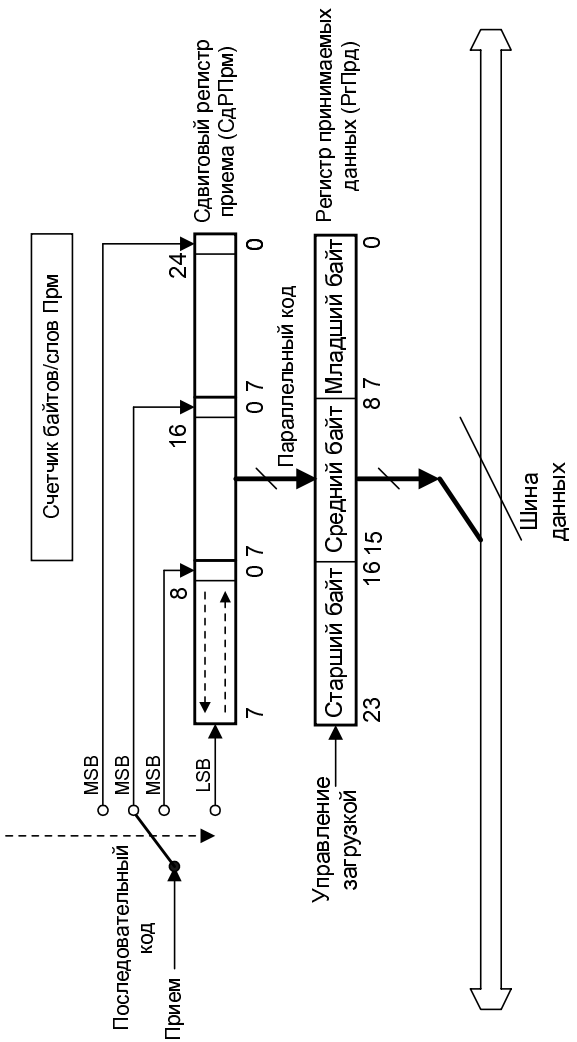


Рис. 8.23. Базовый последовательный синхронный интерфейс

Передающая часть (передатчик) и приемная часть (приемник) в большинстве процессоров полностью разделены и независимы; исключение составляют процессоры фирмы Motorola, в которых, как будет показано, передатчик и приемник имеют общие устройства, работающие в режиме разделения времени между ними. Интерфейс может работать с различными форматами передаваемых/принимаемых данных; формат определяется программно в одном из регистров управления интерфейсом либо при инициализации процессора, либо в ходе выполнения программы с помощью команд бит-манипуляции. На рис. 8.23 указаны три возможных формата: один байт, два байта и три байта. В общем случае формат данных на передаче и приеме может быть различным (что и показано на рисунке), поскольку в абсолютном большинстве процессоров приемная и передающая части интерфейса работают автономно. Рассмотрим отдельно работу передатчика и приемника.

Передача данных

Данные, поступающие из ЦПУ через шину данных, записываются в регистр передаваемых данных РгПрд. Длина передаваемого слова (формат данных в канале) определяется возможностями подключенной к процессору внешней периферии и задается установкой соответствующих битов в регистре управления интерфейсом.

Цикл передачи начинается с момента записи данных в РгПрд, в этот же момент формируется сигнал кадровой синхронизации. Данные из РгПрд в параллельном коде пересылаются в сдвиговый регистр передачи СдРПрд, если он к этому моменту пуст, т. е. если предыдущее слово передано полностью. Из СдРПрд слово в последовательном коде поступает на вывод передачи. Содержимое СдПРД может передаваться в одном из двух направлений:

- начиная со старшего бита (MSB);
- начиная с младшего бита (LSB).

На рис. 8.23 показан вариант MSB, что соответствует верхней стрелке, помещенной внутри сдвигового регистра; вариант LSB соответствует нижней стрелке. Загрузкой регистров управляет логическая схема так, что очередное слово передачи записывается в РгПрд сразу после передачи первого бита содержимого сдвигового регистра.

При передаче возможно состояние процессора, когда уже переданные данные будут отправляться повторно. Это произойдет в том случае, когда СдРПрд пуст, а в РгПрд данные еще не поступили. Такое состояние передатчика называют *ошибкой повторной передачи*, на которую реагирует регистр управления и посылает на логическую схему сигнал ошибки. Логическая схема принудительно из регистра нулевых данных (РНД) записывает в РгПрд нулевую комбинацию, которая и будет передана.

Прием данных

Данные в последовательном коде согласно направлению передачи (начиная с MSB или с LSB) поступают в сдвиговый регистр приема (СдРПрм) и после его заполнения автоматически в параллельном коде пересылаются в регистр принимаемых данных РгПрм.

Важно обратить внимание на следующее: если данные передавались, начиная с младшего бита, то и прием должен осуществляться также с младшего бита, т. е. сдвиг принимаемых данных необходимо производить от старшего бита в сторону младшего, чтобы приходящий последним бит данных оказался на своем месте — в 23-м разряде сдвигового регистра.

Принятое слово из СдРПрм в параллельном коде поступает на регистр принимаемых данных, а сдвиговый регистр обнуляется. Возможна ситуация, когда СдРПрм заполнен и готов к пересылке данных в РгПрм, который в этот момент оказывается занятым данными предыдущего приема. Такое состояние, называемое *ошибкой переполнения приемника*, фиксируется регистром управления интерфейсом, под воздействием которого логическая схема формирует запрос на прерывание приемника, и данные из СдРПрм не пересылаются в РгПрм, пока не освободится регистр приема данных.

Базовый синхронный последовательный интерфейс обладает независимыми каналами приема и передачи и снабжен *двойной буферизацией*, т. е. на любом канале есть своя пара регистров — сдвиговый и данных, причем синхронизация по каждому каналу может отличаться и быть как внутренней, так и внешней. Двойная буферизация позволяет поддерживать *непрерывность потока данных*: данные могут передаваться в/из регистров приема/передачи в то время, как выполняется передача/прием.

Процессоры DSP56K, TMS320C24x, TMS320C5000 имеют по одному подобному порту, а процессоры ADSP-21xx — до двух портов, которые называются SPORT0 и SPORT1 и имеют следующие приоритеты прерываний:

- наибольший приоритет SPORT0 — передача;
- SPORT0 — прием;
- SPORT1 — передача;
- наименьший приоритет SPORT1 — прием.

В семействе DSP56xx кроме рассмотренного интерфейса существует интерфейс последовательной передачи, который имеет следующие особенности (рис. 8.24):

- приемник и передатчик не разделены; одни и те же устройства обеспечивают прием/передачу с разделением по времени (мультиплексирование);
- сдвиговый регистр приема/передачи является 8-разрядным, поэтому прием и передача данных осуществляется побайтно;
- между регистром принимаемых/передаваемых данных и сдвиговым регистром размещены регистры упаковки/распаковки 24-разрядных данных.

При передаче данные распаковываются на байты в трех регистрах распаковки/упаковки, откуда байты в параллельном коде поочередно пересылаются в сдвиговый регистр. Возможны два варианта последовательной передачи/приема байтов:

старший — средний — младший;

младший — средний — старший

и два варианта направлений передачи битов — от старшего MSB или от младшего LSB, которые описаны выше.

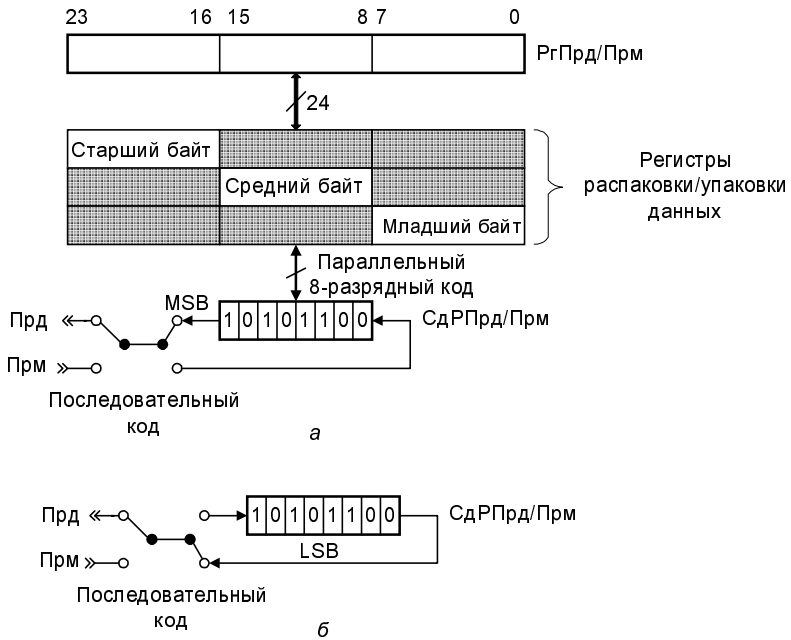


Рис. 8.24. Распаковка/упаковка данных в процессорах; *а* — передача и прием начинаются со старшего бита; *б* — передача и прием начинаются с младшего бита

Прием данных осуществляется на тот же сдвиговый регистр, с которого в параллельном коде данные размещаются на своих местах в тех же регистрах распаковки/упаковки, которые теперь служат для упаковки. После приема последнего байта данные в параллельном коде пересылаются на соответствующие места регистра приема/передачи, где упаковываются (объединяются) в полное 24-разрядное слово.

В новейших и перспективных процессорах применяются и другие модификации базового последовательного синхронного интерфейса, которые рассматриваются ниже.

Интерфейс с компандированием

Интерфейс с компандированием (рис. 8.25) по μ- и A-законам отличается от базового последовательного синхронного интерфейса (см. рис. 8.23) включением между регистрами РгПрд и СдРПрд компрессора, а между регистрами РгПрм и СдРПрм — экспандера. Однако разрядность регистров передаваемых и принимаемых данных обычно существенно превышает разрядность компрессированных данных, поэтому существует задача размещения компрессируемых данных в РгПрд и экспандируемых данных в РгПрм с целью согласования их форматов с форматами данных в ЦПУ. Рассмотрим на примере процессоров TMS320C6201/C6701, каким образом размещаются данные в 32-разрядных регистрах.

На передаче компрессируемое 16-разрядное слово представляет собой младшее слово 32-разрядного регистра (рис. 8.26), которое выравнивается слева до 14 или 13 разрядов с сохранением знака, как на рис. 8.6. С выхода компрессора 8-разрядные данные в параллельном коде записываются в сдвиговый регистр передачи.

На приеме согласование формата данных происходит несколько сложнее. Принятые 8-разрядные компрессированные данные записываются в 16-разрадный буферный РгПрм и после расширения в экспандере записываются в 32-разрядный регистр выравнивания. Процедура выравнивания состоит из двух этапов: сначала формируется 16-разрядное слово согласно закону компандирования, затем полученное слово размещается в 32-разрядном регистре РгПрм. Возможны три варианта выравнивания в регистре РгПрм (рис. 8.27):

- ❒ слово занимает младшие разряды, старшие разряды обнуляются;
- ❒ слово занимает младшие разряды, старшие разряды заполняются знаком (расширение знака);
- ❒ слово занимает старшие разряды, младшие разряды обнуляются.

Вариант выравнивания выбирается пользователем установкой соответствующих битов в регистре управления последовательным портом.

Замечание

При отсутствии аппаратной реализации компандирования описанные варианты выравнивания задаются программой пользователя.

Компандер можно применить для исследования эффектов компандирования внутренних данных при отладке, для чего сам последовательный интерфейс не используется для обмена данными и отключается от выводов Прд/Прм. В этом случае, разумеется, на передаче и приеме устанавливается одинаковый закон компандирования. Компандирование внутренних данных возможно двумя методами, путь данных для которых показан на рис. 8.25 штриховой стрелкой.

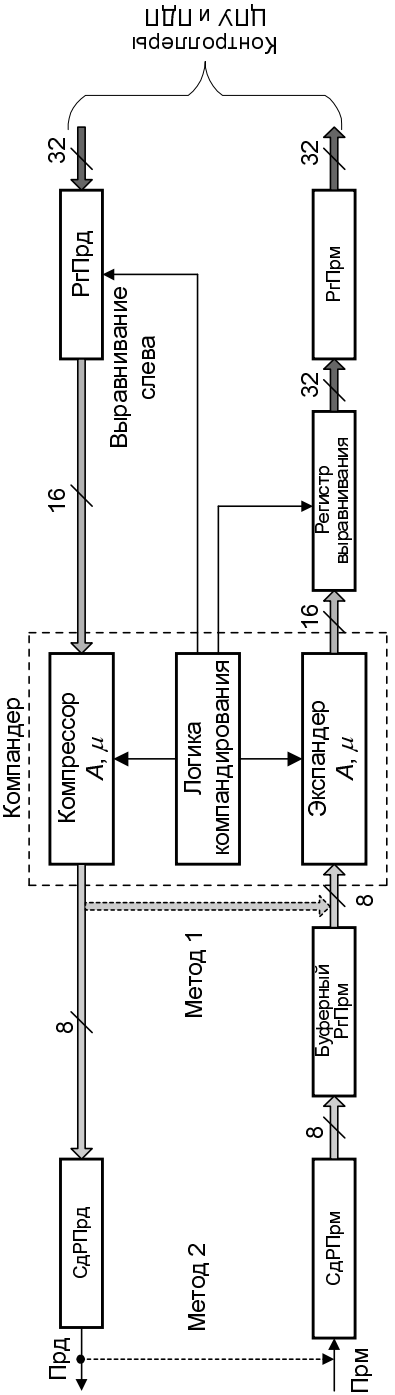


Рис. 8.25. Интерфейс с компандированием

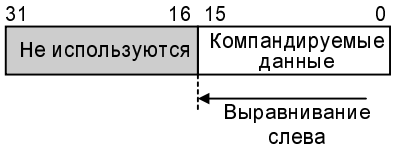


Рис. 8.26. Формат данных в РгПрд

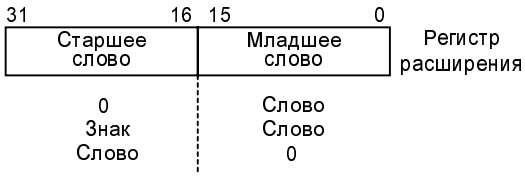


Рис. 8.27. Варианты выравнивания данных на приеме

- *Метод 1 — без прохождения сдвиговых регистров.* Регистры принимаемых и передаваемых данных соединяются с помощью логики компандирования. Передаваемые данные компрессируются, а затем экспандируются. Преимущество данного метода состоит в его быстроте. Недостаток состоит в отсутствии синхронизации с контроллерами ЦПУ и DMA, что препятствует организации потока данных.
- *Метод 2 — с прохождением сдвиговых регистров.* Прерывания приема и передачи или события синхронизации позволяют синхронизировать контроллеры ЦПУ и ПДП с рассматриваемыми преобразованиями и организовывать потоки данных. Недостаток указанного метода в том, что время компандирования зависит от выбранной скорости передачи данных.

8.6.2. Буферизированный последовательный порт

Архитектура современных процессоров нацелена на достижение максимально возможной скорости обработки данных. До недавнего времени серьезным препятствием повышению быстродействия была организация обмена данными между собственно последовательным портом и внутренней памятью процессора. Этот обмен выполнялся и во многих ЦПОС продолжает выполняться через ЦПУ (через его входные регистры), что существенно замедляет работу самого ЦПУ, который вынужден расходовать свой временной ресурс на операции обмена (пересылок). Во избежание потери временного ресурса ЦПУ новейшие модификации процессоров (ADSP-21xx, TMS320C54x и др.) снабжены буферизированными последовательными портами.

Буферизированный последовательный порт предназначен для непосредственного обмена данными в виде 8-, 10-, 12- и 16-разрядных слов между внутренней памятью процессора и внешней последовательной периферией (кодеки, последовательные АЦП и т. п.), минуя ЦПУ, с возможностью работы в режиме непрерывного потока данных.

Буферизированный порт (рис. 8.28) состоит из собственного модуля последовательного порта и блока автобуферизации.

Модуль буферизированного порта (МБП) предназначен для организации сопряжения с внешней периферией, управления блоком автобуферизации (БАБ), формирования необходимых сигналов прерывания для ЦПУ и содержит:

- устройство передачи, включающее регистр передачи (БРгПрд), сдвиговый регистр передачи (БСдРПрд), систему тактовой и кадровой синхронизации;
- устройство приема, включающее сдвиговый регистр приема (БСдРПрд), регистр приема (БРгПрд), систему тактовой и кадровой синхронизации;
- регистр управления портом (РгУП);
- регистр управления блоком автобуферизации (РгУБАБ);
- логику управления портом;
- устройство управления прерываниями.

Устройства передачи и приема и их регистры подобны соответствующим устройствам базового последовательного порта и выполняют те же функции; сдвиговые регистры программно недоступны. Тактовая и кадровая синхронизация передачи и приема независима и может быть как внешней, так и внутренней.

Блок автобуферизации (БАБ) предназначен для пересылки данных между последовательным портом и внутренней памятью процессора без вмешательства ЦПУ. В состав блока входят размещенные в памяти регистры:

- адреса передачи (РгАПрд);
- размера буфера передачи (РгБПрд);
- адреса приема (РгАПрм);
- размера буфера приема (РгБПрм);
- регистр управления автобуферизацией (РгУАБ) в составе устройства управления БАБ.

Регистры адресов и размеров буфера являются 11-разрядными, а регистр управления — 16-разрядным.

Блок автобуферизации управляет доступом к буферам передачи/приема через интерфейс внутренней памяти и работой ЦПУ с помощью сигналов прерываний. Регистры адресов обеспечивают циклическую адресацию буферов передачи и приема, причем каждый из них имеет свой блок генерации адреса. Буферы передачи и приема размещаются в специально отводимой

области внутренней памяти (согласно карте памяти). Типовой объем этой области составляет $2K$ слов. При необходимости эта область может использоваться как память общего назначения, но она является единственной, где возможна автобуферизация.

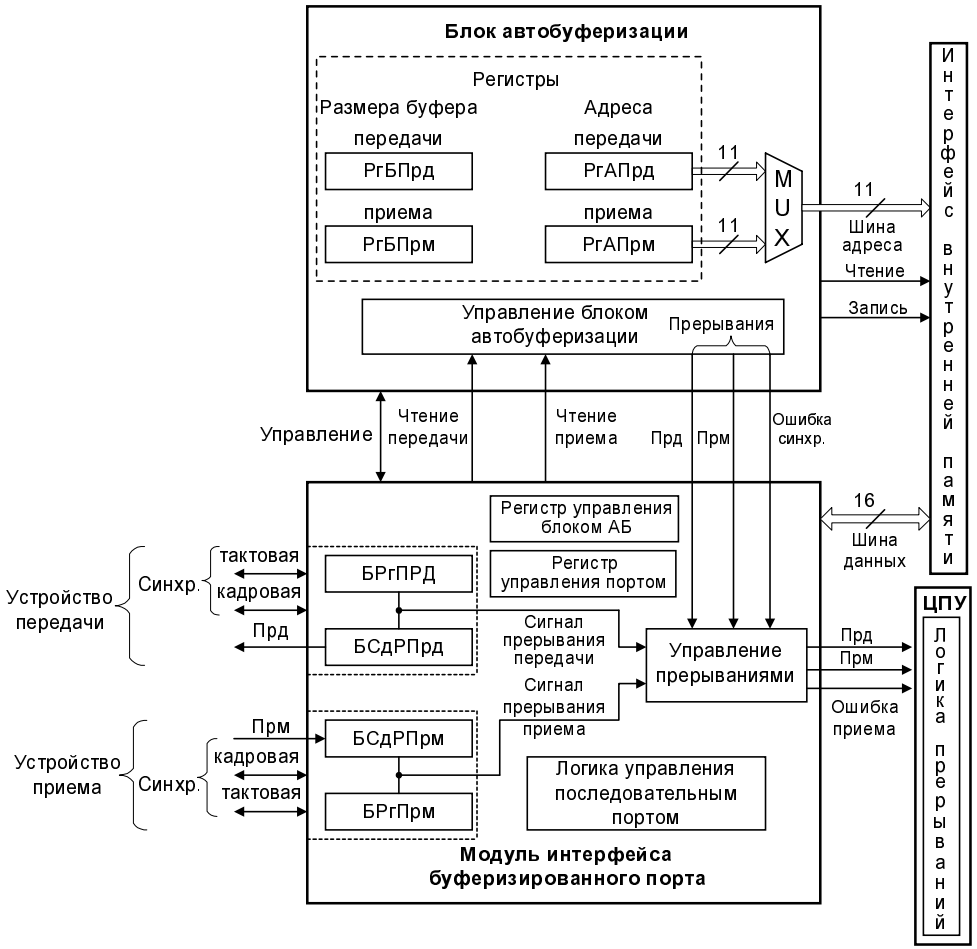


Рис. 8.28. Буферизированный последовательный порт

Режим автобуферизации

Смысл режима автобуферизации (АБ) состоит в том, чтобы обеспечить *циклический прямой доступ* к выделенному буферу с использованием механизма циклической адресации *(см. главу 3)* для приема и передачи полного блока данных. Автобуферизация позволяет использовать непрерывную передачу данных с внутренней кадровой синхронизацией.

Режим АБ может устанавливаться по каждому направлению (передачи или приема) самостоятельно парой регистров: адреса и размера буфера (рис. 8.29). Эти регистры полностью определяют адрес:

- верхней границы;
- нижней границы;
- текущей ячейки выделяемого буфера.

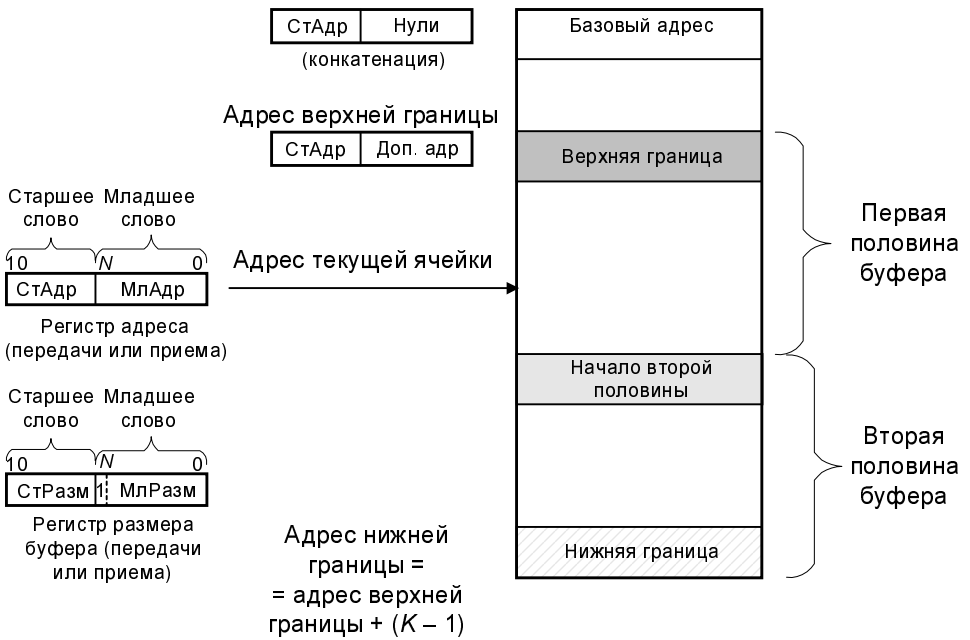


Рис. 8.29. Режим автобуферизации

Границы буфера задаются в пределах отведенной для буферизации области памяти объемом $2K$ согласно карте памяти. Установка границ осуществляется записью необходимых данных в регистрах адреса и размера буфера при инициализации процессора. Отводимая для буферизации область памяти может быть конфигурирована как память данных, память программ или и то, и другое вместе. Буферы передачи и приема могут размещаться в независимых пространствах памяти, в перекрывающихся пространствах или полностью совпадать. Перекрытие или совпадение буферов позволяет осуществлять передачу из буфера и прием в тот же буфер.

Минимально допустимый размер буфера равен 2, а максимально допустимый размер может составлять $2K$, т. е. 2048. Десятичное число 2048 соответствует 11-разрядному двоичному числу $2^{11} \Rightarrow 1000000000$, поэтому для адресации внутри области $2K$ достаточно 11-разрядного двоичного числа.

Однако все 11-разрядные регистры читаются как 16-разрядные, пять старших разрядов которых воспринимаются нулевыми, а оставшиеся 11 выравниваются справа, занимая разряды с 0 по 10.

Рассмотрим, каким образом определяется пространство памяти, занимаемое буфером.

Величина K, равная размеру буфера, записывается в N младших разрядов регистра размера (приема или передачи) при инициализации процессора. Эти разряды образуют младшее слово (МлРазм), у которого N-й бит всегда равен 1. Оставшиеся $(11 - N)$ разрядов составляют старшее слово (СтРазм), все биты которого нулевые. Размер буфера K определяет:

- допустимые относительные адреса верхней границы буфера (ОАВГ);
- объем пространства L, в котором может размещаться предполагаемый буфер.

В любом случае величины ОАВГ и L должны представлять собой степень двойки:

$$L = 2^n;$$
$$\text{ОАВГ} = m \cdot 2^n, \tag{8.11}$$

где $m = 0, 1, \ldots, (2K / L) - 1$.

Регистр адреса также состоит из старшего (СтАдр) и младшего (МлАдр) слов. Текущий адрес внутри буфера определяется полным содержимым регистра адреса.

Прежде чем размещать буфер в памяти, необходимо определить его базовый адрес, относительно которого определяется верхняя граница буфера. В образовании базового адреса участвует только та пара регистров размера и адреса, которая имеет наибольшее значение N. Базовый адрес (БА) формируется *конкатенацией* (соединением) старшего слова адресного регистра и комбинации из N нулей (рис. 8.30).

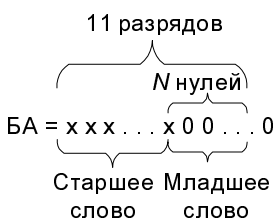


Рис. 8.30. Формирование базового адреса

Относительный адрес верхней границы (ОАВГ) представляет собой сумму БА и любого из допустимых адресов верхней границы (ДАВГ)

$$\text{ОАВГ} = \text{БА} + \text{ДАВГ}.$$

Тогда адрес нижней границы (АНГ) буфера определится суммой

$$\text{АНГ} = \text{ОАВГ} + (K - 1).$$

Замечание

Любой буфер размера от 1024 до 2048 должен начинаться с базового адреса 10000000000, т. е. у таких буферов БА и ОАВГ совпадают всегда.

Инициализацией МлАДР задается начальный адрес, по которому произойдет обращение в пределах буфера при старте программы. Обычно в качестве начального адреса выбирается ОАВГ.

Внутренней логикой АБ буфер разделяется на две половины (части). Адресуемое пространство первой части буфера охватывает $\lfloor K/2 \rfloor$ ячеек, второй части — $(K - \lfloor K/2 \rfloor)$ ячеек, где $\lfloor\ \rfloor$ означает обычное округление. Это говорит о том, что первая часть может быть на одну ячейку больше второй.

Приведем пример (рис. 8.31). Пусть размер буфера передачи равен 5, а буфера приема — 8. Тогда содержимое регистров размера имеет вид:

регистр размера буфера передачи	регистр размера буфера приема
0 0 0 . . . 0 **1 0 1** (xxx**5**h)	0 0 0 . . . **1 0 0 0** (xxx**8**h)

Величина N соответственно равна 3 и 4, а базовые адреса буферов вследствие конкатенации трех и четырех нулей со старшим словом регистра адреса окажутся такими:

БА буфера передачи	БА буфера приема
x x x . . . x **0 0 0** (xxx**X**h)	x x x . . . **0 0 0 0** (xxx**0**h)

Отсюда следует, что общий базовый адрес равен БА буфера приема. Однако относительные адреса верхней границы рассматриваемых буферов будут разными, поскольку для буфера передачи ОАВГ кратен 8 (0008h, 0010h, ..., 07F8h), а для буфера приема ОАВГ кратен 16 (0010h, 0020h, ..., 07F0h). Выберем ОАВГ буфера передачи 0008h, а буфера приема — 0010h. Адресные пространства буферов и их частей показаны на рис. 8.31.

Адресация начинается с верхней границы. Любое обращение к буферу сопровождается *инкрементированием адресного регистра.* После достижения нижней границы адресный регистр по сигналу с устройства управления блоком АБ перезагружается относительным адресом верхней границы, т. е. осуществляется алгоритм циклической адресации.

Режим автобуферизации задается пользователем установкой необходимых битов в регистре управления БАБ модуля последовательного порта (см. рис. 8.26). Сигналы прерывания, подаваемые на ЦПУ при передаче и прие-

ме каждого слова, в режиме АБ не генерируются. Вместо них на ЦПУ выдаются сигналы прерывания передачи и приема лишь в случаях, если:

- пересекается адресное пространство одной из частей буфера или буфер полностью (сигналы "прерывание Прд" и "прерывание Прм");
- обнаружена ошибка синхронизации (сигнал "Ошибка синх.").

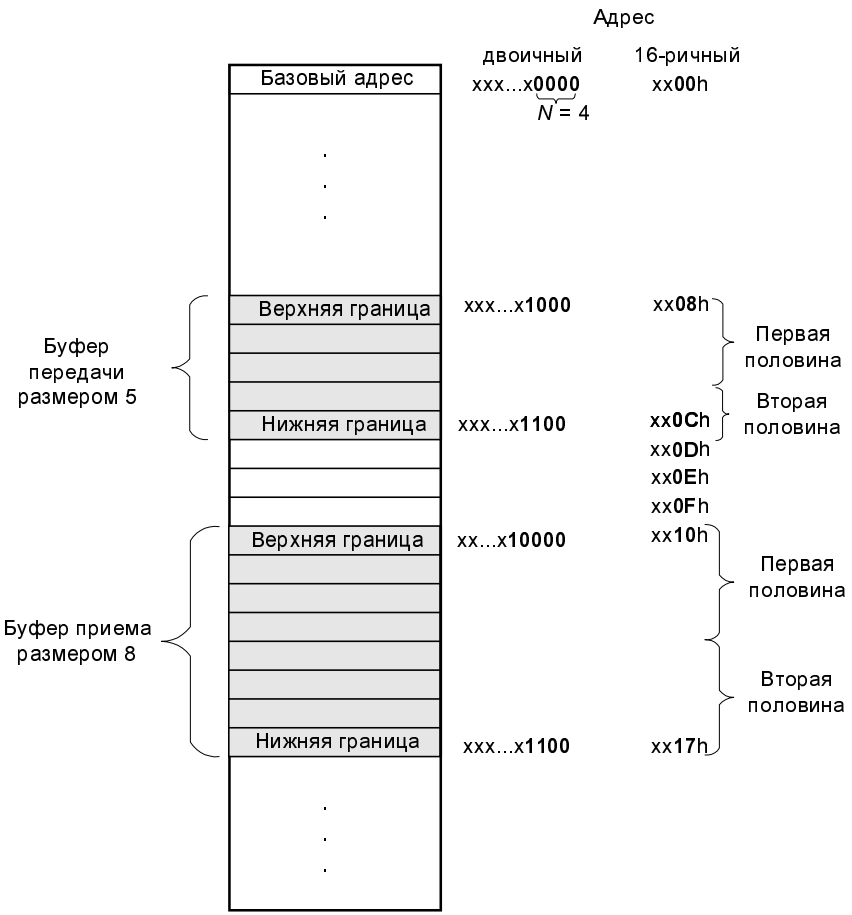


Рис. 8.31. Пример формирования буферов

Адресная шина используется для передачи и приема в режиме разделения времени. При передаче формируется строб "чтение", и по шине данных слово пересылается в регистр передачи БРгПрд и далее в сдвиговый регистр БСдРПрд. Принимаемое слово через регистры устройства приема поступает на шину данных и далее пересылается в буфер согласно адресу приема в момент воздействия строба "запись".

В процессе приема и передачи вследствие воздействия импульсных помех возможна ошибка тактовой или кадровой синхронизации, которая выражается в виде пропуска или появления посторонних импульсов. При обнаружении ошибки синхронизации выдается сигнал прерывания "Ошибка синх.", который указывает, что последовательным интерфейсом потеряны одно или несколько слов.

Таким образом, протокол автобуферизации состоит в следующем:

- ❒ блок автобуферизации выполняет операции доступа к памяти буфера;
- ❒ соответствующий адресный регистр инкрементируется, если еще не достигнута нижняя граница буфера;
- ❒ генерируется прерывание приема или передачи, если пересечена первая половина буфера или его нижняя граница;
- ❒ выполняется автоблокировка блока автобуферизации (при выборе этой функции во время инициализации процессора), если пересечена половина буфера или его нижняя граница.

8.7. Контроллер прямого доступа к памяти

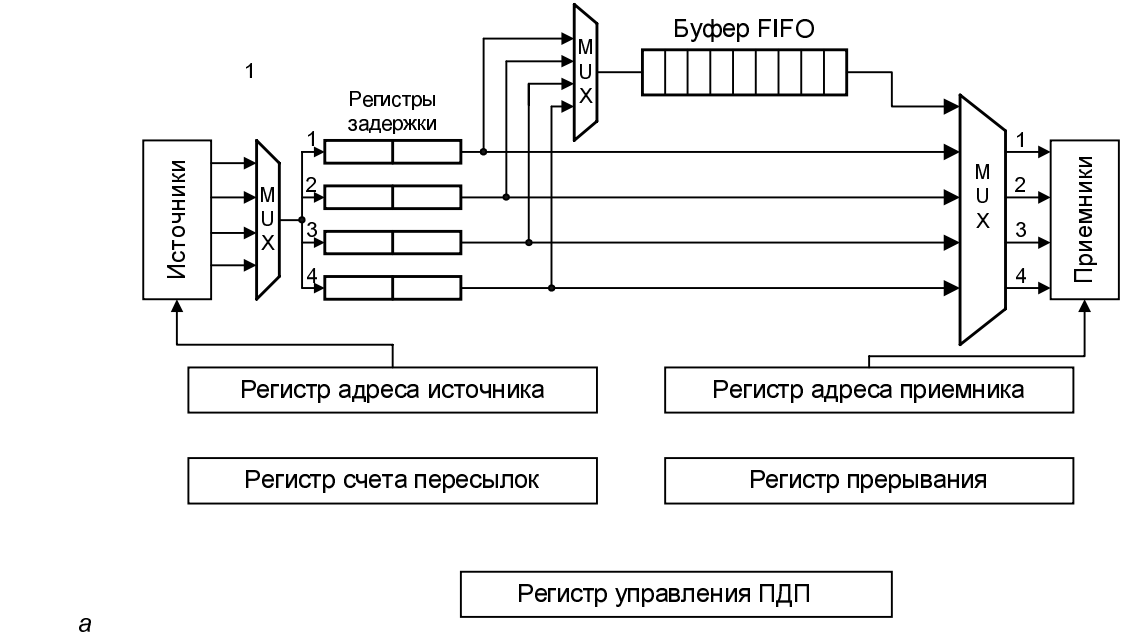
Контроллер прямого доступа к памяти (ПДП) предназначен для организации пересылок между различными областями карты памяти без вмешательства ЦПУ. Все пересылки ПДП производятся в фоновом режиме относительно работы ЦПУ, что существенно увеличивает производительность процессора. Контроллеры ПДП входят в состав процессоров семейств TMS320C5xxx, TMS320C6xxx, ADSP-21xx.

Замечание

В иностранной литературе для термина "прямой доступ к памяти" используется аббревиатура DMA (Data Memory Access).

Контроллер ПДП (рис. 8.32, *а*) имеет несколько независимо программируемых каналов и снабжен рядом регистров:

- ❒ *общим регистром управления ПДП*, который своим состоянием определяет:
  - режим работы контроллера;
  - характер изменения адресов источника и приемника (инкремент, декремент, синхронный);
- ❒ *регистрами адресов* источников и приемников для каждого канала ПДП;
- ❒ *регистрами счетчика пересылок*, управляющего размером кадров и блоков пересылок по каждому каналу ПДП;
- ❒ *регистром разрешения прерываний* ЦПУ/ПДП.



а

б

Рис. 8.32. Принцип работы контроллера прямого доступа к памяти

Контроллер ПДП обеспечивает пересылки данных в виде (рис. 8.32, *б*):

- элемента, разрядность которого программируется и может составлять 8, 16 или 32 бита;
- кадра, состоящего из программируемого количества элементов;
- блока, состоящего из программируемого количества кадров.

Пересылка элемента в каждом канале выполняется за два шага:

1. Чтение элемента (данных) из ячейки памяти, адрес которой определяется регистром адреса источника ПДП.
2. Запись прочитанного элемента в ячейку памяти, адрес которой определяется регистром адреса приемника ПДП.

Пересылка элемента завершается только по окончании процессов чтения и записи. В конце каждой операции чтения или записи адрес источника или приемника модифицируется (обновляется) согласно правилу (инкремент, декремент), установленному в общем регистре управления ПДП. По окончании каждой записи декрементируется счетчик пересылок ПДП.

- *Пересылка кадра.* Пересылаемые элементы, число которых на каждом канале ПДП в общем случае неодинаково и устанавливается пользователем в регистре счетчика пересылок, собираются в кадр. Количество составляющих кадр элементов называется длиной кадра.
- *Пересылка блока.* Пересылаемые кадры, число которых на каждом канале ПДП в общем случае неодинаково и устанавливается пользователем, собираются в кадр. Количество составляющих блок кадров называется длиной блока.

Длина кадров и блоков устанавливается в регистре счетчика пересылок по каждому каналу ПДП при инициализации процессора. Допустимые пределы, в которых может изменяться эта длина, зависят от разрядности регистра счетчика пересылок. Регистр разрядности m допускает $2^m - 1$ элементов в одном кадре и столько же кадров в одном блоке. Например, если $m = 16$ (что типично для большинства процессоров), максимальная длина кадра составит 65 535 элементов; такое же максимальное количество кадров может содержать один блок. Таким образом, в одном блоке по одному каналу ПДП допустимо пересылать до 4 294 836 225 элементов.

Пересылка кадра и блока завершается генерацией сигнала прерывания.

8.7.1. Синхронизация каналов ПДП

Каналы ПДП синхронизируются с прерываниями со стороны внешней периферии или внутренних устройств процессора. Каждому каналу доступны четыре механизма синхронизации, включаемых согласно определенному пользователем событию, вызывающему прерывание (например, прерывание

по таймеру, прерывание канала ПДП, прерывание хост-порта и т. д.), и возбуждает соответствующую пересылку:

- *нет синхронизации*: все прерывания ПДП блокируются, и канал ПДП выполняет чтение и запись;
- *синхронизация источника* (чтение): канал ПДП ожидает соответствующий сигнал прерывания и при его поступлении выполняет чтение по адресу, определяемому регистром адреса источника ПДП;
- *синхронизация приемника* (запись): канал ПДП ожидает соответствующий сигнал прерывания и при его поступлении выполняет запись по адресу, определяемому регистром адреса приемника ПДП;
- *синхронизация кадра*: канал ПДП ожидает соответствующий сигнал прерывания и при его поступлении выполняет пересылку кадра.

8.7.2. Генерация адреса ПДП

Генерация адреса ПДП осуществляется с помощью индексных регистров. На каждом канале ПДП вычисляется *базовый адрес* пересылки чтения и записи элемента, для чего используется индексная адресация *(см. главу 5)* на величину N, глобального (общего) индекса, равную размеру элемента. По умолчанию значение индекса $N = 0$. Размер элемента равен количеству байтов, содержащихся в одном элементе:

разрядность элементов	размер элементов
8	1
16	2
32	4

При выборе инкремента или декремента индекс равен размеру элемента. Например, если адресация источника установлена на инкремент и передается 16-разрядный элемент, адрес источника инкрементируется на $N = 2$ после каждой пересылки чтения.

Базовый адрес элемента не зависит от того, является ли пересылаемый элемент (пересылка) последним в текущем кадре или нет. С целью определения окончания кадра вводятся дополнительные индексы элемента и кадра, которые записываются в общий индексный регистр выбранного канала и могут находиться в пределах от −32 768 до 32 767 каждый. Эти индексы воздействуют на установку адреса следующим образом:

- *индекс элемента*: определяет ту величину, которая должна быть добавлена к базовому адресу источника или приемника, если пересылка соответствующего элемента не является последней в текущем кадре;
- *индекс кадра*: добавляется к базовому адресу источника или приемника, если пересылка соответствующего элемента является последней в теку-

щем кадре; такая установка осуществляется при пересылках как одиночного кадра, так и при многокадровых пересылках.

8.7.3. Система приоритетов ПДП

Система приоритетов ПДП построена с точки зрения состязаний между ПДП и ЦПУ с одной стороны и между каналами ПДП — с другой.

Приоритет ПДП относительно ЦПУ

Каждому каналу ПДП может быть независимо определен более высокий приоритет установкой соответствующих битов в регистре управления соответствующим каналом.

Приоритет между каналами ПДП

Контроллер ПДП имеет фиксированную схему приоритетов, в которой нулевому каналу присвоен высший приоритет, а каналу с наибольшим номером — низший.

Состязания между каналами ПДП и ЦПУ

Если за одно и то же устройство состязаются каналы ПДП и ЦПУ, сначала осуществляется арбитраж между каналами ПДП, а затем между выбранным каналом с наивысшим приоритетом и ЦПУ. Если канал обладает меньшим приоритетом, чем ЦПУ, доступ к устройству первым предоставляется ЦПУ, а данный канал и каналы с более низким приоритетом ставятся в очередь. И наоборот: если ЦПУ обладает меньшим приоритетом относительно выбранного канала, то сначала обслуживается выбранный канал и каналы с более высоким, чем у ЦПУ, приоритетом, а ЦПУ ставится в очередь.

Пример. Рассмотрим принцип работы контроллера ПДП (см. рис. 8.32) на примере процессоров TMS320C6201/C6701, имеющих четыре канала ПДП. Источниками и приемниками могут быть:

- ❒ интерфейс внешней памяти;
- ❒ внутренняя память программ;
- ❒ внутренняя память данных;
- ❒ шина внутренней периферии.

Пусть необходимо 32-разрядные элементы данных переслать из внешней памяти во внутреннюю память данных. Регистр управления определяет источник — канал 1 и приемник — канал 3. Регистры адресов выставляют соответствующие адреса чтения и записи. Данные из внешней памяти через мультиплексор поступят на третью пару регистров задержек (первый шаг пересылки — *чтение*) и затем через второй мультиплексор в ячейку внут-

ренней памяти данных (второй шаг пересылки — *запись*). По окончании выполнения чтения/записи адреса источника/приемника независимо инкрементируются (или декрементируются — в зависимости от настройки) на 4, а по завершении пересылки декрементируется счетчик пересылок. Начало очередной пересылки (чтения) по времени совпадает с записью в текущей пересылке.

При поступлении запроса на прерывание от устройства с высшим приоритетом данные от источника будут накапливаться в буфере FIFO, имеющем глубину 9. Общая память накопления данных вместе с двумя регистрами задержки составляет 11 ячеек. После обслуживания прерывания накопленные данные будут переданы приемнику сначала из буфера, затем из регистров задержек, и возобновится описанный ранее процесс пересылки.

8.8. Порт с временным разделением каналов

Порт с временным разделением каналов, или TDM-порт (Time-Division Multiplexing, мультиплексирование с разделением по времени), предназначен для обмена данными между несколькими процессорами в режиме временного разделения при реализации многопроцессорных задач.

TDM-порт представляет собой расширенный вариант базового синхронного порта *(см. разд. 8.6.1)* и может работать в двух режимах: автономном и мультипроцессорном. Выбор режима осуществляется пользователем установкой требуемого бита (обычно TDM) в регистре управления TDM-портом. Автономный режим соответствует работе БСП, поэтому ниже рассматривается только мультипроцессорный режим.

Идея временного разделения каналов состоит в объединении группы процессоров с целью быстрого обмена данными между ними, когда каждому процессору внутри кадра выделяется промежуток времени (*сегмент*), только в течение которого процессор может передавать данные другим процессорам или самому себе, а в оставшиеся сегменты способен принимать данные от других процессоров. Такие порты содержат процессоры TMS320C5x, 'C54x, а также процессоры платформ 'C5xxx и 'C6xxx как часть более сложных портов.

TDM-порт содержит восемь размещенных в памяти регистров (рис 8.33):

- регистр управления портом TDM (ТРгУпр);
- регистр приема данных (ТРгПрм);
- сдвиговый регистр приема (ТСдПрм);
- регистр передачи данных (ТРгПрд);
- сдвиговый регистр передачи (ТСдПрд);

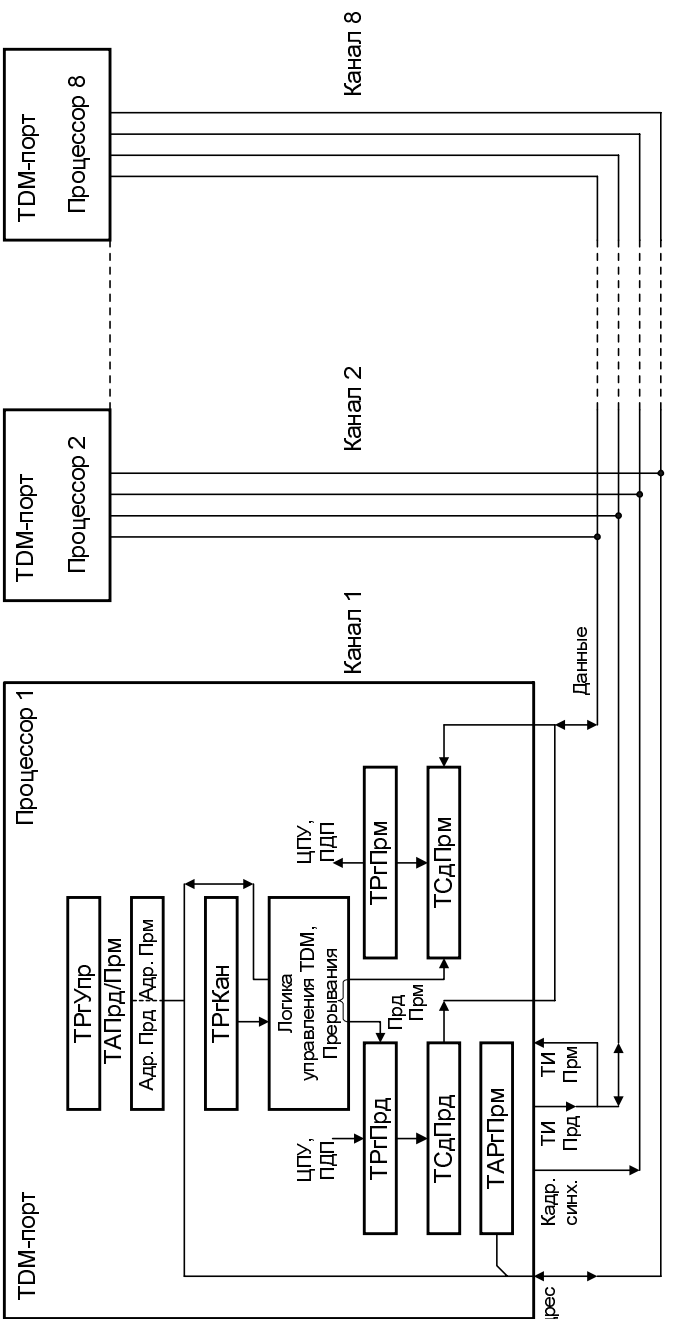


Рис. 8.33. Принцип работы TDM-порта

- регистр выбора канала (ТРгКан);
- адресный регистр передачи/приема (ТАПрд/Прм);
- адресный регистр приема (ТАРгПрм).

Все регистры, за исключением сдвиговых, программно доступны. Регистры приема, передачи и сдвиговые имеют то же назначение и работают так же, как и в базовом синхронном порте.

Регистр выбора канала (ТРгКан) определяет, в каком сегменте процессор может передавать данные.

Адресный регистр передачи/приема (ТАПрд/Прм) содержит в старшем слове адрес передачи (какому процессору передаются данные), а в младшем — присвоенный процессору адрес приема.

Адресный регистр приема (ТАРгПрм) содержит различную информацию о состоянии адресной линии интерфейса TDM, о чем сказано ниже.

Процессоры подключаются параллельно к четырехпроводной шине, содержащей линии (см. рис. 8.33):

- тактовой синхронизации;
- кадровой синхронизации;
- данных;
- адреса.

Принцип работы TDM-порта состоит в следующем. Каждому процессору присваивается номер канала и выделяется *свой сегмент* в кадре для передачи данных другим процессорам (рис. 8.34). Номер канала, соответствующий номеру сегмента (например, канал 1 и сегмент 0), размещается в ТРгКан при инициализации порта.

Кадр содержит восемь сегментов. Работа всех TDM-портов синхронизируется одним процессором — источником сигналов синхронизации (на рис. 8.34 это процессор 1). Частота тактовой синхронизации ТИ обычно равна 1/4 частоты генератора СИ. Каждый сегмент содержит 16 битов (данные Прд и Прм синхронизируются по переднему фронту). Длительность одного бита равна одному периоду ТИ, кроме первого бита, длительность которого равна 1/2 периода ТИ. Импульс кадровой синхронизации формируется каждые 128 периодов ТИ и совпадает с нулевым битом седьмого сегмента.

Каждый процессор и соответствующий ему канал может быть *активным* или *пассивным*. Канал является активным, только если процессор осуществляет передачу данных на отведенном ему сегменте. Так, канал 1 может быть активным на нулевом сегменте, остальные каналы в это время пассивны и принимают данные, передаваемые первым процессором по линии данных.

Одновременно с передачей данных на адресной линии выставляется адрес передачи, который в текущем сегменте анализируется всеми (включая и активный!) процессорами.

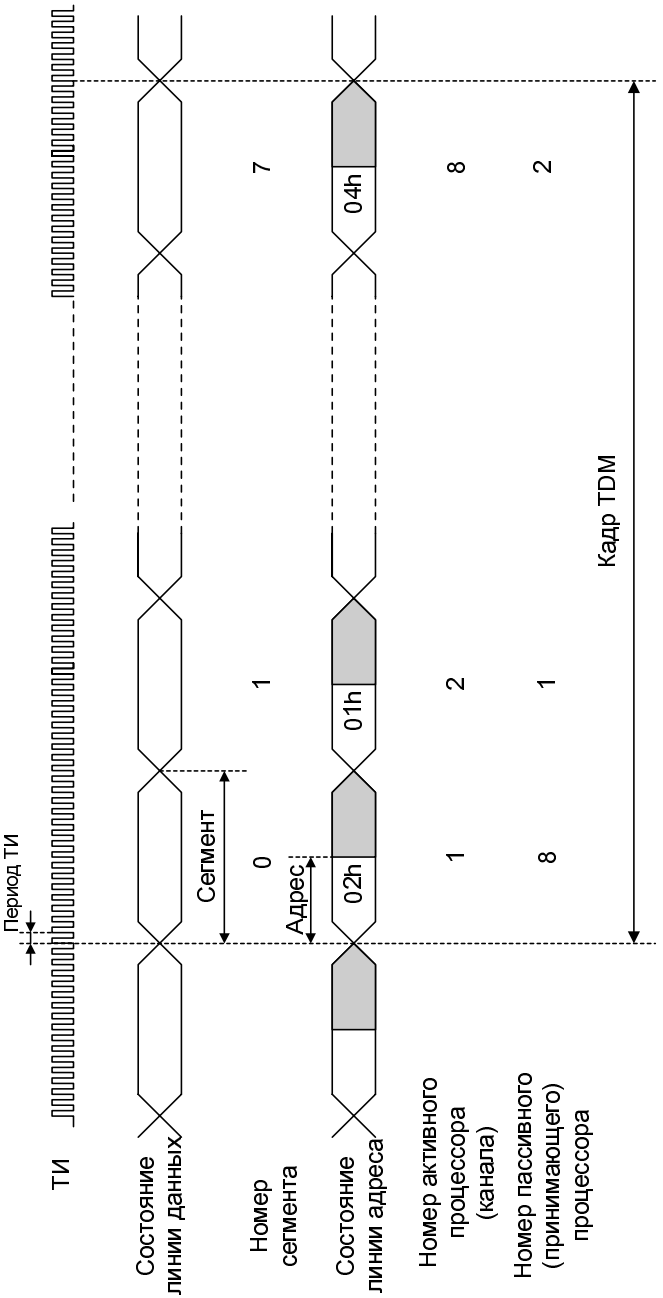


Рис. 8.34. Синхронизация системы TDM

Анализ адреса передачи происходит побитовым его сопоставлением с адресом приема, записанным в младшем слове ТАПрд/Прм. Достоверный прием данных выполняется теми процессорами, чьи адреса приема совпадают с адресом передачи, т. е. обмен данными может осуществляться как по принципу "каждый-с-каждым", так и по принципу "один-со-многими".

При совпадении адресов передачи и приема логикой управления TDM-порта формируется сигнал прерывания Прм, который подтверждает достоверность принятых данных и разрешает пересылку данных, поступивших на регистр ТСдПрм, в регистр приема и далее в контроллер ЦПУ или ПДП.

Адрес передачи, выставляемый активным процессором, является 8-битовым и пересылается, начиная с младшего бита в первой половине сегмента (вторая половина адресного сегмента не занята).

Замечание

Адрес передачи может устанавливаться в ТАПрд/Прм как при инициализации, так и при выполнении программы, поэтому активный процессор может передавать данные самому себе.

На рис. 8.33 стрелками показано, как передаются данные в нулевом, первом и седьмом сегментах, когда последовательно являются активными процессоры 1, 2 и 8. Например, в первом сегменте активным является процессор 2, выставивший адрес передачи 01h, поэтому достоверный прием данных будет осуществляться только первым процессором, адрес приема которого совпадает с выставленным адресом передачи.

8.9. Многоканальный буферизированный последовательный порт

Многоканальный буферизированный последовательный порт (МкБПП) представляет собой высокоскоростной синхронный последовательный порт с прямым доступом к памяти, возможностью многоканальной работы и совместимый со стандартами E1/T1, SCSA и MVIP.

Кроме возможностей базового синхронного порта, МкБПП обеспечивает:

- прямое подключение к шинам внешней периферии сетевых стандартов:
  - E1/T1 — мультиплексирования до 32 каналов при организации телефонных стыков;
  - ST-BUS (Stream Protocol — Broadcast and Unknown Server) — обмена данными в режиме потока данных;
  - AC97 — сопряжения с аудиокодеком стандарта Audio Codec'97;
  - SPI — 4-проводного интерфейса последовательного порта с протоколом по принципу "ведущий-ведомый", с двунаправленными шинами передачи и приема;

- многоканальный обмен данными при количестве каналов до 128;
- переменную разрядность данных 8, 12, 16, 20 и 32 бита;
- встроенное аппаратное компандирование по A- и μ-законам;
- программируемую систему синхронизации;
- SCSA (Signal Computing System Architecture) — архитектура вычислительных систем по обработке сигналов;
- MVIP (Multi-Vendor Integration Protocol) — объединенный многоплатформенный протокол.

Каналы передачи и приема МкБПП (рис. 8.35) работают независимо, соответствующие регистры передачи, приема и сдвига выполняют те же функции, что и в БСП.

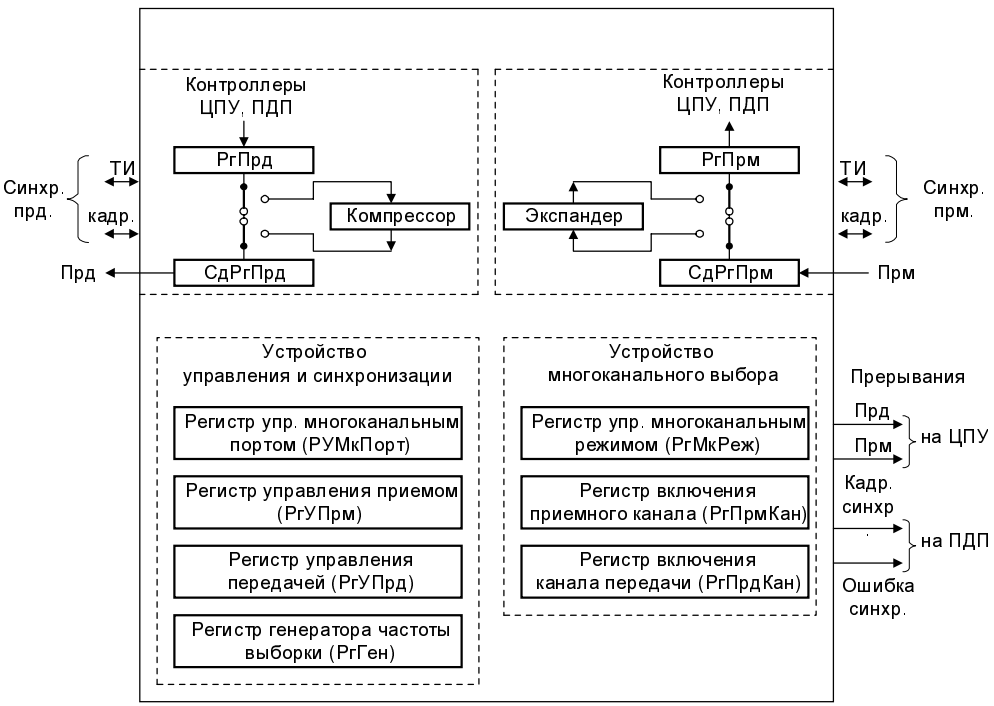


Рис. 8.35. Структурная схема МкБПП

Многоканальный обмен данными осуществляется в режиме временного разделения каналов подобно TDM-порту.

Компандирование по A- и μ-законам достигается подключением компандера с помощью регистра управления МкБПП.

Система синхронизации МкБПП обладает рядом особенностей, позволяющих организовывать непрерывный поток данных с различной разрядностью

внутри кадра. Кадр передачи и приема может характеризоваться следующими основными параметрами:

- количеством фаз;
- количеством элементов в каждой фазе;
- количеством битов на элемент;
- задержкой данных относительно сигнала кадровой синхронизации.

Кадр (рис. 8.36) может содержать до двух *фаз* (фаза 1 и фаза 2); максимальное количество элементов на кадр равно 128 для однофазного кадра и 256 для двухфазного. Количество битов на элемент выбирается пользователем из ряда 8, 12, 16, 20, 24 и 32, причем разрядность и количество элементов для каждой фазы определяется независимо. На рис. 8.36 приведен пример двухфазного кадра, содержащего 2 элемента по 12 битов в первой фазе и 3 элемента по 8 битов во второй фазе. Поток данных в кадре является непрерывным, т. е. отсутствуют паузы между фазами и элементами.

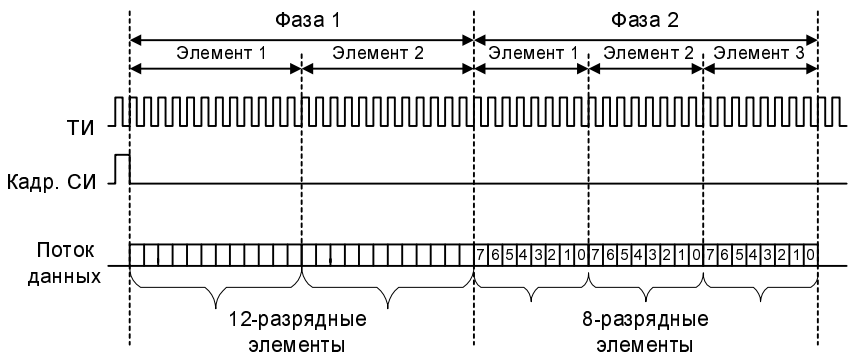


Рис. 8.36. Пример двухфазного кадра

Длина кадра, как и прежде, определяется количеством элементов (временных сегментов, или каналов), содержащихся в кадре; *длина элемента* — количеством содержащихся в нем битов. Управляя длиной кадра и длиной элемента, можно существенно сократить количество пересылок данных, а потому и время, затрачиваемое на пересылки при приеме и передаче. Например, пусть в однофазном кадре содержится четыре 8-разрядных элемента. Тогда контроллер ЦПУ или ПДП должен выполнить четыре пересылки из регистра принимаемых данных РгПрм и столько же в РгПрд. Но этот кадр можно запрограммировать как однофазный, состоящий из одного 32-разрядного элемента — в этом случае контроллеру потребуется выполнить только по одной пересылке на передаче и приеме, что сократит время занятости шины в четыре раза.

Задержка данных относительно сигнала кадровой синхронизации зависит от того, какой *стандарт* имеет интерфейс подключаемой периферии, и может составлять 0, 1 или 2 бита.

При отсутствии задержки передаваемые данные синхронизируются по переднему (возрастающему) фронту, а принимаемые — по заднему (падающему) фронту сигнала кадровой синхронизации.

Задержка на 1 бит является типовой и необходима, например, при сопряжении с AC97, использующем свойства двухфазного кадра: первая фаза состоит из одного 16-разрядного элемента, вторая — из двенадцати 20-разрядных элементов. Задержка в один элемент первой фазы связана с тем, что сигнал кадровой синхронизации длительностью в один элемент захватывает передаваемый последним нулевой бит двенадцатого элемента второй фазы предыдущего кадра (передача/прием начинается со старшего, 15-го бита).

Замечание

Стандартным является однофазный кадр с количеством элементов от 1 до 128 и задержкой данных относительно сигнала синхронизации на один бит, поскольку сигнал синхронизации предшествует начальному биту кадра и сам имеет длительность в один бит.

Задержка на два бита используется в интерфейсе E1/T1, где длительность сигнала кадровой синхронизации составляет один бит, а за ним следует "бит организации цикла", предшествующий потоку данных в кадре. На приеме бит организации цикла отбрасывается, а на передаче на его месте многоканальный порт устанавливает пустой интервал, заполняемый битом организации цикла другим прибором, имеющим указанный выше интерфейс.

8.9.1. Устройство управления и синхронизации МкБПП

Регистры устройства управления МкБПП (см. рис. 8.35) определяют конфигурацию порта и параметры каналов передачи и приема:

- одно- или двухфазную кадровую синхронизацию;
- длину кадра;
- количество битов в элементе;
- режимы компандирования;
- величину задержки данных относительно сигнала кадровой синхронизации.

Регистр генератора тактовой частоты дает возможность пользователю выбрать разнообразные параметры тактовой и кадровой синхронизации.

Устройство управления посылает также сигналы прерывания приема и передачи на контроллеры ЦПУ и ПДП:

- на каждом последовательном элементе;
- в конце подкадра, содержащем не больше 16 элементов, внутри кадра;

- при обнаружении сигналов кадровой синхронизации;
- при ошибке кадровой синхронизации.

8.9.2. Устройство многоканального выбора

Устройство многоканального выбора управляет работой в многоканальном режиме и содержит (см. рис. 8.35):

- регистр управления многоканальным режимом (РгМкРеж);
- регистр включения канала передачи (РгПрмКан);
- регистр включения канала приема (РгПрмКан).

Многоканальный режим устанавливается в РгМкРеж независимо для каналов передачи и приема при конфигурировании МкБПП в режиме однофазного кадра.

Каждый кадр представляет собой поток данных, содержащий до 128 элементов, разделяемых во времени. Эти 128 элементов группируются на 8 подкадров (от 0 до 7) по 16 элементов в каждом. Подкадры распределяются по двум секциям *A* и *B*: подкадры с четными номерами (0, 2, 4, 6) размещаются в секции *A*, подкадры с нечетными номерами (1, 3, 5, 7) — в секции *B*.

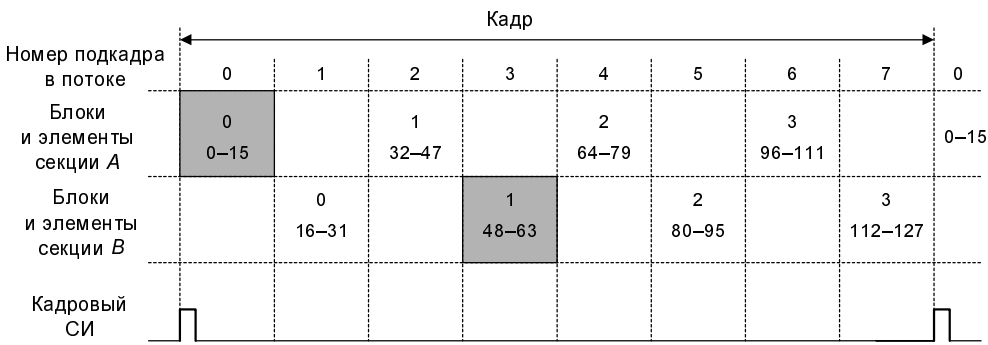


Рис. 8.37. Разбиение потока данных на секции

Одновременно могут быть доступны только два любых подкадра, принадлежащих разным секциям, т. е. один кадр с четным номером, а второй — с нечетным. Таким образом, обеспечивается одновременный доступ только к 32 элементам из допустимых 128. На рис. 8.37 показано, что доступны только блок 0 секции *A,* содержащий элементы 0—15, и блок 1 секции *B*, содержащий элементы 48—63 кадра потока данных. Управление доступом к подкадрам приема и передачи осуществляется регистрами включения каналов передачи и приема. В конце каждого подкадра, содержащего 16 элементов или меньше, генерируется прерывание передачи и приема на контроллеры ЦПУ или ПДП. Это прерывание является сигналом перехода к новой секции.

8.10. Хост-порт

Большинство сложных многопроцессорных систем нуждаются в управлении со стороны только одного из приборов (процессоров), составляющих систему. Этот прибор называют ведущим, а остальные — ведомыми. Ведущим может быть компьютер, другой процессор, микроконтроллер. Ведущий прибор в технике ЦПОС получил наименование *хост-прибора*, т. е. "хозяина" системы (англ. *host* — хозяин).

Другой задачей в многопроцессорных системах является повышение скорости обмена данными, что возможно при переходе от последовательного порта к параллельному, когда скорость обмена данными между хост-прибором и ведомыми приборами приближается к скорости работы шины данных, а операция чтения или записи может производиться за один машинный цикл.

Указанные задачи решаются организацией параллельных портов, называемых хост-портами.

Хост-порт — это 8- или 16-разрядный параллельный порт ввода/вывода, выделяемый в процессоре и предназначенный для прямого обмена данными между хост-прибором и процессором.

Хост-прибор воспринимает любой из подключенных к нему процессоров как периферийное устройство, содержащее доступную специально выделенную область внутрикристальной памяти — хост-память. Всем процессорам в системе присваиваются индивидуальные адреса, по которым хост-прибор осуществляет управление отдельным процессором.

Хост-порт организован таким образом, что он не влияет на работу и быстродействие остальных устройств процессора.

Все современные и перспективные процессоры, включая платформы TMS320C5xxx и TMS320C6xxx, семейства процессоров ADSP-21xx и фирмы Motorola, снабжены интерфейсами хост-портов (HIP, Host Interface Port).

Интерфейс хост-порта (хост-интерфейс) в зависимости от образца процессора может быть сконфигурирован для работы с 8- или 16-разрядными данными, иметь раздельные шины адреса и данных или мультиплексируемую шину данных/адреса.

Рассмотрим работу обобщенного хост-интерфейса (рис. 8.38) при условиях:

- данные в процессоре 16-разрядные;
- шина данных/адреса 8-разрядная мультиплексированная;
- внутрикристальная хост-память имеет объем 2 Кбайт.

Хост-порт включает в себя:

- три регистра для связи хост-прибора с ЦПУ процессора:
  - регистр управления (РгУпрХост), доступный как хост-прибору, так и процессору;

- мультиплексированный регистр хост-данных передачи/приема (РгДХост), доступный только хост-прибору; регистр содержит либо данные, которые считаны из памяти и должны быть переданы хост-прибору, либо данные, поступившие от хост-прибора для записи в память;
- регистр хост-адреса (РгАХост), доступный только хост-прибору;

❒ логику управления хост-интерфейсом;

❒ вспомогательный канал ПДП.

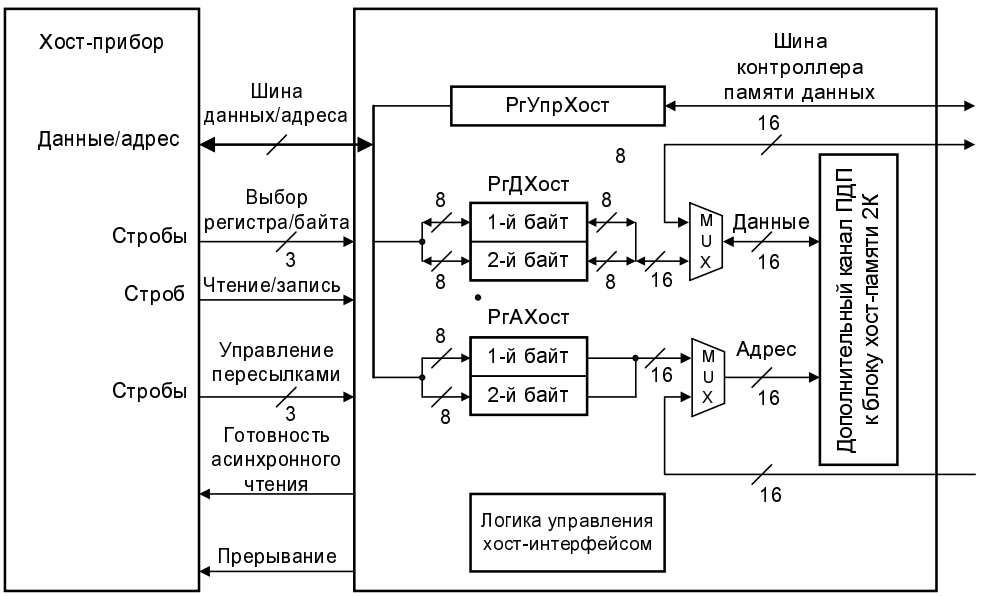


Рис. 8.38. Структурная схема обобщенного хост-интерфейса

Хост-интерфейс может работать в одном из двух режимов:

❒ в режиме группового доступа;

❒ в режиме только хост-интерфейса.

Режим группового доступа является обычным режимом работы, когда хост-память доступна как хост-прибору, так и самому процессору. В этом режиме асинхронные обращения со стороны хост-прибора синхронизируются внутренней системой процессора и в случае конфликта между процессором и хост-прибором при одновременном обращении к памяти на чтение или запись хост-прибор имеет приоритет, а процессор ожидает один цикл, пока не завершится выполнение обращения со стороны хост-прибора. При обращении хост-прибора к хост-регистру данных логика управления интерфейсом автоматически обеспечивает доступ к выделенному двунаправленному блоку хост-памяти для завершения пересылки данных.

Режим "только хост-интерфейс" исключает доступ процессора к хост-памяти. Этот режим позволяет обращаться хост-прибору к памяти данных даже в тех случаях, когда процессор находится в состоянии сброса, пониженного потребления мощности или в состоянии останова внутрикристальной периферии и ЦПУ.

Рассмотрим последовательность действий в режиме группового доступа при обращении хост-прибора к процессору для пересылки данных из хост-прибора в хост-память процессора.

Хост-прибор начинает обращение к процессору последовательным вызовом регистра управления, регистра адреса и регистра данных. Сигнал "номер вызываемого регистра" указывает, какой из перечисленных регистров вызывается хост-прибором. Первым вызывается регистр управления, который определяет:

- ❒ порядок следования байтов (полуслов);
- ❒ прерывание ЦПУ со стороны хост-прибора;
- ❒ прерывание хост-прибора со стороны ЦПУ;
- ❒ запрос вызова хост-прибора;
- ❒ значение сигнала, выставляемого хост-прибором.

При вызове адресного регистра на шине адреса/данных устанавливается адрес в хост-памяти и формируется строб адреса для загрузки адресного регистра. Данные из хост-памяти записываются в регистр данных.

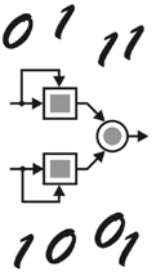
Поскольку внутренние слова процессора и адрес являются 16-разрядными, все пересылки должны состоять из двух, следующих друг за другом, байтов. Сигнал "номер байта", поступающий от хост-прибора, указывает, какой из двух байтов должен пересылаться первым. Регистр управления по сигналу "идентификация полуслова" определяет, какой из пересылаемых байтов является старшим в 16-разрядном слове. По сигналам "строб чтения" и "чтение/запись" данные выставляются на шине данных. Аналогичным образом происходит пересылка данных из хост-прибора в процессор.

Чтение и запись массивов данных, размещенных в последовательно расположенных ячейках памяти, может происходить в режиме автоинкремента, причем чтение данных осуществляется постинкрементированием адресного регистра, а запись данных — преинкрементированием.

Хост-прибор может прервать ЦПУ и перевести процессор в режим пониженного потребления мощности установкой специально отведенных битов в регистре управления. Процессор со своей стороны также может послать хост-прибору сигнал прерывания, однако *прерывание между байтами одного слова недопустимо.*

Вследствие асинхронной работы хост-интерфейса в процессоре могут появиться отложенные циклы. При обнаружении такой ситуации процессор выдает сигнал "готовности", позволяющий включать режим ожидания на хост-приборе до завершения отложенных циклов.

Глава 9



Подготовка программ пользователя. Языки программирования

9.1. Этапы разработки программы

Процесс разработки программ описан во многих пособиях и книгах. Часто его называют жизненным циклом разработки программы [43]. Он включает в себя следующие этапы (рис. 9.1).

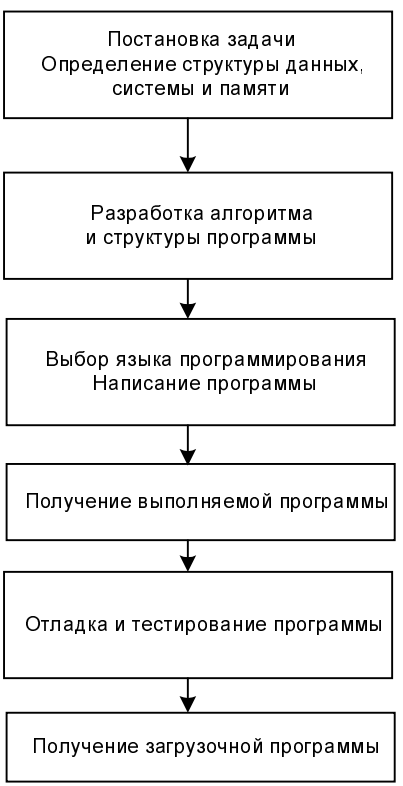


Рис. 9.1. Этапы разработки программы

- **Постановка и формулировка задачи, описание данных**. На этом этапе определяется назначение программы, формулируются требования к исходным и выходным данным, их структуре и форматам, методам ввода и вывода в проектируемой системе. В соответствии со структурой и объемом данных выбирается общий вид системы и памяти, методы и порты ввода/вывода. Эта информация является предварительной и может уточняться в процессе дальнейшей работы, однако она необходима для работы над программой. В частности алгоритмы могут зависеть от структуры данных.

- **Разработка алгоритма и структуры программы**. На этом этапе выбирается метод решения задачи и разрабатывается алгоритм его реализации. Формируется структура программы в виде функциональных модулей, и определяется интерфейс модулей, т. е. конкретные входные и выходные параметры, через которые они получают и передают данные. При этом важно выделить стандартные модули, для реализации которых можно использовать или приспособить стандартные решения и библиотеки стандартных функций. Весьма удобным и наглядным средством представления алгоритма и связей между функциональными модулями является схема алгоритма [12].

  На этом этапе могут уточняться требования к выбору процессора. В задачах, в которых большую часть необходимых действий составляют вопросы управления, может оказаться выгодной ориентация на использование гибридных процессоров с ядром DSP и микроконтроллера *(см. разд. 2.5)*.

- **Выбор языка программирования**. Для программирования процессоров DSP используются язык C и язык ассемблера. Выбор языка программирования определяется многими факторами (процессором, сложностью задачи, наличием компиляторов программ на языке C), в том числе уровнем подготовки разработчика. Некоторые рекомендации по поводу языка будут приведены ниже. Следует указать, что выбранный язык определяет формулировку задачи и алгоритма, необходимые для непосредственного перехода к написанию программы. При использовании языка ассемблера нужна "ассемблерная" формулировка, предполагающая описание на уровне команд и инструкций этого языка.

- **Написание программы**. При разработке программы рекомендуется руководствоваться методами структурного программирования [12]. Основные идеи этого метода включают две составляющие: проектирование сверху вниз и модульное программирование.

  Проектирование сверху вниз предполагает постепенную детализацию всей программы и ее отдельных составляющих и должно использоваться еще на этапе разработки алгоритма.

  Модульное программирование предусматривает разделение программы на логические части — модули и последовательное программирование каждого модуля. Модули организуются как законченные функциональные

блоки, содержащие отдельные независимые процедуры, которые могут быть объединены в общую программу. Модули должны иметь хорошо определенный интерфейс, т. е. четко выделенные входные и выходные параметры, через которые они получают и передают исходные данные и полученные результаты. Модульный принцип построения программы имеет ряд преимуществ, некоторые из которых приведены ниже:

- с относительно небольшими модулями проще работать;
- модуль, являющийся законченным функциональным блоком, допускает индивидуальную отладку и тестирование;
- отдельные модули могут разрабатываться независимо различными специалистами, что ускоряет процесс получения конечного программного продукта;
- модульный принцип позволяет создавать и использовать библиотеки стандартных (типовых) процедур.

Запоминающие устройства современных микропроцессорных систем разбиваются обычно на отдельные блоки (страницы, сегменты, пространства); модульный принцип построения программы позволяет распределять модули по различным блокам памяти. Модулями законченной прикладной программы с функциональной точки зрения могут являться:

- набор подпрограмм и объединяющая их головная программа;
- отдельные процедуры, последовательное соединение которых образует конечный продукт — библиотеки, включающие отдельные подпрограммы, процедуры;
- конечный продукт — библиотеки, включающие отдельные подпрограммы, процедуры, наборы данных, например, коэффициентов для реализации различных вариантов фильтров

и т. д.

Необходимым условием хорошей программы является также ее подробное комментирование и документирование.

- **Получение исполняемого кода программы**. Исполняемый код программы создается с помощью средств, входящих в пакет поддержки разработки. Эти программы описываются в *разд. 9.3*.
- **Отладка и тестирование программы**. Отладка программы представляет собой процесс поиска в ней ошибок (предполагая, что ошибки в программе есть). Если программа работает правильно, то она подлежит тестированию. Однако после тестирования программа должна быть снова подвергнута отладке. Таким образом, тестирование устанавливает наличие ошибки, а отладка позволяет выяснить ее причину [12, 16, 18].

Отладка представляет собой достаточно творческий процесс, и ее задачей является локализация ошибки, т. е. нахождение соответствующего

места в программе, которое вызывает ошибку. Возможности тестирования необходимо закладывать в программу еще на этапе разработки.

Тестирование программы проводится с использованием некоторого набора тестовых данных или сигналов. Тестовые данные должны обеспечивать проверку всех ветвей алгоритма. Для систем цифровой обработки информации, функционирующих в реальном масштабе времени, тестирование в обязательном порядке должно включать измерение времени выполнения всей программы и отдельных ее частей. Для этого могут использоваться специальные программы — *профилировщики*. При тестировании программы проверке могут также подвергаться некоторые показатели системы, связанные с программой, в частности объем кодов и данных (соответственно требуемый объем памяти) и потребляемая мощность. Последний показатель важен для различных переносных систем с батарейным питанием.

Отладка и тестирование программы могут привести (и, как правило, приводят) к необходимости возврата к ранним этапам проектирования для устранения ошибок в постановке задачи, разработке алгоритма, написании программы и т. д. Таким образом, разработка программы так же, как и весь процесс проектирования, является итерационным процессом.

- **Получение загрузочной программы**. На этапе отладки и тестирования программы применяются исполняемые коды программы в форматах, содержащие много "лишней" информации, используемой только для отладки. Эта информация увеличивает объем программы и не нужна при нормальном функционировании разрабатываемой системы. Кроме того, формат загрузочной программы, т. е. программы, загружаемой в память проектируемой системы, зависит от организации используемой памяти. Длина слова в памяти может отличаться от длины слова процессора. Все это приводит к необходимости изменения формата кода программы.

9.2. Языки ассемблера

9.2.1. Особенности языка

Для программирования систем, построенных на основе цифровых сигнальных процессоров различных фирм, применяются язык ассемблера и язык высокого уровня С.

Основные общие особенности языка ассемблера (чаще не совсем точно называемого просто ассемблером; отличие будет оговорено чуть ниже) сигнальных процессоров совпадают с особенностями всех языков подобного типа. Отметим их основные особенности [43].

Языки ассемблера являются машинно-ориентированными языками и, следовательно, для любого типа процессоров существует свой язык. Почти каждая

команда ассемблера эквивалентна команде на машинном языке процессора. Однако программирование на ассемблере, по сравнению с программированием на машинном языке (на уровне машинных кодов), существенно облегчается за счет возможности использования символического обозначения всех элементов программы (кодов операций, адресов ячеек памяти, программ и данных, переменных и констант, операндов и т. д.). Используемые символические обозначения элементов обычно отражают их содержательный смысл. При программировании на языке ассемблера разработчик может не заботиться о распределении памяти или назначении конкретных адресов операндам. В ассемблере допускается оформление повторяющейся последовательности команд как одной макрокоманды. Соответствующие версии языка, допускающие использование макрокоманд, называют *макроассемблерами*. Кроме того, ассемблеры позволяют в той или иной форме использовать при программировании стандартные структуры, например, цикл, разветвление.

При программировании на ассемблере доступны все ресурсы системы и конкретного процессора (регистры, стек, память и т. д.). Это позволяет получать эффективные программы с точки зрения их времени выполнения и объема памяти, необходимого для размещения программы. Проблемы, связанные с конкретной аппаратурой и периферийными устройствами процессора лучше и удобнее решать на языке ассемблера. Однако программирование на ассемблере предполагает знание архитектуры и свойств процессора, т. е. всего того, что входит в понятие "программная модель процессора".

Современные версии языков ассемблера предоставляют программисту ряд возможностей, характерных для языков высокого уровня, таких как условное ассемблирование, организация циклов арифметического и условного типа, т. е. позволяют использовать стандартные логические структуры, рекомендуемые методами структурного программирования.

Следует отметить, что знание языка ассемблера остается необходимым условием получения "хороших" программ и при использовании для программирования процессоров языка С. В этом случае программа на ассемблере, как правило, является промежуточным этапом для получения выходной исполняемой программы. Кроме того, все компиляторы языка С поддерживают включение в программу модулей на языке ассемблера. Это может оказаться полезным для лучшего решения аппаратных проблем и получения эффективной программы с точки зрения времени выполнения и используемой памяти.

9.2.2. Структура программы

Секции и модули

В этом разделе и далее рассматриваются общие особенности языков ассемблера, во всяком случае, языков ассемблера наиболее распространенных ЦПОС, рассматриваемых в данном пособии (фирмы TI, Motorola, ADI, ZiLOG, Lucent, LSI).

Как отмечено в *разд. 9.1*, согласно принципам структурного программирования прикладная программа обычно представляет совокупность нескольких единиц — *модулей*. Физической единицей программы является *программный модуль (файл)*. Логической единицей программы на ассемблере является *секция* или *сегмент*. Функциональный модуль программы (подпрограмма, процедура и т. д.) может совпадать с секцией или объединять несколько секций.

Использование нескольких программных модулей для получения общей программы обусловлено существованием больших и сложных программ, совместной работой нескольких программистов, удобством разработки программы.

Наличие нескольких функциональных модулей обусловлено, прежде всего, принципами структурного программирования, присутствующими даже в неявном виде в любой "хорошей" программе.

Механизм секций поддерживается современными языками ассемблера в связи с использованием в процессорах памяти с достаточно разветвленной (и иногда сложной) структурой, памяти, состоящей из блоков различного назначения и с различными характеристиками. Программу, содержащую несколько секций, можно загружать в разные блоки памяти, распределяя различные секции программы в зависимости от назначения в разные блоки памяти. Например, часть программы, представляющую команды инициализации процессора, которые выполняются один раз при включении системы и время выполнения которых не критично, можно разместить в медленной внешней памяти, а критичные с точки зрения времени выполнения фрагменты — в быстрой внутренней памяти процессора.

Отдельный программный модуль не связан с определенным количеством секций — модуль может содержать и одну и несколько секций, в свою очередь секция способна располагаться в нескольких программных модулях.

Каждая секция может являться отдельным функциональным элементом программы. При этом секция может быть независимо настроена на размещение в определенном месте (в соответствии с требуемыми типом и адресом памяти) используемого пространства памяти. Эта настройка на конкретные адреса так же, как и размещение секций в строгом порядке, осуществляется компоновщиком или непосредственно ассемблером (если он позволяет получить абсолютную программу, т. е. программу с абсолютными адресами).

Секции программы, прежде всего, разделяются по назначению. Как правило, во всех языках используются (иногда этим и ограничиваются) секции трех видов: секции кодов программы, секции инициализируемых данных программы (то есть данных, которые загружаются в память) и секции неинициализируемых данных (то есть данных, появляющихся в процессе выполнения программы). Секции могут иметь имя, задаваемое программистом

(пользователем). Допустимое количество секций в программе различается в разных версиях ассемблера и в некоторых вариантах может быть создано до 65 535 различных секций.

Предложения (строки) ассемблера

Исходная программа на языке ассемблера состоит из последовательности утверждений, которые называют также *ассемблерными строками* или *предложениями*. Предложение ассемблера может располагаться в нескольких строка на экране дисплея или в файле при использовании для продолжения строки специальных символов.

В ассемблерной строке могут быть записаны команды или инструкции процессора, директивы ассемблера, макрокоманды, команды препроцессора и комментарии. Еще раз следует отметить, что здесь рассматриваются общие принципы построения различных ассемблеров и в конкретных вариантах каких-либо из перечисленных выше конструкций может не быть.

Например, в ассемблере TI нет команд препроцессора, которые используются, в частности, а ассемблерах фирм ADI и Lucent. Команды препроцессора применяются для изменения ассемблерного кода. Они начинаются с символа `#`.

Примеры команд:

- `#include` — включает в текст программы содержимое указанного файла;
- `#define` — определяет макрокоманды;
- `#if #else ... #endif` — дает возможность ассемблеру выбора в зависимости от выполнения некоторого условия одной из двух последовательностей предложений ассемблера.

Следует отметить, что действия, выполняемые этими и некоторыми другими командами препроцессора, в ассемблерах иных фирм выполняются директивами ассемблера.

Директивы ассемблера не порождают машинные команды и какие-либо действия в процессоре; они задают структуру программы, сообщают транслятору и компоновщику информацию о том, что им надо делать с командами и данными.

Примеры директив:

- `.section` — определяет начало новой секции;
- `.rsect` — определяет начало секции (ассемблер Lucent);
- `.PRECISION` — определяет число знаков при представлении числа с плавающей точкой (ассемблер ADI);
- `.GLOBAL` — определяет глобальные символы (общие в различных программных модулях);

- `.bss` — резервирует место в памяти для неинициализируемых данных;
- `.data` — определяет начало секции данных;
- `.end` — определяет конец программы на ассемблере;
- `.equ` — присваивает значение некоторому символу (варианты в различных версиях: `.set`, `.word` и др.);
- `.org` — задает абсолютный начальный адрес для секции;
- `.ref`, `.def` — директивы, связывающие и определяющие общие символы в различных программных модулях.

Команды или инструкции процессора порождают машинные команды и выполняются в заданной последовательности во время работы процессора. Различные команды рассмотрены в *главе 6*, а макрокоманды представлены ниже.

Комментарии не влияют на результат трансляции и служат для пояснения и описания программы. Комментарии без изменений переносятся в файл, получаемый после трансляции, — листинг трансляции. Вся строка ассемблера может являться комментарием. В этом случае она начинается каким-либо специальным символом, например `*` (звездочка) или `;` (точка с запятой). Иногда используется стиль языка C, и комментарием считается все, заключенное в символы `/*  */`. В ассемблере фирмы ADI комментарии, записанные на одной строке, могут начинаться символами `//` (стиль C++); комментарии, расположенные в нескольких строках, заключаются в фигурные скобки `{ }`.

Строка (предложение) программы на ассемблере обычно делится на несколько полей, разделенных одним или более пробелами. В строке могут быть следующие поля:

- поле метки;
- поле мнемоники;
- поле операнда;
- поля операции 2 и операнда 2;
- поля перемещения данных (поля параллельных пересылок);
- поле комментария.

В некоторых вариантах могут добавляться дополнительные поля, например поле условия.

Чаще всего строка имеет следующий формат:

```
[метка[:]]  <мнемоника>  [операнд]  [; комментарий]
```

Здесь, как и обычно, при описании системы команд и синтаксиса языка, для обозначения необязательного элемента конструкции использованы квад-

ратные скобки [ ]. Таким образом, метка, операнд и комментарий являются необязательными элементами и могут отсутствовать.

Примеры ассемблерных строк:

```
A_XS    EQU     0
A_PS:   EQU     $100
a_ps:   EQU     $200
a_xs:   EQU     $300
MM2     add     x       ; команда сложения, MM2 — метка
MM3
        add     z,a     ; команда сложения, MM3 — метка
```

Поле метки

Метка в общем случае является необязательным элементом ассемблерной строки. Для некоторых директив наличие метки обязательно. Метка начинается в первой позиции строки и может содержать (как и любой идентификатор) алфавитно-цифровые знаки (A—Z, a—z, 0—9, _ и $), первым из которых должна являться буква.

Если в первой позиции строки стоит пробел или символ ; (точка с запятой), то считается, что метка отсутствует. Однако, если метка заканчивается двоеточием, то она может начинаться с любой позиции. Двоеточие не входит в состав метки.

Разрешено размещать метку отдельной строкой, но оставшиеся части предложения ассемблера могут располагаться на следующих строках без знака продолжения строки.

Поле мнемоники

Поле мнемоники начинается после первого пробела в строке и заканчивается одним или более пробелами. Поле мнемоники содержит одно из следующих утверждений:

- мнемоническое обозначение команды процессора, например, add, move (описание команд процессора *см. в главе 6*);
- мнемоническое обозначение макрокоманды (см. ниже);
- директиву ассемблера.

Директивы ассемблера не порождают машинные команды и какие-либо действия в процессоре; они задают структуру программы, сообщают транслятору и компоновщику информацию о том, что им необходимо делать с командами и данными. Ассемблер проверяет допустимость кодов мнемоник по своей внутренней таблице кодов команд и директив, а затем по таблице макрокоманд. В случае отсутствия используемых мнемоник выдается сообщение об ошибке.

Поле операнда

Поле операнда начинается сразу за пробелом (или пробелами), заканчивающим поле мнемоники, и, в свою очередь, завершается одним или более пробелами. В поле операнда могут быть записаны константы, символы и выражения, состоящие из символов и констант. Эти конструкции языка описаны ниже. Если команда или директива требуют нескольких операндов, то отдельные операнды разделяются запятыми, но не пробелами. Интерпретация поля операнда зависит от мнемоники соответствующей команды или директивы.

В алгебраических ассемблерах *(см. разд. 9.2.3)* для записи команды вместо отдельных полей мнемоники и операндов используется общее поле команды.

Изменение формата записи команды порождает в некоторых случаях и другие отличия в записи строки ассемблера. Так, в ассемблере процессоров платформы C6000 *(см. разд. 2.4)* фирмы TI строка ассемблера имеет вид

```
[метка[:]]  [знак параллельной операции]  [условие]  <мнемоника>
[исполнительный элемент]  [операнд]  [; комментарий]
```

Такая структура строки ассемблера вызвана сложной конструкцией команды данных процессоров, в которой указываются параллельно производимые операции, может выполняться проверка некоторого условия и должен указываться один из 8 исполнительных функциональных элементов, который будет производить операцию.

9.2.3. Мнемонические и алгебраические ассемблеры

В языках ассемблера используются два способа (вида) записи команд процессора: *алгебраический* и *мнемонический*, и, соответственно, существуют алгебраический и мнемонический ассемблеры. Например, трехоперандная команда сложения может быть записана следующим образом:

Мнемоническая форма	Алгебраическая форма
`ADD  Xmem, Ymem, Acx`	`ACx = Xmem + Ymem`

При мнемонической форме для записи команды применяются два поля — поле мнемоники команды (`ADD`) и поле операндов (`Xmem`, `Ymem`, `Acx`). (Могут быть команды комбинированные, осуществляющие несколько действий с несколькими полями операндов.) Мнемоника команды указывает, что надо выполнять над операндами, и расшифровывается в описаниях и списках команд. Как правило, мнемоники представляют сокращения английских слов (или полные слова), обозначающих определенные действия. Например, перемещение данных (move) часто обозначается мнемоникой `MOV`.

При алгебраической форме записи команда представляет собой выражение — "компонент языка, определяющий способ вычисления некоторого значения" [40]. В выражения входят операторы и операнды. Оператор является символом операции.

Примеры операторов:

- `*` — умножение;
- `/` — деление;
- `%` — операция по модулю;
- `+` — сложение;
- `-` — вычитание;
- `<<` — левый сдвиг;
- `>>` — правый сдвиг;
- `&` — поразрядная операция И;
- `|` — поразрядная операция ИЛИ.

Операнд определяет объекты, над которыми производятся операции. В качестве таких объектов могут быть значения (константы, переменные) или в какой-либо форме адреса конкретных значений (регистры процессора, ячейки памяти). Соответственно при алгебраической записи с помощью операторов прямо указываются действия, которые необходимо выполнить над операндами.

Приведем в качестве примеров эквивалентные записи команд для процессора TMS320C55 в мнемонической и алгебраической формах ассемблера.

Мнемоническая форма	Алгебраическая форма
Операции сложения:	
`ADD [src,] dst`	`dst = dst + src`
`ADD k4, dst`	`dst = dst + k4`
`ADD ACx << #SHIFTW, Acy`	`ACy = ACy + (ACx << #SHIFTW)`
`ADD Xmem, Ymem, ACx`	`ACx = (Xmem << #16) + (Ymem <<#16)`
`ADD K16, Smem`	`Smem = Smem + K16`
Операции сдвига и инициализации памяти:	
`MOV Cmem, Smem`	`Smem = Cmem`
`MOV K8, Smem`	`Smem = K8`
Операции умножения с накоплением и округлением результата:	
`MAC[R] ACx, Tx, ACy[, ACy]`	`ACy = rnd(ACy + (ACx * Tx))`
`MAC[R] ACy, Tx, ACx, ACy`	`ACy = rnd((ACy * Tx) + ACx)`
`MACK[R] Tx, K8, [ACx,] Acy`	`ACy = rnd(ACx + (Tx * K8))`

В некоторых версиях поддерживаются варианты записи выражений, свойственные языкам высокого уровня, например эквивалентны записи операции сложения:

Мнемоническая форма	Алгебраическая форма
`AC0 = AC0 + *AR4`	`AC0 + = *AR4`

Алгебраическая форма записи команд на первый взгляд представляется более общей, не зависящей от конкретного процессора. Однако могут быть особенности записи операторов и особенно обозначений операндов в ассемблерах различных процессоров и фирм.

Кроме того, при записи выражений для некоторых операций фактически используется мнемоника этих операций. Например, эквивалентные команды процессора TMS320C55 в мнемонической и алгебраической формах ассемблера имеют вид:

Мнемоническая форма	Алгебраическая форма
Помещение значения в стек	
`PSH  src`	`push(src)`
Извлечение из стека	
`POP  dst`	`dst = pop()`
Повторение команды	
`RPT  k16`	`repeat(k16)`

Тонкости правил записи и разделения операндов и различных частей команды могут отличаться и приводятся в описаниях конкретных языков.

Для процессоров платформы C5000 фирмы TI разработаны два варианта ассемблеров с мнемонической и алгебраической формой записи. Ассемблеры для процессоров фирмы ADI и Lucent используют алгебраическую форму записи.

Наибольшие особенности во всех смыслах, в том числе и в форме записи команд, имеет ассемблер фирмы ADI.

Примеры записи команд:

```
[IF условие] AR = xop + yop;    // команда сложения; AR, xop, yop — регистры
[IF условие] MR = xop * yop;    // команда умножения; MR, xop, yop — регистры
```

Большинство команд ассемблера этой фирмы так же, как приведенные выше, могут сопровождаться необязательным условием ее выполнения.

Метка с помощью, которой отмечается расположение команды для последующей ссылки, размещается в начале строки команды или на предшествующей строке. Метка заканчивается символом : (двоеточие).

9.2.4. Основные конструкции ассемблера

При записи операндов и меток используются различные конструкции языка. Рассмотрим основные из этих конструкций.

Константы

Константа является величиной, которая не изменяется в течение всего времени выполнения программы. Константы бывают числовыми и строчными.

Числовые константы могут быть записаны в одной из трех систем счисления — двоичной, десятичной или шестнадцатеричной. Отрицательные константы ассемблером записываются в дополнительном коде (для положительных чисел представление в дополнительном и прямом кодах совпадают, *см. главу 3*). Длина внутреннего представления константы определяется возможностями процессора и директивами/командами, в которых указана константа. Например, для 24-разрядных процессоров ассемблер может использовать внутреннее представление констант длиной 24 и 48 двоичных разрядов (6 и 12 шестнадцатеричных разрядов) — одно или два слова процессора соответственно в зависимости от значения.

Система счисления обозначается спецификатором, который может либо предшествовать константе, либо записываться за ней. В качестве спецификаторов применяются:

- `H`, `h`, `$` — шестнадцатеричная система;
- `D`, `d` — десятичная система;
- `B`, `b`, `%` — двоичная система;
- `O` (`o`) — восьмеричная система.

В ассемблере ADI к спецификатору, стоящему перед числом, добавляется знак `#`.

Примеры записи констант в ассемблере Motorola:

- `%100001` соответствует 33;
- `$77` соответствует 119;
- `29` соответствует 29.

Дробная десятичная константа может быть записана в одной из двух форм: дробное число с разделительной десятичной точкой или экспоненциальная форма с записью основания системы счисления `E` (`e`).

Примеры записи дробных констант в ассемблере ZiLOG:

- `1.2`
- `2.e-5`
- `0.5E2`

Строчная (знаковая) *константа* записывается в виде последовательности из одного или нескольких знаков, заключенных в одиночные кавычки. Внутреннее представление знаковой константы соответствует 8-разрядным значениям кода ASCII для каждого знака. Если в константе определено меньше знаков, чем необходимо для заполнения 24-разрядного слова (или двойного слова), при трансляции слева дописываются нули вместо недостающих знаков.

Констант в ассемблере ZiLOG:

- `'a'` — константа эквивалентна 97 (десятичная) или 61 (шестнадцатеричная);
- `'C'` — константа эквивалентна 67 (десятичная) or 43 (шестнадцатеричная);
- `'\'` — константа эквивалентна 39 (десятичная) or 27 (шестнадцатеричная).

Цепочки знаков — строки

Некоторые директивы ассемблера используют в качестве операнда строку или цепочку, которая представляет собой последовательность знаков, заключенных в кавычки. Если необходимо включить в строку кавычку, то в это место включаются две последовательных кавычки. Длина цепочки определяется директивой, в которой используется цепочка.

Внутреннее представление знаков строки соответствует коду ASCII.

Наример, `'ONE SIMBOL'` определяет цепочку, состоящую из 10 знаков.

Цепочки знаков используются в качестве имен файлов, имен секций и т. д.

Символы (символические имена)

Символы применяются в качестве меток или идентификаторов в операндах и выражениях. Символ есть строка буквенно-цифровых знаков (A—Z, a—z, 0—9, $ и _) длиной до 512 символов. В качестве знака в строке не может использоваться пробел. Заглавные и строчные буквы ассемблеры обычно воспринимают как различные (например, символы `ABC` и `abc`), если при трансляции не применяется специальные опция.

Символы, которые используются как метки, становятся идентификаторами адресов ячеек памяти, в которых хранятся кодовые слова программы. Они могут применяться в поле метки только один раз.

Символы, используемые в поле операнда, становятся идентификаторами переменных величин, т. е. идентификаторами адресов ячеек памяти данных, где хранятся эти величины. Эти символы должны быть определены либо директивами ассемблера непосредственно в программном файле, либо в командном файле компоновки.

Символы действуют только внутри некоторого программного модуля, т. е. для каждого символа имеется некоторая область видимости. Соответствующие правила определения и задания области действия символа приводятся в описаниях конкретных версий ассемблера.

Некоторые символы (помимо мнемоник команд и директив ассемблера) являются стандартно определенными или предопределенными, т. е. они зарезервированы ассемблером и не могут использоваться в качестве идентификаторов.

Выражения

Выражением является последовательность констант, символов, функций, объединенных арифметическими операторами и круглыми скобками. Выражение может быть операндом или частью операнда команды процессора или директивы.

Выражение вычисляется при трансляции, т. е. ассемблер может взять на себя задачу расчета фактического значения операнда. Смысл результата вычисления определяется местоположением выражения в программе. Например, результат может являться адресом или значением какой-либо величины (например, величины сдвига). Вычисление выражения подчиняется правилам алгебры и булевой алгебры.

Три главных фактора задают порядок вычисления выражения:

- круглые скобки;
- ранги арифметических операторов;
- направление вычислений.

Выражения обычно вычисляются слева направо, если другой порядок вычисления не определяется круглыми скобками или рангами операций. Внутри скобок направление вычислений также слева направо при равенстве рангов операций.

Примеры выражений будут рассмотрены ниже.

Выражения могут быть абсолютными и относительными (перемещаемыми). Абсолютное выражение состоит из абсолютных величин, например, констант или символов (меток), определяемых в абсолютном режиме трансляции.

Относительное выражение состоит из относительных величин или композиции абсолютных и относительных величин.

Операторы

Оператор является символом операции, которая должна быть выполнена. Примеры некоторых операторов приведены выше. Среди операторов выделяют унарные, которые используются с одним операндом (например, знаки

числа). Ранги операторов задают порядок выполнения соответствующих операций в выражении.

Функции

В некоторых ассемблерах используются *встроенные функции* для поддержки преобразования данных, сравнения строк, математических вычислений и т. д.

Пример математических функции ассемблера Motorola:

- `MAX` — выбор максимального значения;
- `MIN` — выбор минимального значения;
- `RND` — случайное число;
- `SGN` — знаковая функция;
- `SIN` — синус.

Эти и другие подобные функции можно использовать в выражениях ассемблера.

9.2.5. Средства макроассемблера

Язык макроассемблера отличается от языка ассемблера наличием макросредств, облегчающих составление программ. К таким средствам относят макрокоманды, средства организации повторений (циклов) отдельных команд и блоков команд, организацию условного ассемблирования (трансляции) и т. д.

Следует отметить, что макросредства не создают другой конечный продукт, но избавляют программиста от некоторой рутинной работы, например, от многократной записи одной команды при необходимости ее повторения.

Организация условного ассемблирования может производиться как с помощью макросредств, так и с помощью директив условного ассемблирования.

Макрокоманды

При разработке программ часто возникает необходимость в повторении (иногда с модификациями, при других параметрах) некоторой группы команд. Такие группы повторяющихся команд можно оформить как процедуру или подпрограмму. Макроассемблер позволяет применить иной вариант краткой ссылки на часто используемую последовательность команд: такую последовательность можно определить один раз как большую команду — макрокоманду с уникальной мнемоникой, не совпадающей с мнемоникой команд процессора. Макрокоманду после определения ее в начале программы можно рассматривать как входящую в систему команд процессора.

Таким образом, использование подпрограмм (процедур) и макрокоманд позволяет сократить длину исходной программы и повысить ее "читаемость". Но между этими конструкциями языка имеются существенные различия.

При использовании подпрограмм в текст исходной программы включается команда CALL, которая реально будет выполняться в процессоре. При ее обработке происходит передача управления в другое место памяти программ.

При использовании макрокоманды ее имя (мнемоника) ассемблером заменяется при трансляции на последовательность команд, определяемых этой мнемоникой. Вышеуказанное приводит к тому, что время выполнения программы при использовании макрокоманд сокращается (по сравнению с использованием подпрограмм) за счет отсутствия команд переходов. При этом длина выполняемой программы (и место, занимаемое программой в памяти) увеличивается за счет многократного повторения команд процессора, входящих в макрокоманду.

Макрокомандами можно также объявить часто используемые программистом типовые последовательности действий (процедуры) и создать из них библиотеку. При использовании библиотеки нет необходимости включения текстов макрокоманд в каждую исходную программу.

Макрокоманды могут быть вложенными друг в друга.

Макроопределение и макровызов

Макроопределением является набор исходных ассемблерных строк, включающий команды процессора, макрокоманды и директивы ассемблера, который должен выполняться процессором в соответствии с определяемой макрокомандой.

Макроопределение обычно располагается либо в начале исходной программы (во всяком случае, до вызова макрокоманды), либо в библиотеке макроопределений, вызываемой специальной директивой, например MACLIB.

Макроопределение обычно имеет следующую структуру:

```
<имя макрокоманды>  MACRO  [список параметров]  [;комментарий]

    последовательности команд и директив — тело макрокоманды

ENDM
```

Директива начала (MACRO) и директива конца макро (ENDM) являются *макродирективами*, специфичными в каждом ассемблере, которые открывают и завершают макроопределение соответственно.

Имя макрокоманды размещается в поле метки, т. е. начинается с первой позиции ассемблерной строки. Последовательность параметров макрокоманды [список параметров] записывается через запятую в поле операнда ассемблерной строки. Параметры в макроопределении являются формальными и заменяются на фактические после макровызова. Макроопределение

может не иметь параметров. Через параметры в макроопределении могут задаваться любые операнды и метки.

Макровызов имеет следующую структуру:

```
[метка]  <имя макрокоманды>  [список параметров]  [;комментарий]
```

Имя макрокоманды, расположенное в поле мнемоники команды, должно совпадать с именем, расположенным в поле метки соответствующего макроопределения.

Последовательность величин (или символов), являющихся фактическими параметрами, записывается через запятую в поле операнда ассемблерной строки. Параметры по количеству и порядку следования должны соответствовать формальным параметрам в макроопределении. Каждый параметр может быть константой или выражением любого типа, распознаваемого ассемблером.

Приведем пример (ассемблер Motorola). Представленная ниже последовательность строк определяет макрокоманду с именем `INCX1`:

```
INCX1  MACRO
         move  X:(R2),X0
         move  X0,A0
         inc   A
         move  A0,X:(R2)
       ENDM
```

Макрокоманда `INCX1` будет вызвана при помещении в текст исходной программы строки

```
MM1  INCX1
```

с именем макрокоманды `INCX1` в поле мнемоники. Макроассемблер поменяет эту строку на команды процессора, т. е. на последовательность команд, записанных в макроопределении.

Макрокоманда `INCX1` имеет ограниченное применение, т. к. может быть использована только с ячейкой памяти `X:(R2)`. Допустимо расширение сферы применения макрокоманды, и она может использоваться с любой ячейкой памяти, если ввести формальный параметр.

Макроопределение:

```
INCX3  MACRO  ARG       ; передача всего операнда
         move  ARG,X0    ; параметр передается как текст
         move  X0,A0
         inc   A
         move  A0,ARG
       ENDM
```

При использовании макровызова с фактическим параметром, например, `MM3 INCX3 X:(R2)` этот макровызов будет заменяться последовательностью команд, в которой можно использовать произвольный операнд.

9.2.6. Средства организации стандартных структур

Ассемблеры обычно имеют набор средств, которые позволяют при написании программ избавиться от некоторых формальных операций типа повторения команд процессора и использовать такие стандартные логические структуры, как циклы разных типов и ветвления, рекомендуемые методами структурного программирования. Ассемблер генерирует коды, соответствующие стандартным структурам, и вставляет их в исходный текст программы. Это позволяет получать более эффективные программы и улучшает их читаемость.

К средствам организации стандартных структур можно отнести перечисленные ниже.

Команды повторения

Процессоры имеют команды, позволяющие аппаратно организовать повторение одной или нескольких команд, т. е. осуществлять арифметический цикл.

Например, повторение одной команды заданное число раз:

```
REP  #25      ; DSP56XXX Motorola
RPT  k16      ; TMS320C5000 TI
```

Повторение блока команд:

```
RPTB  <метка> ; TMS320C5000 TI
```

В соответствии с указанной конструкцией будут повторяться все команды между данной и отмеченной меткой `<метка>`.

Команды организации цикла

Некоторые процессоры имеют команды, позволяющие организовать цикл, например:

```
DO <метка> UNTIL <условие> ; ADSP21xx  ADI
DO .. ENDDO                ; DSP56XXX Motorola
```

Циклы различного типа можно осуществлять с помощью директив ассемблера. Например:

```
.FOR   TO    DO     ENDF; ассемблер Motorola
$REPEAT               ; ассемблер ZiLOG
.$WHILE                ; ассемблер ZiLOG
```

Организация ветвлений

Для организации ветвлений в программе можно использовать директивы ассемблера. Например:

```
#IF  #ELSE  #ENDIF   ; ассемблер ADI
.IF  .ELSE  .ENDIF   ; ассемблер TI
.IF  .ELSE  .ENDI    ; ассемблер Motorola
.$I  .$ELSE .$ENDIF  ; ассемблер ZiLOG
```

9.2.7. Совместимость ассемблеров различных процессоров

Как уже отмечалось, языки ассемблера являются машинно-ориентированными языками и, следовательно, для каждого типа процессоров существует свой язык. Фирмы — производители процессоров выпускают ЦПОС семействами, совместимыми на уровне команд, и эти семейства имеют общий ассемблер. Помимо собственно ассемблера общим является весь пакет программных средств поддержки разработки, включающий также С-компилятор и другие продукты. Приведем примеры подобных семейств.

Семейства процессоров фирмы TI

- TMS320C1x/C2x/C24x/C5x, TMS320C54x, TMS320C55x, TMS320C3x/4x

  Процессоры совместимы на уровне ассемблера, но не объектных кодов. Выбор типа процессора для генерации нужного кода осуществляется директивой `.version`.

- TMS320C6000(C62x/C64x/C67x)

  Платформа процессоров C6000 включает как процессоры с фиксированной точкой C62x и C64x, так и процессоры с плавающей точкой C67x. Однако ассемблер (так же, как С-компилятор) — общий. Это объясняется тем, что процессоры обладают практически общим ядром ЦПУ, и C67x и C64x представляют некоторые расширения C62x. Все команды, производимые в C62x (фиксированная точка), выполняются также в C67x и C64x. В C67x добавлены команды обработки данных в формате с плавающей точкой. В C64x расширен набор команд процессора с фиксированной точкой, прежде всего за счет введения и обработки новых типов данных. Идентификация типа процессора производится путем записи соответствующих кодов в регистр управления процессора.

Следует отметить, что ассемблеры различных семейств процессоров TI имеют много общего. Это касается конструкций языка, состава директив, пра-

вил их записи, опций, состава пакетов и т. д. Отличия относятся, главным образом, к системе команд процессоров.

Семейства процессоров фирмы Motorola

Фирма Motorola выпускает несколько семейств процессоров с фиксированной запятой [34]: DSP5600, DSP56300, DSP56600, DSP56800.

Для программирования этих процессоров используется общий язык ассемблера, описанный в [34]. Однако программа ассемблеров у каждого семейства своя (asm56k.exe, asm56300.exe, asm56600.exe, asm56800.exe), так же, как и свои компиляторы языка С. Некоторые другие программы, входящие в пакеты поддержки разработки для различных семейств, одинаковы, например компоновщики, библиотекари.

Семейства процессоров фирмы ADI

Общее программное обеспечение имеют 16-разрядные процессоры с фиксированной запятой ADSP-217x, ADSP-210x, ADSP-216x и ADSP-2115.

Семейство процессоров ADSP-218x, ADSP-219x обладает общим программным обеспечением, включая ассемблер, заключенным в интегрированную среду VisualDSP, в состав которой входят и отладчики.

Семейства 32-разрядных процессоров ADSP-21000 SHARC и ADSP-TS001 TigerSHARC имеют различное программное обеспечение, также заключенное в интегрированную среду VisualDSP.

ZiLOG

Процессоры фирмы ZiLOG семейств Z893x3, Z893x1, Z89C00 имеют общий ассемблер ZMA; идентификация типа процессора производится с помощью опции при вызове ассемблера.

Lucent Technologies

Семейства процессоров DSP1600 и DSP16000 поддерживают различные пакеты программного обеспечения, причем в пакет обеспечения процессоров DSP1600 не входит С-компилятор.

Следует также сказать, что имеется определенная тенденция к унификации ассемблеров различных фирм; в частности весьма похожи ассемблеры фирм ZiLOG и TI.

9.3. Получение исполняемой программы. Состав пакетов программного обеспечения процессоров DSP

Структура и состав пакетов программного обеспечения поддержки разработки (средств разработки) соответствуют основным этапам создания программы. В состав пакета входят программы, обеспечивающие проведение всех этапов разработки. Примерный состав пакета разработки приведен на рис. 9.2.

Рис. 9.2. Состав типового пакета программного обеспечения поддержки разработки

Обобщенный процесс подготовки выполняемой программы (этапы написания программы и получения исполняемого кода) для процессоров при использовании языка ассемблера показан на рис. 9.3. На этом рисунке прямоугольниками обозначены системные программы, входящие в пакет программного обеспечения подготовки программ пользователя. Исходные программы (файлы) и файлы, получаемые в результате работы системных программ, соответствуют на рис. 9.3 овалам. Таким образом, прямоугольник (символ "процесс") отражает как некоторую программу, так и процесс преобразования, осуществляемый этой программой.

Исходная программа подвергается с помощью собственно ассемблера трансляции, в результате которой получается промежуточная объектная программа. В процессе трансляции ассемблер обнаруживает синтаксические ошибки — нарушения правил языка — и выдает листинг трансляции, содержащий исходную программу, коды команд, диагностическую информацию и сообще-

ния об ошибках. При наличии ошибок необходимо вернуться к исходной программе для ее коррекции.

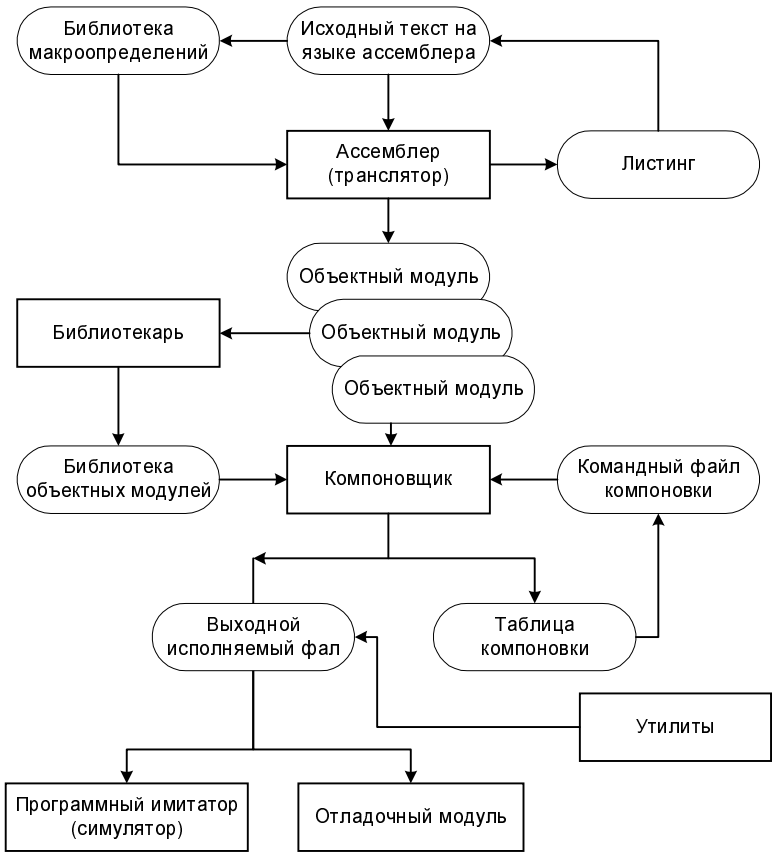


Рис. 9.3. Процесс подготовки исполняемой программы

Языки ассемблера ЦПОС (так же, как и другие подобные языки) допускают использование макрокоманд, которыми можно объявить повторяющуюся последовательность команд процессора. Описания макрокоманд можно включать в исходный текст программы. Допустимо также создать библиотеку макрокоманд, включающую типовой набор операций для программ конкретного пользователя. Макрокоманды из библиотеки разрешено использовать в исходных программах без дополнительного описания при подключении библиотеки к трансляции.

Для создания библиотеки макрокоманд служат специальные программы — библиотекари или архиваторы. *Архиватор* предоставляет возможность собирать группу файлов в единственный файл архива, называемый *библиотекой*.

Дополнительно архиватор позволяет изменять библиотеку, удаляя, заменяя, извлекая или добавляя компоненты. Одно из наиболее полезных приложений архиватора — формирование библиотеки объектных модулей.

В некоторых случаях процесс получения объектной программы более сложен.

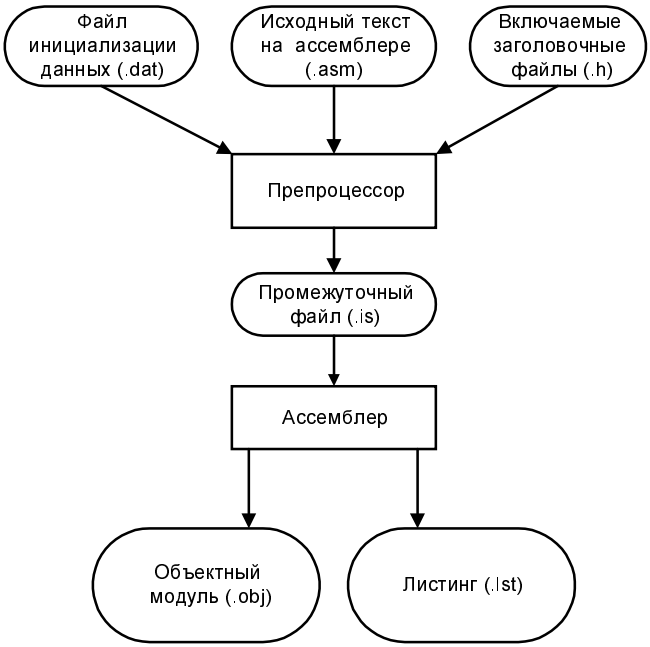


Рис. 9.4. Подготовка объектного модуля при использовании ассемблера ADI

На рис. 9.4 показан процесс получения объектного модуля при использовании ассемблера фирмы ADI. Этот ассемблер включает препроцессор, на вход которого подаются файлы инициализации данных, собственно текст программы и включаемые файлы. Последние могут содержать описания макрокоманд и файлы описания целевого процессора, на который ориентируется разрабатываемая программа.

9.3.1. Абсолютные и перемещаемые программные модули

В процессе подготовки выполняемой программы после трансляции можно получить перемещаемые и абсолютные программные модули, состоящие из перемещаемых и абсолютных сегментов кодов и данных. Определим эти, а также некоторые другие используемые понятия.

Объектный модуль (объектный код) — код, полученный в результате трансляции на машинный язык или близкий к нему язык программы, записанной на некотором исходном языке.

Выполняемый загрузочный модуль — программа, представленная в виде, пригодном для загрузки в память системы и ее выполнения процессором. Загрузочный модуль содержит программу на машинном языке и некоторую служебную информацию, необходимую для ее размещения в памяти.

Абсолютный адрес — число, однозначно указывающее положение данных или кодов в памяти.

Абсолютный (неперемещаемый) программный модуль — программный модуль, использующий абсолютные (фактические) адреса.

Перемещаемый (относительный) программный модуль представляет собой программу, которая может быть настроена на загрузку и выполнение в любой области памяти. В такой программе все требуемые адреса выражаются относительно общей точки отсчета — начала программы. Настройка перемещаемых модулей на абсолютные адреса производится компоновщиком в процессе подготовки выполняемой (загрузочной) программы. При этом все относительные адреса заменяются абсолютными. Использование перемещаемых программных модулей позволяет получать единую выполняемую программу из нескольких программных модулей или секций, если она содержит несколько секций.

9.3.2. Компоновка

Следующим после трансляции этапом подготовки выполняемой программы является *компоновка* одного или (как правило) нескольких самостоятельно разработанных и оттранслированных программных модулей с помощью компоновщика (см. рис. 9.3).

Компоновщик (редактор связей или линкер) решает следующие задачи.

- Объектные программы, полученные в результате трансляции, не привязаны к каким-либо определенным адресам ячеек в памяти (команды, константы и переменные объектной программы не привязаны к каким-либо конкретным адресам памяти), т. е. они являются перемещаемыми и могут выполнятся при размещении в разных местах памяти. Первой задачей компоновки является привязка или настройка перемещаемых модулей (объектной программы), полученных после трансляции, к определенным абсолютным адресам памяти. При этом выходная выполняемая программа может быть получена в виде нескольких выходных секций (не обязательно совпадающих с входными), каждая из которых может быть независимо предназначена для размещения или выполнения в различных областях и типах памяти.

- Второй задачей компоновки является объединение нескольких программных модулей в одну исполняемую программу. Процесс объединения модулей не является чисто механическим процессом соединения.

  Компоновщик производит объединение отдельных объектных модулей, которые транслировались независимо друг от друга. При этом возможно использование библиотеки объектных модулей. Во время объединения модулей проверяются и устанавливаются необходимые связи между ними через общие константы, переменные и адреса переходов (метки). Объединение модулей и секций производится в заданном формате, т. е. модули могут состыковываться в определенном порядке и расставляться с промежутками (или "дырками") в памяти. При этом промежутки могут заполняться некоторыми задаваемыми величинами.

  При объединении модулей в одну программу возникают взаимные обращения (ссылки) к символьным компонентам (переменные, константы, метки, адреса и т. д.), используемым в различных модулях. Именно межмодульные обращения объединяют различные модули в единую программу. Информация о возможных межмодульных обращениях вносится в каждый модуль при его написании с помощью директив ассемблера и сохраняется в объектной программе после трансляции. Конкретные численные значения символьных компонентов могут задаваться как при написании отдельных модулей, так и на этапе компоновки.

  Ассемблер и компоновщик имеют специальные директивы, позволяющие объединять модули с общими компонентами.

- При компоновке возможно получение на выходе промежуточной перемещаемой программы, которую можно использовать в качестве одного из исходных модулей при следующих этапах компоновки.

  Компоновщик, помимо основной выполняемой программы, выдает файл (таблицу компоновки), показывающий результат компоновки: порядок следования и схему распределения исходных модулей по конкретным адресам, а также другую диагностическую информацию.

Компоновщик решает свои задачи в соответствии с указаниями, вводимыми программистом. Эти указания могут задаваться в виде опций и ключей командной строки вызова компоновщика, либо указываться командным файлом компоновки. Командный файл (или опции) должен отражать структуру и состав проектируемой системы, в части конфигурации памяти (блоки памяти и распределение адресов), состава устройств ввода/вывода и т. д., а также содержать указания (директивы) о последовательности соединения входных секций в выходные и распределении выходных секций исполняемой программы по блокам памяти. Таким образом, для компоновщика необходима информация, подготовленная на первом и втором этапах разработки программы, описанных в *разд. 9.1*. При отсутствии подобных указаний компоновщик по умолчанию распределяет программу в соответствии с некоторой стандартной картой памяти.

Библиотека объектных модулей, которая может содержать типовой набор процедур: инициализация вектора прерываний, перемещение программ и данных и т. д., создается программой, называемой *библиотекарем*.

В языках некоторых процессоров компоновка не является обязательной. Например, ассемблер процессоров фирмы Motorola позволяет получить абсолютную исполняемую программу непосредственно после трансляции без проведения компоновки.

9.3.3. Отладка и тестирование программы

Полученная в результате компоновки выполняемая программа в дальнейшем может использоваться различным образом в зависимости от этапа подготовки конечного программного продукта и его назначения.

Прежде всего, производятся отладка и тестирование программы. При этом используются программные имитаторы (симуляторы), отладочные модули и эмуляторы. В состав имитаторов и отладчиков, как правило, входят и средства профилирования. Например, в отладчиках фирмы TI можно измерять время выполнения (в периодах тактовой частоты), количество обращений к функциям, количество циклов для определенной области программы на ассемблере или С, функции языка С. Отладочное программное обеспечение может включать большое количество различных программ, объединенных визуальными оболочками, в том числе и операционные системы реального времени, драйверы аналоговых устройств, интегрированные среды для моделирования и выполнения программ ЦОС.

Процесс отладки программ описывается в *главе 10*.

После неоднократного выполнения отладки и тестирования получается вариант программы, удовлетворяющий заданным требованиям. Для получения загрузочной программы может оказаться необходимым выполнить одно из следующих действий:

- преобразование программы с помощью конвертора из COFF-формата в один из других форматов для последующего использования в устройствах, работающих в форматах, отличных от COFF;
- при необходимости преобразование длины слова программы в соответствии с длиной слова используемых кристаллов памяти;
- запись полученной программы в память с помощью программатора (для последующего использования в реальных системах).

На этапе написания и отладки программы используются также различные утилиты для манипуляций с объектными файлами. В частности для преобразования объектных файлов в абсолютные и относительные на языке ассемблера, получения таблиц переменных и перекрестных ссылок на переменные и т. д.

В качестве примера можно привести состав пакета обеспечения средств разработки процессоров семейства DSP56300 фирмы Motorola:

- Asm56300 — ассемблер;
- dsplnk — компоновщик;
- dsplib — библиотекарь;
- Run563 — программный имитатор (симулятор);
- cldinfo — информация о размере памяти объектного файла формата COFF;
- cldlod — конвертер из формата COFF в формат LOD;
- cofdmp — утилита дампа памяти файла формата COFF;
- srec — конверсионная утилита.

9.3.4. Библиотеки функций и информационная поддержка

Для облегчения процесса разработки систем цифровой обработки сигналов фирмы — производители процессоров, а также другие фирмы разрабатывают библиотеки алгоритмов и подпрограмм, ориентированных на обработку сигналов. Примером такой библиотеки может служить TMS320C54x DSPLIB (TI) — оптимизированная библиотека функций обработки сигналов на языке С для процессоров TMS320C54X. Она включает более 50 универсальных подпрограмм обработки сигналов. Эти подпрограммы ориентированы на применение в приложениях, работающих в реальном масштабе времени, для которых скорость выполнения программ весьма важна. Используя эти средства, можно достигать высоких скоростей выполнения программ. Подпрограммы, включенные в библиотеку, делятся на несколько различных функциональных групп:

- БПФ;
- фильтрация;
- адаптивная фильтрация;
- корреляция;
- математика;
- тригонометрия;
- матричные вычисления.

Аналогичные библиотеки предоставляет разработчикам фирма Motorola, например "Suite56. Библиотека программного обеспечения DSP". Библиотека содержит коды программ цифровой обработки (DSP) и документацию для ряда прикладных программ, включая обработку речи, цифровую связь, фильтрацию, преобразование и обработку изображений. Основные библио-

течные подпрограммы включают арифметические операции, матричные операции, логарифм, и алгоритмы сортировки.

Библиотеки функций можно найти на сайтах фирм, адреса которых приведены в *приложении 3*. Кроме того, на сайтах фирм можно найти большое количество различных программ для решения конкретных задач. Алгоритмы решения различных задач приводятся в статьях типа "Application Report" (фирма TI) или "Application Notes" (фирма AD), которые можно также обнаружить на фирменных сайтах. Например, статьи на сайте фирмы AD на тему:

- использование библиотеки C-FFT для процессоров 21xx;
- программирование на C семейства ADSP-2100, руководство, примеры;
- выбор и использование FFT's для ADSP-21xx;
- программирование таймера на C для ADSP-21xx.

На сайте TI для процессоров платформы TMS320C5000 в разделе "Application" (Применение) приведено порядка сотни различных программ для построения модемов, кодеров, декодеров различных стандартов, генераторов и т. д.

Собственные оптимизированные библиотеки функций ЦОС предлагают также другие фирмы, например "АО Инструментальные системы".

Важным моментом при разработке программного обеспечения для процессоров является *информационная поддержка*. Как правило, на сайтах фирм имеется полная документация по самим ЦСП и пакетам программ поддержки разработки, материалы по их применению, отчеты об использовании, а также готовые примеры алгоритмов и программ, которые помогают правильно ориентироваться в проблеме и существенно облегчают работу за счет использования готовых стандартных решений. Список URL-адресов фирм приведен в *приложении 3*.

9.3.5. Использование интегрированных оболочек для подготовки и моделирования программ ЦОС

Для решения все усложняющихся проблем подготовки программного обеспечения фирмы предлагают для разработчика интегрированные наборы программ, позволяющие существенно ускорить и облегчить процесс разработки. Эти наборы включают в себя как аппаратные, так и программные элементы.

Аппаратным элементом является набор отладочных модулей — прототипов с продуманной архитектурой и широкими функциональными возможностями. Их использование позволяет оценить разрабатываемую систему и отработать ее решения на живом устройстве на самых ранних стадиях разработки.

Примером интегрированного набора фирмы TI является CodeComposer Studio — интегрированная среда разработчика, имеющая удобный графический интерфейс в сочетании с мощными средствами конфигурирования и отладки, ориентированными на ЦОС приложения. При ее разработке было достигнуто оптимальное сочетание визуальных средств разработки с мощностью и возможностями продукта. Система предназначена для обеспечения максимального удобства разработчику. Ее применение позволяет в несколько раз сократить сроки разработки и отладки ЦОС-систем. Являясь полностью функционально законченным продуктом, Code Composer Studio позволяет, не выходя из отладочной среды, редактировать, компилировать и отлаживать программы. Мощные возможности анализа и отладки систем в реальном времени дают возможность отлаживать и анализировать поведение системы без остановки процессора. Она включает DSP/BIOS — библиотеку планировочных, инструментальных и коммуникационных функций, обеспечивающих анализ и обмен данными в реальном времени; аппаратные средства эмуляции и оценки, которые дают возможность отладки программ непосредственно на кристалле и оценки производительности на ранних этапах цикла разработки.

Фирма AD предлагает аналогичный по назначению пакет VisualDSP++, который является удобной в работе средой разработки программ. Он включает интегрированную среду разработки (IDE) и отладчик. VisualDSP++ допускает управление проектом от начала до конца внутри единственного интерфейса и поддерживает SHARC DSP и TigerSHARC семейства DSP на платформе Windows 9*x*, Windows NT и Windows 2000. Поддержка для ADSP-218X и ADSP-219x семейства DSP доступна с марта 2001 года.

Интегрированные среды объединяют в себя мощные средства для инженерных и научных расчетов и визуализации полученных данных. Многие современные пакеты поддерживают визуальное программирование на основе блок-схем.

9.3.6. Matlab

Одним из средств разработки программного обеспечения для процессоров DSP является такой инструмент, как Matlab (фирма MathWorks Inc.) [29, 31]. В последней версии этого пакета возможно использование составляющих "Motorola DSP Developer's Kit" (Комплект разработчика ЦПОС Motorola) и "Developer's Kit for Texas Instruments DSP" соответственно для процессоров фирм Motorola и TI.

Matlab включает пакеты DSP Blockset и Simulink, позволяющие проектировать и отлаживать различные цифровые устройства (в частности цифровые фильтры). Пакет DSP Blockset с помощью интерфейса Real-Time Workshop позволяет генерировать ANSI C-код по результатам моделирования DSP в Simulink. Полученные коды можно использовать в прикладных программах.

Пакеты Developer's Kit позволяют разрабатывать и моделировать системы, основанные на особенностях конкретных процессоров (семейства DSP Motorola 56xxx, семейства TI C5000 и C6000) в среде Matlab и осуществлять двустороннюю связь с интегрированными оболочками пакетов ЦПОС. Средство Developer's Kit дает возможность реализовать алгоритмы, используя ассемблеры DSP или коды C, и выполнять полученные объектные программы непосредственно в пределах Matlab или Simulink на выбранном имитаторе DSP. Применению Matlab для решения задач ЦОС посвящен цикл статей [9].

9.4. Рекомендуемый путь построения программы

Для достижения лучшей эффективности исполняемой программы фирма TI рекомендует разрабатывать ее в соответствии с алгоритмом, схема которого приведена на рис. 9.5. Этот алгоритм носит достаточно общий характер, не использует никаких особенностей процессоров TI и, естественно, может применяться и для процессоров других фирм.

На первом этапе разрабатывается программа на стандартном языке C, при этом знание каких-либо особенностей процессора не является обязательным. После компиляции полученная исполняемая программа подвергается отладке и тестированию, целью которых является проверка, удовлетворяет или нет программа заданным условиям и показателям. Одним из основных показателей программы является время выполнения (это особенно важно для систем, работающих в реальном масштабе времени). Проверка времени выполнения может проводиться с помощью стандартных или специально разработанных средств профилирования.

Другими важными показателями могут являться объемы кодов и данных, определяющие требуемый объем (и соответственно стоимость памяти), и потребляемая мощность, которая также связана со временем выполнения кодов и их содержанием. Последний показатель особенно важен для систем с батарейным питанием.

Если заданные условия выполняются, то на этом процесс разработки программы может быть закончен. Если же заданные условия не выполняются, то необходима дальнейшая работа над программой.

На втором этапе работы подключаются средства оптимизации алгоритма и компиляторов языка C для процессов ЦПОС. С целью оптимизации алгоритма может производиться его трансформация, направленная на использование комбинированных циклов вместо последовательности простых, возможность организации параллельных вычислений и т. д.

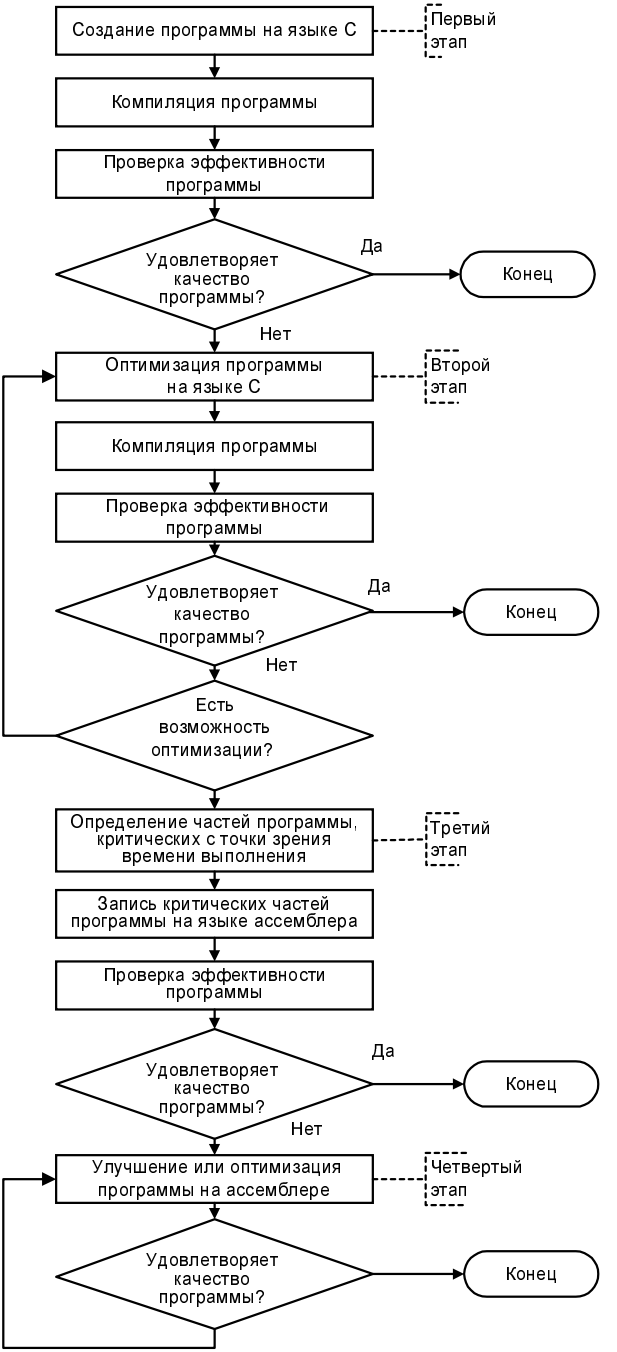


Рис 9.5. Путь построения выполняемой программы

Необходимо обратить внимание еще раз на следующий момент. Подвергать оптимизации с целью повышения эффективности нужно программу, работающую правильно и не имеет смысла улучшать программу, неверно или некорректно решающую поставленную задачу.

Средства оптимизации имеются во всех компиляторах *(см. разд. 9.6)*. Оптимизация программы производится, например, путем упрощения выражений, организации и упрощения циклов, использования регистров для хранения и передачи переменных и т. д. Оптимизаторы для повышения эффективности программы учитывают архитектуру и состав операционных устройств конкретных процессоров. Уровень оптимизации, задаваемый компилятору, способен изменяться, поэтому второй этап может выполняться неоднократно, в том числе и за счет "ручной" оптимизации. Некоторые рекомендации по оптимизации программы приводятся обычно в фирменных руководствах.

Если на уровне языка С достигнуть нужной эффективности программы не удается, то выполняется третий этап. В написанной программе выделяются фрагменты, критичные с точки зрения поставленных условий эффективности. В частности, в любой программе можно выделить фрагменты, время выполнения которых вообще не критично, (например инициализация процессора, настройка режимов и устройств периферии) и фрагменты, время выполнения которых в основном и определяет общее время выполнения, (например, многократно циклически повторяющиеся команды обработки отсчетов входного сигнала). По некоторым оценкам до 80% общего времени выполнения программы занимают 20% ее кодов. Выделенные важные фрагменты программы нужно переписать на языке ассемблера, используя все возможные особенности архитектуры и системы команд конкретного процессора. Многие процессоры имеют большое количество специализированных команд, направленных на решение задач цифровой обработки, например команды для реализации БПФ (ADSP-2116x) КИХ- и БИХ-фильтров (TMS320C54x и C55x). Полученные фрагменты на ассемблере можно вставлять в общую программу на языке С, используя соответствующие конструкции языка.

В некоторых пакетах поддержки разработки имеются оптимизаторы программы на языке ассемблера. В частности, такие оптимизаторы имеют пакеты фирмы Motorola и пакеты поддержки разработки процессоров платформы С6000 фирмы TI. Их можно применять на четвертом этапе работы с программой. При этом могут потребоваться и некоторые "творческие усилия" по улучшению алгоритма и программы, а также изменения в структуре системы, в частности изменения в структуре и характеристиках памяти, например для размещения критичных фрагментов программы и соответствующих данных может понадобиться использование более "быстрой" памяти. Следует отметить, что написание вручную эффективной программы на ассемблере процессоров С6000 вряд ли возможно. Это определяется параллелизмом работы 8 операционных элементов и сложной структурой конвей-

ера. Оптимизатор ассемблерного кода выполняет работу по учету этих, а также некоторых других возможностей процессора.

Действия по оптимизации программы естественно требуют знания архитектуры процессоров и возможных алгоритмов решения задач ЦОС. Оптимизация достигается, как правило, на пути компромисса между различными показателями системы и используемой программы.

9.5. Размещение программ в памяти

9.5.1. Секции программы и блоки памяти

Структура программ систем обработки сигналов достаточно сложна. В программах используются коды и данные, отличающиеся по назначению, возможности модификации в процессе работы, частоте и кратности выполнения. Например, в любой программе существуют команды инициализации процессора, которые выполняются один раз (или достаточно редко), команды обработки входного сигнала, способные выполняться многократно, константы (например, коэффициенты фильтров), которые считываются многократно, и т. д. Это приводит к тому, что требования к быстродействию выполнения различных фрагментов кодов и времени выборки/записи различных констант и переменных существенно отличаются.

Сложная структура программ определяет требования к структуре памяти процессоров ЦПОС, которая оказывается достаточно сложной. Память процессоров состоит из блоков разного назначения (ROM, RAM, EPROM), с различным быстродействием и возможностями доступа в одном командном цикле (SARAM, DARAM), которые размещаются как внутри, так вне процессора.

На уровне создания использование сложной структуры программ поддерживается механизмом секций ассемблера. Программу, состоящую из нескольких секций, можно загружать в несколько блоков памяти, распределяя различные секции программы в зависимости от назначения в различные блоки памяти. Например, часть программы, представляющую команды инициализации процессора, которые выполняются один раз при включении системы и время выполнения которых не критично, можно разместить в медленной (например, внешней) памяти, а критичные с точки зрения времени выполнения фрагменты — в быстрой памяти процессора.

Каждая секция может быть независимо настроена на размещение в определенном месте (в соответствии с требуемыми типом и адресом) используемого пространства памяти. Эта настройка на конкретные адреса, так же, как и размещение секций в определенном порядке, осуществляется компоновщиком или непосредственно ассемблером (если он позволяет получить абсолютную программу, т. е. программу с абсолютными адресами). При этом выходная выполняемая программа может быть получена в виде нескольких

выходных секций (не обязательно совпадающих с входными), каждая из которых способна оказаться независимо предназначенной для размещения или выполнения в различных областях и типах памяти. Таким образом, на этапе компоновки реализуется разработанная ранее структура системы и алгоритма ее функционирования. Эта структура закладывается в файлы, управляющие процессом компоновки.

9.5.2. Начальная загрузка программы

Если система постоянно выполняет одну и ту же программу с неизменными исходными данными, то простейшим вариантом с точки зрения использования является размещение этой программы и исходных данных в постоянной памяти типа ROM. Такое размещение связано с выполнением следующих условий:

- если память внешняя, то она должна удовлетворять требованиям быстродействия;
- для упрощения системы в целом лучше использовать только внутреннюю память, т. е. процессор с внутренней ROM;
- запись информации в постоянную память типа ROM производится при ее изготовлении; стоимость этого процесса связана с тиражом изделия (особенно внутренней ROM процессора).

В связи с достаточно широким разбросом возможных вариантов построения систем номенклатура выпускаемых процессоров ЦПОС имеет достаточно широкий разброс вариантов внутренней памяти, как с точки зрения объема, так и наличия памяти разного вида: RAM, ROM, EPROM (Flash). Объем памяти типа ROM, как правило, меньше объема RAM, а достаточно часто она вообще отсутствует (в частности в процессорах фирмы ADI).

В связи с отсутствием внутренней памяти типа ROM, а иногда ее малым объемом возникает задача размещения и начальной загрузки исполняемой программы. Возможно несколько вариантов построения системы.

1. Для размещения программы и ее выполнения используется внешняя память типа ROM. В этом случае она должна обладать требуемым (как правило, достаточно высоким) быстродействием. Кроме того, во всех процессорах интерфейс обращения к внешней памяти более ограничен с точки зрения количества доступов по сравнению с внутренней и в ряде случаев (например, необходимость считывания из внешней памяти кодов и данных) будет "тормозить" выполнение программы.
2. Размещение программы и исходных данных в "медленной" внешней памяти и загрузка ее для выполнения во внутреннюю память процессора.

Естественно, возможные и используемые на практике варианты размещения программы не ограничиваются двумя приведенными выше. Находят приме-

нение различные компромиссные варианты, при которых для различных частей программы применяются и внутренняя ROM, и внешняя ROM, как "медленная", так и "быстрая". Кроме того, в современных сложных системах существуют варианты реализации с динамической перезагрузкой программ и данных в процессе работы.

Таким образом, достаточно часто возникает задача загрузки (переписывания) программы во внутреннюю память.

Эта задача в принципе решается следующим образом. В памяти типа ROM (независимо от ее расположения) записывается модуль программы, осуществляющий переписывание на этапе инициализации системы основной программы из внешней ROM во внутреннюю память типа RAM. В некоторых процессорах подобный загрузчик предусматривается при изготовлении, причем он автоматически активизируется после сброса или подачи питания. Данный начальный загрузчик (boot loader) в глоссарии ЦПОС TI определяется так: встроенный сегмент кода, который перемещает исполняемый код от внешнего источника до памяти программы при включении питания.

В процессорах ADI начальный загрузчик переписывает 256 слов программы, адрес которых передается в программный счетчик. Загруженный таким образом начальный фрагмент программы должен содержать команды перезаписи всей остальной программы. Загрузчик может работать в нескольких режимах, выбор которых определяется сигналами на внешних выводах процессора.

В процессорах TI загрузчик может заполнять всю внутреннюю память через последовательный или параллельные порты. Режим работы определяется сигналами на внешних выводах.

9.5.3. Оверлейные программы

Оверлейными называют программы, которые имеют различное расположение при загрузке и при выполнении. Таким образом, программы, перемещаемые с помощью начального загрузчика, являются оверлейными. Менять свое расположение для выполнения может не вся исполняемая программа, а отдельные ее части, критичные к времени выполнения.

Основной проблемой, возникающей при использовании оверлейных программ, является необходимость присваивания символам, и в частности меткам (которые могут являться адресами переходов), значений в соответствии с местом выполнения, а не загрузки. Эта проблема решается по-разному в ассемблерах различных фирм. В ассемблере Motorola введены соответствующие указания транслятору с помощью директивы `ORG`, операнды которой указывают как место загрузки программы, так и место ее выполнения. В ассемблере TI адреса загрузки и выполнения указываются в командном файле компоновки, и используется специальная директива `.label` для задания меток при загрузке.

9.6. Языки C/C++ и архитектуры, дружественные к языку C

Практически для всех процессоров ЦПОС язык C является одним из допустимых языков программирования. C-компиляторы процессоров ЦПОС поддерживают язык C, отвечающий стандарту ANSI (American National Standards Institute). Язык C, отвечающий этому стандарту, описан в [17]. Компиляторы языка C++ поддерживают стандарт языка ANSI/ISO, приведенный в [38]. Язык C++ поддерживается компиляторами процессоров ADSP-21k фирмы ADI, процессорами платформы C6000 и TMS329C28x фирмы TI, процессорами StarCore SC100 и некоторыми другими. Планируется разработка компилятора C++ для платформы C5000 фирмы TI.

Достоинствами языка C помимо естественных достоинств языка высокого уровня является следующее.

- Язык C представляет собой (несмотря на существование "диалектов") стандартный язык программирования (стандарт ANSI) высокого уровня. Поэтому программы на C легко переносимы на любой процессор ЦПОС, поддерживающий этот язык.
- Значительную часть программ, реализуемых на ЦПОС, составляют сложные управляющие структуры. Эти составляющие программ занимают малую часть общего времени выполнения программы, однако требуют значительных усилий при их разработке. Поэтому такие части программ легче создавать на языке высокого уровня C.
- При использовании языка C программист, вообще говоря, не должен знать архитектуру процессора, который он программирует (при этом, однако, он не получит эффективной "хорошей" программы), архитектура процессора закладывается в компилятор и используемые оптимизаторы.
- Компиляторы C для ЦПОС допускают введение в программу фрагментов, директив и конструкций на ассемблере. Подобные вставки можно использовать для частей программы, которые лучше писать на ассемблере, например для частей, критичных к времени выполнения.
- Процессоры с архитектурой WLIV *(см. разд. 2.4)* имеют несколько функциональных модулей, работающих параллельно. Для того чтобы писать эффективную программу на ассемблере, в этом случае требуется хорошее знание процессора и ассемблера и достаточно большие усилия. Язык C для программирования таких процессоров представляется единственно приемлемым.
- Как правило, исполняемая программа, полученная с использованием языка C, имеет больший объем и медленнее выполняется. Однако последние версии C-компиляторов имеют достаточно эффективные оптимизаторы и позволяют получать программы, не сильно уступающие

программам на языке С. В табл. 9.1 приведены данные, иллюстрирующие эффективность С-компиляторов по сравнению с ассемблером с точки зрения времени выполнения для процессора С62х (тактовая частота 300 МГц) фирмы TI (см. Comp_bench.zip по адресу **http://www.ti.com**).

Таблица 9.1. *С-компиляторы и ассемблер*

Алгоритм	Ассемблер		Язык С		Эффективность С, %
	Кол-во тактов	Время, мкс	Кол-во тактов	Время, мкс	
40 отсчетов КИХ-фильтра на 10 коэффициентов	227	0,76	269	0,90	84
БИХ-фильтр на 16 коэффициентов	32	0,11	32	0,11	100
Поиск минимума в таблице на 2304 значения	1175	3,92	1309	4,36	90
Перемножение с накоплением (два вектора по 40 отсчетов)	50	0,17	51	0,17	98

- При использовании языка С распространен вариант создания программ, при котором после написания общей программы на языке С выявляются критичные с точки зрения эффективности программы фрагменты, и затем эти фрагменты переписываются или исправляются "вручную" на ассемблере *(см. разд. 9.4)*. Отметим, что из-за сложности распределения действий между 8 операционными устройствами дополнительная полностью "ручная" оптимизация полученного кода на уровне ассемблера для С6000 и других подобных процессоров более затруднительна.

К недостаткам языка С с точки зрения использования его для ЦПОС, особенно ЦПОС с фиксированной точкой, является то, что он не поддерживает тип данных с фиксированной точкой.

Язык С++, позволяющий вводить пользовательские типы данных, а также поддерживающий объектно-ориентированное программирование, предоставляет пользователю большие возможности.

Рассмотрим формирование исполняемой программы с использованием языка С.

Компилятор языка С программной оболочкой объединяется с набором других программ, в частности с ассемблером и компоновщиком. В качестве выходных файлов компилятора могут быть получены файлы с программой на языке

ассемблера, объектные модули или исполняемые программы за один шаг. На этапе компоновки программы можно подключать библиотеку функций языка С и модули, написанные на ассемблере. Составляющими частями компилятора являются препроцессор, оптимизатор и кодогенератор.

Препроцессор компилятора выполняет синтаксический анализ, управляет формированием различных информационных файлов.

Оптимизатор анализирует программу с точки зрения возможностей повышения эффективности, уменьшения размера и изменяет ее. Изменения могут производиться, например, с целью упрощения выражений, организации и упрощения циклов, использования регистров для хранения и передачи переменных, устранения неиспользуемых инструкций и т. д. Оптимизаторы учитывают архитектуру и состав операционных устройств конкретных процессоров для повышения эффективности программы. При этом могут применяться различные уровни оптимизации, определяющие производимые оптимизатором действия.

Кодогенератор компилятора формирует программу на языке ассемблера. При компиляции может формироваться множество различных файлов с информационными и диагностическими сообщениями.

В качестве примера на рис. 9.6 показаны этапы выполнения компиляции и используемые при этом программы для процессоров фирмы TI TMS320.

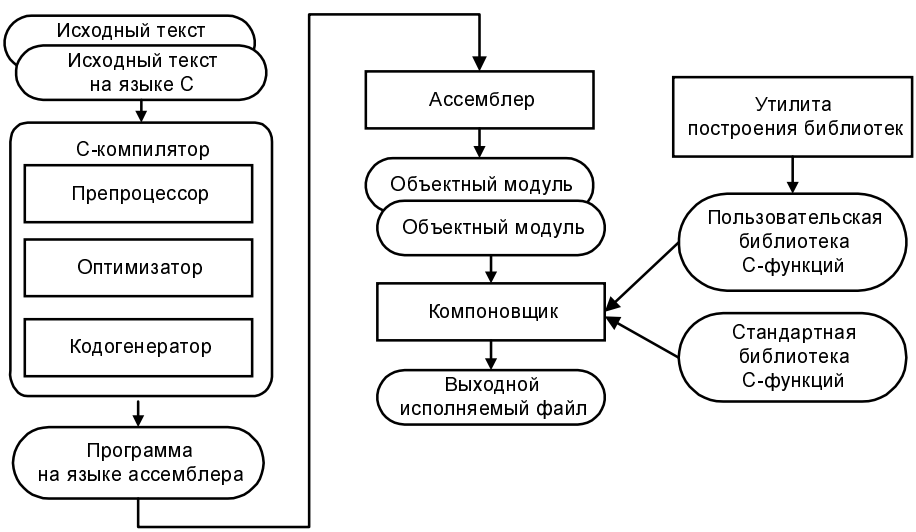


Рис. 9.6. Этапы получения исполняемой программы при использовании языка С

Программа оболочки компилятора вызывает индивидуальные программы с желательными параметрами и опциями или выполняет все программы за

один проход. Программа оболочки может вызывать синтаксический анализатор (препроцессор), оптимизатор, генератор кода, утилиту межсписка, ассемблер и компоновщик.

Синтаксический анализатор читает С-файл исходного текста и выполняет предварительную обработку, проверяет синтаксис и формирует промежуточный файл, который используется как входной для генератора кода или оптимизатора. Он также обрабатывает макроопределения и расширения, включаемые файлы и директивы условной трансляции.

Оптимизатор читает промежуточный файл, сгенерированный препроцессором, и производит различную оптимизацию, чтобы повысить быстродействие выполняемой программы. Оптимизация включает упрощение инструкции и выражений, распределение переменных в регистрах, упрощение циклов, и объединение нескольких исходных файлов в один модуль, чтобы исполнять оптимизацию уровня одного файла. Оптимизатор также делает начальный анализ циклов конвейерной обработки команд, этот процесс будет закончен генератором кода. Оптимизатор создает промежуточный файл, который является входным для кодогенератора.

Генератор кода формирует программу на языке ассемблера, оптимизированную с точки зрения использования конвейера команд процессора. Эти файлы могут содержать команды процессора, директивы ассемблера и макродирективы.

Ассемблер транслирует файл текста ассемблера в объектные файлы формата COFF.

Компоновщик объединяет объектные файлы в единственную выполнимую программу формата COFF. Компоновщик распределяет перемещаемые секции и символы в памяти и разрешает вопросы с различными внешними ссылками между входными файлами. На этапе компоновки могут подключаться библиотеки стандартных функций языка С, библиотеки функций языка С и собственные библиотеки пользователя. Последние могут быть образованы с помощью утилиты построения библиотеки (Library-build utility).

На этапе компиляции может быть опцией подключена утилита Interlist. В результате выполнения этой утилиты выводится текст программы, в котором показывается, каким образом каждое утверждение языка С переводится на язык ассемблера. Утилита позволяет проверить ассемблерный код, сгенерированный для каждой инструкции С, и тем самым проверить и выбрать действия, производимые оптимизатором.

Примерно таким же образом язык С используется при программировании процессоров фирмы Motorola. Программирование процессоров фирмы ADI несколько отличается.

В качестве примера можно привести состав пакета С-компилятора семейства цифровых процессоров DSP56000 фирмы Motorola, который включает:

- g56k — интегрированную управляющую программу;
- mcpp — препроцессор языка С (стандарт ANSI);
- g56-cc1 — оптимизирующий С-компилятор;
- asm56000 — ассемблер;
- dsplnk — компоновщик;
- dsplib — библиотекарь;
- gdb56 — отладчик на уровне языка С;
- run56 — симулятор;
- srec, cldlod, cofdmp — различные инструментальные средства манипуляции с содержимым объектного файла.

На эффективность работы компилятора С влияет архитектура процессора. Архитектура обычных ЦПОС с несколькими пространствами памяти, большим количеством шин, комбинированными и специализированными командами не способствует тому, чтобы с помощью компилятора языка высокого уровня получить эффективную программу (см. [2] по адресу **http://www.embedded.com**). Язык С гораздо более эффективен, когда процессор имеет большой, универсальный набор регистра и объединенное пространство памяти. Кроме того, архитектура процессоров с плавающей точкой более ориентирована на использование языков высокого уровня. Компилятору трудно оптимальным образом учесть все аппаратные возможности процессора. С этой точки зрения употребляется термин "архитектура ЦПОС, дружественная к С" (см. "Compiler-friendly Architectures for DSP" по адресу **http://www.zsp.com**). В этой работе в качестве таких архитектур отмечаются VLIW (процессоры платформы С6000 TI) и суперскалярная архитектура (процессоры DSP16000 ZSP).

В статье "DSP Benchmark Results for the Latest WLIV-Based Processors" (см. wliv_icspat00.pdf по адресу **http://www.bdti.com**) приводятся результаты тестирования и сравнения эффективности архитектур WLIV и улучшенной стандартной *(см. разд. 2.4)* с точки зрения применения С-компиляторов.

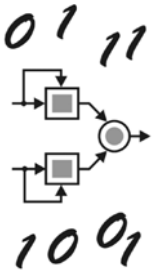
При тестировании проверялись показатели программ, полученных с помощью языка С, по сравнению с программами, написанными на ассемблере, на примерах задач БПФ, реализации БИХ-фильтра и декодера Витерби. Соответствующие данные приведены в табл. 9.2. По размеру кода программы на С для WLIV-процессоров проигрывают программам на ассемблере в 1,5 раза, процессоры улучшенные стандартной архитектуры — в 11,5 раз. По скорости выполнения программы на С этих двух архитектур уступают программам на ассемблере соответственно: БПФ — в 7 и 9 раз, БИХ-фильтр — в 2 и 3 раза, декодер Витерби — в 3,5 и 11 раз.

Таблица 9.2. *Отношение кода после компилятора к коду, полученному с использованием ассемблера*

Тип архитектуры процессора	Задача	Соотношение кодов
Улучшенная стандартная	БПФ	9
VLIW	БПФ	5
Улучшенная стандартная	КИХ-фильтр	2,2
VLIW	КИХ-фильтр	1,9
Улучшенная стандартная	Декодер Витерби	11,5
VLIW	Декодер Витерби	3,5

Архитектурами, оптимизированными для компиляторов C/C++ (по мнению соответствующих производителей процессоров) являются также архитектуры Star Core (Motorola), суперскалярные архитектуры (ZSP) и некоторые другие, в основе которых лежит архитектура процессоров RISC с сокращенным набором простых команд [21].

Глава 10



Средства разработки и отладки систем цифровой обработки сигналов

Разработка систем ЦОС является сложным и трудоемким процессом, относящимся к области высоких компьютерных технологий. Процедура разработки включает в себя:

- постановку задачи;
- разработку или выбор оптимального алгоритма решения задачи;
- выбор процессора и реализация алгоритма, в конечном счете, в виде программы на языке ассемблера выбранного процессора;
- отладка программы, т. е. устранение возможных ошибок и оптимизация ее с целью минимизации временного ресурса, необходимого для ее выполнения, а в ряде случаев и памяти;
- создание на базе выбранного ЦПОС аппаратного устройства, реализующего заданные свойства и характеристики системы;
- отладка разработанной системы.

Таким образом, система ЦОС представляет собой совокупность программного и аппаратного продукта. Для ее создания необходимы инструментальные средства разработки и отладки. Последние делятся на два обширных класса: *программные* и *аппаратные* средства.

К программным средствам разработки и отладки систем ЦОС относятся:

- ассемблеры;
- линкеры;
- компиляторы;
- отладчики;
- симуляторы (программные имитаторы команд процессора).

Ассемблеры, линкеры и компиляторы рассмотрены в *главе 9*, поэтому здесь из программных средств описываются отладчики и симуляторы; основное внимание в данной главе уделяется аппаратным средствам.

10.1. Аппаратные средства отладки

К аппаратным средствам отладки (рис. 10.1) относятся аппаратные эмуляторы и проверочные модули.

Аппаратные эмуляторы предназначены для отладки программного и аппаратного обеспечения процессоров в режиме реального времени. Они работают под управлением ведущего компьютера (хост-компьютера), оснащенного необходимым интерфейсом и программным обеспечением — отладчиками. Различают три группы эмуляторов:

- эмуляторы-приставки, замещающие в системе ЦОС отлаживаемый процессор;
- внутрисхемные, или сканирующие, эмуляторы, позволяющие не изымать из системы отлаживаемый процессор;
- внутрикристальные эмуляторы, работающие совместно с внутрисхемными эмуляторами и представляющие собой одно из устройств внутренней периферии процессора.

Проверочные модули предназначены для быстрой отладки программного обеспечения ЦПОС в реальном времени и исследований разрабатываемых алгоритмов и программ. Различают два типа проверочных модулей:

- стартовые наборы;
- отладочные модули.

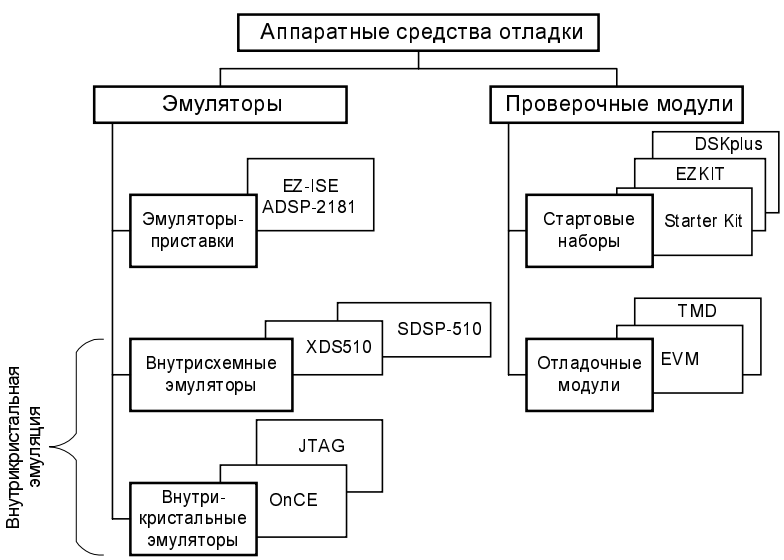


Рис. 10.1. Классификация аппаратных средств отладки

10.1.1. Внутрисхемные эмуляторы-приставки

Внутрисхемный эмулятор-приставка представляет собой плату, содержащую аппаратный имитатор отлаживаемого процессора и дополнительное устройство управления имитатором. Перед использованием эмулятора-приставки процессор извлекается из системы ЦОС и на его место подключается плата (рис. 10.2). Подключение платы к системе осуществляется с помощью гибкого кабеля, количество и назначение контактов которого идентично выводам замещаемого процессора.

Управление всем процессом отладки осуществляется с хост-компьютера, содержащего инсталлированный отладчик, и начинается с загрузки в эмулятор той программы, которую в дальнейшем будет выполнять процессор. Плата подключается к компьютеру через один из стандартных параллельных портов (например, через COM1 или COM2), в зависимости от образца эмулятора. На плату должно подаваться электропитание.

Подобные эмуляторы (например, EZ-ICE процессора ADSP-2181) обеспечивают выполнение трассировки программы в реальном времени.

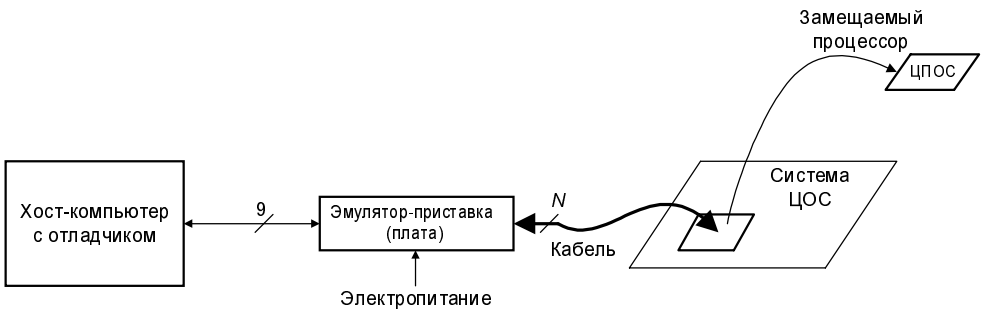


Рис. 10.2. Подключение эмулятора-приставки (*N* — количество контактов процессора)

Эмуляторы-приставки еще не так давно были единственно возможным средством внутрисхемной эмуляции, несмотря на то, что сразу были видны их серьезные недостатки:

- ❒ высокая стоимость;
- ❒ недостаточная надежность;
- ❒ большое энергопотребление;
- ❒ изменение электрических характеристик цепи, к которой подключается эмулятор, вследствие чего могут возникать ошибки синхронизации.

На смену эмуляторам-приставкам пришли другие, рассматриваемые ниже, внутрикристальные средства отладки.

10.1.2. Внутрикристальные средства отладки

Внутрикристальная отладка осуществляется *без извлечения процессора* из системы ЦОС с целью непосредственного контроля за выполнением программы процессором на полной скорости без каких-либо ограничений. Средства внутрикристальной отладки обеспечивают прямой доступ к регистрам, памяти и периферии процессора.

Для внутрикристальной отладки необходимо иметь (рис. 10.3):

- внутрисхемный эмулятор, обеспечивающий связь между хост-компьютером и процессором;
- внутрикристальный эмулятор, являющийся элементом внутренней периферии и содержащий логику отладки процессора.

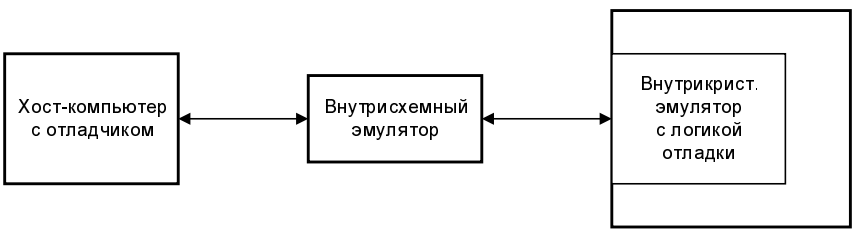


Рис. 10.3. Принцип внутрикристальной отладки

Рассмотрим перечисленные средства.

Внутрисхемный эмулятор

Внутрисхемный эмулятор включает в себя плату-контроллер, кабель, буфер и переходную приставку (рис. 10.4). Эмулятор устанавливается между компьютером и процессором. Связь с компьютером обеспечивается установкой платы в слот ISA (или в слот PCI в новейших компьютерах), а связь с внутрикристальным эмулятором процессора осуществляется по специально организованному интерфейсу.

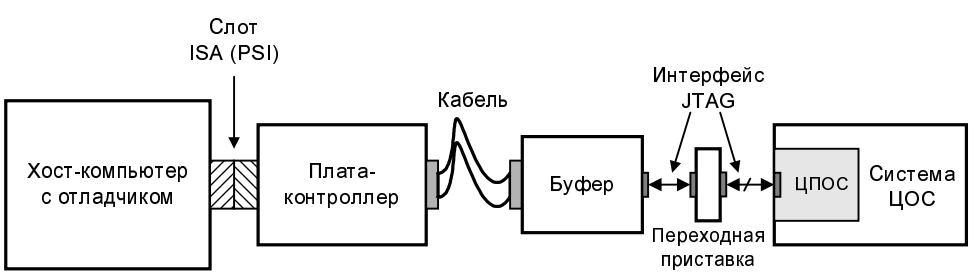


Рис. 10.4. Подключение внутрисхемного эмулятора

Большинство процессоров содержат 5-проводный сигнальный последовательный интерфейс стандарта IEEE 1149.1, известный как JTAG (по названию комитета, утверждающего стандарты: Joint Test Action Group — Объединенная рабочая группа по автоматизации тестирования). Сам интерфейс JTAG имеет четырнадцать контактов. Этот стандарт используется во всех процессорах фирмы Texas Instruments, Analog Devices, а также в процессорах семейства DSP563xx фирмы Motorola. В других процессорах фирмы Motorola применяется собственный последовательный сигнальный интерфейс, подобный JTAG, но с меньшим количеством сигнальных линий.

Типовым внутрисхемным эмулятором является XDS-510, выпускаемый фирмой Texas Instruments для своих процессоров. Перечень других приборов дан в табл. 10.1, где аббревиатура PP означает, что эмулятор подключается к параллельному порту компьютера, аббревиатура L указывает на пониженное напряжение питания интерфейса с возможностью плавной регулировки от 1,6 до 3,4 В.

Таблица 10.1. *Внутрисхемные эмуляторы фирмы Texas Instruments*

Эмулятор	Интерфейс	Слот
XDS–510	JTAG 3/5 В JTAG 1,6 — 3,6 В	ISA
SDSP–510	JTAG 3/5 В MPSD 3/5 В JTAG 1,6 — 3,6 В	ISA
SDSP–PCI SDSP–PCIL	JTAG 3/5 В JTAG 1,6 — 3,6 В	PCI
SDSP–PP SDSP–PPL	JTAG 3/5 В JTAG 1,6 — 3,6 В	Параллельный порт

Подключение компьютера к последовательному отладочному порту процессора через внутрисхемный эмулятор делает доступным отладочные средства процессора. Часто этот метод называют *сканирующей эмуляцией*: внутрикристальная логика отладки контролирует функционирование кристалла и останавливает процессор при достижении точки останова, определяемой пользователем.

Достоинствами внутрикристальной (сканирующей) эмуляции являются:

- использование небольшого количества выводов (не более четырех) закрепленного канала на процессоре;
- предоставление возможности пользователю выполнять разнообразные действия на процессоре без изъятия его из системы ЦОС, например:
  - загружать программы;
  - проверять и изменять содержимое регистров и памяти;

- устанавливать и удалять точки останова;
- проверять содержимое конвейера процессора после достижения точки останова

и т. д.;

- ❒ поддержка максимальной производительности процессора;
- ❒ снижение энергопотребления без изменения электрических характеристик системы.

Внутрикристальный эмулятор

Внутрикристальный эмулятор является отладочным средством, обеспечивающим быстрый и независимый доступ к внутренним регистрам, памяти и периферии процессора.

Как правило, производители процессоров не описывают конструкцию внутрикристального эмулятора, поэтому представление о его свойствах можно получить по наблюдениям за результатами сканирования на программном отладчике. Исключение составляет фирма Motorola, предоставляющая документацию по своим внутрикристальным эмуляторам, имеющим торговую марку OnCE (On-Chip-Emulator). Тем не менее, можно полагать, что внутрикристальные эмуляторы других фирм построены на тех же принципах, что и эмуляторы OnCE. По этой причине здесь рассматривается внутрикристальный эмулятор фирмы Motorola, а для сокращения записи используется его обозначение OnCE.

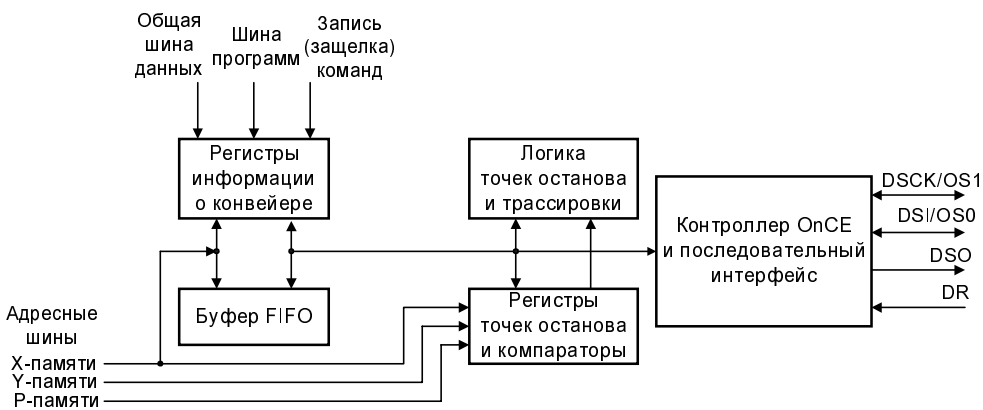


Рис. 10.5. Структурная схема OnCE

В состав OnCE входят (рис. 10.5):

- ❒ контроллер и последовательный интерфейс;
- ❒ регистры точек останова и компараторы;

- ❒ логика точек останова и трассировки;
- ❒ регистры информации о конвейере;
- ❒ буфер FIFO.

Управление OnCE осуществляется сигналами:

- ❒ тактовой синхронизации/состояния процессора 1 (вывод DSCK/OS1):
  - тактовая синхронизация (DSCK) в режиме отладки выполняется на частоте 1/8 тактовой частоты процессора, причем передаваемые данные синхронизируются по возрастающему фронту СИ, а принимаемые — по падающему;
  - сигнал "состояния процессора 1" (OS1) формируется вне режима отладки и при аппаратном сбросе; совместно с сигналом OS0 образует сигнал, содержащий информацию о состоянии процессора (табл. 10.2);
- ❒ ввода данных/состояния процессора 0 (вывод DSI/OS0):
  - данные или команды (DSI) в режиме отладки поступают из внутрисхемного эмулятора в последовательном формате, начиная с бита MSB;
  - сигнал "состояния процессора 0" (OS0) формируется вне режима отладки и при аппаратном сбросе; совместно с сигналом OS1 образует сигнал, содержащий информацию о состоянии процессора (табл. 10.2);

Таблица 10.2. *Сигналы о состоянии процессора*

Состояние выводов		Состояние процессора
OS1	OS0	
0	0	Нормальное
0	1	Останова или ожидания (низкого потребления энергии)
1	0	Занятости оперативной памяти
1	1	Занятости оперативной памяти или ПДП

- ❒ вывода данных/подтверждения (вывод DSO):
  - вывод содержимого одного из регистров OnCE, определенного последней командой, поступившего от внутрисхемного эмулятора;
  - формирование сигнала "подтверждения" готовности OnCE принять команду при входе в режим отладки;
  - формирование сигнала "подтверждения" после приема команды чтения, означающего, что запрашиваемые данные доступны, и начинается передача данных;
  - формирование сигнала "подтверждения" после приема команды записи, означающего, что OnCE готов к приему данных;

- запроса отладки (вывод DR): по сигналу "запроса отладки" от внутрисхемного эмулятора процессор заканчивает выполнение текущей команды, сохраняет информацию о конвейере команд, входит в режим отладки и ожидает команды по выводу DSI; вывод используется также для сброса контроллера OnCE.

Процессор может входить в режим отладки по программному запросу, внешнему запросу и механизмам точек останова и трассировки. Рассмотрим эти возможности отдельно.

- *Программный запрос отладки* возможен при нормальной работе по условной команде отладки `DEBUGcc`: если заданное условие `cc` истинно, процессор входит в режим отладки после команды, непосредственно следующей за командой `DEBUGcc`.
- *Внешний запрос отладки* поступает на вывод DR:
  - при нормальной работе по сигналу внутрисхемного эмулятора; перед входом процессора в режим отладки заканчивается выполнение текущей команды;
  - по аппаратному сбросу; перед входом в режим отладки процессор прекращает выполнение команд;
  - по состояниям STOP и WAIT; после формирования сигнала подтверждения DSO выполняется команда `STOP` (`WAIT`), процессор входит в режим отладки и останавливается после выполнения очередной команды.
- *Запрос отладки по механизмам точек останова и трассировки* осуществляется по достижении нуля счетчиков останова и трассировки соответственно (логика точек останова и трассировки рассматривается ниже).

Контроллер OnCE и последовательный интерфейс

Контроллер OnCE предназначен для управления всей логикой внутрикристальной отладки. Он содержит следующие блоки (рис. 10.6):

- регистр команд (РгКом);
- счетчик битов (СчБит);
- декодер (Дек);
- регистр состояния и управления (РгСУ).

Работа контроллера и последовательного интерфейса будет ясна после рассмотрения функций каждого из перечисленных блоков.

- *Регистр команд* (РгКом) является 8-разрядным сдвиговым регистром, получающим данные в последовательном формате по линии "ввод данных" (DSI). После получения восьмого бита РгКом приостанавливает прием новых данных. Фиксируемые данные, содержащие команды внутрикри-

стальному эмулятору, в параллельном коде пересылаются в декодер. Команда имеет строго заданную структуру и определяет:

- регистр OnCE, используемый в качестве источника/приемника для чтения/записи;
- необходимость выполнения команды, хранящейся в регистре информации о конвейере;
- направление пересылки данных — чтение/запись;
- вход в режим отладки.

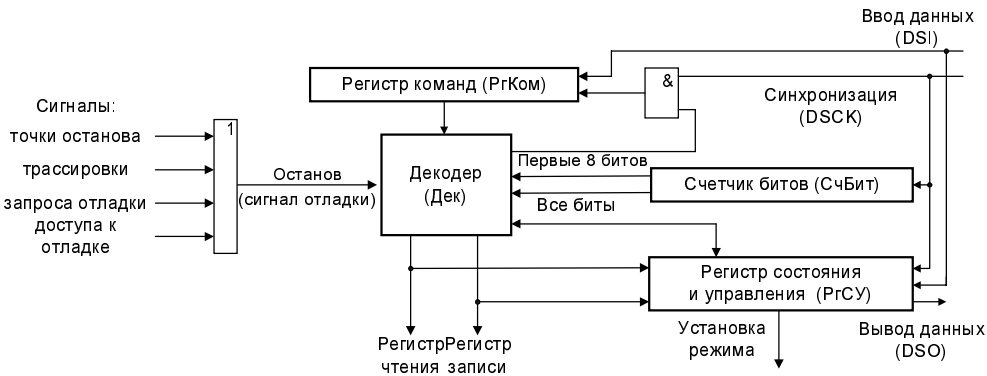


Рис. 10.6. Контроллер и последовательный интерфейс OnCE

- *Счетчик битов* (СчБит) является 5-разрядным, работающим на инкремент с автоматической перезагрузкой после достижения нулевого состояния, поэтому может считать от 31 до 0. Счетчик предназначен для выдачи на декодер двух сигналов, связанных со сдвигом данных в OnCE или из OnCE. Первый сигнал выдается после сдвига первых 8 битов; это означает, что для выборки доступна очередная 8-разрядная команда. Второй сигнал соответствует окончанию сдвига (приема/передачи) всего 24- или 32-разрядного слова (в зависимости от образца процессора). Счетчик сбрасывается при переходе процессора в режим отладки по сигналу подтверждения или по аппаратному сбросу.
- *Декодер* (Дек) управляет работой внутрикристального эмулятора согласно:
  - 8-разрядной команде, поступающей из регистра команд;
  - двум сигналам счетчика битов;
  - двум сигналам останова процессора: один поступает из РгСУ, другой — из схемы "ИЛИ".

  По указанным сигналам декодер генерирует стробы управления:
  - чтением или записью выбранных регистров OnCE;

- регистром команд (по восьмому биту СчБит);
- регистром состояния и управления.

❒ *Регистр состояния и управления* (РгСУ) предназначен для:

- индикации причины, по которой наступил режим отладки;
- управления точками останова в ограниченных областях P-, X-, Y-памяти; точки останова могут фиксироваться по чтению, записи, по ПДП и др., а также могут быть запрещены;
- установки режима трассировки.

Логика точек останова

Точки останова можно устанавливать в памяти программ, данных или внутри той адресуемой области, где могла бы выполняться программа. Логика точек останова имеет два совершенно идентичных независимо работающих блока: один — для памяти программ, другой — для памяти данных. Каждый блок содержит (рис. 10.7):

❒ защелку адреса памяти;

❒ регистры хранения адресов верхней и нижней границы памяти;

❒ компараторы адресов верхней и нижней границ;

❒ счетчик точек останова.

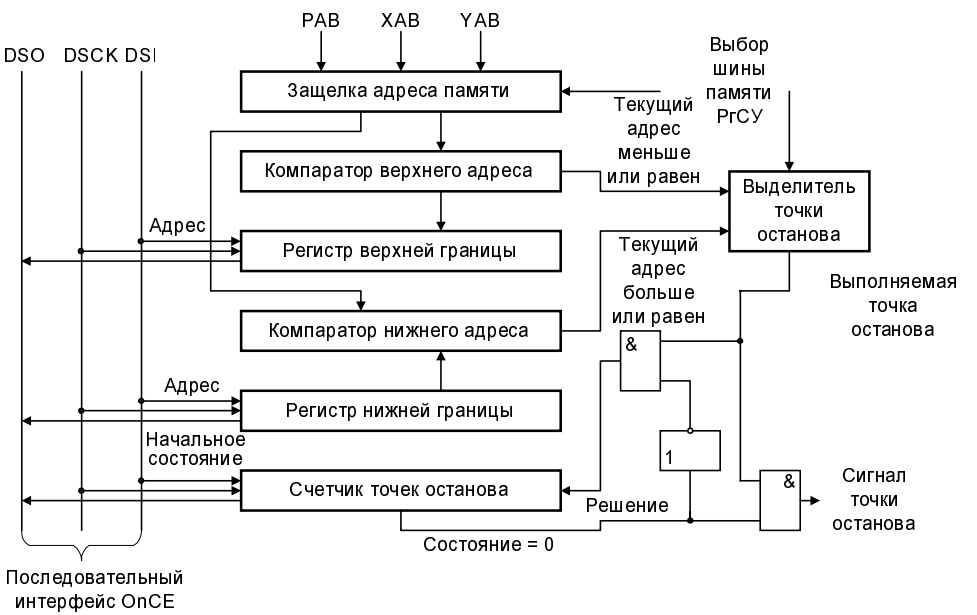


Рис. 10.7. Логика точек останова

Логика точек останова работает следующим образом. Из внутрисхемного эмулятора через последовательный ввод DSI поступают адреса верхней и нижней границ памяти, в пределах которых предполагается организация точек останова; через этот же ввод загружается счетчик точек останова величиной, соответствующей количеству циклов обращения к памяти, которое должно предшествовать выполнению точки останова. При каждом обращении к памяти счетчик инкрементируется. После того как состояние счетчика станет равным 0, первое же обращение к памяти вызовет переход процессора в режим отладки. Согласно команде, поступившей в регистр команд (см. рис. 10.6), происходит выбор адресной шины памяти для защелки и выделения точки останова.

Сказанное означает, что во время отладки при каждом обращении к выбранной памяти выполняются следующие операции:

- текущий адрес выбранной памяти фиксируется в защелке, откуда пересылается в компараторы верхнего и нижнего адресов;
- компараторы сравнивают текущий адрес с верхней и нижней границами области, в которой должен происходить останов; если адрес находится в пределах заданных границ (включая верхний и нижний адреса), оба компаратора выдают сигнал "истинно" на выделитель точки останова;
- выделитель точки останова при наличии трех входных сигналов "истинно" формирует сигнал "выполняемая точка останова", который поступает на две схемы "И";
- счетчик точек останова инкрементируется, если на него не поступил сигнал "решение" от первой схемы "И"; внутренний сигнал точки останова появится на выходе второй схемы "И" лишь в том случае, когда состояние счетчика окажется равным нулю и произойдет очередное обращение к выбранной области памяти; появление этого сигнала обеспечит останов по адресу, зафиксированному в защелке, при этом счетчик не инкрементируется вследствие подачи на него сигнала "решение";
- на последовательном выводе DSO формируется сигнал подтверждения готовности OnCE, передаваемый внутрисхемному эмулятору.

Логика трассировки

Под *трассировкой* понимается пошаговый (от команды к команде) или поблочный контроль за ходом выполнения программы и состоянием выводов процессора в реальном времени. Трассировка осуществляется *вне режима отладки* и позволяет разработчику отлаживать те участки программы, которые не являются линейными или представляют собой бесконечные циклы.

Работа логики трассировки (рис. 10.8) подобна работе участка логики точек останова, связанного со счетчиком точек останова. Главным элементом логики трассировки является счетчик трассировки.

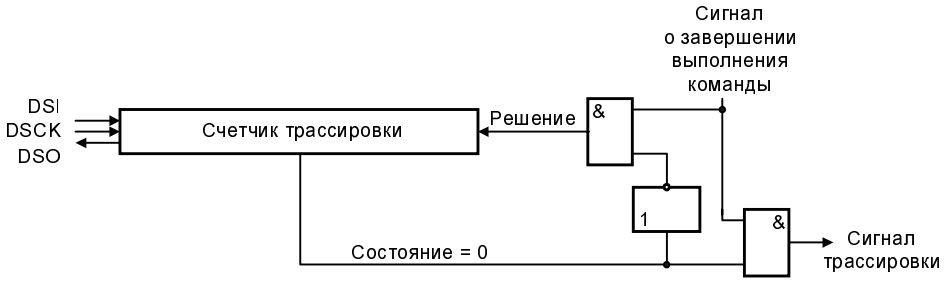


Рис. 10.8. Логика трассировки

Инициализация режима трассировки осуществляется с использованием программного отладчика, инсталлируемого в компьютер. Для установки режима трассировки необходимо:

- загрузить в программный счетчик процессора начальный адрес блока команд, которые должны быть выполнены в реальном времени *до вхождения в режим трассировки*;
- установить в РгСУ (см. рис. 10.6) биты разрешения трассировки;
- загрузить счетчик трассировки через вывод DSI величиной $(N-1)$, где N — количество команд, которые должны быть выполнены в реальном времени в режиме трассировки;
- выйти из режима отладки с помощью команды отладчика.

После выхода из режима отладки процессор начинает работать в реальном времени, счетчик трассировки декрементируется при выполнении каждой команды отлаживаемой программы. По завершении выполнения команды, вызвавшей последний декремент счетчика трассировки (установилось состояние 0), выдаются два сигнала:

- сигнал "решение"; по этому сигналу предотвращается дальнейшая работа счетчика, на выводе DSO устанавливается уровень, указывающий на переход процессора в режим отладки и на ожидание им очередного запроса со стороны хост-компьютера;
- сигнал "трассировки", который воздействует на декодер (см. рис. 10.6) и переводит процессор в режим отладки; в свою очередь, декодер сбрасывает биты трассировки в РгСУ.

Замечание

На некоторых процессорах трассировка позволяет также определять действительное время выполнения трассируемого участка программы, т. е. выполнять *профилирование*.

Информация о конвейере

Возврат процессора из режима отладки к нормальной работе достигается восстановлением состояния конвейера, что достигается с помощью регистров информации о конвейере (рис. 10.9). Рассмотрим каждый из регистров.

- *Регистр шины программ* сохраняет состояние этой шины, установленное при последнем обращении к памяти программ непосредственно перед входом процессора в режим отладки. Регистр доступен для чтения и записи как со стороны шины, так и со стороны последовательного интерфейса OnCE.
- *Регистр общей шины данных* хранит данные, которые записываются процессором и могут выводиться (читаться) через вывод DSO по команде чтения этого регистра, задаваемой пользователем на компьютере.
- *Регистр защелки команды* фиксирует команду, непосредственно предшествующую режиму отладки; регистр может читаться только через вывод DSO, причем на него влияют операции режима отладки, поэтому его содержимое должно быть восстановлено внешним управлением при возвращении в нормальный режим работы.

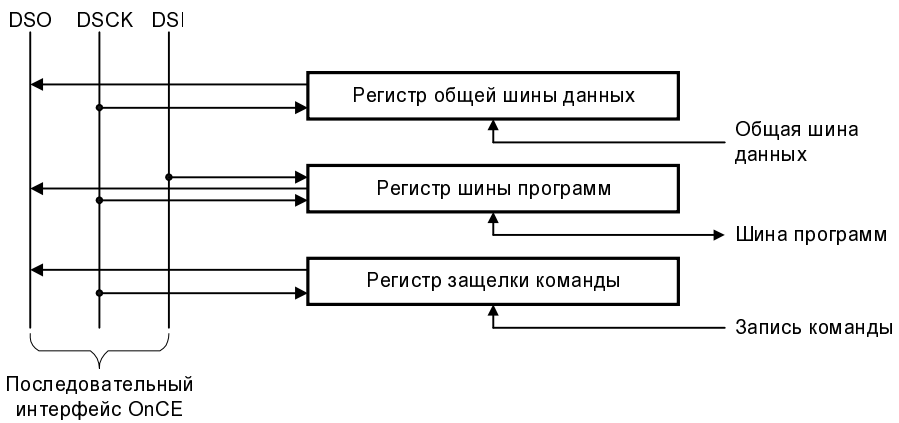


Рис. 10.9. Регистры информации о конвейере

Более полная информация о конвейере содержится в блоке регистров адресной шины программ (рис. 10.10). В состав блока входят:

- *два регистра адресной шины адреса* (только для чтения через вывод DSO), дающие информацию о конвейере, когда введен режим отладки:
  - *регистр адреса выбранной команды* хранит адрес последней команды, которая была выбрана перед вхождением в режим отладки;
  - *регистр адреса декодируемой команды* содержит адрес той команды, которая должна была бы декодироваться, если бы процессор не входил в режим отладки;

- *буфер типа FIFO*, состоящий из пяти регистров, хранит адреса последних пяти команд, выполненных перед входом в режим отладки;
- *циклический указатель буфера*, представляющий собой 3-разрядный счетчик от 0 до 4;
- *выходной сдвиговый регистр*, обеспечивающий передачу адреса через последовательный вывод DSO в хост-компьютер.

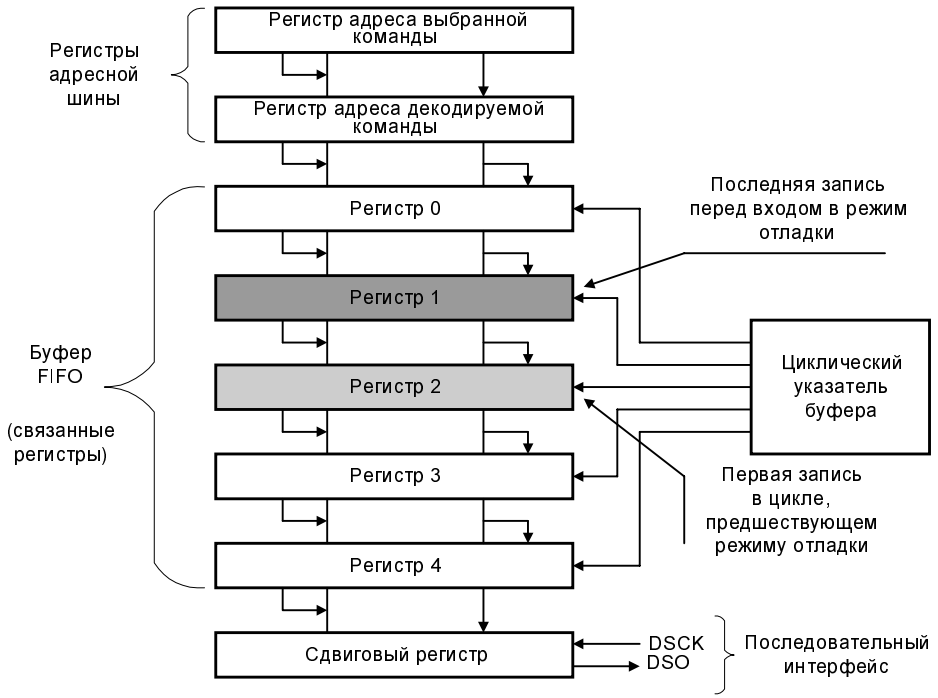


Рис. 10.10. Блок регистров адресной шины программ

Блок работает следующим образом.

Все регистры буфера являются *связанными*, т. е. имеющими один и тот же адрес. Всякое обращение к буферу сопровождается инкрементированием счетчика, состояние которого будет соответствовать номеру очередного регистра. Таким образом, регистры буфера *последовательно доступны* контроллеру команд через общий FIFO-адрес. Операции, выполняемые в режиме отладки, не оказывают воздействия на регистры буфера.

При входе в режим отладки циклический указатель буфера будет ссылаться на регистр, содержащий адрес первой из пяти ранее выполненных команд. Дальнейшие чтения регистров будут происходить последовательно от первого адреса к адресу последней команды.

Например, пусть к моменту входа в режим отладки буфер был заполнен и указатель имел состояние 2. Это означает, что последней была запись в регистр 1, после которой произошло инкрементирование указателя, и он стал ссылаться на регистр 2, куда ранее был записан первый адрес из всей пятерки адресов.

10.1.3. Проверочные модули

Проверочные модули, как было сказано ранее, включают в себя стартовые наборы и отладочные модули. Они представляют собой готовые платы, предназначенные для отладки программного обеспечения ЦПОС в реальном времени, исследований возможностей и характеристик разрабатываемых алгоритмов и программ, а также для обучения работе с ЦПОС.

Стартовые наборы

Стартовые наборы (Starter Kit) предназначены для обучения работе с ЦПОС и разрабатываются для каждого процессора отдельно. Стартовый набор позволяет:

- изучить характеристики и принцип работы того или иного процессора;
- отладить не слишком сложные программы;
- выполнить несложное макетирование;
- проверить возможность применения процессора для использования его в конкретной задаче.

В состав стартового набора входят: плата, программное обеспечение и комплект документации.

На плате устанавливаются: процессор (как правило, младший в серии), аудио- или другой кодек (в зависимости от назначения процессора), последовательные и параллельные порты и др. Стартовый набор подключается к компьютеру через параллельный порт.

Отладочные модули

Отладочные модули, разрабатываемые под конкретные процессоры и конструируемые в виде платы, предназначены для проверки в реальных условиях разработанного алгоритма, реализованного в виде программы на языке ассемблера соответствующего процессора.

Отладочный модуль позволяет:

- произвести быструю отладку и оптимизацию алгоритма;
- изготовить на базе платы фактически законченное устройство;
- использовать установленную на плате периферию;

- практически избавиться от стадии макетирования устройства и значительно сократить время и затраты на этапе изготовления прототипа устройства.

Плата устанавливается в компьютер с интерфейсом ISA (PCI). Обобщенная структурная схема отладочного модуля показана на рис. 10.11. Обычно на плате устанавливаются:

- целевой процессор;
- разнообразная память данных/программ (асинхронная SRAM, синхронная динамическая SDRAM и др.);

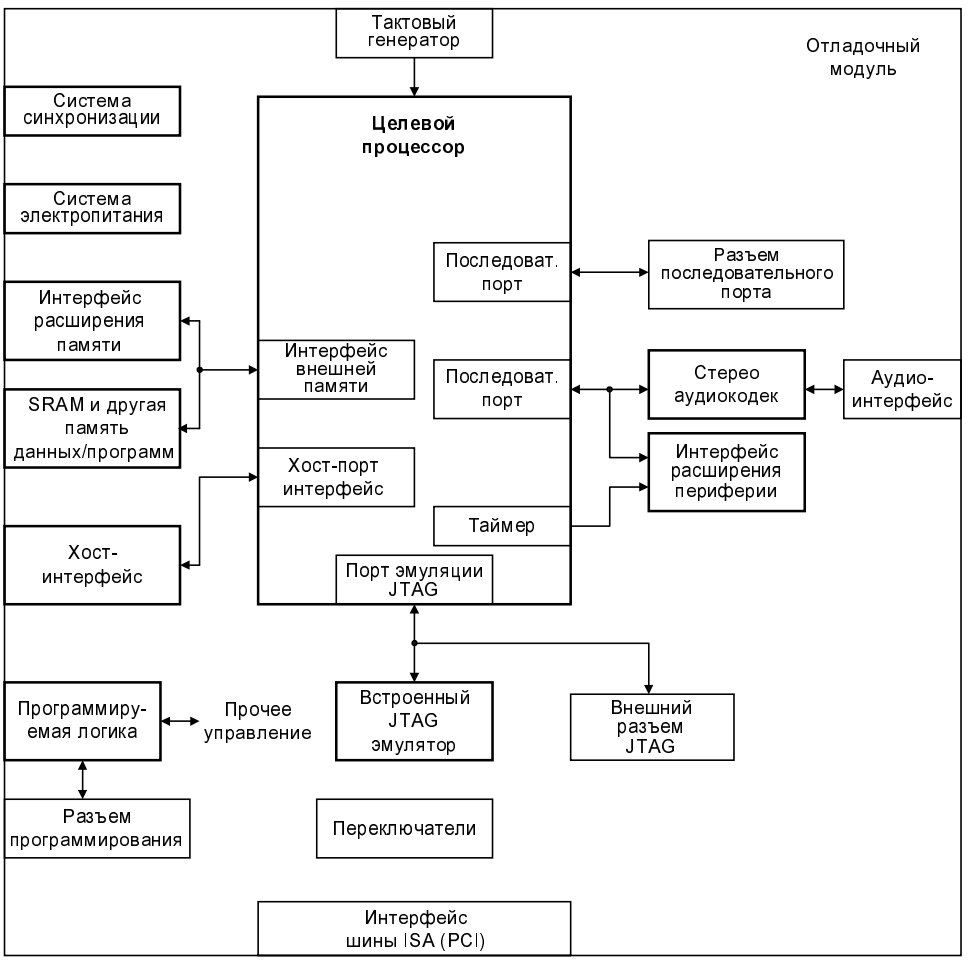


Рис. 10.11. Обобщенная структурная схема отладочного модуля

- тактовые генераторы и система синхронизации;
- интерфейсы расширения памяти и периферии для подключения дополнительной памяти и другой периферии, установленных на дочерних платах;
- хост-интерфейс;
- разнообразный аудиоинтерфейс, включающий стереовходы микрофона и линейные стереовыходы;
- разнообразная программируемая логика для управления платой;
- встроенный JTAG-эмулятор;
- система электропитания;
- группа переключателей, обеспечивающих конфигурирование модуля независимо от компьютера;
- интерфейс шины ISA (PCI) для подключения к компьютеру.

Кроме перечисленного оборудования, плата может содержать другие устройства и разъемы; например, отладочный модуль C6000 EVM имеет внешний разъем питания и разъем для подключения внутрисхемного эмулятора XDS510 с целью использования модуля вне компьютера. Отладочный модуль SETMobile, разработанный под платформу TMS320C5000, имеет набор высокоскоростных как синхронных, так и асинхронных последовательных портов со скоростью обмена до 80 Мбит/с и возможностью подключения к стандартным синхронным потокам.

10.2. Программные средства отладки

10.2.1. Симуляторы системы команд

Симуляторы системы команд, или просто *симуляторы*, являются программами, имитирующими работу того или иного процессора только на уровне его команд. Симуляторы отображают:

- команды программы, разработанной для процессора;
- состояние регистров и результаты вычислений;
- состояние памяти;
- время выполнения команды в циклах процессора,

а также другие характеристики и параметры результатов выполнения исследуемой программы.

Обычно перед испытанием на аппаратных средствах программа (или ее отдельные блоки) проверяется на симуляторе, хотя опытные разработчики систем ЦОС ради экономии времени и удобства используют только отладочные модули.

Симуляторы являются ключевым инструментом для отладки и оптимизации программного продукта и предоставляют пользователю возможность как пошагового, так и автоматического выполнения программы, а также управлять содержимым регистров и памяти. Симуляторы разрабатываются для каждого образца процессора и могут существенно отличаться по своим характеристикам.

Основными показателями, отличающими симуляторы от других программных средств разработки, являются: точность, скорость, сложность, поддержка отладки и оптимизации, поддержка точек останова. Рассмотрим эти показатели.

- **Точность.** С точки зрения процесса вычисления представляют интерес два рода точности: функциональная точность и точность синхронизации.
  - *Функциональная точность* отображает верность, с которой симулятор моделирует функциональные возможности процессора. Функциональная точность оценивается вероятностью, с которой некоторая последовательность команд, обрабатывающая определенный массив данных, будет давать те же результаты на симуляторе, что и на самом процессоре. Функциональная точность симулятора несколько меньше, чем точность процессора. Неточности в симуляторах обнаруживаются в процессе их эксплуатации, о чем поставщики симуляторов сообщают своим пользователям.
  - *Точность синхронизации* характеризует ту достоверность, с какой симулятор моделирует временные соотношения между операциями внутри процессора. Дело в том, что, как было показано ранее, некоторые команды выполняются процессором более чем за один командный цикл, поэтому такие команды могут затрачивать различное время для своего выполнения в зависимости от событий, происходящих в процессоре (например, блокировки прерывания или конфликт шин внутренней памяти). Добиться точного моделирования временных соотношений в симуляторе практически невозможно, поскольку чем точнее воспроизводятся временные согласования, тем ниже скорость работы симулятора.
- **Скорость работы симулятора.** Симуляторы обладают существенно более низкой скоростью, чем имитируемые процессоры, поэтому отладка сложных алгоритмов, таких как БПФ или линейное предсказание, на симуляторе может оказаться малоэффективной.
- **Сложность.** Этот показатель характеризует возможность симулятора имитировать работу не только ядра процессора, но и периферии: внутрикристальных таймеров, портов и интерфейсов. Большинство современных симуляторов имитируют интерфейсы, что позволяет разработчику увеличивать функциональную точность симулятора путем имитации, например, многопроцессорных структур.

- **Поддержка отладки и оптимизации.** Разработчик взаимодействует с симулятором через буферную программу, называемую отладчиком. Современные отладчики (например, CodeComposer) могут использоваться как с симулятором, так и с внутрисхемным эмулятором данного процессора. Возможности отладки и оптимизации определяются именно отладчиками, которые обсуждаются ниже.
- **Поддержка точек останова.** Все симуляторы поддерживают точки останова — определенные пользователем условия, которые принуждают симулятор остановиться. Различают простые и сложные точки останова:
  - простой точкой останова называют определенную пользователем точку программы, при достижении которой симулятор должен остановиться;
  - сложной точкой останова называют точку, обусловленную каким-либо выражением; например, "остановить программу при обращении по адресу P:250h" или "остановить программу после пятикратного выставления числа 20h на шину данных".

10.2.2. Отладчики

Отладчиком называют буферную программу, предоставляющую разработчику необходимый интерфейс и обеспечивающую функциональные возможности симулятора и внутрисхемного эмулятора. Современные отладчики характеризуются высокой производительностью, простотой использования. Они имеют разнообразный интерфейс:

- *символьно-ориентированный*, работающий в режиме командной строки. Этот интерфейс обеспечивает текстовый обмен информацией в пределах одиночного окна. Такие интерфейсы могут создавать трудности при отладке сложных программ;
- *символьный*, работающий в режиме окна. Экран делится на окна для отображения различных видов информации. Именно такой подход применяется в современных отладчиках;
- *графический*, работающий в режиме окна. Данный интерфейс является наиболее гибким и удобным, обладает доступной разнообразной настройкой изображения на экране. Отладчики с подобными интерфейсами производятся фирмами Analog Devices, Texas Instruments, Motorola.

Обеспечение функциональных возможностей симуляторов и внутрисхемных эмуляторов достигается:

- *символьным отлаживанием*, которое поддерживается всеми современными отладчиками; символьное отлаживание означает, что обращение к переменным происходит не с помощью указания их адресов, а путем использования их символьных имен. Символьное отлаживание является важным как для языка ассемблера, так и для программирования на языке высокого уровня;

- *отладкой на уровне исходного кода*, которая означает возможность обращения с объектами программы путем ссылки непосредственно на исходный текст: в окне отображается исходный текст программы (на языке ассемблера или языке высокого уровня) и высвечивается каждая строка, которую проходит симулятор в пошаговом режиме;
- *отладкой данных* при переходе от одной точки программы к другой *с выбором форматов* представления: двоичное, шестнадцатеричное, десятичное целое и др.; для этого предусматриваются окна наблюдения, отображающие содержимое регистров и памяти. Переменные в окнах наблюдения обновляются всякий раз после точки останова или после выполнения команды в пошаговом режиме;
- *профилированием*, позволяющим выяснить, в каком месте или за счет каких областей (участков) программы неоптимально расходуется временной ресурс. Большинство приложений ЦОС весьма чувствительно к времени, затрачиваемому на выполнение программы, при разработке которой как непосредственно на языке ассемблера, так и с использованием С-компиляторов далеко не всегда удается минимизировать время, затрачиваемое на выполнение программы. Профилирование может осуществляться как покомандно, так и позонно, по желанию пользователя. В случае *покомандного профилирования* отладчик предоставляет список выполненных команд и количество исполнений каждой команды либо во всей программе, либо в заданном блоке, который ограничивается начальным и конечным адресом. При *позонном профилировании* отладчик фиксирует время, в течение которого доступны различные зоны программы. Зоны представляют собой участки программы, выделяемые пользователем; это могут быть подпрограммы и различные функционально законченные блоки. Полученная при профилировании информация помогает пользователю принять решение об оптимизации программы с целью сокращения времени ее выполнения;
- *дизассемблированием* — восстановлением программы на языке ассемблера по объектному коду; это дает возможность разработчику не только следить за ходом выполнения программы от команды к команде, но быстро корректировать обнаруживаемые ошибки без перезагрузки всей программы;
- *протоколированием команд*; это свойство обеспечивает возможность создавать логический файл из некоторой последовательности команд: в протоколе записываются не только выполненные команды, но и результат на выводе отладчика; это означает, что при многократном повторении сеанса отладки с целью сокращения времени можно организовать выполнение команд из заранее созданного логического файла.

Замечание

Большинство современных отладчиков поддерживает отлаживание как ассемблерных программ, так и программ, написанных на языке высокого уровня с использованием С-компиляторов.

10.2.3. Интегрированные отладочные средства

Появление мощных процессоров, подобных семействам TMS320C5xxx и TMS320C6xxx, вызвало необходимость в новых отладчиках, которые, в дополнение к рассмотренным в *разд. 10.2.2* свойствам, предоставляли бы пользователю более широкие возможности и удобства с целью сокращения сроков разработки и отладки систем ЦОС. В связи с этим появилось новое поколение отладочных средств, получивших наименование "интегрированная среда отладки"; сюда относятся среда разработчика CodeComposer и среда разработчика CodeComposerStudio.

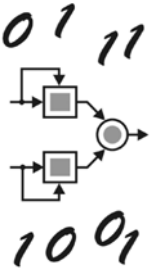
Среда разработчика CodeComposer обладает следующими новыми возможностями:

- редактирования, компилирования и отлаживания программы, не выходя из отладочной среды;
- одновременного отлаживания на языках С и ассемблера нескольких программ, написанных для разнородных процессоров с отдельными окнами отладки;
- создания собственных сценариев на С-подобном внутреннем языке GEL и встраивания их в интерфейс отладчика;
- выполнения сценариев при попадании на заданную точку останова;
- установки *точек подключения* с заданием условий выполнения каждой точки; точка подключения используется для:
  - обзора сигналов в определенные пользователем моменты;
  - подачи сигналов из файлов в задаваемые точки системы;
  - считывания сигналов в заданных точках и записи их в файл;
  - фиксирования состояния ограниченной области памяти.

Среда разработчика CodeComposerStudio включает в себя все возможности CodeComposer и дополнительно обеспечивает:

- непрерывный обмен между процессором и отладчиком в реальном времени;
- анализ и отладку в реальном времени;
- визуализацию состояния системы в реальном времени, а также БПФ, временного анализа, многих видов стандартных диаграмм и т. д.;
- подключение нескольких различных отладочных систем;
- работу с несколькими различными процессорами на одном канале JTAG;
- сетевой менеджмент проектов, позволяющий координировать совместную разработку проектов группой.

Глава 11



Разновидности и характеристики ЦПОС

11.1. Квалификационные параметры и характеристики ЦПОС

Современные процессоры ЦПОС являются сложными устройствами с большими возможностями. Фирмы-производители ЦПОС выпускают большое количество самых разнообразных процессоров с различными характеристиками. Особенности архитектуры ЦПОС, важные для реализации алгоритмов ЦОС, были рассмотрены в *главе 2*. ЦПОС используются в самых различных областях, начиная с применений в устройствах радиолокации и заканчивая бытовыми приборами. Естественно, не существует идеального процессора для всех областей применений. Для каждой из них при реализации различных алгоритмов ЦОС оказываются важными те или иные характеристики процессоров (см. "Choosing a DSP Processor", choose_2000.pdf, **http://www.bdti.com**). Рассмотрим характеристики ЦПОС, которые важны при выборе процессора для конкретной разработки и обычно приводятся в различных таблицах для сравнения. Некоторые из этих характеристик уже упоминались ранее.

- **Тип арифметики. Форма представления данных с плавающей или с фиксированной точкой**. Формы представления данных с ПТ и ФТ рассматривались в *разд. 2.3* и *главе 3*. Не повторяя всего сказанного ранее, напомним, что форма с ПТ является более гибкой и удобной при разработке системы обработки сигналов, однако соответствующие процессоры являются более сложными и дорогими устройствами.
- **Разрядность данных**. Все обычные ЦПОС с плавающей точкой используют слово данных длиной в 32 бита. Для ЦПОС с фиксированной точкой обычный размер слова данных — 16 битов. ЦПОС фирмы Motorola применяет слово данных в 24 бита. Большинство процессоров допускают обработку с двойной точностью.
- **Быстродействие**. Одним из самых важных параметров с точки зрения конкретных применений является быстродействие процессора. Для характеристики быстродействия используют различные параметры, однако все они определяют только конкретные стороны проблемы. Реальнее характеризует быстродействие системы время решения различных реальных

задач и тестов. Соответствующий материал приведен далее в *разд. 11.2*. В настоящей главе рассматриваются некоторые технические характеристики процессоров, определяющие быстродействие.

- **Тактовая частота работы процессора** и связанное с ней **время командного цикла (цикла ЦПУ)**. Как правило, при описаниях процессоров обычно указывается внешняя тактовая частота, подаваемая на процессор. Она может отличаться от внутренней частоты работы из-за наличия системы деления или умножения частоты (системы PLL). Для последних процессоров, в которых внешняя частота может изменяться в широких пределах, чаще указывают внутреннюю частоту работы процессора.

  Время командного цикла связано с внутренней частотой работы процессора. Оно определено в *разд. 2.2.3*. Так как отдельная операция в процессоре может выполняться как за несколько циклов, так и за один, время командного цикла является самой неоднозначной характеристикой быстродействия процессора. К тому же, в некоторых процессорах используется параллельное выполнение команд и параллельная работа нескольких операционных модулей. Поэтому время цикла полностью не характеризует реально выполняемую процессором работу.

  Тактовая частота работы процессоров фирмы TI отражается в их маркировке *(см. приложение 4)*.

- **Количество миллионов команд, выполняемых за секунду MIPS** (Million instructions per second). В ЦПОС используются различные команды, в том числе комбинированные, в соответствии с которыми одновременно выполняется несколько операций. Кроме того, существуют процессоры с несколькими АЛУ, в которых применяются длинные команды, а так же процессоры с архитектурой VLIW *(см. разд. 2.4)*. Таким образом, одной команде в разных процессорах соответствует различная выполняемая работа. Поэтому характеристика MIPS неоднозначно определяет быстродействие процессора.

- **Количество миллионов операций за секунду MOPS** (Millions operations per second). Эта характеристика более однозначно, по сравнению с другими, характеризует быстродействие, т. к. учитывает выполнение параллельных команд и одновременную работу нескольких операционных модулей. Однако нет стандартного определения операции. Иногда к выполняемым операциям относят и выборки команд, и запись в память полученных результатов.

- **Количество миллионов операций с плавающей точкой за секунду MFLOPS** (Millions of floating-point operations per second). Эта характеристика используется в процессорах с плавающей точкой. К ней относится все сказанное относительно MOPS.

- **Количество операций MAC в единицу времени** *(см. главу 2)*. Возможный путь определения производительности состоит в выборе единой простой

операции для целей сравнения. Для прикладных программ ЦОС естественным является выбор операции умножения/накопления MAC, которая является основной для алгоритмов ЦОС. Однако в указанных алгоритмах применяются и другие операции помимо этой.

- **Объем и разновидности внутренней памяти (ROM, OTP ROM, RAM, Flash, кэш)**. Эти характеристики определяют многие параметры и возможности разрабатываемой системы. Наличие памяти типа *ROM* (ПЗУ, программируемого при изготовлении процессора) позволяет заказывать ЦПОС с записанной программой работы системы. Такой вариант использования ЦПОС экономически оправдан при крупносерийном производстве. Память типа *OTP ROM* (One Time Programmable ROM, однократно программируемое ПЗУ) позволяет моделировать и тестировать систему при отработке программного обеспечения, а также изготавливать единичные и мелкосерийные образцы. Память типа *Flash* [21], [41] позволяет неоднократно перезаписывать программу и данные в процессоре, в том числе и на "рабочем месте", т. е. непосредственно на изготовленной плате системы. Объем и разновидности памяти типа *RAM* определяют возможности построения системы без использования внешней памяти, как для хранения данных, так и загружаемой программы *(см. разд. 2.6 и 9.5)*. Достоинства и работа кэш-памяти рассматривались в *разд. 2.6.3*.

  Тип технологии и разновидности используемой внутренней памяти процессоров фирмы TI отражаются в их маркировке *(см. приложение 4)*.

- **Объем адресного пространства памяти** определяется разрядностью шины адреса (единой для внешней памяти) и характеризует возможный общий объем памяти, используемой в системе.

- **Boot Loader**. Начальный загрузчик выполняемой программы во внутреннюю память *(см. разд. 9.5)*.

- **Количество и разновидности портов последовательного ввода информации** определяют возможности системы с точки зрения связи с различными внешними устройствами *(см. главу 8)*.

- **Внутренние периферийные устройства**. Используемые в ЦПОС периферийные устройства рассмотрены в *главе 8*. Следует отметить, что в процессорах существуют периферийные устройства, которые условно можно разделить на *устройства общего применения* (типа таймеров) и *проблемно-ориентированные устройства* (АЦП, кодеки, компрессоры, сопроцессоры и т. д.). Последние облегчают построение специализированных цифровых систем типа обработки звука и т. д.

- **Наличие и количество каналов DMA (Direct Memory Access)**. Прямой доступ к памяти *(см. главу 8)* позволяет общаться с внешними устройствами, в том числе записывать отсчеты входного сигнала (выводить полученные отсчеты выходного сигнала) без использования ресурсов и затрат време-

ни ЦПУ. Это очень эффективная особенность, облегчающая построение высокопроизводительных систем.

- **Напряжение питания, потребляемый ток**. Характеристики процессора, особенно важные при построении переносимых систем с батарейным питанием. С этими характеристиками связан такой показатель, как потребляемая мощность. Следует отметить, что потребляемая мощность существенно зависит от выполняемой программы и, как правило, не приводится.

  Многие производители предлагают низковольтовые (3,3 В, 2,5 В или 1,8 В) версии процессоров, которые потребляют гораздо меньше мощности, чем пятивольтовые эквиваленты при той же производительности.

  Процессор может работать в различных режимах, в том числе в режиме ожидания (Idle), при нахождении в котором ряд внутренних модулей отключается и не потребляет энергии. Поэтому потребляемый ток иногда приводят для различных режимов работы.

  Многие современные процессоры с пониженным напряжением питания используют различное напряжение для ЦПУ и периферийных устройств. Например, процессор фирмы ADI ADSP-2186M использует 2,5 В для питания ядра (ЦПУ) и 3,3 В для устройств ввода/вывода. Некоторые ЦПОС позволяют программно отключать неиспользуемые периферийные устройства.

  Напряжение питания и потребляемая мощность процессоров фирмы TI, а также ADI отражаются в их маркировке *(см. приложение 4)*.

- **Комбинированные относительные показатели типа "мощность/ток/быстродействие"**. Естественным свойством любых электронных устройств, в том числе и ЦПОС, является повышение потребления мощности при увеличении быстродействия. Поэтому многие фирмы в качестве показателя эффективности процессора используют удельные относительные показатели потребления энергии или тока, отнесенные к некоторой единице быстродействия, например показатель ma/MIPS применяется для характеристики процессоров TMS320 и ADSP. Иногда в такие удельные показатели включают и стоимость процессора.

- **Наличие различных средств и информационных ресурсов сопровождения разработки**. Виды сопровождения разработки цифровой системы на конкретном процессоре могут быть самые разнообразные. К ним можно отнести:

  - наличие и состав пакетов программного обеспечения разработки *(см. главу 9)*;
  - наличие и состав средств отладки систем, в том числе визуальных средств *(см. главу 10)*;
  - наличие и доступность документации, информационная поддержка, в том числе и через Internet;

- существование библиотек стандартных программ и математических функций *(см. главу 9)*;
- наличие совместимых с процессором устройств преобразования данных АЦП и ЦАП, а также устройств поддержки энергопотребления (стабилизаторы, контроллеры питания и т. д.).

Помимо рассмотренных выше, есть также другие характеристики ЦПОС, которые необходимо учитывать при разработке цифровой системы обработки сигналов. К ним можно, например, отнести допустимый температурный диапазон работы, тип корпуса.

Выбор процессора для конкретной разработки целиком определяется назначением разрабатываемой системы.

Для недорогих устройств типа мобильных телефонов, дисководов, игровых приставок, которые выпускаются в большом количестве, стоимость, сравнительная простота и потребляемая мощность являются определяющими. Сложность и стоимость разработки подобной системы играют меньшую роль.

При обработке больших объемов данных по сложным алгоритмам (например, обработка звука, сейсмические исследования) определяющими являются другие характеристики — эффективность процессора, простота разработки, возможность построения многопроцессорной системы.

11.2. Сравнение производительности процессоров

Как отмечено в *разд. 11.1*, стандартные характеристики ЦПОС типа "тактовая частота работы процессора", MIPS, MOPS, MAC далеко не однозначно оценивают его быстродействие.

Наиболее правильным подходом к оценке быстродействия является использование набора тестовых программ, реализующих стандартные алгоритмы ЦОС (см. "Choosing a DSP Processor", choose_2000.pdf, **http://www.bdti.com**), и оценка времени их выполнения различными ЦПОС. Такой подход использовала компания Berkeley Design Technology, Inc (BDTI) — см. "The BDTImark2000: A Measure of DSP Execution Speed", BDTImark2000.pdf, **http://www.bdti.com**. BDTI специализируется на анализе, эталонном тестировании, оценке и развитии новых технологий и микропроцессоров, разрабатывает прикладные программы ЦОС. Заказчиками BDTI являются все ведущие производители ЦПОС.

BDTI использует для оценки процессоров как отдельные алгоритмы ЦОС, так и разработанную оценку скорости работы процессора, полученную на наборе эталонных алгоритмов ЦОС (BDTImark). BDTI использует не полные программы, а упрощенные прикладные ядра программ. Эти ядра — маленькие

фрагменты кода DSP, которые формируют ключевые части больших прикладных программ. Ядра, например, включают: вычисления отсчета фильтра КИХ и БИХ, реализация "бабочки" БПФ. Все эталонные тесты написаны полностью в ассемблере и тщательно оптимизированы для ЦПОС, на котором они выполняются. Большинство эталонных программ пишутся BDTI, и лишь некоторые — изготовителями ЦПОС. Во всех случаях функциональные возможности и эффективность эталонных программ независимо проверяются BDTI.

В 2000 году эталонный набор был модифицирован и создана оценка BDTImark2000 (см. "The BDTImark2000: A Measure of DSP Execution Speed", BDTImark2000.pdf, **http://www.bdti.com**). Набор включает следующие задачи ЦОС:

- реализацию фильтров КИХ и БИХ;
- перемножение векторов;
- реализацию декодера Витерби;
- БПФ

и некоторые другие задачи.

Оценка использует "родной" формат данных (ФТ и ПТ) тестируемых процессоров. BDTI оговаривает, что BDTImark является оценкой быстродействия выполнения процессором задач ЦОС в обобщенном виде, но не определенных прикладных программ. Возможности процессора при выполнении одной конкретной прикладной программы не отражаются в BDTImark.

Кроме того, BDTI оценивает полное время разработки системы, использование памяти выполняемыми программами.

Ниже приводятся оценки различных ЦПОС по материалам, опубликованным BDTI[1]. Оценки носят, прежде всего, сравнительный характер и интересны для сопоставления различных процессоров.

В табл. 11.1 приведено время реализации комплексного КИХ-фильтра.

Таблица 11.1. *Время реализации комплексного КИХ-фильтра*

Фирма	ЦПОС	Тип арифметики	Тактовая частота, МГц	Время, мкс
ADI	ADSP-2189	ФТ	75	40,5
Lucent Technologies	DSP1620	ФТ	120	27

[1] См. "The BDTImark2000: A Measure of DSP Execution Speed", BDTImark2000.pdf; "Independent DSP Benchmarks: Methodologies and Results. BDTI, 1999", benchmark_icspat99.pdf; "Independent DSP Benchmark Results for the Latest Processors. BDTI, 2000", benchmark_000407.pdf; "DSP Benchmark Results for the Latest VLIW-Based Processors. BDTI, 2000", vliw_icspat00.pdf, по адресу **http://www.bdti.com**.

Таблица 11.1 *(окончание)*

Фирма	ЦПОС	Тип арифметики	Тактовая частота, МГц	Время, мкс
TI	TMS320C549	ФТ	120	24
Lucent Technologies	DSP16210	ФТ	120	14,2
Motorola	DSP56311	ФТ	150	19
LSI Logic	LSI401Z	ФТ	200	8,2
TI	TMS320C6202	ФТ	250	5,5
TI	TMS320C31	ПТ	40	77
ADI	ADSP-21065	ПТ	60	40,5
ADI	ADSP-21160	ПТ	100	15,5
TI	TMS320C6701	ПТ	167	11,5

В табл. 11.2 представлены оценки теста BDTImark1999 скорости работы процессоров (чем больше, тем лучше), полученные по решению набора задач.

Таблица 11.2. *Оценки BDTImark скорости работы процессоров*

Фирма	ЦПОС	Тип арифметики	Тактовая частота, Гц	Скорость BDTImark1999
TI	TMS320VC549	ФТ	120	30
Lucent Technologies	DSP16210	ФТ	120	45
Lucent Technologies	DSP1620	ФТ	120	22
Motorola	DSP56311	ФТ	150	38
LSI Logic	LSI401Z	ФТ	200	70
TI	TMS320C6202	ФТ	250	123
TI	TMS320C6701	ПТ	167	65

В табл. 11.3 приведено время реализации КИХ-фильтра.

Таблица 11.3. *Время реализации КИХ-фильтра*

Фирма	ЦПОС	Тип арифметики	Тактовая частота, МГц	Время, мкс
TI	TMS320C5416	ФТ	160	4,5

Таблица 11.3 *(окончание)*

Фирма	ЦПОС	Тип арифметики	Тактовая частота, МГц	Время, мкс
TI	TMS320C5510	ФТ	160	2,5
TI	TMS320C6203	ФТ	300	1,0
Motorola	MSC8101	ФТ	300	0,5
TI	TMS320C64xx	ФТ	600	0,3

В табл. 11.4 указаны объемы памяти, занимаемые программой при реализации КИХ-фильтра. Как отмечалось в *главе 2*, процессоры с архитектурой типа VLIW (TMS320C62xx, TMS320C64xx) менее экономно расходуют память, чем процессоры с улучшенной стандартной архитектурой (TMS320C55xx) и стандартной архитектурой (TMS320C54xx). Это хорошо иллюстрируется данными, приводимыми в таблице.

Таблица 11.4. *Объемы памяти, занимаемые программой при реализации КИХ-фильтра*

Фирма	ЦПОС	Тип арифметики	Объем программы, байт
TI	TMS320C54xx	ФТ	220
TI	TMS320C55xx	ФТ	300
Motorola	MSC8101	ФТ	420
TI	TMS320C62xx	ФТ	760
TI	TMS320C64xx	ФТ	810

В табл. 11.5 представлены оценки BDTImark2000 скорости работы процессоров (чем больше, тем лучше), полученные по решению набора задач.

Таблица 11.5. *Оценки BDTImark2000 скорости работы процессоров*

Фирма	ЦПОС	Тип арифметики	Тактовая частота, МГц	Скорость BDTImark2000
ADI	ADSP-21xx	ФТ	75	230
Motorola	DSP56800	ФТ	80	110
Motorola	DSP56300	ФТ	150	450
ADI	ADSP-219x	ФТ	160	420
TI	TMS320C54xx	ФТ	160	500
Lucent Tecnologies	DSP164xx	ФТ	170	810

Таблица 11.5 *(окончание)*

Тип арифметики	Тактовая частота, МГц	Скорость BDTImark2000	Фирма	ЦПОС
ФТ	300	1920	TI	TMS320C62xx
ФТ	300	3430	Motorola	MSC8101 StarCore SC140
ПТ	66	250	ADI	ADSP-2106x
ПТ	80	410	ADI	ADSP-2116x
ПТ	130	2530	Intel	Pentium III
ПТ	167	820	TI	TMS320C67xx

В табл. 11.6 приведено время выполнения комплексного БПФ на 256 точек в различных процессорах. Для сравнения приведены также оценки для процессора общего назначения Pentium III.

Таблица 11.6. *Время выполнения комплексного БПФ*

Фирма	ЦПОС	Тип арифметики	Тактовая частота, МГц	Время, мкс
Motorola	DSP56311	ФТ	150	58
TI	TMS320C5416	ФТ	160	65
TI	TMS320C6203	ФТ	300	9
Motorola	MSC8101	ФТ	300	6
TI	TMS320C6701	ПТ	167	21
Intel	Pentium III	ПТ	1000	9,5

Оценки BDTI по влиянию типа архитектуры на быстродействие процессора и сравнению программ, полученных с использованием языка С и ассемблера, рассматриваются в *разд. 2.5* и *9.6* соответственно.

11.3. Разновидности ЦПОС с точки зрения назначения

Как было показано в *главе 2*, все ЦПОС имеют общие возможности, определяемые алгоритмами ЦОС и, прежде всего, все они хорошо "умеют" выполнять операции MAC — умножения с накоплением, характерные для всех задач ЦОС. Тем не менее, ЦПОС разделяют на *процессоры общего применения* и *проблемно-ориентированные* (application specific) *процессоры* [1].

Проблемная ориентация ЦПОС заключается, в первую очередь, в использовании внутренних специализированных периферийных устройств и в меньшей степени в добавлении специализированных команд.

Приведем примеры проблемно-ориентированных процессоров.

- **Семейство TMS320C2000 (TI)**. Данное семейство предназначено для реализации систем цифрового управления [20]. Семейство привлекательно для реализации систем тяговых приводов с векторным управлением (электромобили, локомотивы, малые транспортные средства), а также приводов, работающих на активную нагрузку (краны, подъемники, канатные дороги, лифты), изделий массового спроса (бытовой и офисной техники). Привлекательной стороной семейства является предельно низкое потребление энергии. Семейство включает уникальную комбинацию различных встроенных периферийных устройств: память типа Flash, преобразователи АЦП, контроллер локальной промышленной сети (CAN-контроллер) для организации локальной промышленной сети, коммуникационный (SCI) и периферийный (SPI) интерфейсы, процессоры событий, генераторы сигналов ШИМ и т. д.
- **Семейство ADMC (ADI)** аналогично по назначению семейству TMS320C2000. Примерами могут служить процессоры ADMSF326 (27, 28), имеющие встроенную память типа Flash, а также ADMC401, ADMC331 и др.
- **Процессор LSI403Z (LSI Logic Corporation)**, оптимизированный для устройств инфраструктуры систем и сетей связи, передачи речи по сетям, обработке аудиосигналов. За счет наличия специализированных устройств ввода/вывода обеспечивается прямая связь со стандартными интерфейсами передачи данных. Специализированные команды: команды сравнения, выбора, сложения для декодирования Витерби, команды операций с отдельными разрядами, 32-разрядные логические операции.
- **Процессоры DSP16xx (Lucent Tecnologies)**. Процессоры разработаны для применения в системах сотовой связи, реализации базовых и терминальных станций. Устройство содержит специализированный модуль манипуляции разрядов (BMU) и сопроцессор исправления ошибок (ECCP) для повышения эффективности декодирования Витерби. Процессоры включают ряд периферийных устройств для обмена сигналами цифровой беспроводной связи.
- **Семейство DSP56600 (Motorola)**. Семейство предназначено для использования в радиосетях передачи информации, в сотовых сетях и других областях, где требуется реализация задач ЦОС и задач управления. В состав процессоров семейства входят ядро ЦПОС и микроконтроллер.
- **Процессор TMS320DRE200 (TI)** применяется для решения всех задач при приеме сигналов. Проектируемая система включает процессор ЦПОС и все

компоненты, необходимые для реализации цифрового приема, включая аналоговые части и схемы АЦП, ЦАП, усилитель мощности и т. д. Процессор предназначен для производителей цифровой техники. Используя открытую программную часть, заказчики могут быстро добавлять различные прикладные программы. Например, заказчик может интегрировать декодирование MP3 к цифровому радио. Это позволяет изготовителям легко строить разные системы цифрового радио, от основных до сложных.

- **Процессор TMS320DSCx — C21, C24 (TI)**. Устройство интегрирует два ядра обработки — ЦПОС TMS320C5000 и RISC-микроконтроллер. Процессор предназначен для обработки изображений в цифровых камерах в реальном масштабе времени при одновременном контроле за системными функциями типа непрерывного автоматического управления размером окна, фокусировки, процессом записи. ЦПОС позволяет производить в реальном масштабе времени сжатие видео- и аудиосигналов, так что камера может делать запись видео с аудио непрерывно, допуская использование видеофильма без обширных и дорогостоящих дискретных буферов памяти.

  Процессор поддерживает различные стандарты средств и форматы файлов типа JPEG, MPEG1, MPEG4, M.-JPEG, H.263, MP3, AAC, QuickTime и AVI. Программируемый характер процессора дает возможность изготовителям камеры не отставать от изменяющихся стандартов и расширять их номенклатуру и особенности. TMS320DSCx также поддерживает обработку во многих других конечных видах оборудования, включая сканеры, принтеры, цифровые видеомагнитофоны и сотовые телефоны.

  Процессор используется в камерах Kodak DX3500 и DX3600.

- **Процессор TMS320DA250 (TI)** предназначен для решения различных задач, связанных с использованием Internet Audio. Он позволяет строить портативные устройства, включает интерфейс USB, устройства управления батарейным питанием. Процессор может также применяться для реализации цифровых камер со сжатым аудио, сотовых телефонов и цифровых динамиков, аудиоплееров для воспроизведения записей в различных форматах. ЦПОС совместим по архитектуре и программной реализации с семейством TMS320C54xx.

- **Процессор TMS320C5472 (TI)** служит для реализации систем IP-телефонии, т. е. систем передачи телефонных сигналов по сети Internet с помощью соответствующих протоколов передачи сообщений. Устройство объединяет ядро ЦПОС TMS320C54x и ЦПУ с архитектурой RISC ARM7TDM фирмы ARM Ltd. ЦПУ ARM7TDM реализует операционную систему реального времени (ОС РВ). Ядро ЦПОС имеет следующие характеристики: 16 разрядов, тактовая частота 100 МГц, внутренняя память 32 Кбайт слов, часть которой используется для связи с ЦПУ, 50 МГц ARM7TDM, который имеет свою память 16 Кбайт. Периферийные устройства включают: 2 дуплексных буферизованных многоканальных порта

(McBSPs), 6-канальный контроллер DMA, двухпортовый модуль связи с сетью Internet, асинхронный приемопередатчик UART, таймеры и т. д. Напряжение питания 1,8 В для ядра и 3,3 В для периферии.

- **Процессор DSP56366 (фирма Motorola)** поддерживает системы цифровой обработки звука. На эти применения ориентированы порты и интерфейсы приема/передачи аудиосигналов по различным протоколам.
- **Семейства CS481x, CS492x, CS49300 (фирма Cirrus Logic Inc.)**. Семейства предназначены для реализации самых разнообразных устройств, связанных с обработкой, кодированием и декодированием аудиосигналов в разнообразных форматах: стерео и многоканальных аудиодекодеров Dolby Digital, MPEG, DTS, DVD-дисков реализации разнообразных звуковых эффектов.

  Они представляют набор высококачественных аудиопериферийных устройств, включающих мультимедиа "кодер-декодеры", стерео-АЦП и ЦАП-конвертеры, интерфейсы ввода/вывода звуковых сигналов IEC60958 и др., с 24-разрядным ЦПОС с ФТ. Производительность используемых ЦПОС лежит в пределах от 17 до 86 MIPS. Элементы данного семейства предназначены для применения как в составе отдельных устройств типа плееров и игровых приставок, так и в качестве встраиваемых элементов. Процессоры используют программы реализации эффектов, декодирования и т. д., разработанные фирмой. Конкретные устройства могут быть реализованы на ROM- и RAM-основе. В последнем случае программа при включении питания переписывается во внутреннюю память из внешней памяти типа ROM.

  Среди различных ЦПОС следует выделить ядра ЦПОС, не предназначенные для самостоятельного использования. Примерами могут служить ядра нескольких ЦПОС.

- **Ядро Smart Core (фирма DSP Group)**. Фирма DSP Group проектирует, разрабатывает и торгует ядрами для реализации устройств цифровой обработки сигналов. Они предназначены для использования в качестве платформ для проблемно-ориентированных интегральных схем ЦОС (ASDSPs) и не доступны как автономные изделия.

  При разработке конкретного процессора ядра могут объединяться с любыми ячейками/функциями, доступными в соответствующей библиотеке модулей (память типа ROM/RAM, блоков ввода/вывода, периферийных устройств, и т. д.). Время разработки сокращается за счет наличия законченного набора инструментальных средств с большими библиотеками программных функций и модулей схемы. Результатом являются уникальные ИС, приспособленные к реализации определенных алгоритмов и программ. Считается, что на сегодняшнем конкурентоспособном рынке такие решения высоко рентабельны. Фирма DSP Group предлагает четыре разновидности ядер ЦПОС: PalmDSPCore, TeakDSPCore, OakDSPCore и PineDSPCore. Самые современные ядра PalmDSPCore и TeakDSPCore

оптимизированы для потребления малой мощности и высокой эффективности при реализации устройств. Они предназначены для широкого спектра приложений в беспроводной телефонии, передаче данных, обработки речи и встраиваемых систем управления.

Ядро OakDSPCore реализует 16-разрядные операции с ФТ и имеет следующие характеристики:

- цикл ЦПУ 25 нс;
- питание 5 или 3,3 В;
- умножение 16×16 с результатом в 32 разряда;
- выполнение операции MAC (умножение с накоплением) за один цикл;
- 36-разрядное АЛУ;
- четыре 36-разрядных аккумулятора;
- операции над отдельными разрядами;
- поддержка программными средствами: макроассемблер, компоновщик, С-компилятор, имитаторы, отладчики.

Ядро PalmDSPCore содержит два устройства для выполнения операций MAC, допускает параллельное использование ядер для обработки 16-, 20- и 24-разрядных данных. Тактовая частота работы ядра 210 МГц.

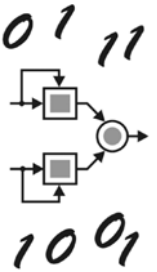
- **Ядро CARMEL Core (фирма Infineon Technologies)**. Аналогичным по назначению является CARMEL Core фирмы Infineon Technologies. Особенностью ядра выступает высокая степень параллелизма при выполнении команд. Оно способно одновременно генерировать четыре адреса, исполнять четыре арифметических операции и две передачи данных (2 операции MAC, 2 операции в АЛУ и 2 перемещения данных). В ядре применяется запатентованная архитектура со словом команды перестраиваемой длины (CLIW, Configurable Long Instruction Word). Архитектура CLIW позволяет получить возможности и эффективность архитектуры VLIW при низкой цене традиционных ЦПОС. Разрешается создавать до 1024 новых команд CLIW, каждая из которых может быть составлена из четырех арифметических и двух команд перемещения. Система команд может составляться, исходя из потребностей конкретной программы. При переменной длине команд повышается плотность их размещения в памяти и эффективность ее использования.

  Некоторые характеристики ядра:

  - тактовая частота работы ядра 250 МГц (командный цикл 4 нс);
  - малая потребляемая мощность 0,5 мВ/МГц при напряжении 1,2 В.

  Оценка скорости работы процессоров BDTImark2000 *(см. разд. 11.2)* для данного ядра равна 1850 при частоте 250 МГц, соответствующие данные для других процессоров приведены в табл. 11.5.

Приложение 1



Сравнительная таблица параметров процессоров

Ниже приведены данные в основном для упоминаемых в пособии процессоров, а также российских процессоров серии 1867 (табл. П.1.1).

Примечания к табл. П.1.1:

1. Другая единица измерения оговаривается ("K" означает умножение на $2^{10} = 1024$).
2. MFLOPS операций с плавающей точкой или MOPS операций с фиксированной точкой.
3. Миллионов 32-разрядных операций MAC.
4. Внутренняя частота; имеется система PLL, поэтому внешняя тактовая частота может быть ниже.
5. ЦПОС имеет сопроцессор для реализации цифровых фильтров.
6. ЦПОС имеет встроенный микроконтроллер.
7. 31,5 Кбайт ROM типа Flash память программ, 2 Кбайт ROM типа Flash память данных.
8. ЦПОС имеет встроенный коммуникационный контроллер, сопроцессор реализации цифровых фильтров.
9. Работа портов определяется коммуникационным контроллером.
10. Многопроцессорная система — 4 независимых ядра типа ADSP-21060.
11. ЦПОС, ориентированный на управление двигателями с соответствующей периферией; совместим с ADSP-21xx.
12. Интерфейс двухканального кодека для аудиоприменений.
13. Встроенный 4-канальный 8-разрядный АЦП.
14. ЦПОС имеет встроенный сопроцессор/декодер Витерби и Турбо-декодер.
15. Процессор является совокупностью двух независимых ЦПОС, связанных интерфейсом; данные приведены для одного ЦПОС.
16. Миллионов операций MAC.
17. Процессор состоит из двух ядер DSP16000 с общей периферией, ядра имеют локальные и разделяемые блоки памяти; имеются устройства, осуществляющие межпроцессорную связь через прерывания и буфера.
18. Память типа Flash.
19. ЦПОС имеет 8 каналов 10-разрядного АЦП с временем преобразования 900 нс и 8 генераторов ШИМ.
20. ЦПОС имеет 5 каналов 10-разрядного АЦП с временем преобразования 500 нс и 7 генераторов ШИМ.
21. ЦПОС имеет 16 каналов 10-разрядного АЦП с временем преобразования 375 нс и 16 генераторов ШИМ.
22. Ядро C27x содержит микроконтроллер со временем командного цикла 53 нс.
23. MFLOP — миллионов операций с плавающей точкой.
24. ЦПОС содержит два ядра с локальными ЗУ данных и программ и общей памятью данных (4 Кбайт); локальные ЗУ данных отличаются по объему; данные приведены для одного ядра.

Таблица П1.1. *Сравнительные характеристики параметров процессоров*

Фирма	Процессор	Разрядность данных	Тип арифметики	Частота, МГц	Время цикла, нс	MIPS[1]	Внутренняя память RAM, слов[1]		ROM, слов[1]	DMA	Таймеры	Последовательные порты	Напряжение ядра, В
							Данные	Программа					
ГП НИИЭТ	М1867ВМ1	16	ФТ	20	200	5	144	0	1,5	0	0		5
ГП НИИЭТ	Л1867ВМ2	16	ФТ	40	100	10	544	0	4	0	1	1	5
Analog Devices	ADSP-2101	16	ФТ	25	40	25	1K	2K	0		1	2	5
Analog Devices	ADSP-2105	16	ФТ	20	50	20	0,5K	1K			1	1	5
Analog Devices	ADSP-2171	16	ФТ	16,67	30	33	2K	2K			1	2	5
Analog Devices	ADSP-2184L	16	ФТ	20	25	40	4K	4K	0	2	1	2	3,3
Analog Devices	ADSP-2185L	16	ФТ	26	19	52	16K	16K		1	1	2	3,3
Analog Devices	ADSP-2189M	16	ФТ	75	13	75	56K	48K		2	1	2	2,5
Analog Devices	ADSP-2192[24]	16	ФТ	24,576	6,25	160	32(64)K	16K		5	1	2	2,5

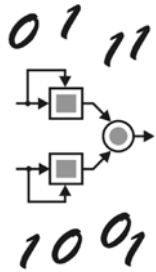
Analog Devices	ADSP-21061	32	ПТ	50	20	50	1 Мбит 2-портовая			6		2	5
Analog Devices	ADSP-21160N	32	ПТ	100	10	600[2]	4 Мбит 2-портовая			14		2	1,8
Analog Devices	TS001 (TigerSharc)	8/16/32	ПТ	150	6,7	300[3]	6 Мбит общей памяти			14		4	2,5
Analog Devices	AD14160[10]	32	ПТ	40	25		16 Мбит общей памяти						5
Analog Devices	ADMC401[11]	16	ФТ	13	38,5	26	1K	2K	2		1	2	5
LSI Logic	LSI401Z	16	ФТ	200[4]	5	800	48K 2-портовая		2	1	2	2	2,5
Lucent Technologies	DSP1620	16	ФТ	120[4]	8,3	120	32K 2-портовая		4	2	1	1	3
Lucent Technologies	DSP16210	16	ФТ	100[4]	10	200	62K 2-портовая		8	2	2	2	3
Lucent Technologies	DSP16410C[17]	16	ФТ	200[4]	5	800[16]	96K 3-портовая для ядра			3	4	3	1,8
Motorola	DSP56002	24	ФТ	80[4]	25	40	0,5K	0,5K	0,5		1	2	5
Motorola	DSP56307[5]	24	ФТ	100[4]	10	100	48K	16K		6	1	3	2,5
Motorola	DSP56311[5]	24	ФТ	150[4]	6,7	150	96K	32K		6	1	2	1,8

Таблица П1.1 *(окончание)*

Фирма	Процессор	Разрядность данных	Тип арифметики	Частота, МГц	Время цикла, нс	MIPS[1]	Внутренняя память RAM, слов[1]		ROM, слов[1]	DMA	Таймеры	Последовательные порты	Напряжение ядра, В
							Данные	Программа					
Motorola	DSP56652[6]	16	ФТ	70[4]	14,3	70	20K	0,5K	48		2	2	1,8
Motorola	DSP56F826	16	ФТ	80[4]	25	40	4K	0,5K	33,5[7]		1	3	2,5
Motorola	MSC8101[8]	16	ФТ	300[4]	3,3	1200	256K общая			16	1	4[9]	1,5
TI	320LC206	16	ФТ	80	25	40	4,5K общая		32		1	2	3,3
TI	320LF241[19]	16	ФТ	20	50	20	544K общая		8[18]		2	2	3,3
TI	320LF2401A[20]	16	ФТ	40	25	40	1K общая		8[18]		2	1	3,3
TI	320LC2404A[21]	16	ФТ	40	25	40	1,5K общая		16		4	2	3,3
TI	320C27x[22]	16	ФТ	150	7	150							1,8
TI	320VC33	32	ПТ	150	13	150[23]	34K общая			1	2	1	1,8
TI	320C541	16	ФТ	40[4]	25	40	5K		28		1	2	5
TI	320LC549	16	ФТ	80[4]	12,5	80	32K		16		1	3	3,3
TI	320VC5416	16	ФТ	160[4]	6,25	160	128K		16	6	1	3	1,6

TI	320VC5420[15]	16	ФТ	100[4]	10	100	64K	32K		6	1	3	1,8
TI	320VC5509	16	ФТ	144[4]	7	288	128K общая		32	6	2	3	1,5
TI	320VC5510	16	ФТ	200[4]	5	400	160K общая		16	6	2	3	1,6
TI	320C6202	8/16/32	ФТ	250[4]	4	2000	128 Кбайт	256 Кбайт		4	2	3	1,8
TI	320C6416[14]	8/16/32	ФТ	600[4]	1,67	4800	128 Кбайт кэш данных L1D; 128 Кбайт кэш программ L1P; 8 Мбит кэш L2			64	3	3	1,2
TI	320C6701	32	ПТ	167[4]	6	1000[23]	512 Кбайт	512 Кбайт		4	2	2	1,9
TI	320C6712	32	ПТ	100[4]	10	600[23]	4 Кбайт кэш данных L1; 4 Кбайт кэш программ; 64 Кбайт кэш L2			16	2	2	1,8
ZiLOG	Z89321	16	ФТ	20	50	20	0,5K	0	4		1	2^{12}	5
ZiLOG	Z89223[13]	16	ФТ	20	50	20	0,5K	0	8		3	1	5

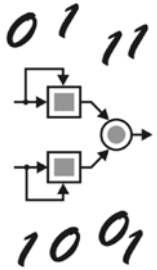
Приложение 2



Описания процессоров на русском языке

- TMS32010
  - Цифровой процессор обработки сигналов TMS32010 и его применение/Под ред. А. А. Ланнэ — Л.: ВАС, 1990.
  - Остапенко А. Г. и др. Цифровые процессоры обработки сигналов Справочник. — М.: Радио и связь, 1994.
- Процессоры фирмы Motorola
  - Куприянов М. С., Матюшкин Б. Д. Цифровая обработка сигналов. — СПб.: Политехника, 1998.
  - Куприянов М. С. и др. Техническое обеспечение цифровой обработки информации. Справочник. — СПб.: Форт, 2000.
  - Солонина А. И., Улахович Д. А., Яковлев Л. А. Цифровые процессоры обработки сигналов фирмы Motorola. — СПб.: БХВ-Петербург, 2000.
- ADSP-21xx
  - Руководство пользователя по сигнальным микропроцессорам семейства ADSP-2100: Пер. с англ. — СПб.: Санкт-Петербургский государственный электротехнический университет, 1997.
  - Цифровые сигнальные процессоры. Книга 1. — М.: Микроарт, 1996.
- Процессоры TI, Motorola, ADI и некоторые другие
  - Корнеев В. В., Кисилев А. В. Современные микропроцессоры. — М.: НОЛИДЖ, 1998.

Приложение 3



Источники информации

Приводимые данные не претендуют на исчерпывающую полноту.

Периодические издания

Журналы на русском языке

- Chip News: новости о микросхемах
  http://chipnews.gaw.ru
- Инженерная микроэлектроника: журнал для инженеров разработчиков
  http://chipnews.gaw.ru
- Цифровая обработка сигналов
- Компоненты и технологии
  http://www.compitech.ru

Журналы на английском языке

- IEEE Signal Processing magazine, Published bimonthly by the IEEE Signal Processing Society
  http://www.ieee.org/organizations/pubs/magazines/sp.htm
- Signal Processing, An International Journal: A publication of the European Association for Signal Processing (EURASIP)
  http://www.elsevier.com/inca/publications/store/5/0/5/6/6/2/
- EDN Access, The Design Source for Engineers and Managers Worldwide
  http://www.ednmag.com/ednmag
- DSP Engineering Magazine
  http://www.dspengineering.com/

Фирмы — производители ЦПОС

- Государственное предприятие "Научно-исследовательский институт электронной техники (ГП "НИИЭТ")

  http://www.electronicengineering.vrn.ru/

- Analog Devices Inc. (ADI)

  http://www.analog.com/

- DSP Group, Inc.

  http://www.dspg.com/

- LSI Logic Corporation

  http://www.lsilogic.com

- Lucent Technologies

  http://www.lucent.com/

- Motorola, Inc.

  http://www.motorola.com/

- Texas Instruments Incorporated

  http://www.ti.com

- ZiLOG, Inc.

  http://www.zilog.com

Поставщики продукции и консультанты

- **SCAN Ltd**. Продукция Texas Instruments: поставки, средства разработки, техническая поддержка, свободное ПО, форум

  http://www.texas.ru

- **"Сканти-Рус"** — официальный представитель компании Texas Instruments в России

  http://www.scanti.ru

- **AUTEX Ltd**. Представитель фирмы Analog Devices Inc. Продукция, техническая поддержка

  http://www.autex.ru

- **АВТЭКС Санкт-Петербург** — дилер компании АВТЭКС, официального дистрибьютора Analog Devices. Техническая поддержка, указатель характеристик компонент, литература

  http://www.autex.spb.ru

- **Представительство фирмы Motorola**. Продукты, литература, обучение
  http://www.mot.ru
- **Фирма ZiLOG**
  http://www.zilog.com/sales/europe/russia.html
  Представительства в России:
  - Eurodis Microdis Electronics: **http://www.eurodis.com/**
    Москва +7 095 535-63-98
    Санкт-Петербург +7 812 232-97-12
  - Gamma Ltd.
    Москва +7 095 965-36-83
    Санкт-Петербург +7 812 321-61-60

Разработчики систем

Согласно статье А. Соснина. "DSPA 2000" (Компоненты и технологии, № 9, 2000).

- "СКАН Инжиниринг-Телеком", г. Воронеж. Фирма является инженерным центром московской фирмы "СКАН" — официального дистрибьютора фирм Xilinx (весь спектр кристаллов и средств разработки ПЛИС) и Texas Instruments (TI) в России, а также поставляет программное обеспечение системного уровня фирм System View, Mentor Graphics. Производит разработку устройств ЦОС на заказ, начиная от проработки схемы и заканчивая аппаратной реализацией.
- АО "Инструментальные системы", г. Москва. Фирма занимается разработкой и производством программно-аппаратных средств ЦОС для применения в таких областях, как телекоммуникации, радиосвязь, гидроакустика, мультимедиа, промышленный контроль и управление и др. Предлагаемые компанией устройства построены на базе DSP Texas Instruments (TI) TMS320C6000 и Analog Devices (AD) ADSP-21060/21160, на ПЛИС высокой емкости Altera FLEX 10 Кбайт, Xilinx Virtex, поддерживают мультишинную архитектуру. Аппаратные средства предназначены как для работы в автономном режиме, так и для работы в составе вычислительных комплексов.

Web-сайты с полезной информацией

- DSP on EDN
  http://www.ednmag.com/ednmag/verticalmarkets/dsp.asp

- Berkeley Design Technology, Inc. (BDTI)

  http://www.bdti.com

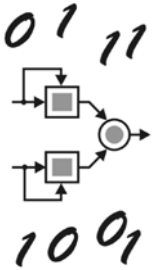
- Конференция "Цифровые сигнальные процессоры (DSP) и их применение"

  http://www.telesys.ru/wwwboards/dsp/1/wwwboard.html

- Конференция "DSP WORLD Conferreces on Digital Processing Solutions and Technologies"

  http://www.dspworld.com

Приложение 4



Маркировка процессоров TMS320 фирмы TI

Обозначение процессора TI можно разбить на несколько полей.

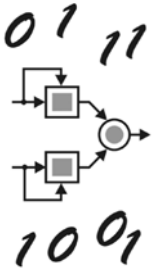
Пример

Обозначение	TMS	320	C	25	FN	L	40
N поля	1	2	3	4	5	6	7

Используемые в полях символы:

- **поле 3 — тип технологии**:
  - BC — КМОП с начальным загрузчиком;
  - C — КМОП;
  - E — КМОП с памятью типа EPROM;
  - F — КМОП с памятью типа Flash;
  - LBC — низковольтовая КМОП с начальным загрузчиком;
  - LC — низковольтовая КМОП;
  - LF — низковольтовая КМОП с памятью типа Flash;
  - P — КМОП с памятью типа OTPROM;
  - VC — очень низковольтовое устройство;
  - UC — питание: ядро 1,8 В периферия 1,8—3,6 В;
  - UVC — питание: ядро 1,2 В периферия 1,2—2,75 В;
- **поле 4 — номер устройства**;
- **поле 5 — тип корпуса**;
- **поле 6 — допустимый температурный диапазон**:
  - A: −40 — +85 °C;
  - H: 0 — +85 °C;
  - L: 0 — +70 °C;
  - M: −55 — +125 °C;
  - S: −55 — +100 °C;
- **поле 7 — тактовая частота процессора**.

Приложение 5



Принятые сокращения

ADI (Analog Devices Inc.)	фирма — производитель ЦПОС
AGU (address generation unit)	устройство генерации адреса
ASIC (Application-Specific Integrated Circuit)	проблемно-ориентированная интегральная схема
ASICs (Application Specific Integrated Circuits)	специализированные или проблемно-ориентированные интегральные схемы
BFU (Bit Functional Unit)	устройства обработки отдельных разрядов
cDSP (customizable Digital Signal Processors)	цифровые сигнальные процессоры с перестраиваемой конфигурацией
COFF (Common Object File Format)	формат записи объектного файла
DARAM (Dual Access RAM)	ОЗУ с двойным доступом
dma (data memory address)	адрес памяти данных
DMA (direct memory access)	прямой доступ к памяти
DSP (Digital Signal Processor)	цифровой сигнальный процессор
EPROM	тип памяти РПЗУ
EVM (Evaluation Module)	отладочный модуль
FFT (Fast Fourier Transform)	быстрое преобразование Фурье
FIFO (First-in-First-out)	первым вошел — первым вышел
Flash	тип памяти РПЗУ
FPGA (Field Programmable Gate Arrays)	один из методов реализации PLD
HIP (Host Interface Processor)	процессор с хост-интерфейсом
JPEG (Joint Photographic Experts Group)	объединенная фотографическая экспертная группа
JTAG (Joint Test Action Group)	объединенная рабочая группа по автоматизации тестирования

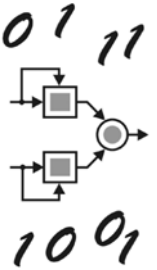
LPC (Linear Prediction Coder)	вокодер с линейным предсказанием
LSI	фирма-производитель
LUCENT	фирма — производитель ЦПОС
MAC	базовая операция ЦОС: операция умножения и добавление (накопление) результатов умножения (а также устройство, осуществляющее эту операцию)
MFLOPS (Millions of Floating-point Operations Per Second)	количество миллионов операций с плавающей точкой за секунду
MIPS (Million Instructions Per Second)	количество миллионов команд, выполняемых за секунду
MOPS (Millions Operations Per Second	количество миллионов операций за секунду
Motorola	фирма — производитель ЦПОС
MPEG (Moving Pictures Experts Group)	экспертная группа по движущимся изображениям
MVIP (Multi-Vendor Integration Protocol)	объединенный многоплатформмный протокол
OnCE (On-Chip Emulator)	внутрикристальный эмулятор
OTP ROM (One Time Programmable ROM)	однократно программируемое ПЗУ
PLDs (Programmable Logic Devices)	программируемые логические устройства
PLL	параллельное логическое устройство
pma (program memory address)	адрес памяти программ
RAM (random access memory)	оперативное запоминающее устройство
RISC	архитектура процессора с набором упрощенных команд типа "регистр, регистр → → регистр"
ROM (Read Only Memory)	постоянное запоминающее устройство
SARAM (Single Access RAM)	ОЗУ с одиночным доступом
SCSA (Signal Computing System Architecture)	архитектура вычислительных систем по обработке сигналов
SIMD (Single-Instruction Myltiplay-Data)	"одна команда — много данных"
SISD (Single-Instruction Single-Data)	"одна команда — одно данное"

TDM (Time-Division Multiplexing)	мультиплексирование с разделением по времени
TI (Texas Instruments Incorporated)	фирма — производитель ЦПОС
VAD (Voice Activity Detector)	определитель активности голоса
VLES (Variable Length Execution Set)	команды переменной длины
VLIW (Very Long Instructions Word)	очень длинное слово команды
ZiLOG	фирма-производитель

АБ	автобуферизация
АЛУ	арифметико-логическое устройство
АНГ	адрес нижней границы
АУВР	арифметическое устройство вспомогательных регистров
АФ	адаптивный фильтр
АФНЧ	антиэлайсинговый фильтр нижних частот
АЦП	аналого-цифровой преобразователь
АЧХ	амплитудно-частотная характеристика
БА	базовый адрес
БАБ	блок автобуферизации
БИС ЗУ	большая интегральная схема запоминающего устройства
БИХ	бесконечная импульсная характеристика (тип фильтра)
БПФ	быстрое преобразование Фурье
ВУ	внешнее устройство
ВША	внешняя шина адреса
ВШД	внешняя шина данных
ГТИ	генератор тактовых импульсов
ДКП	дискретное косинусное преобразование
ДПФ	дискретное преобразование Фурье
ДПХ	дискретное преобразование Хартли
КИХ	(конечная импульсная характеристика) тип фильтра
ЛП	линейное предсказание

МБП	модуль буферизированного порта
МкБПП	многоканальный последовательный буферизированный порт
МНК	метод наименьших квадратов
ОАВГ	относительный адрес верхней границы
ОДПФ	обратное дискретное преобразование Фурье
ПД	память данных
ПДП	прямой доступ к памяти
ПЗУ	постоянное запоминающее устройство
ПП	память программ
ПТ	плавающая точка, способ представления данных
Р	шина результата
РПУ	речепреобразующее устройство
СКО	среднеквадратическая ошибка
СКС	сигнал кадровой синхронизации
СЛЗ	согласующая линия задержки
СФНЧ	сглаживающий фильтр нижних частот
ТИ	тактовый импульс
УГА	устройство генерации адреса
ФТ	фиксированная точка, способ представления данных
ФЧХ	фазочастотная характеристика
ЦАП	цифро-аналоговый преобразователь
ЦОС	цифровая обработка сигналов
ЦПГ	цифровой преобразователь Гильберта
ЦПОС	цифровой процессор обработки сигналов
ЦПУ	центральное процессорное устройство
ЦСП	цифровой сигнальный процессор
ЦФ	цифровой фильтр
ШАПД	шина адреса памяти данных
ШАПП	шина адреса памяти программ
ШДПД	шина данных памяти данных
ШДПП	шина данных памяти программ

Приложение 6



Список литературы

1. Cravotta R. DSP Directory 2001/EDN Magazine. Issue of EDN, 2001, March 29.
2. Don Morgan A DSP for Every Application/Embedded Systems Programming (magazine). Vol. 12, 1999, № 4, April.
3. Eyre J., Bier J. The Evolution of DSP Processor/IEEE Signal Processing magazine, 2000, March.
4. Philip Lapsley, Jeff Bier, Amit Shoham, Edward A. Lee. DSP Processor Fundamentals, Architecture and Features. — New York: IEEE Press, 1997.
5. Real time digital signal processing applications with Motorola's DSP56000 family. Prentice Hall, Englwood Cliffs, NI, 1990.
6. Robert I. Simpson. Digital Signal Processing Using The Motorola DSP Family. Prentice Hall, Englwood Cliffs, NI07632, 1994.
7. Steven W. Smith The Scientist and Engineer's Guide to Digital Signal Processing. Second Edition. California Technical Publishing San Diego, California, 1999.
8. Адаптивные фильтры: Пер. с англ./Под ред. К. Ф. Н. Коуэна и П. М. Гранта. — М: Мир, 1988.
9. Анохин В. В., Ланнэ А. А. и др. MATLAB для DSP. Цикл статей/ChipNews, 2000, № 2—4, 7, 9; 2001 № 2; Цифровая обработка сигналов, № 2, 2000.
10. Аппаратные средства проектирования комплексов ЦОС на базе процессоров TMS320/ChipNews, 1999, № 3.
11. Бетелин В. Б., Грузинова Е. В., Кольцова А. А. и др. Архитектура цифровых процессов обработки сигналов. — М.: РАН, Научный совет по комплексной проблеме "Кибернетика", 1993.
12. Ван Тасселл Д. Стиль, разработка, эффективность, отладка и испытание программ: Пер. с англ. — М.: Мир, 1985.
13. Гольденберг Л. М., Матюшкин Б. Д., Поляк М. Н. Цифровая обработка сигналов. Справочник. — М.: Радио и связь, 1985.
14. Гольденберг Л. М., Матюшкин Б. Д., Поляк М. Н. Цифровая обработка сигналов. Учебное пособие для вузов. — М.: Радио и связь, 1990.

15. Гончаров Ю. Новое поколение ЦСП Texas Instruments/Компоненты и технологии, 2001, № 1.
16. Карниган Б., Пайк Р. Практика программирования: Пер. с англ. — СПб.: Невский Диалект, 2001.
17. Карниган Б., Ритчи Д. Язык программирования Си: Пер. с англ. — 3 изд-е испр. — СПб.: Невский Диалект, 2000.
18. Кипер Сэм и др. Тестирование программного обеспечения: Пер. с англ. — К.: Duesoft, 2000.
19. Козаченко В., Обухов Н., Горбунов В. и др. Высокопроизводительные встраиваемые системы управления двигателями на базе сигнального микроконтроллера TMS320F241/Chip News, 2000, № 5.
20. Козаченко В., Грибачев С. Перспективная серия микроконтроллеров фирмы Texas Instruments 240x для систем цифрового управления двигателями/Chip News, 1999, № 9.
21. Корнеев В. В., Кисилев А. В. Современные микропроцессоры. — М.: НОЛИДЖ, 1998.
22. Куприянов М. С. и др., Техническое обеспечение цифровой обработки сигналов. — СПб.: Наука и техника, 2000.
23. Куприянов М. С., Матюшкин Б. Д. Цифровая обработка сигналов. — СПб.: Политехника, 1998.
24. Ланнэ А. А., Семенов О. Б., Фукс В. А.. Цифровые процессоры обработки сигналов фирмы Motorola. — М.: Motorola Inc., 1997.
25. Ланнэ А. А., Улахович Д. А. Рациональный способ вычисления $\sqrt{x_1^2 + x_2^2}$ в задачах цифровой фильтрации//В сб. "Вопросы аналого-цифровой обработки и формирования сигналов". — Ярославль: ЯрГУ, 1992.
26. Марпл-мл. С. Л. Цифровой спектральный анализ и его приложения: Пер. с англ. — М: Мир, 1990.
27. Новые DSP — новый рывок в производительности//Пер. М. Ахметова/ChipNews, 2000, № 10.
28. Оппенгейм А., Шафер Р. Цифровая обработка сигналов. — М.: Связь, 1979.
29. Потемкин В. Г. Система инженерных и научных расчетов Matlab 5.x. В 2 т. М.: Диалог-МИФИ, 1999.
30. Рабинер Л., Голд Б. Теория и применение цифровой обработки сигналов. — М.: Мир, 1978.
31. Рудаков П. И., Сафонов И. В. Обработка сигналов и изображений. Matlab 5x/Под общ. ред. к. т. н. В. Г. Потемкина. — М.: Диалог-МИФИ, 2000.

32. Руководство пользователя по сигнальным микропроцессорам семейства ADSP-2100: Пер. с англ. — СПб.: СПГУ, 1997.

33. Соловьев А., Веселов М. Семейство DSP-микроконтроллеров фирмы Analog Devices для встроенных систем управления двигателями/Chip News, 1999, № 1.

34. Солонина А. И., Улахович Д. А., Яковлев Л. А. Цифровые процессоры обработки сигналов фирмы Motorola. — СПб.: БХВ-Петербург, 2000.

35. Солонина А. И., Яковлев Л. А. Основы построения микропроцессорных систем. — Учебн. пособие. — Л.: ЛЭИС, 1991.

36. Соснина А. DSPA 2000. Компоненты и технологии, № 1, 2001.

37. Справочная книга по математической логике: В 4 ч./Под ред. Дж. Барвайса//Ч. III. Теория рекурсии: Пер. с англ. — М.: Наука, 1992.

38. Страуструп Б. Язык программирования C++. 3-е изд./Пер. с англ. — СПб.; М.: Невский Диалект — "Изд-во БИНОМ", 1999.

39. Компоненты и технологии. Рынок. Компоненты. Технологии. 2000, № 8.

40. Толковый словарь по вычислительным системам/Под ред. В. Иллингуорта и др.: Пер. с англ. — М.: Машиностроение, 1989.

41. Угрюмов Е. П. Цифровая схемотехника. — СПб.: БХВ-Петербург, 2000.

42. Шилд Г. Справочник программиста на C/C++: Пер. с англ./Учебн. пособие. — М.: Издательский дом "Вильямс", 2000.

43. Юров В. Assembler. — СПб.: Питер, 2000.

Предметный указатель

А

Б

В

Г

Д

З

И

К

Л

М

Н

О

П

Р

С

Т

У

Ф

Х

Ц

Ч

Ш

Э

Я