

Х. Крейгон

АРХИТЕКТУРА КОМПЬЮТЕРОВ И ЕЕ РЕАЛИЗАЦИЯ

Книга рекомендована Учебно-методическим
советом по прикладной математике и информатике
УМО университетов РФ в качестве учебного пособия
по архитектуре компьютеров.

Перевод с английского
К. Г. Финогенова

под редакцией
члена-корр. РАН, профессора
Л. Н. Королева



Москва «Мир» 2004

COMPUTER ARCHITECTURE AND IMPLEMENTATION

HARVEY G. CRAGON

University of Texas at Austin



CAMBRIDGE
UNIVERSITY PRESS

УДК 681.3
ББК 32.97
К79

Крейгон Х.

К79 **Архитектура компьютеров и ее реализация: Пер. с англ.— М.: Мир, 2004. — 416 с.**
ISBN 5-03-003379-3

Книга известного американского специалиста и педагога является ясным и точным введением в архитектурные принципы и технику реализации современных компьютеров. Рассматриваются системы команд, методы оценок производительности различных технических решений, назначение и функционирование основной, дисковой и кэш-памяти, узлов процессора, систем ввода-вывода, конвейеров и других элементов современного компьютера.

Для преподавателей и студентов университетов технических вузов, в учебных планах и программах которых значатся курсы по архитектуре компьютеров, а также для всех интересующихся современными методами проектирования компьютеров.

УДК 681.3
ББК 32.97



Издание осуществлено при поддержке Российского фонда
фундаментальных исследований по проекту 02-01-14107

Редакция литературы по информатике и новой технике

ISBN 5-03-003379-3 (русск.)
ISBN 0-521-65168-9 (англ.)

© Cambridge University Press 2000
© перевод на русский язык, оформление,
«Мир», 2004

Оглавление

От редактора	8
Предисловие	9
Об этой книге и ее авторе	12
ГЛАВА 1. ОСНОВЫ КОМПЬЮТЕРОВ	13
1.0. Введение	13
1.1. Фон-неймановская модель	14
1.2. Фон-неймановская архитектура	19
1.2.1. Фон-неймановская система команд	19
1.2.1. Интерпретация команд	25
1.2.3. Ограничения фон-неймановской системы команд	29
1.3. Немного истории	30
1.3.1. Предшественники фон-неймановской системы команд	30
Литература	42
Упражнения	43
ГЛАВА 2. МОДЕЛИ И ОЦЕНКИ ПРОИЗВОДИТЕЛЬНОСТИ	45
2.0. Введение	45
2.1. Модели производительности	45
2.2. Закон Амдала	60
2.3. Закон Мура	63
2.4. Кривая обучаемости	66
2.5. Закон Гроша	69
2.6. Стационарная производительность	70
2.7. Переходная характеристика	72
Литература	73
Упражнения	74
ГЛАВА 3. ПРОЕКТИРОВАНИЕ НАБОРА ПРИКЛАДНЫХ КОМАНД	76
3.0. Введение	76
3.1. Лексический уровень	77
3.2. Архитектура набора команд (система команд)	78
3.2.1. Адреса	81
3.2.2. Операции	91
3.2.3. Типы данных	107
3.2.3.1. Пользовательские типы данных	108
3.2.3.2. Типы данных для управления программой	125
3.2.4. Состав команд	131

3.3. Архитектурные стили CISC и RISC	138
3.4. Статическая и динамическая статистики команд	145
3.5. Арифметика	153
3.5.1. Сложение и вычитание	154
3.5.2. Умножение и деление	158
3.5.3. Арифметика плавающей запятой	163
3.5.3.1. Точность	166
Литература	175
Упражнения	176
 ГЛАВА 4. СИСТЕМЫ ПАМЯТИ	 179
4.0. Введение	179
4.1. Иерархическая память	181
4.2. Страничная виртуальная память	181
4.3. Кэш-память	197
4.4. Реальная память с перекрытием	222
4.5. Кэш-память виртуальных и реальных адресов	232
4.6. Сегментированная виртуальная память	234
4.7. Дисковая память	238
4.8. Действие системы	243
Литература	244
Упражнения	245
 ГЛАВА 5. ПРОЕКТИРОВАНИЕ УПРАВЛЯЮЩИХ ЦЕПЕЙ ПРОЦЕССОРА	 247
5.0. Введение	247
5.1. Аппаратное управление	250
5.2. Микропрограммное управление	260
Литература	269
Упражнения	270
 ГЛАВА 6. КОНВЕЙЕРНЫЕ ПРОЦЕССОРЫ	 272
6.0. Введение	272
6.1. Модели производительности	273
6.2. Разбиение конвейера	276
6.3. Задержки в конвейере	281
6.3.1. Задержки переходов	282
6.3.2. Структурные конфликты	299
6.3.3. Взаимозависимость данных	302
6.4. Прерывания	308
6.5. Суперскалярные конвейеры	311

6.6. Конвейерные процессоры и требования, предъявляемые к памяти	316
Литература	317
Упражнения	318
 ГЛАВА 7. ВВОД-ВЫВОД	320
7.0. Введение	320
7.1. Архитектура системы ввода-вывода	321
7.2. Требования к устройствам ввода-вывода	332
7.3. Шины и контроллеры	336
7.3.1. Примеры шин	344
7.4. Последовательная передача данных	344
7.5. Тактовая синхронизация	350
7.6. Теория очередей	353
7.6.1. Модель открытой системы с очередью	354
7.6.2. Модель закрытой системы с очередью	361
Литература	364
Упражнения	365
 ГЛАВА 8. РАСШИРЕНИЯ НАБОРА КОМАНД	366
8.0. Введение	366
8.1. Поддержка ядра операционной системы	367
8.2. Поддержка виртуальной памяти	376
8.3. Поддержка прерываний	385
8.4. Поддержка ввода-вывода	387
8.5. Поддержка кэш-памяти	391
8.6. Мультипроцессорная поддержка	396
8.7. MMX — другие усовершенствования прикладной ISA	401
8.7.1. Другие виды специальной поддержки ISA Intel	407
Литература	408
Упражнения	409
 Предметный указатель	410

От редактора

Читателю предлагается перевод книги Н. G. Cragon «Computer Architecture and Implementation». Она написана как учебник по курсу «Архитектура компьютеров», который читается в большинстве зарубежных и отечественных высших учебных заведений, специализирующихся на подготовке специалистов в области компьютерных наук и компьютерной инженерии (Computer Science и Computer Engineering).

За последние несколько лет резко усложнилась архитектура вычислительных машин массового производства, и прежде всего микропроцессоров, на основе которых создаются персональные компьютеры, серверы, кластеры и супермощные мультимикропроцессорные системы с массовым параллелизмом.

Для специалистов важно знать, как устроены современные процессоры, как работает кеш-память, как преобразуется виртуальный адрес в физический, как обрабатываются потоки команд и данных, что такое суперскалярность, как происходит взаимодействие процессора с устройствами ввода-вывода, как работает система прерываний. На все эти и на многие другие вопросы можно найти ответ, прочитав или изучив эту книгу.

Важным достоинством книги следует считать желание автора объяснить, для чего разработчики компьютеров включают в архитектуру процессора те или иные средства, что это дает в части повышения производительности, удобства при работе трансляторов, возможностей операционных систем, улучшения интерфейса с пользователем.

Рассказывая об архитектурных особенностях вообще, в качестве конкретных примеров автор в основном использует сведения о структуре Pentium Pro и IBM 360/370/390, не оставляя без внимания интересные особенности процессоров других фирм. Книгу, следовательно, нельзя рассматривать как еще одно описание какой-то конкретной архитектуры. Таких описаний конкретных архитектур выпущено достаточно количество, автор их не повторяет и в этом также состоит достоинство предлагаемой читателю книги.

Книга снабжена примерами, упражнениями и обширной библиографией, достаточно хорошо иллюстрирована.

Несомненно, эта книга может быть использована как полезное учебное пособие для преподавателей и студентов вузов и университетов, в учебных планах и программах которых значатся курсы по архитектуре компьютеров.

Л.Н. Королев

Предисловие

Эта книга предназначена для использования в качестве учебника для студентов старших курсов и рассчитана на один семестр. Предполагается, что студенты уже знакомы с логическим проектированием с помощью конечных автоматов и программированием на языке ассемблера. Для изучения книги полезно также предварительно пройти курс структур данных и основ операционных систем.

Две основные темы пронизывают эту книгу. Первая представляет собой обзор основных идей и принципов проектирования архитектуры и реализации компьютеров с некоторым уклоном в сторону истории предмета. Мы склонны забывать роль наших предшественников в области компьютерного проектирования; несколько страниц, посвященных этому вопросу, помогут восстановить справедливость.

Вторая тема — принципы и использование аналитического моделирования производительности компьютера. Я убежден, что выпускники факультетов инженерных специальностей должны взять на вооружение инженерный подход к изучению архитектуры и реализации компьютера. Различные проекты выпускаемых изделий должны проверяться с помощью аналитических моделей. Так что собой представляет инженерный подход? Приведем две цитаты.

«Инженер отличается от техника прежде всего тем, что он может формулировать и вычислять силы и отклонения, концентрации и потоки, напряжения и мощности, которые нужны для анализа предложенного проекта в аналитической форме по различным критериям отказов. Способность вычислять определяет способность предсказывать производительность проекта еще до того, как он будет воплощен и испытан» [1].

«...Гораздо проще иметь смутное представление обо всем без разбора, чем дойти до истинной сущности единственного вопроса, какой бы простой она не оказалась» [2].

Помимо этих двух основных тем книга познакомит студентов с новой терминологией. Как только вы начинаете изучать новый предмет, вы неминуемо сталкиваетесь с новыми терминами. В этой книге каждый раз, когда вводится новое слово или термин, делается попытка дать этому термину определение и привести возможные синонимы. Мы рекомендуем студентам иметь экземпляр толкового словаря по компьютерной тематике. Можно также обратиться к интерактивному словарю компьютерных терминов по адресу в Интернете <http://wombat.doc.ic.ac.uk/foldoc/index.html>.

К сожалению, в области компьютеров нет единого общепринятого набора терминов. Например, архитектура Intel x86 использует тер-

мин «линейный адрес», но я предпочитаю, вслед за IBM S370 и S390, говорить «виртуальный адрес». Причина этого предпочтения заключается в том, что реализация виртуальной памяти в IBM S370 была выполнена раньше, чем в Intel x86, и, соответственно, терминология IBM имеет право на приоритетное использование. В этой книге, где это возможно, используются такие приоритетные термины.

В главе 1 на примере фон-неймановской архитектуры вводятся основные принципы архитектуры компьютеров. Студентам следует знать, что именно предложил фон Нейман и каковы были ограничения его архитектуры. В последующих архитектурах эти ограничения были преодолены, однако базовые принципы остались неизменными.

В качестве иллюстраций на протяжении всей книги будут использоваться два архитектурных семейства: IBM S360/370/390 и Intel x86 вплоть до Pentium Pro с MMX-расширением. Эти семейства представляют два наиболее широко используемых в наши дни процессора. Семейство IBM заняло лидирующие позиции в области больших вычислительных машин. Семейство Intel используется приблизительно в 90% всех персональных компьютеров. Именно с этими семействами, скорее всего, столкнутся студенты, закончив обучение.

Последовательность изложения в этой книге несколько отличается от многих других учебников. Книга начинается с обзора компьютеров в главе 1, посвященного архитектуре фон Неймана и особенностям некоторых проектов его предшественников.

В главе 2 рассматриваются аналитические модели производительности, кривая обучения и другие способы оценки результатов. Аналитическое моделирование проходит красной нитью сквозь всю книгу^{*)}.

В главе 3 рассматриваются базовые вопросы проектирования наборов прикладных (пользовательских) команд. Здесь вводится систематика архитектуры, использующая понятия адресов, типов данных и операций. Даются основы компьютерной арифметики вместе с IEEE-стандартом плавающей запятой.

Поскольку во главу угла при изучении компьютера мы поставили его память, в главе 4 излагаются вопросы реализации памяти. Я убежден, что студенты должны приступать к изучению процессора лишь после того, как они познакомятся с организацией памяти, так как состояние процесса, представляющее собой собственно снимок памяти, является основой решения целого ряда проблем. Я также начинаю обсуждение систем памяти с объяснения понятия виртуальной памяти, так как именно она предоставляет функциональные возможнос-

^{*)} Под аналитическим моделированием автор понимает получение формулы для вычисления той или иной характерной величины, даже если эта формула очень проста и представляет собой, например, сумму или произведение двух чисел. В русском языке термин «модель» имеет гораздо более возвышенное значение, что вызвало определенные трудности при переводе. Мы сочли возможным оставить в большинстве случаев терминологию автора, предпослав переводу это замечание. — *Прим. перев.*

ти, необходимые для выполнения больших программ с большими наборами данных.

С точки зрения проектировщика функциональные возможности тесно связаны с производительностью; поэтому после виртуальной памяти описывается кэш-память. Между двумя крайностями — кэш-памятью и дисками — рассмотрены память с чередованием и динамическая память DRAM с произвольным доступом. Повсюду в этой главе для демонстрации компромиссных решений в проектировании используется модель производительности системы памяти с иерархической организацией.

Вслед за главой, посвященной памяти, рассматриваются вопросы реализации процессоров. Прежде всего, в главе 5 дается сравнение аппаратной и микропрограммной реализаций. Далее в главе 6 излагаются идеи и методы построения конвейеров. Повсюду в изложение вплетены расчеты на основе модели производительности вместе с функциональными описаниями вариантов реализации процессора.

Глава 7 посвящена вводу-выводу. В этой главе мы оцениваем рабочую нагрузку вычислительной системы и показываем, как в современных системах удовлетворяются требования производительности. Здесь, в частности, рассматриваются вопросы проектирования шины, принципы последовательной передачи данных, а также способы установки тактовой синхронизации. Кроме того, в главу включено краткое введение в теорию очередей.

В заключительной главе 8 описываются расширения архитектур прикладных наборов команд, служащие для поддержки операционной системы, виртуальной памяти, кэш-буферов и мультипроцессорной организации. Глава завершается рассмотрением MMX-расширения архитектуры Pentium Pro.

Конечная цель этой книги — облегчить студентам задачу чтения современной компьютерной литературы, оказать им помощь в понимании и оценке предлагаемых идей. Я попытался достичь этой цели, раскрыв перед студентами базовые принципы архитектуры и реализации современных процессоров и продемонстрировав богатый, хотя и неустановившийся мир терминологии этой отрасли знаний.

Литература

1. Petroski, H. (1996). *Invention by Design*, Harvard U. Press, Cambridge, MA, pp. 89–90.
2. Descartes, R. (1962). *Rules for the Direction of Mind*.

Об этой книге и ее авторе

Этот учебник является ясным и точным введением в архитектурные принципы и технику реализации современных компьютеров. Две важные темы проходят красной нитью сквозь всю книгу. Первая — это обзор важнейших концепций и стратегий проектирования компьютерной архитектуры и компьютерной организации. Вторая — введение в использование аналитического моделирования для оценки производительности процессора и компьютера в целом.

Автор начинает с изложения классической фон-неймановской архитектуры, после чего предлагает детальное описание ряда моделей и методов оценки производительности компьютера. Далее рассматривается проектирование прикладных наборов команд, включая особенности RISC-архитектуры. Отличительной чертой книги является ее построение: во главу угла поставлена память компьютера, которая рассматривается еще до перехода к вопросам реализации процессора. Помимо базовых принципов, лежащих в основе современных процессоров, автор останавливается на таких вопросах, как конвейерные процессоры, реализация ввода-вывода, модели очередей, архитектуры с расширенным набором команд. Каждая тема иллюстрируется ссылками на особенности реальных архитектур процессоров IBM и Intel.

В книгу включено большое количество жизненных примеров и свыше 130 упражнений для домашнего освоения. В целом она представляет собой идеальное учебное пособие по односеместровому университетскому курсу, посвященному вопросам архитектуры и реализации компьютеров.

Харви Крэйгон является заслуженным профессором технических наук Техасского университета в г. Остин. Он — член Национальной технической академии США, пожизненный член IEEE, член Ассоциации вычислительной техники. Х.Крэйгон издал свыше 50 научных работ и является автором 9 патентов в разных областях вычислительной техники.

ГЛАВА 1

ОСНОВЫ КОМПЬЮТЕРОВ

1.0. Введение

Компьютер общего назначения приобрел решающую роль в мировом сообществе. Компьютеры управляют поистине всем, от автомобильных систем зажигания до регистрации олимпийских рекордов. Настоящая книга отражает односеместровый университетский курс, посвященный базовым концепциям компьютеров, безотносительно к различиям, вносимым их конкретными реализациями, будь то большие универсальные вычислительные машины, серверы, рабочие станции, персональные компьютеры или встроенные в аппаратуру контроллеры. Нас здесь будут интересовать лишь общие принципы, которые мы будем иллюстрировать примерами, взятыми из реализаций конкретных процессоров.

Роль компьютеров в современном мире многогранна. Они могут хранить информацию, например, в персональных базах данных; преобразовывать и запоминать информацию, как это делают текстовые процессоры; создавать новую информацию, например, при исчислении налогов. Компьютеры также обладают способностью отыскивать информацию во Всемирной интернетовской паутине.

Столь всепроникающее использование компьютеров обязано тому факту, что они являются универсальными приборами общего назначения. Другими словами, аппаратура компьютера с помощью записанных в нее программ может трансформироваться в широчайший набор устройств различного назначения. Пример такой трансформации можно найти в технике обработки текстов. В течение многих лет компания Wang Laboratories была основным поставщиком машин для обработки текстов, основанных на специализированных процессорах со встроенными командами. Не сумев осознать революцию, надвигающуюся вместе с появлением персональных компьютеров, компания Wang упорно придерживалась этой тупиковой производственной стратегии и в конце концов была вытеснена с рынка.

Другим примером использования универсальной природы компьютера является история развития автоматизации автомобильных систем зажигания. Когда требования охраны окружающей среды вынудили автомобильную индустрию заняться разработкой систем зажигания с автоматическим управлением, автомобильная компания Ford подошла к делу не так, как компании Chrysler и General Motors. Пос-

ледние две компании, обладая относительно стабильным финансовым положением, пошли по пути разработки для своих систем зажигания специализированных электронных устройств. Компания же Ford, испытывающая серьезные финансовые трудности, решила использовать для той же цели микропроцессор, который, хотя и обходился немного дороже в производстве, однако не требовал серьезных затрат на разработку специализированных схем. Применив микропроцессор, компания Ford в дальнейшем смогла с относительно небольшими затратами модифицировать свой контроллер для различных конфигураций двигателей, изменяя лишь постоянное запоминающее устройство (ПЗУ), содержащее программу. С другой стороны, компании Chrysler и General Motors быстро обнаружили, что для каждой конфигурации автомобиля им надо было разрабатывать новый уникальный контроллер, что существенно увеличивало издержки производства. Микропроцессорное управление, впервые примененное на практике компанией Ford, сейчас признано всеми и является промышленным стандартом.

До наступления эпохи компьютеров общего назначения разрабатывалось и строилось множество специализированных компьютеров. К ним можно отнести дифференцирующую машину Бэббиджа (приблизительно 1835 г.), компьютер ABC Атанасова и Берри в университете штата Айова (конец 1930-х годов), а также машину Z3 Конрада Зуса (приблизительно то же время). Известны также компьютер Colossus из Бенчли Парк (использовавшийся для взлома немецких шифров во Вторую мировую войну) и ENIAC (сокращение от electronic numerical integrator and computer — электронный цифровой интегратор и вычислитель), машина из Пенсильванского университета, программируемая с помощью штекерной панели. Эти компьютеры будут подробнее описаны в разделе 1.3.1.

Каждый из упомянутых выше компьютеров существовал лишь в одном экземпляре. В те времена отсутствовал стандарт, который позволил бы унифицировать базовую архитектуру компьютера и дал бы возможность разработчикам будущих машин сконцентрировать свои усилия на развитии этой общей базы. Такая возможность была найдена с появлением фон-неймановской модели.

1.1. Фон-неймановская модель

Фон-неймановская модель компьютерной архитектуры была впервые описана в 1946 г. в знаменитой работе Буркса, Голдштейна и фон Неймана [4]. К тому времени уже был построен целый ряд компьютеров и подобных им устройств, первым из которых можно считать разработку Ч. Бэббиджа, однако простая структура компьютера с хранимой

программой впервые была описана в этой основополагающей работе. Авторы подчеркивали, что и команды, и данные состоят из битов, не имеющих каких-либо отличительных признаков. Таким образом, для хранения и тех, и других может использоваться общая память. Разница между этими двумя объектами состоит лишь в механизмах доступа и в окружающем их контексте; программный счетчик указывает на команды, в то время как через регистр эффективного адреса происходит обращение к данным. Если по какой-либо причине, например в результате ошибки программиста, команды и данные оказались перепутаны в памяти, поведение программы будет непредсказуемо. До того как фон Нейман предложил свою архитектуру общего адресного пространства, компьютеры использовали отдельные устройства памяти для хранения команд и данных. Одна из таких машин была построена Говардом Айкеном в Гарвардском университете, отчего такой стиль разработки получил название Гарвардской архитектуры¹⁾.

Вариант фон-неймановской архитектуры, широко используемой сегодня в различных реализациях калькуляторов, получил название теговой архитектуры. В таких устройствах каждый тип данных в памяти имеет сопутствующий ему тег (признак), который описывает тип данного: команда, число с плавающей запятой (такие числа находят применение в инженерной практике), целочисленное значение и т. д. Если калькулятору дается команда прибавить число с плавающей запятой к целому, выполняется сравнение тегов; целое преобразуется в число с плавающей запятой, после чего выполняется сложение, и результат представляется также в виде числа с плавающей запятой. Можете попробовать выполнить такую операцию на вашем калькуляторе и убедиться, что так и будет.

Все варианты фон-неймановской архитектуры, разработанные после 1946 г., подтверждают, что эта архитектура выдержала проверку временем и является классической. Ее можно совершенствовать, но заложенный в нее основополагающий принцип остается. В настоящем разделе фон-неймановская архитектура описывается, как набор вложенных конечных автоматов. Деталям этой архитектуры посвящен раздел 1.2.1.

Следует еще раз подчеркнуть огромное значение фон-неймановской архитектуры, которая начиная с 1950 г. выступала и выступает в качестве унифицирующей концепции во всех компьютерных устройствах. Эта концепция предоставляет программисту упорядоченную методологию разработки и единый интерфейс. Взглянув на описание любого современного компьютера или микропроцессора, вы немедленно найдете в нем основные компоненты: память, центральный процессор (central processing unit, CPU), цепи управления, а также цепи ввода-вывода.

¹⁾ Фон-неймановская архитектура известна также под названием Принстонской (в отличие от Гарвардской).

Интерфейс программиста с фон-неймановской архитектурой единообразен. Программист знает, что команды выполняются друг за другом в заданном им порядке. В действительности при использовании параллельных процессоров, обсуждаемых в гл. 6, последовательность выполнения команд нарушается, однако с точки зрения программиста сохраняется порядок, указанный в программе.

В разные годы предлагались другие модели компьютерной архитектуры, отличные от фон-неймановской. Однако простота и порядок фон-неймановской архитектуры оказались непревзойденными; ни одна из предложенных альтернативных концепций не воплотилась в коммерческий компьютер.

Эквивалент конечного автомата

Компьютер определяется как комбинация памяти, процессора и системы ввода-вывода. Ввиду того что память занимает в компьютере центральное положение, в нашей книге сначала обсуждается память (гл. 4) и лишь после этого процессор (гл. 5 и 6).

Три указанных выше компонента компьютера могут быть представлены в виде набора вложенных конечных автоматов. Память хранит команды и данные. И те, и другие направляются в логический узел, затем данные (а в некоторых разработках и команды) модифицируются процессорной логикой и возвращаются в память. Этот поток информации можно представить в виде конечного автомата, показанного на рис. 1.1.

Информация, содержащаяся в памяти, называется состоянием процесса. Входы компьютера подключены к памяти и становятся таким образом элементами состояния процесса. Выходы компьютера являются выходами состояния процесса в памяти.

Следующий уровень абстракции показан на рис. 1.2. Логический блок, изображенный на рис. 1.1, заменен другим конечным автоматом. Этот второй конечный автомат использует в качестве памяти регистры процессора. В число регистров, подробнее обсуждаемых в гл. 3, входят программный счетчик, регистры общего назначения, а также различные специализированные регистры. Логический узел состоит

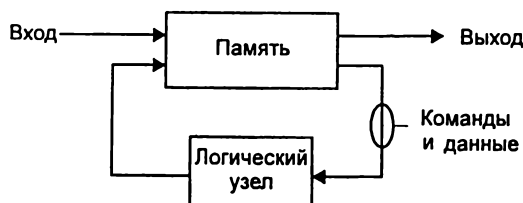


Рис. 1.1. Конечный автомат I.

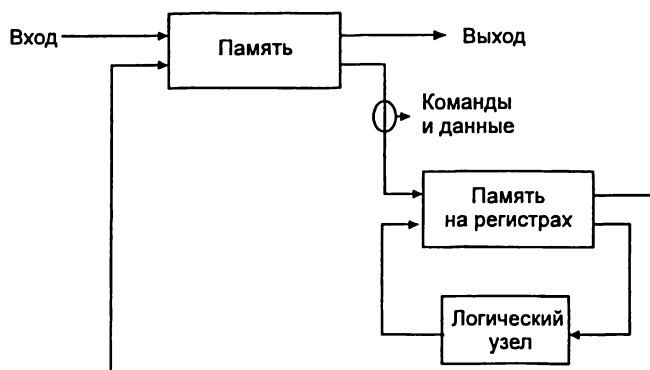


Рис. 1.2. Конечный автомат II.

из арифметико-логического устройства (ALU) и других логических схем, требуемых для расшифровки команд.

Информация, содержащаяся в регистрах, называется состоянием процессора. Состояние процессора — это 1) информация, требуемая для интерпретации команды, и 2) информация, передаваемая вперед от команды к команде, куда входят, например, значение программного счетчика, а также различные теги и флаги. Если выполняется переключение контекста процессора, сохраняется именно состояние процессора, чтобы при обратном переключении можно было восстановить прерванное состояние²⁾.

При обсуждении микропрограммирования в гл. 5 мы увидим, что логический блок с рис. 1.2 также может быть реализован в виде конечного автомата. Такого рода реализация приводит к трем уровням состояний: состояние процесса, состояние процессора и состояние процессора микромашины.

Рассмотрим теперь важнейшие компоненты компьютера, начав с памяти. Как уже отмечалось выше, память представляет собой пространство, содержащее состояние процесса, которое, в свою очередь, состоит из команд и данных. Пространство команд предназначено не только для выполняемой программы, но и для операционной системы, компиляторов, интерпретаторов и прочего программного обеспечения.

Процессор читает команды и данные, выполняет обработку данных и возвращает результаты в память, вследствие чего состояние процесса обновляется. Таким образом, первое требование к памяти состоит в ее быстродействии; другими словами, операции чтения и записи должны выполняться с минимальным запаздыванием.

²⁾ Переключение контекста сохраняет текущее состояние процессора и восстанавливает сохраненное ранее состояние.

Помимо этого к памяти предъявляются еще два противоречивых требования: память одновременно должна быть и очень большой, и очень быстрой. Стоимость памяти всегда является важным фактором; весьма желательно, чтобы память была недорогой. Эти требования приводят к идее иерархической памяти. Часть памяти, ближайшую к процессору, делают относительно небольшой по объему, но чрезвычайно быстрой и, соответственно, сравнительно дорогой. В качестве более далекой от процессора памяти используются магнитные диски, очень медленные, но зато дешевые. Производительность иерархических систем памяти приближается к производительности быстрой памяти, в то время как общая стоимость близка к стоимости дешевой дисковой памяти. Эти характеристики являются результатом концепции локализации, обсуждаемой в гл. 4. Локализация программ и данных проявляется в высокой вероятности того, что очередной запрос процессора на команду или данные будет обслужен с помощью части памяти, ближайшей к процессору.

Процессор, иногда называемый CPU, является устройством, реализующим логику и регистры, изображенные на рис. 1.2. Эта часть системы выбирает из памяти команды, декодирует их, находит операнды, выполняет затребованную операцию и возвращает результат в память. Сложность процессора определяется 1) сложностью набора команд и 2) объемом аппаратных средств для распараллеливания операций, позволяющих существенно повысить производительность системы.

Как показано на рис. 1.2, в компьютере должны быть предусмотрены какие-то средства помещения в память входных данных и команд и извлечения из памяти результатов. За эти перемещения отвечает система ввода-вывода. Входные и выходные устройства могут обладать различной пропускной способностью и быстродействием. Например, клавиатура относится к числу устройств с низкой пропускной способностью, в то время как цветной дисплей обычно характеризуется высокой скоростью работы. Между ними лежат характеристики таких устройств, как диски, модемы и сканеры.

Управление вводом-выводом может принимать различные формы. Так, в простых случаях каждая передача может выполняться процессором. Однако существуют полностью автономные системы ввода-вывода, например системы прямого доступа к памяти (direct memory access, DMA), которые сочетают высокую скорость передачи с минимальным вмешательством процессора. Системы ввода-вывода будут обсуждаться в гл. 7.

Формальное описание возможностей процессора, способов его взаимодействия с памятью и возможностей организации ввода-вывода заключено в его системе команд, называемой также архитектурой набора команд (instruction set architecture, ISA). Система команд характеризует вид компьютера со стороны программиста. Детали воплощения набора команд в аппаратуре процессора, влияющие на

производительность компьютера, называют реализацией системы команд.

1.2. Фон-неймановская архитектура

В этом разделе описывается фон-неймановская система команд. Эта архитектура, если не считать ввода-вывода, является завершенной и может служить отправной точкой для обсуждения в последующих главах. Черты фон-неймановской архитектуры можно найти в любой из используемых сегодня архитектур; поэтому отчетливое понимание фон-неймановской архитектуры будет весьма полезно при изучении современных компьютеров. В этой книге фон-неймановская архитектура, которая, кстати, была реализована в целом ряде компьютеров, используется скорее в качестве простого примера, чем как запутанный пример простой архитектуры. Когда в 1952 г. в Принстонском университете был наконец построен фон-неймановский компьютер, он получил название «компьютера Института углубленного изучения» (Institute for Advanced Studies, IAS).

Фон-неймановская архитектура состоит из трех основных подсистем: узла выполнения команд, арифметического узла и памяти (рис. 1.3). Ключевой чертой этой архитектуры является то, что команды и данные совместно используют общее адресное пространство. В результате источником адресов памяти является единственное устройство — узел выполнения команд. Выход памяти, в зависимости от того, какой объект выбирается — команда или данные — направляется либо в узел выполнения команд, либо в арифметический узел. Как будет показано в дальнейших главах, возможность обработки команд как данных можно рассматривать и как благословение, и как проклятие.

1.2.1. Фон-неймановская система команд

Фон-неймановская система команд (ISA) весьма проста и содержит всего 21 команду. Фактически эту архитектуру можно назвать первым

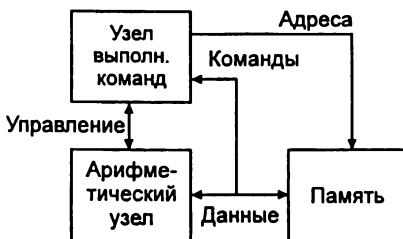


Рис. 1.3. Фон-неймановская архитектура.

процессором с сокращенным набором команд (reduced instruction set computer, RISC)³⁾. Как и в любой системе команд, в ней имеются три компонента: адреса, типы данных и операции. Систематика этих трех компонентов будет развита далее в гл. 3; ниже обсуждаются три компонента фон-неймановской системы команд.

Адреса

Адреса в системе команд определяют архитектурный стиль — организацию памяти, а также способы поиска операндов и сохранения результатов. В простой системе команд имеются лишь две адресуемые памяти: основная память и аккумулятор.

Основная память в фон-неймановской системе команд характеризуется линейным произвольным доступом и эквивалентна динамической памяти с произвольным доступом (DRAM), используемой в современных процессорах. Технология 1940-х годов ограничивала доступный объем памяти с произвольным доступом (RAM) очень небольшими величинами, поэтому память адресовалось прямым 12-битовым адресом, входящим в состав 20-битовых команд⁴⁾. Модификации адреса, например базовая относительная адресация или индексирование, не были предусмотрены. Форматы команд описаны ниже в разделе типов данных.

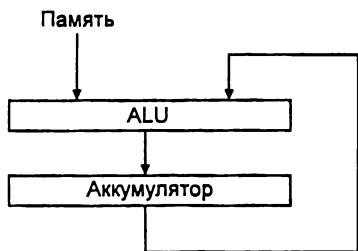


Рис. 1.4. Локальная память аккумулятора.

Локальная память в процессоре представляет собой единственный аккумулятор, как это показано на рис. 1.4. Регистр-аккумулятор получает результаты из ALU с двумя входами, предназначенными для получения одного данного из памяти, а второго из того же аккумулятора. Таким образом, в команде должен присутствовать лишь адрес памяти, поскольку в качестве второго адреса подразумевается аккумулятор.

³⁾ Набор команд, предложенный Ван-дер-Полем в 1956 г., содержал единственную команду.

⁴⁾ Разработчики Принстонского компьютера IAS встретились с такими трудностями в реализации памяти, что использовали лишь 1 К слов с 10-битовыми адресами.

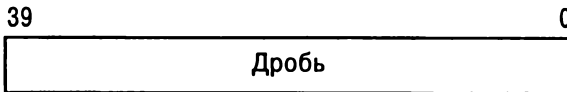
Типы данных

Фон-неймановская система команд содержит два типа данных: дроби и команды. Команды рассматриваются как особый тип данных, так как над ними можно выполнять операции как над данными. Такие программы называют самомодифицирующимся кодом. В наше время использование такого кода считается дурным тоном. Однако современные архитектуры, например семейства Intel x86, должны поддерживать это свойство ради совместимости с такими унаследованными программами, как MS-DOS.

Память состоит из 4096 слов по 40 бит каждое; в одном слове памяти помещается одна дробь или две команды.

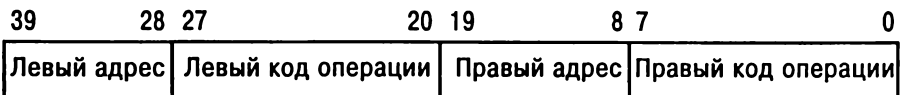
Дроби

40-битовое слово содержит дробь в виде дополнения до двух; диапазон значений составляет $-1 \leq f < +1$:



Команды

В 40-битовом слове размещаются две 20-битовые команды. В каждую команду входят 8-битовый код операции и 12-битовый адрес. Обратите внимание, что при наличии всего 21 команды можно было сократить поле команды и увеличить число битов в поле адреса. Прямой адрес памяти занимает 12 старших битов каждой команды. Пары адрес — код операции в каждом слове носят название левых и правых:



Регистры

На рисунке 1.5 приведена блок-схема фон-неймановского компьютера. Здесь не показаны цепи для ввода-вывода; хотя в исходной работе они были только упомянуты, во всех реализациях описываемой архитектуры цепи ввода-вывода фактически присутствовали.

Фон-неймановский процессор содержит 7 регистров, поддерживающих интерпретацию команд, выбираемых из памяти. Все эти ре-

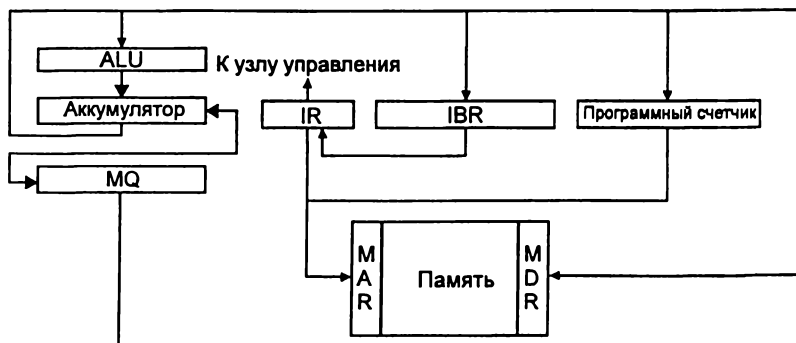


Рис. 1.5. Блок-схема фон-неймановской архитектуры. MQ – регистр множителя и частного, IR – регистр команд, IBR – буферный регистр команд, MAR – регистр адреса памяти, MDR – регистр данного памяти.

гистры вместе с их функциями перечислены в табл. 1.1. Обратите внимание на то, что два регистра явным образом адресуются с помощью команд и определены в системе команд (эти регистры называют архитектурными), в то время как остальные шесть в системе команд не определены, но используются узлом управления для пересылки битов при выполнении команды (их называют регистрами реализации).

Таблица 1.1. Регистры фон-неймановской системы команд

Название	Функция
Архитектурные регистры	
Аккумулятор АС, 40 бит	Содержит выход ALU после арифметической операции, загруженное из памяти данное, старшие биты произведения или делитель для операции деления.
Регистр множителя и частного MQ, 40 бит	Содержит временное данное, например, множитель, младшие биты произведения или частное от операции деления.
Регистры реализации	
Программный счетчик РС, 12 бит *	Содержит указатель памяти. В РС находится адрес следующей выбираемой из памяти пары команд.
Буферный регистр команд IBR, 40 бит	Содержит выбранную из памяти пару команд.

Таблица 1.1. *Продолжение*

Название	Функция
Регистр команд IR, 20 бит	Содержит текущую команду в течение времени ее декодирования в узле управления.
Регистр адреса памяти MAR, 12 бит	Содержит адрес памяти в течение цикла памяти (чтения или записи). MAR получает адрес для выборки команды из счетчика или адрес из адресного поля команды для чтения или записи данного.
Регистр данного памяти MDR, 40 бит	Содержит данное (команду или данные) в цикле обращения к памяти.

^{*}) Программный счетчик отличается от остальных регистров. Он может быть загружен значением при выполнении команды перехода, что делает его архитектурным регистром, но не может быть прочитан или записан, что делает его регистром реализации.

Операции

В фон-неймановской системе команд предусмотрены операции трех типов:

- пересылки между аккумулятором, регистром множителя-частного и памятью,
- операции ALU, например сложение, вычитание, умножение и деление,
- команды безусловных и условных переходов, приводящие к изменению программного потока⁵⁾.

Фон-неймановская архитектура содержит 21 команду, что достаточно для программирования любого алгоритма. Все эти команды представлены в табл. 1.2. Следует, однако, заметить, что современные системы команд требуют существенно меньшего числа команд. Команды разделены на три группы: пересылки, ALU и управления. Такая группировка команд типична для всех компьютеров, построенных на базе фон-неймановской архитектуры. В таблице 1.2 использована более поздняя терминология, чем та, которой пользовался фон Нейман.

⁵⁾ Многие историографы вычислительной науки приписывают фон-неймановской архитектуре первое использование условных переходов в компьютере с хранимой программой. Более ранние компьютеры не обладали этим свойством; подпрограммы реализовывались в виде встроеного кода.

Таблица 1.2. Фон-неймановская система команд

Команды пересылки	
1. $AC \leftarrow MQ$	Пересылка числа из MQ в аккумулятор.
2. $M(x) \leftarrow AC$	Пересылка числа из аккумулятора в ячейку x памяти. Адрес памяти x извлекается из младших 12 бит команды.
3 ^{*)} . $M(x, 28:39) \leftarrow AC(28:39)$	Замена левых 12 бит левой команды, расположенной по адресу x в памяти, левыми 12 бит аккумулятора ^{*)} .
4 ^{*)} . $M(x, 8:19) \leftarrow AC(28:39)$	Замена левых 12 бит правой команды, расположенной по адресу x в памяти, левыми 12 бит аккумулятора ^{*)} .
Команды ALU	
5. $ACc \leftarrow M(x)$	Очистка аккумулятора и прибавление числа из ячейки с адресом x в памяти.
6. $AC \leftarrow ACc - M(x)$	Очистка аккумулятора и вычитание числа из ячейки с адресом x в памяти.
7. $AC \leftarrow ACc + M(x) $	Очистка аккумулятора и прибавление абсолютного значения числа из ячейки с адресом x в памяти.
8. $AC \leftarrow ACc - M(x) $	Очистка аккумулятора и вычитание абсолютного значения числа из ячейки с адресом x в памяти.
9. $AC \leftarrow AC + M(x)$	Прибавление числа из ячейки с адресом x в памяти к числу в аккумуляторе.
10. $AC \leftarrow AC - M(x)$	Вычитание числа из ячейки с адресом x в памяти из числа в аккумуляторе.
11. $AC \leftarrow AC + M(x) $	Прибавление абсолютного значения числа из ячейки с адресом x в памяти к числу в аккумуляторе.
12. $AC \leftarrow AC - M(x) $	Вычитание абсолютного значения числа из ячейки с адресом x в памяти из числа в аккумуляторе.
13. $MQc \leftarrow M(x)$	Очистка регистра MQ и прибавление к нему числа из ячейки с адресом x в памяти.
14. $ACc, MQ \leftarrow M(x) \times MQ$	Очистка аккумулятора и умножение числа из ячейки с адресом x в памяти на число в MQ; помещение старших 39 бит результата в аккумулятор и младших 39 бит результата в MQ.

^{*)} Эти команды пересылают адресную часть команды между памятью и аккумулятором; они необходимы для поддержки модификации адреса. Индексная адресация, включаемая в наши дни во все архитектуры, не была еще изобретена.

^{*)} Запись $M(x, 0:19)$ обозначает правые 20 бит ячейки $M(x)$; $M(x, 20:39)$ обозначает левые 20 бит и т. д.

Таблица 1.2. Продолжение

15. $MQ_c, AC \leftarrow AC + M(x)$	Очистка аккумулятора и деление числа в аккумуляторе на число из ячейки с адресом x в памяти; оставление остатка в аккумуляторе и помещение частного в MQ .
16. $AC \leftarrow AC \times 2$	Умножение числа в аккумуляторе на 2.
17. $AC \leftarrow AC + 2$	Деление числа в аккумуляторе на 2.
Команды управления	
18. Переход на $M(x, 20:39)$	Передача управления на левую команду пары команд в $M(x)$.
19. Переход на $M(x, 0:19)$	Передача управления на правую команду пары команд в $M(x)$.
20. Если $AC \geq 0$, то $PC \leftarrow M(x, 0:19)$	Если число в аккумуляторе ≥ 0 , то управление передается на правую команду в $M(x)$.
21. Если $AC < 0$, то $PC \leftarrow M(x, 20:39)$	Если число в аккумуляторе < 0 , то управление передается на левую команду в $M(x)$.

1.2.2. Интерпретация команд

Интерпретация команды осуществляется с помощью трех шагов, или циклов. Команда выбирается, декодируется и выполняется. В последующих разделах мы обсудим эти три шага.

Выборка команды

На рисунке 1.6 показана в виде блок-схемы часть цикла выборки команды. Поскольку за один раз выбираются две команды, на первом шаге следует определить, требуется ли выборка из памяти. Эта проверка осуществляется путем анализа младшего бита (LSB) программного счетчика (PC). В результате выборка из памяти выполняется на каждое второе состояние PC или после команды перехода. Выборка из памяти помещает левую (L) и правую (R) команды в буферный регистр команд (IBR).

Команды, за исключением команд переходов, выполняются в таком порядке: левая, правая, левая, правая и т. д. Рассмотрим, например, случай, когда только что завершилось выполнение команды R. В IBR нет команды и осуществляется обращение к памяти с целью выборки пары команд.

Обычно далее выполняется команда L. Мы перемещаемся по блок-схеме влево, помещая команду в регистр команд IR. Команда R остается в IBR, готовая для использования в следующем цикле, что экономит один цикл обращения к памяти за следующей командой.

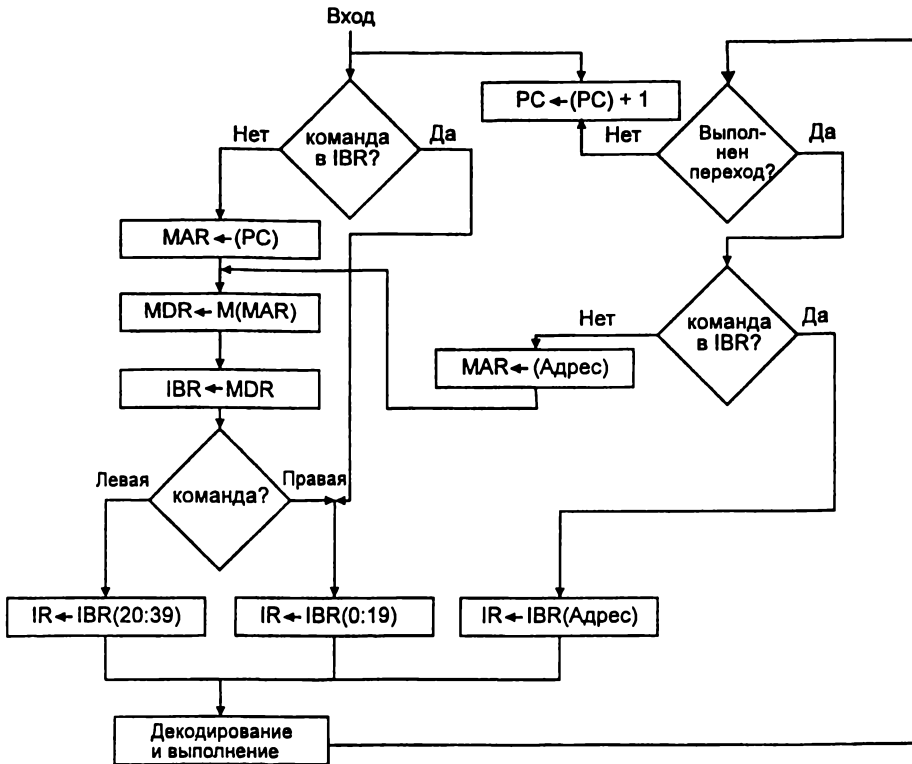


Рис. 1.6. Цикл выборки команды.

Если предыдущей командой была команда перехода на команду R пары команд, команда L не нужна, и в регистр IR помещается команда R. В целом мы имеем такие последовательности команд:

Последовательность Действие

R следует за L

L следует за R

Переход от L к L

Переход от R к R

Переход от L к R

Переход от R к L

Обращение к памяти не требуется.

Инкремент PC, обращение к памяти, использование команды L.

Требуется обращение к памяти; используется команда L.

Требуется обращение к памяти; используется команда R.

Если в том же компьютерном слове, обращение к памяти не требуется.

Если в том же компьютерном слове, обращение к памяти не требуется.

После декодирования и выполнения команды РС инкрементируется для выборки следующей команды и управление передается на точку входа.

Декодирование и выполнение

На рисунке 1.6 операция декодирования команды лишь обозначена. Однако команда была помещена в регистр IR. Как показано на рис. 1.7, комбинаторная логика в узле управления декодирует код операции и определяет, какая конкретно команда должна быть выполнена. Другими словами, декодирование аналогично предложению CASE многих языков программирования. На рисунке 1.7 приведена блок-схема выполнения двух команд с номерами 21 и 6. После выполнения команды управление возвращается в цикл выборки команды, как это показано на рис. 1.6.

Последовательность операций в цикле интерпретации команды определяется «защитым» (аппаратно реализованным) конечным автоматом, который будет обсуждаться в гл. 5. Каждое состояние этого автомата описывается в виде блок-схемы, для представления состояний предусматриваются триггеры, а для перехода из состояния в состояние используются логические узлы. После изобретения микропрограммирования блок-схема стала реализовываться в виде последовательности команд, выполняемых микромашиной. Другими словами, управление осуществляется не аппаратно реализованным конечным автоматом, а вторым компьютером.

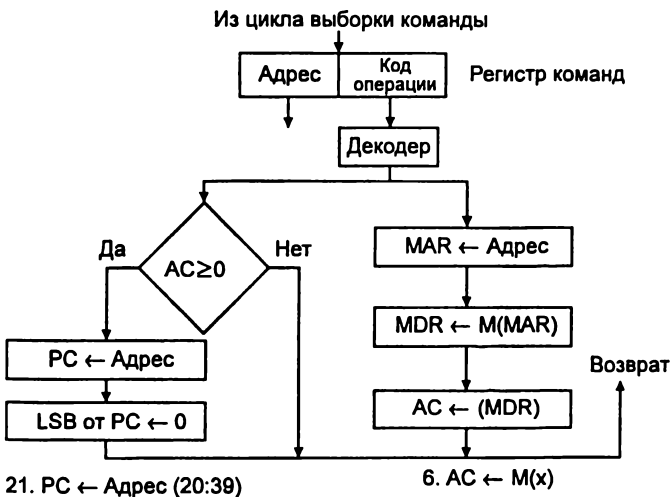


Рис. 1.7. Декодирование и выполнение.

Пример программы

Сложность программирования в рамках фон-неймановской системы команд без операций индексирования иллюстрируется примером, представленным в табл. 1.3. Пусть нам надо вычислить сумму двух векторов длиной 1000^{*)}:

$$C_i = A_i + B_i$$

Вектор **A** хранится по адресам 1001–2000, вектор **B** по адресам 2001–3000, а для вектора **C** зарезервированы ячейки 3001–4000. Программа прежде всего инициализирует три ячейки памяти со значением счетчика цикла 999 (для получения адресов одноименных элементов векторов), константой 1000 и константой 1 (для декремента счетчика цикла).

Таблица 1.3. Программа сложения векторов

Адрес	Данное/Команда	Комментарии
0	999	Счетчик
1	1	Константа
2	1000	Константа
Внутренний цикл для каждого сложения		
3L	AC ← M(3000)	Загрузка B_i
3R	AC ← AC + M(2000)	$B_i + A_i$
4L	M(4000) ← AC	Сохранение AC
Проверка счетчика цикла и продолжение или завершение		
4R	AC ← M(0)	Загрузка счетчика
5L	AC ← AC – M(1)	Декремент счетчика
5R	Если AC ≥ 0, переход на M(6, 0:19)	Проверка счетчика на 0
6L	Переход на M(6, 20:39)	Останов
6R	M(0) ← AC	Сохранение счетчика
Модификация адресов (декременты)		
7L	AC ← AC + M(1)	Инкремент счетчика
7R	AC ← AC + M(2)	Добавление константы
8L	M(3, 8:19) ← AC(28:39)	Сохранение модифицированного адреса в 3R
8R	AC ← AC + M(2)	Добавление константы

^{*)} Под векторами автор подразумевает одномерные массивы по 1000 элементов каждый. — *Прим. перев.*

Таблица 1.3. Продолжение

Адрес	Данное/Команда	Комментарии
9L	$M(3, 28:39) \leftarrow AC$	Сохранение модифицированного адреса в 3L
9R	$AC \leftarrow AC + M(2)$	Добавление константы
10L	$M(4, 28:39) \leftarrow AC$	Сохранение модифицированного адреса в 4L
10R	Переход на $M(3, 20:39)$	Безусловный переход на 3L

1.2.3. Ограничения фон-неймановской системы команд

Для фон-неймановской системы команд характерен ряд ограничений, отчетливо проявляющихся в программе сложения векторов (табл. 1.3). Первое ограничение уже отмечалось в разделе 1.2.2: отсутствие средств автоматической модификации адреса, такой, как в современных процессорах. В результате для индексации массива адреса в командах должны модифицироваться другими командами. Это и есть самомодифицирующийся код, который легко приводит к программным ошибкам.

Поскольку во времена разработки фон-неймановской системы команд не существовало понятия модульного программирования, архитектура не предоставляет регистра базы для разделения команд и данных.

Другое важное ограничение также может быть проиллюстрировано на примере программы сложения векторов в табл. 1.3. В рассматриваемой архитектуре программный счетчик является регистром реализации^{*)}. Все современные процессоры имеют архитектурный программный счетчик; в этом случае его содержимое можно сохранять и восстанавливать, что позволяет реализовать программные концепции подпрограмм и вызовов процедур. Эти концепции не могут быть реализованы в фон-неймановской системе команд, где программный счетчик является элементом реализации.

Наконец, как уже отмечалось в разделе 1.2.1, в фон-неймановской системе команд ввод-вывод был только кратко упомянут. Реализация ввода-вывода в этом и других компьютерах будет рассмотрена в гл. 7.

^{*)} Под регистром реализации понимается программно изменяемый регистр, под архитектурным регистром понимается регистр, модифицируемый аппаратурой. — *Прим. ред.*

1.3. Немного истории

Индексация. Видимо, индексация была впервые использована в системе команд машины Mark 1, разработанной Килберном и Уильямсом в Манчестерском университете в 1946—1949 гг. и изготовленной корпорацией Ferranti. Первая машина Ferranti была установлена в 1951 г. Хотя этот компьютер знаменит в основном разработкой и использованием виртуальной памяти, не менее значимой представляется пионерская работа по внедрению индексации. Индексация представлялась в виде дополнительной функции, называемой индексным регистром («B lines», или «B box»).

IBM 704, выпущенная в 1954 г., имела три индексных регистра. Эти регистры вместе с использованием плавающей запятой предоставили аппаратную поддержку для разработки языка FORTRAN [3]. Любопытно отметить, что машина IBM 701, впервые установленная в 1953 г., требовала модификации базового адреса, как это было и в фон-неймановской системе команд.

Подпрограммы. Подпрограмма представляет собой программный фрагмент, который выполняется только в случае вызова его из главной программы. После завершения подпрограммы продолжается выполнение главной программы. Поскольку подпрограммы требуют обеспечения возврата в главную программу, необходимым условием их использования является возможность сохранения содержимого программного счетчика. В фон-неймановской системе команд такая возможность не предусмотрена, и подпрограммы выполняться не могут.

Эта проблема была впервые решена Уилером под руководством Мориса Уилкса для машины Кембриджского университета EDSAC [15]. Программный счетчик стал архитектурным регистром, допускающим сохранение и восстановление его содержимого. Подпрограммы будут подробнее рассмотрены в гл. 3.

1.3.1. Предшественники фон-неймановской системы команд

Из компьютеров, появившихся еще до предложения фон-неймановской системы команд, следует отметить машины Бэббиджа, Атанасова, Цузе, Айкена и Эккерта—Мочли. Однако еще до этих машин в течение нескольких столетий изучались и разрабатывались математические и вычислительные средства; особенно важными оказались изобретение логарифмов и достижения численного анализа. Эти темы выходят за рамки настоящей книги; для получения дополнительной информации можно обратиться к работам [7] и [8]. Читателям, интересующимся ранними компьютерами, мы рекомендуем работы [10]

и [11]. Эта тема широко освещается в журнале *IEEE Annals of the History of Computing* (вышел уже 21-й том). Серия репринтов Института Чарлза Бэббиджа [5] позволяет познакомиться со статьями пионеров компьютерной техники.

Ранние компьютеры разрабатывались для решения задач двух типов. Во-первых, возникла необходимость в составлении таблиц функций; машины Бэббиджа, Айкена и Эккерта—Мочли были построены именно для этой цели. Второй класс задач составляли решения систем линейных уравнений; эти проблемы решались машинами Атанасова и Цузе. Решение и тех, и других задач людьми для сколько-нибудь нетривиальных случаев требовало непомерных усилий, причем по мере увеличения сложности задач возрастала вероятность ошибок даже при использовании для арифметических действий механических устройств. Решение очень сложных задач было практически невозможно, так как требовало слишком много времени и сопровождалось обилием ошибок⁶⁾.

Чарлз Бэббидж (1792—1871) по праву считается отцом современных компьютеров. В начале 19-го столетия возникла острая необходимость в издании надежно вычисленных таблиц функций. Бэббидж предложил использовать для решения этой задачи разностную машину. В 1820—1822 гг. он построил небольшую модель разностной машины, выдающей для функций с постоянной разностью второго порядка шесть значащих цифр; внешний вид этой машины показан на рис. 1.8. Использование разностных методов объяснялось тем, что они требовали только операций сложения и вычитания. Технология выполнения этих операций была уже давно известна, еще с механического сумматора Блеза Паскаля, использовавшего вращающиеся десятичные колеса.

Попытки создать большой вариант разностной машины для 26 десятичных разрядов с разностями шестого порядка, предпринятые в 1836 г., закончились неудачей из-за сложностей в изготовлении механических частей и прекращения финансовой помощи британского правительства. Макет этой большой машины, построенный по чертежам Бэббиджа, был завершен в 1991 г. и находится в Лондонском музее наук. Хотя сам Бэббидж так и не реализовал свою разностную машину, устройство, основанное на его идеях, было построено в Стокгольме в 1843 г. Эдвардом Шойцем.

Потерпев неудачу с разностной машиной, Бэббидж занялся «аналитической машиной». Он попытался разработать машину, которая могла бы выполнять любые последовательности операций и не была бы рабом техники разностных методов численного анализа. Аналити-

⁶⁾ В течение 30-х годов организация *Администрация развития фабрик* нанимала молодых женщин для вычисления таблиц вручную. Калькуляторы при этом не использовались, поскольку одна из целей этой программы состояла в предоставлении работы максимальному числу людей.

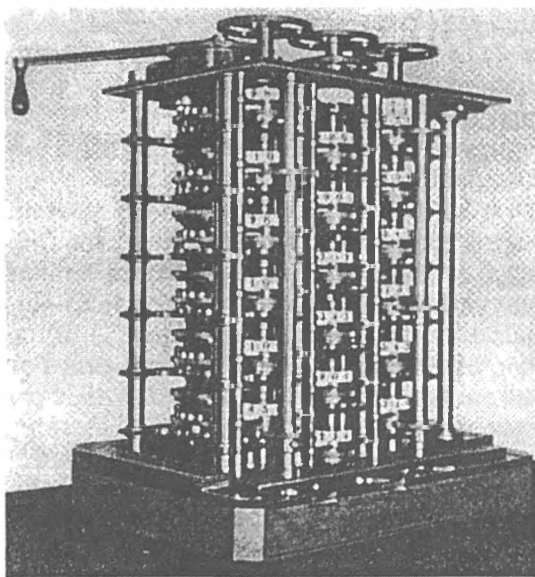


Рис. 1.8. Точная копия разностной машины Бэббиджа (с любезного разрешения корпорации IBM).

ческая машина состояла из трех частей: магазин выполнял функции памяти современных компьютеров, мельница служила арифметическим узлом, третья же часть была управляющей. Очередность работы магазина и мельницы задавалась с помощью перфокарт, технология применения которых была доведена до совершенства в жаккардовом ткацком станке. Бэббидж таким образом впервые обозначил три основных компонента современных компьютеров.

Джон Атанасов (1903—1995), профессор математики и физики в Айовском государственном колледже, создал первую машину, которая вывела компьютерную технологию из эпохи механических приспособлений и реле в эру электроники. Атанасов осознавал потребность в устройстве для решения систем линейных алгебраических уравнений. Он оценил, что время в часах, требуемое механическому калькулятору для решения системы из n линейных уравнений с n неизвестными, составляет

$$\frac{n^3}{64}.$$

Решение больших систем уравнений, например при $n = 30$, требовало бы свыше 50 8-часовых дней и сопровождалось бы при этом множеством ошибок. В духе ученого-физика он приступил к разработке и созданию аппарата для решения таких систем уравнений. Этот аппарат и был реализован в виде специализированного цифрового

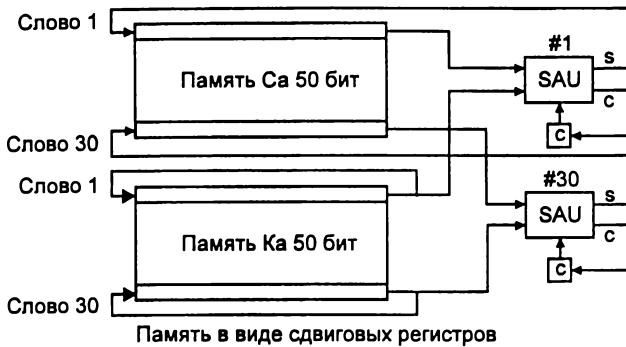


Рис. 1.9. Организация памяти и арифметического узла компьютера ABC.

компьютера, известного под названием компьютера Атанасова–Бери (ABC). Бери был помощником Атанасова в его исследованиях. К сожалению, работа Атанасова не получила широкой известности до 1967–1969 гг., когда в ходе судебного процесса по поводу патента на ENIAC между фирмами Сперр-Рэнд и Ханивел обнаружилось, что Джон Мочли (один из разработчиков ENIAC) посетил Атанасова в 1940 г. или 1941 г. и получил данные по компьютеру ABC.

Помимо решения приступить к конструированию электронного компьютера, Атанасову принадлежит еще целый ряд удачных предложений. Во-первых, арифметические действия должны выполняться с помощью логических операций, а не путем перебора или счета, как в машине Бэббиджа. Во-вторых, данные должны представляться в виде 50-битовых двоичных чисел со знаком, использующих дополнение до 1 (см. гл. 3, где это представление обсуждается более подробно). В-третьих, в качестве памяти предполагалось использовать вращающийся барабан с конденсаторами, предвестник нынешних DRAM. Время оборота барабана составляло 1с; при каждом обороте конденсаторы подзаряжались.

Организация памяти и последовательных арифметических узлов (SAU) показана на рис. 1.9. Предусмотрены два банка памяти, Са и Ка. Каждый банк содержит по 30 слов длиной 50 бит. Коэффициенты и свободный член одного уравнения загружаются в Са, а коэффициенты и свободный член другого — в Ка. Тридцать полных одноразрядных сумматоров и вычитателей складывают или вычитают биты Ка и Са, сохраняя переносы и заменяя бит за битом содержимое Са получаемой суммой. При наличии памяти на 30 слов система может решать до 29 уравнений с 29 неизвестными.

Коэффициенты и свободный член одного уравнения считываются с помощью перфосчитывателя в память Са, другого уравнения — в память Ка. Методом исключения Гаусса один из коэффициентов в Са сводится к нулю. В этом методе 30 коэффициентов в Ка многократно вычитаются из Са, сводя один из коэффициентов в Са (исключаемый

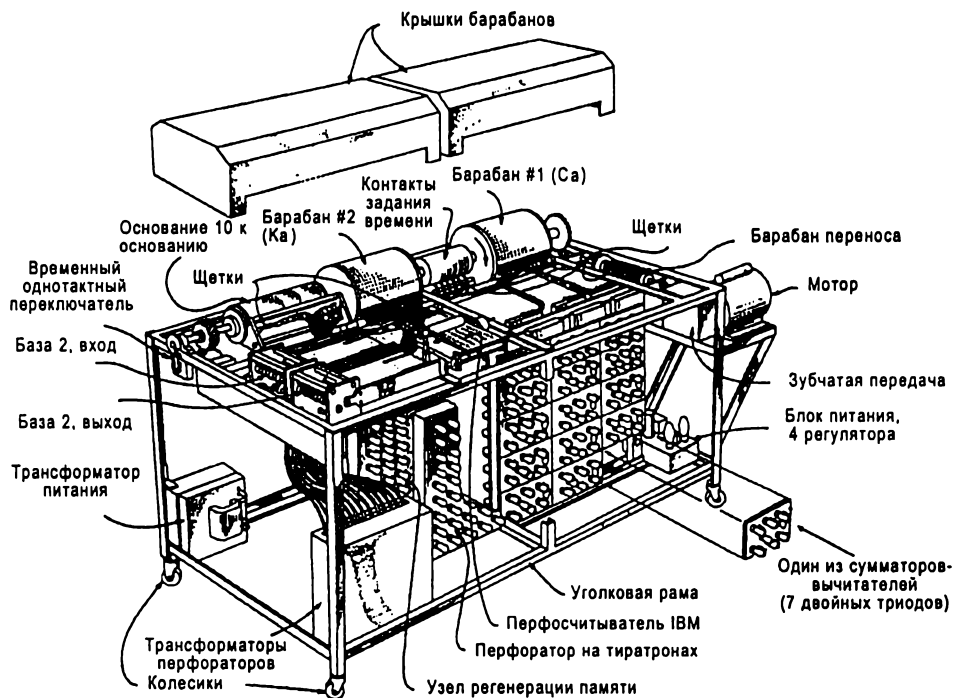


Рис. 1.10. Компьютер Атанасова–Бери (© [2]).

коэффициент) к нулю. Процесс заключается в последовательном делении коэффициентов в каждом цикле на два. Когда коэффициент становится отрицательным, вычитание заменяется сложением; когда коэффициент становится положительным, сложение заменяется вычитанием. Этот процесс напоминает деление без восстановления остатка (см. гл. 3). После того как один из коэффициентов в Са сведен к нулю, результаты выводятся на перфокарту и с перфосчитывателя получают два новых уравнения; таким образом число неизвестных в системе уменьшается на 1. Этот процесс ввода, вычислений и вывода повторяется до исчерпания всех коэффициентов системы.

На рисунке 1.10 [2] показан чертеж окончательного варианта компьютера ABC, прошедшего испытания весной 1942 г. Атанасов сообщал о трудностях надежного вывода на перфокарты. В условиях военного времени Атанасову и его студентам пришлось выполнять военные заказы, и компьютер не был завершен; в конце концов машина была разобрана. Сегодня узлы этой машины можно увидеть в Компьютерном музее в Бостоне. Работающий макет компьютера ABC создан и постоянно экспонируется в Айовском государственном университете.

Конрад Цузе (1910–1995) работал в Германии и в период с 1934 по 1944 гг. он разработал и построил целый ряд машин, предназначенных для решения систем линейных уравнений: Z1, Z2 и Z3. Имея уче-

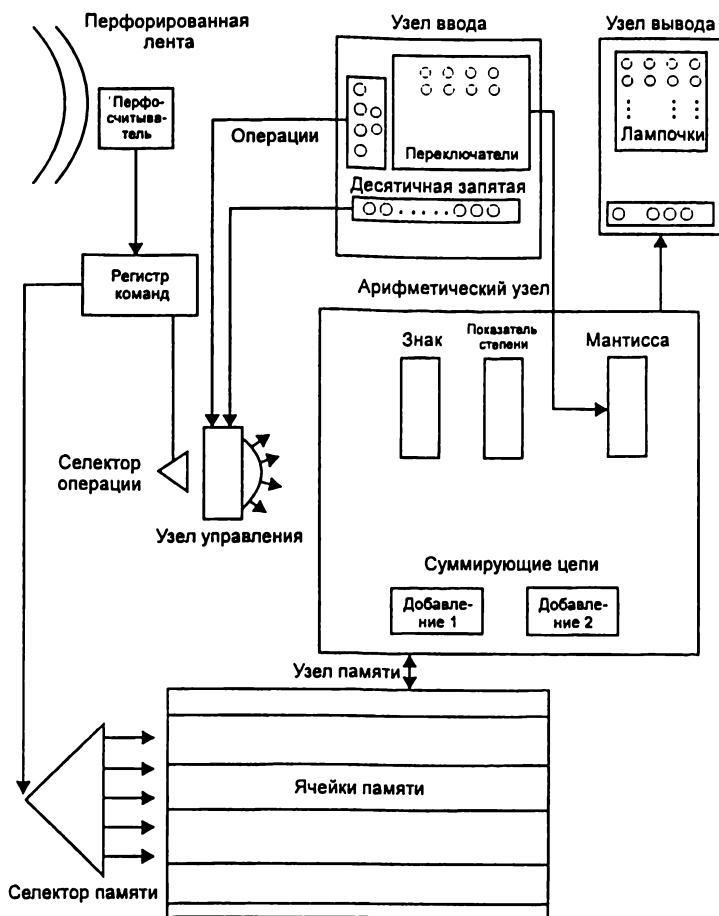


Рис. 1.11. Компьютер Цузе Z3 (© [6]).

ную степень по гражданскому строительству, Цузе был заинтересован в решении задач устойчивости и несущей способности конструкций. Его исследования требовали решения систем линейных уравнений, в частности системы из 30 уравнений с 30 неизвестными. Для получения ответа с помощью механических калькуляторов потребовалось бы несколько месяцев работы [6], [13].

Работа Цузе поддерживалась его семьей; ему также время от времени удавалось получать небольшие субсидии от компании Henschel Aircraft. К сожалению, машина Z3 была уничтожена во время бомбардировки Берлина, однако в 1961–1963 гг. с помощью оригинальных чертежей был построен ее макет.

Z3 представляет собой релейную машину, содержащую приблизительно 2600 реле. Блок-схема Z3 представлена на рис. 1.11 [6]. Любо-

пытно, что исходные данные имеют тип с плавающей запятой (по основанию 2), а ввод программы осуществляется с использованной 35-миллиметровой пленки, на которую нанесена перфорация. Основная память (64 слова по 22 бит плюс бит знака) является непосредственно адресуемой и реализована на реле. В качестве выходного устройства использовалась матрица на лампочках; вывод на печать не предусматривался. Ввод данных осуществлялся с помощью переключателей. Машина Z3 относится к Гарвардской архитектуре, поскольку она имеет отдельную память для программы и данных. Минимальные возможности ввода и вывода не должны удивлять, ведь решение больших систем линейных уравнений является «вычислительно-ориентированной» задачей: немного исходных данных, большой объем вычислений, немного результирующих данных.

Для выполнения операций предназначался базовый комплект из двух регистров. Команды имели форму

$$R2 \leftarrow R1 <\text{операция}> R2$$

Набор команд включал 11 операций — все, что мы ожидаем найти в современном компьютере, за исключением условных переходов, механизма подпрограмм и индексации. Циклы реализовывались с помощью бесконечной перфоленты.

Набор команд вместе с числом тактов, требуемых для выполнения каждой команды, приведен в табл. 1.4. Тактовая частота составляла приблизительно 5,33 Гц. Наконец, в машине использовались в примитивной форме конвейерная обработка и параллельные вычисления. Перфосчитыватель обгонял вычисления на две команды. В результате в процессе вычислений можно было вернуться на две команды назад; для компенсации низкой скорости памяти две операции с памятью могли выполняться заранее, и промежуточный результат мог быть сохранен в регистре и использован в следующей команде. Мы еще вернемся к этим идеям в гл. 6, где будет описана конвейеризация.

Говард Айкен (1900–1973), доцент Гарвардского университета, в 1937 г. выдвинул предложение о разработке машины для вычисления таблиц функций, обычно определяемых с помощью бесконечных рядов. Айкен знал о работах Бэббиджа, но, видимо, не был знаком с предложениями Атанасова и Цузе. Айкен отверг использованный Бэббиджем метод разностей, как имеющий слишком большие ограничения для решения поставленных задач. В своей статье [1] Айкен указал, что предлагаемая машина должна выполнять следующие процессы: «вычисления по формулам и сведение результатов в таблицы, вычисление рядов, решение обыкновенных дифференциальных уравнений первого и второго порядка, численное интегрирование, а также численное дифференцирование эмпирических данных». Для достижения этой цели требуется программируемое устройство с возможностью выполнения обычных арифметических функций.

Таблица 1.4. Набор команд и время их выполнения для машины Z3

Тип	Описание	Такты	Время (с)
Арифметика	Умножение	16	3
	Деление	18	3,4
	Квадратный корень	20	3,75
	Сложение	3	0,56
	Вычитание	4–5	0,75–0,93
	Инверсия		
	Квадрат		
	Отрицание		
	Удвоение		
	Деление пополам		
	Умножение на 10		
Память	Деление на 10		
	Загрузка	1	0,18
Ввод-вывод	Сохранение	0–1	0–0,18
	Чтение с переключателей	9–41	1,68–7,7
	Высвечивание результата	9–41	1,68–7,7

Айкен представил свои предложения компании IBM, получив там одобрение и, что более важно, финансовую и инженерную поддержку. В результате группой опытных инженеров компании IBM была разработана и реализована требуемая машина. Здесь мы видим совершенно другой подход по сравнению с реализацией многих более ранних машин, которые разрабатывались и конструировались созданными для этого группами университетских сотрудников и студентов. IBM имела большой опыт производства табуляторов, машин, использующих перфокарты для выполнения несложных вычислений с управлением от коммутационной панели⁷⁾. Эта технология была естественным образом использована в устройстве, известном теперь под названием автоматического калькулятора с управляемой последовательностью (SCAC). Позже это устройство получило имя «Гарвардский Mark 1» [14]⁸⁾.

Как и аналитическая машина Бэббиджа, и Z3, Гарвардский Mark I имеет разделенные памяти команд и данных. Программа хранится на 24-дорожечной бумажной перфоленте; команды считываются в про-

⁷⁾ Значительная часть работы по расшифровке перехваченных сообщений в США во время Второй мировой войны основывалась на табулирующих машинах, собирающих статистическую информацию о японских сообщениях.

⁸⁾ В те дни термин «компьютер» (вычислитель) относился к людям, выполняющим вычисления. Машина для производства вычислений называлась калькулятором.

МЕХАНИЧЕСКИЙ ПРИВОД СИСТЕМЫ

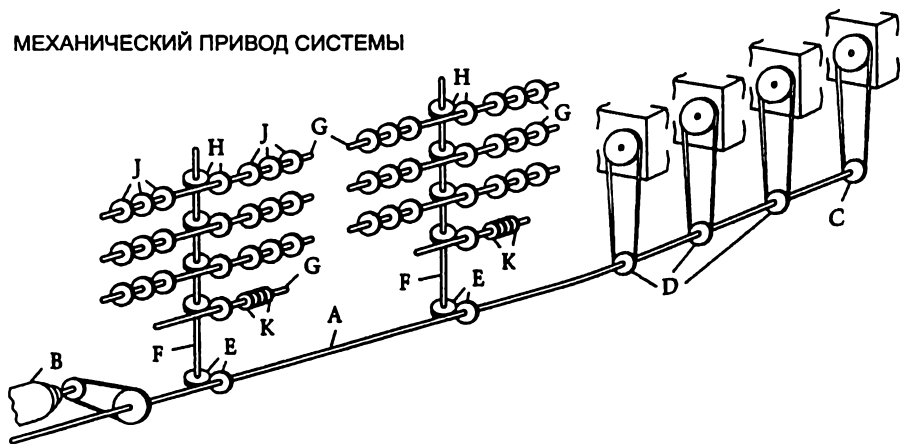


Рис. 1.12. Механический привод машины Гарвардский Mark I (с любезного согласия Гарвардского университета).

цессор по порядку их следования, а средства организации циклов отсутствуют. Память состоит из 60 24-битовых десятичных регистров. Каждый из этих регистров действует как аккумулятор, выполняющая распределенные арифметические функции (в машине нет централизованного арифметического узла, как в фон-неймановской архитектуре). Ввод данных осуществляется с помощью одного из трех перфосчитывателей с бумажной ленты, аналогичных по конструкции считывателю программы.

Регистры реализованы на шаговых переключателях J, приводимых в движение мотором B и вращающимися стержнями A, F и G, как это показано на рис. 1.12. Шаговые переключатели связаны со стержнями с помощью электромеханических муфт. Сложение выполняется следующим образом. Пусть, например, один переключатель имеет номер 3, другой — номер 5, и должно быть выполнено прибавление 3 к 5. Импульсный генератор на основе кулачкового механизма вырабатывает сигналы от переключателя 3, которые заставляют переключатель 5 пройти три шага, после чего муфта расцепляется, оставляя на переключателе число 8. Сегодня этот механизм представляется практически неработоспособным. Тем не менее инженеры IBM построили в 1943 г. весьма надежный компьютер, который эксплуатировался вплоть до 1959 г.⁹⁾

Машинные команды включают обычные операции сложения и вычитания, а также умножения и деления. Другие команды вместе с временами их выполнения перечислены в табл. 1.5. Данные содержат 23

⁹⁾ Когда спустя много лет Айкена спросили, почему он использовал именно технологию IBM, а не какую-то другую, например электронную, он заметил, что деньги были выделены IBM, а не компанией с опытом работы в электронике.

Таблица 1.5. Команды и времена их выполнения для машины Mark I

Команды	Секунды	Такты
Сложение	0,3	1
Вычитание	0,3	1
Умножение	6,0	20
Деление	11,4	38
$\text{Log}_{10}(x)$	68,4	228
10^x	61,2	204
$\sin(x)$	60,0	199

десятичных разряда со знаком. Десятичная запятая позиционируется с помощью штекерных соединений. Однако из-за трудностей программирования десятичной запятой она в течение большей части жизни машины устанавливалась между 15-й и 16-й цифрами. В гл. 3 более подробно описан этот тип данных, называемый смешанным.

Дж. Преспер Эккерт (1919–1995) и Джон Мочли (1908–1980) по контракту с армией Соединенных Штатов разработали в Пенсильванском университете электронный числовой интегратор и вычислитель ENIAC. Работы начались в 1942 г., а в 1946 г. была получена работоспособная машина [9].

Данные составляли 10 десятичных цифр со знаком. Организация машины такая же, как у Mark I в том отношении, что в качестве базовой памяти и вычислительных элементов используются аккумуляторы. Машина реализована на вакуумных лампах; цикл сложения длится $1/5000$ с — более чем в 1600 раз быстрее, чем на машине Mark I. Назначение ENIAC состояло в вычислении и печати таблиц функций, а также артиллерийских таблиц стрельбы. Функциональные переключатели обеспечивали табличный ввод данных в виде 104 слов с 12 десятичными цифрами плюс знак.

ENIAC имеет 20 аккумуляторов, соединенных между собой с помощью пяти шин с образованием примитивного потокового процессора. Три из этих аккумуляторов показаны на рис. 1.13 вместе с пятью шинами [9]. Заметьте, что каждая шина, как и кабели, присоединяющие аккумуляторы к шинам, состоит из 11 проводников, соответствующих 10 цифрам и знаку. В целом машина состоит из 20 соединенных друг с другом калькуляторов. Передатчик константы мог посылать константы во все аккумуляторы. На рисунке 1.13 также показаны цепи для выдачи тактовых сигналов, служащих для синхронизации работы аккумуляторов. Поскольку машина создает на выходе таблицы, в ее состав был включен принтер.

Возможности программирования по сравнению с Mark I ограничены. Аккумуляторы с помощью кабелей подключались через переключики к шинам, получающим тактовые сигналы из центрального

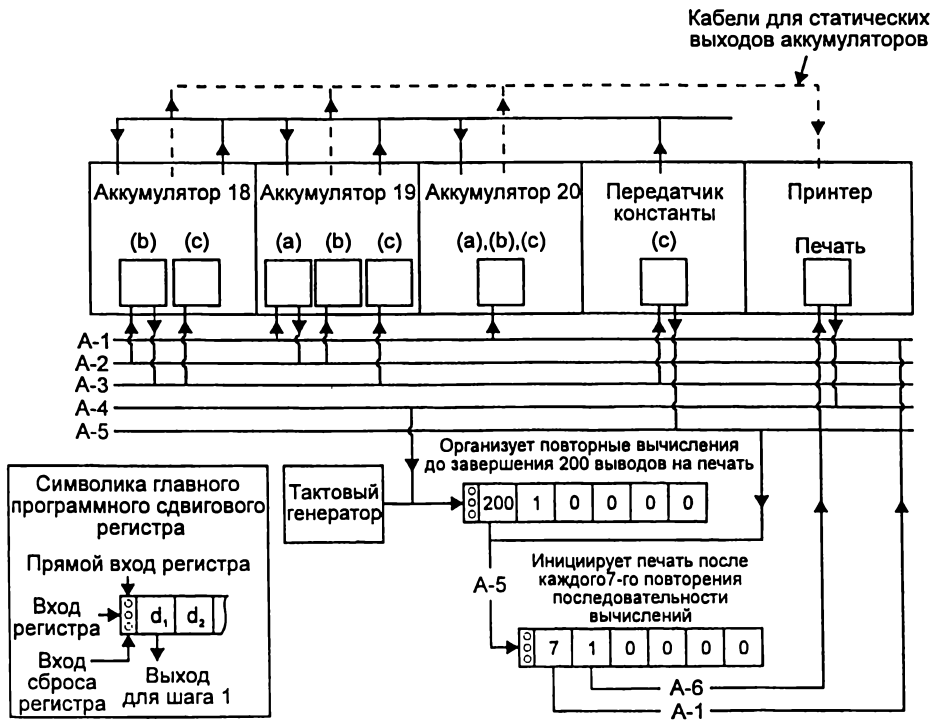


Рис. 1.13. ENIAC, три аккумулятора (© [9]).

источника. Перемычки позволяли установить алгоритм решения. Рассмотрим в качестве примера задачу вычисления таблиц n , n^2 и n^3 для n , $n + 1$, $n + 2$ и т. д., как показано в табл. 1.6.

Таблица 1.6. Вычисление на машине ENIAC таблиц n , n^2 и n^3

Шаг	Операция, начальное значение	Аккумулятор 18 $n = 2$	Аккумулятор 19 $n^2 = 4$	Аккумулятор 20 $n^3 = 8$
1	Прибавление содержимого A19 три раза к A20.	2	4	$8 + 12 = 20$
2	Прибавление содержимого A18 два раза к A19. Прибавление содержимого A18 три раза к A20.	2	$4 + 4 = 8$	$20 + 6 = 26$
3	Прибавление константы 1 к A18, A19 и A20.	$2 + 1 = 3$	$8 + 1 = 9$	$26 + 1 = 27$

Вычисления начинаются с задания начальных значений 2 , 2^2 и 2^3 (2, 4 и 8), вводимых в аккумуляторы 18, 19 и 20. С помощью показанных в таблице команд выполняется вычисление следующих значений. В качестве начальных значений можно было использовать также числа 1, 12 и 13 с получением тех же результатов после большего числа шагов.

Итак, мы видим, что начальные значения 2, 4 и 8 превратились в 3, 9 и 27. В конце каждой итерации результаты выводятся на печать. Обратите внимание на одновременное выполнение шагов 2 и 3. Поскольку эти операции взаимно независимы, они могут выполняться параллельно (эти проблемы обсуждаются в гл. 6).

Нетрудно представить сложности программирования и настройки этой машины. Особенные затруднения вызывал поиск ошибок. К счастью, перепрограммирование выполнялось нечасто, поскольку каждая задача выполнялась на компьютере в течение длительного времени, когда от сеанса к сеансу изменялись только входные константы.

Резюме. Краткая сводка некоторых существенных характеристик машин ABC, Z3, ENIAC и Mark I дана в табл. 1.7.

Таблица 1.7. Характеристики ранних компьютеров

Характеристика	ABC	Z3	ENIAC	Гарвардский Mark I
Цель	Системы линейных уравнений	Системы линейных уравнений	Баллистические таблицы	Функции, задаваемые бесконечными рядами
Основной тип данных	Целое, 50 бит (основание 2)	Плавающая запятая, 22 бит (основание 2)	Целое 10 цифр (основание 10)	Целое, 24 цифры (основание 10)
Память	2, 30 слов $\times$ 50 Конденсатор, регенерация каждую секунду	64 слова $\times$ 22, реле	20 слов $\times$ 10 цифр, вакуумные лампы	60 слов $\times$ 24 цифры, шаговые переключатели ^{*)}
ALU	Централизованно 30, по одному на каждое слово	Централизованно	Аккумуляторы, распределенное	Аккумуляторы, распределенное
Ввод программы	Защита для одного алгоритма	Перфорированная киноплёнка	Перемычки	Перфолента
Ввод данных	Перфокарты	Наборное поле	Переключатели, считыватель с карт IBM	Переключатели, перфолента
Вывод данных	Перфокарты	Лампочки	Перфокарты	Перфокарты, принтер
Логика	Вакуумные лампы, бит-последовательная, слово-параллельная	Реле, слово-параллельная	Вакуумные лампы, слово-параллельная	Электромеханическая, слово-параллельная

^{*)} Вращающийся стержень приводил в движение и синхронизовал все шаговые переключатели в системе, что аналогично общему тактовому генератору современных процессоров.

Таблица 1.7. Продолжение

Характеристика	ABC	Z3	ENIAC	Гарвардский Mark I
Тактовая частота (Гц)	60	5,3	100K	300
Дата ввода в эксплуатацию(г.)	1942	1941	1945	1944
Оценка стоимости (долл. США)	7000	6500	486804	450000

Нам хотелось бы завершить это краткое обсуждение ранних компьютеров замечанием о работах, выполняемых в Великобритании в военное время. Для расшифровки телеграфных сообщений немецкой армии, использующих весьма совершенные шифры, была разработана и построена большая машина на вакуумных лампах. Первая модель этого компьютера, названная Colossus, была введена в эксплуатацию в декабре 1943 г. К концу войны использовались уже 10 таких машин. Однако существование компьютеров Colossus оставалось военной тайной вплоть до 1975 г. [12]

Литература

1. Aiken, H. (1937). «Proposed Automatic Calculating Machine», *IEEE Spectrum*, August 1964. См. также [11].
2. Atanasoff, J.V. (1984). «Advent of Electronic Computing», *IEEE Annals of the History of Computing*, 6:3.
3. Blaauw, G. and Brooks, F.P. Jr. (1997). *Computer Architecture, Concepts and Evolution*, Addison-Wesley, Reading MA.
4. Burkes, A.W., Goldstine, H.H. and von Neumann, J. (1946). «Preliminary Discussions of the Logical Design of an Electronic Computing Instrument», U.S. Army Ordnance Department Report.
5. Cambell-Kelly, Williams M.R., eds. (1985). *The Moore School Lectures, Theory and Techniques for Design of Electronic Digital Computers*, reprinted in the Reprint Series for the History of Computing, Vol. 9, Charles Babbage Institute, MIT Press, Cambridge, MA, Tomash Publishers, Los Angeles.
6. Ceruzzi, P.E. (1981). «The Early Computers of Konrad Zuse, 1935 to 1945», *IEEE Annals of the History of Computing*, 3:3.
7. Chase, G.C. (1980). «History of Mechanical Computing Machinery», *IEEE Annals of the History of Computing*, 2:3.
8. de Solla Price, D. (1984). «A History of Calculating Machines», *IEEE Micro*, 4:1, pp.22—52.

9. Goldstine, H.H. and Goldstine, A. (1996). *The Electronic Numerical Integrator and Computer (ENIAC)*, M.T.A.C. 2, No. 15, 97—110 (1946). Reprinted in [11] and in *IEEE Annals of the History of Computing*, 18:1.

10. Goldstine, H.H. (1972). *The Computer from Pascal to von Neumann*, Princeton U. Press, Princeton, New Jersey.

11. Randell B., ed. (1973). *The Origins of Digital Computer: Selected Papers*, Springer-Verlag, New York.

12. Randell, B. (1976). «The Colossus», Technical Report series No 90, Computing Laboratory, University of Newcastle upon Tyne, U.K.

13. Rojas, R. (1997). «Konrad Zuse's Legacy: The Architecture of the Z1 and Z3», *IEEE Annals of the History of Computing*, 19:2.

14. The Staff of The Computation Laboratory. (1946). *A Manual of Operation For The Automatic Sequence Controlled Calculator*, reprinted in the Reprint Series for the History of Computing, Vol. 8, Charles Babbage Institute, Tomash Publishers, Los Angeles.

15. Wilkes, M., Wheeler, D. and Gill, S. (1951). *The Preparation of Programs for an Electronic Digital Computer*, Addison-Wesley, Cambridge, MA. Reprinted in Reprint Series for the History of Computing, Vol. 1, Charles Babbage Institute, Tomash Publishers, Los Angeles.

Упражнения

1.1. Напишите программу векторного сложения, используя систему команд IBM PowerPC и Intel Pentium.

1.2. Перепишите программу сложения векторов для фон-неймановской системы команд для векторов длиной 300.

1.3. Есть ли какие-либо основания к тому, что в программе векторного сложения сложение векторов начинается с больших адресов, перемещаясь к меньшим?

1.4. Бэббиджа интересовало решение выражения $n^2 + n + 41$ при увеличении целочисленных значений n . Покажите, как эту задачу можно решить на разностной машине для $n = 1, 2, \dots, 10$.

1.5. Получите первые члены функции синуса с помощью хорошо известного разложения в ряд. Затем вычислите первые 10 членов функции синуса, используя метод разностей.

1.6. Воспользуйтесь архитектурой ENIAC и табл. 1.6 для вычисления n , n^2 и n^3 для $n = 1$, $n = 2$ и $n = 3$.

1.7. Модифицируйте программу, приведенную в табл. 1.6, для вычисления с помощью ENIAC n^4 . Подсказка:

$$(n+1)^4 = (n+1)(n+1)(n+1)(n+1).$$

1.8. Напишите программу вычисления детерминанта матрицы 3×3 для Z3. Предположите, что девять входных данных вводятся с наборного поля.

1.9. Покажите, каким образом компьютер ABC решает приведенную ниже систему уравнений. Используйте десятичные значения и постройте график сходимости первого коэффициента к нулю.

$$2x + 5y = 9$$

$$3x + 2y = 8$$

1.10. Сравните характеристики каждой из четырех машин, описанных в табл. 1.7, с характеристиками современных компьютеров или перспективных разработок.

1.11. На рисунке 1.10 показан вращающийся барабан со щетками, используемый для преобразования чисел от основания 10 к основанию 2. Подумайте, как это устройство работает.

1.12. Изобразите последовательность шагов при выполнении команд 1 и 3 из табл. 1.2 в виде блок-схемы, аналогичной рис. 1.7.

ГЛАВА 2

МОДЕЛИ И ОЦЕНКИ ПРОИЗВОДИТЕЛЬНОСТИ

2.0. Введение

Проектировщики всегда оценивают производительность компьютера и влияние, оказываемое на нее изменениями компьютера в целом или его подсистем. Для выполнения этих оценок имеются два общепринятых метода: имитационное моделирование и исследование аналитической модели.

Серьезным недостатком имитационного моделирования является сокрытие характера влияния параметров рабочей нагрузки или архитектуры, действующих в качестве входных параметров модели. Моделирование дает некоторую единую меру производительности, но внутренние процессы остаются в тени^{*)}. С другой стороны, аналитические модели должны явным образом включать в себя каждый из параметров рабочей нагрузки или архитектуры. Поскольку время, требуемое для получения решения при изменении параметров, невелико, их влияние может быть оценено и осознано. Однако эти методы обычно не включают в себя параллелизм выполнения и, кроме того, зачастую приводят к значительным ошибкам. Тем не менее на протяжении этой книги для оценки параметров рабочей нагрузки и архитектуры будут использоваться аналитические модели, так как они позволяют наглядно представить влияние на производительность различных проектных решений. Заметьте, что некоторые параметры аналитической модели, например параметры функционирования памяти, были получены с помощью имитационного моделирования небольших частей системы.

2.1. Модели производительности

Каким образом проектировщик должен измерять производительность компьютера или компьютерной подсистемы при оценке различных вариантов проекта? В компьютерной технике нас обычно интересуют две

^{*)} Спорное утверждение — все зависит от детализации имитационной модели. — *Прим. ред.*

величины: время выполнения задачи и скорость ее выполнения. Например:

Задача А выполняется за время 3 мкс на задачу.

Задача А выполняется со скоростью $3,33 \times 10^5$ ее прогонов в секунду.

В случае обычной рабочей нагрузки (задача А) время и скорость выполнения являются взаимобратными величинами. Поскольку компьютер имеет тактовый генератор, управляющий всеми его функциями, для измерения времени часто используют число тактов этого генератора. Если тактовый период процессора составляет 50 нс, то мы получим:

300 тактов на выполнение задачи А.

Задача А выполняется со скоростью 0,0033 задачи на такт.

Величина CPI

Базовая временная характеристика, используемая при оценке процессоров, определяется как *число тактов на команду* (clocks per instruction, CPI). Эта величина определяет время в тактах, затрачиваемое процессором для выполнения средней команды:

$$CPI = \frac{\text{число тактов}}{\text{число выполненных команд}}$$

Величина, обратная CPI, есть *число команд на один такт* (instructions per clock, IPC). IPC определяет скорость выполнения команд и в некоторых случаях является полезной мерой производительности.

Вычисление CPI можно иллюстрировать следующим примером. Задача, требующая 1×10^6 тактов для выполнения 5×10^5 команд, имеет $CPI = 2,0$. Малые значения CPI указывают на более высокую производительность, чем большие. Для многих процессоров различные команды требуют для своего выполнения неодинаковое число тактов, поэтому CPI является усредненной характеристикой по всем выполненным командам. Поскольку различные программы используют команды в различных комбинациях, CPI сочетает весовой коэффициент использования каждого класса команд (доля от всех команд) с числом тактов на команду каждого класса, давая взвешенное среднее. К этой величине мы еще вернемся.

Повышение быстродействия

Разработчики должны оценивать влияние модификации проекта. Пусть проект А модифицируется в проект В. Будет ли проект В лучше проекта А? Ответ можно получить, используя понятие повышения

быстродействия (speedup). Обратите внимание на то, что повышение быстродействия является безразмерной величиной:

$$\text{повышение быстродействия} = \frac{\text{время A}}{\text{время B}} = \frac{\text{CPI A}}{\text{CPI B}} = \frac{\text{скорость B}}{\text{скорость A}} = \frac{\text{IPC B}}{\text{IPC A}}$$

Если проект В лучше проекта А, повышение быстродействия будет больше 1. Если проект В ухудшает производительность, повышение быстродействия будет меньше 1. Если повышение быстродействия равно 1, производительность не изменилась. Это отношение учитывает, что снижение CPI указывает на повышение производительности, так как число тактов является мерой времени, а команда — мера выполненной работы.

Процентное отношение

Нас также может интересовать разница в производительности двух систем А и В, выраженная в процентах. Будем считать, что скорость А больше скорости В и что время А меньше времени В.

$$\begin{aligned} \text{А быстрее, чем В (по скорости) на } x \% &= 100 \times \frac{\text{скорость А} - \text{скорость В}}{\text{скорость В}} \\ &= 100 \left(\frac{\text{скорость А}}{\text{скорость В}} - 1 \right) \\ &= 100 (\text{повышение быстродействия} - 1) \end{aligned}$$

$$\begin{aligned} \text{А быстрее, чем В (по времени) на } x \% &= 100 \times \frac{\text{время В} - \text{время А}}{\text{время В}} \\ &= 100 \left(1 - \frac{\text{время А}}{\text{время В}} \right) \\ &= 100 \left(1 - \frac{1}{\text{повышение быстродействия}} \right) \end{aligned}$$

Примеры

1. Процессор А выполняет данную задачу за 1 с, в то время как процессор В требует 2 с. На сколько процентов А быстрее В?

Решение

Повышение быстродействия А по отношению к В равно $2/1 = 2$. Таким образом, процессор А на $100(1 - 1/2) = 50\%$ быстрее, чем В.

2. Процессор А выполняет задачу со скоростью 1 задача в секунду, в то время как процессор В выполняет ту же задачу со скоростью 0,5 задачи в секунду. На сколько процентов А быстрее В?

 Решение

Повышение быстродействия составляет $1/0,5 = 2$. Таким образом, А на $100 \times (2 - 1) = 100\%$ быстрее В.

 Замечание

Обратите внимание на то, что одно и то же соотношение производительностей процессоров А и В дает 100% улучшения по скорости и 50% улучшения по времени.

3. Каково максимальное значение процентного отношения производительности двух процессоров по скорости и по времени?

 Решение

Если скорость процессора А бесконечна, максимальное процентное отношение производительностей двух процессоров будет $\infty\%$. При бесконечной скорости время процессора А уменьшается до нуля и максимальное процентное отношение по времени ограничено величиной 100%.

4. Процессор выполняет 100М операций в секунду (здесь М обозначает 10^6). Использование улучшенной памяти дает рост скорости до 125М команд в секунду. Каково процентное увеличение производительности?

 Решение

Повышение быстродействия, возникающее из-за смены памяти, составляет $125М/100М = 1,25$. Таким образом, процентный рост производительности составляет $100 \times (1,25 - 1) = 25\%$.

5. Тот же процессор требует 1×10^{-8} секунд на команду. Эта величина после модификации улучшается до значения 8×10^{-9} секунд на команду. Каков процентный рост производительности?

 Решение

Повышение быстродействия равно $1 \times 10^{-8} / (8 \times 10^{-9}) = 1,25$. Процентное изменение скорости выполнения составит:

$$100 \times (1,25 - 1) = 25\%.$$

Средние и взвешенные средние

Обратимся теперь к важным понятиям среднего и взвешенного среднего. Вполне естественно желание охарактеризовать производительность компьютера одним числом, которое можно сравнить с соответствующей величиной для другого компьютера или использовать как оценку полезности модификации. Вообще мы можем получить

одно число, оценив некоторое количество измерений или наблюдений и определив среднее из этих наблюдений. Применительно к компьютерам мы можем выполнить измерения или наблюдения трех видов: 1) время, требуемое для выполнения задачи, 2) скорость, с которой выполняется задача, и 3) отношение времен или скоростей, названное выше повышением быстродействия.

Для описания усредненных характеристик измерений используются три вида средних значений: арифметическое среднее для времени, гармоническое среднее для скоростей и геометрическое среднее для отношений. Наблюдения, на основании которых вычисляются эти средние, могут иметь одинаковый вес или использоваться с различными коэффициентами, т. е. быть взвешенными. Если в названии среднего не используется слово «взвешенный», то подразумевается, что наблюдения входят в среднее с одинаковым весом. Выбор того или иного среднего зависит от вида изучаемых данных и от способа их оценки [3].

Во многих последующих обсуждениях взвешенных средних под термином «вес» будет пониматься либо относительная вероятность появления события, измеряемая в единицах $x/100$, либо частота появления события также в $x/100$. У нас будут встречаться случаи, когда контекст модели заставит нас вместо термина «вес» использовать понятия частоты или вероятности появления. Как правило, эти три термина являются синонимами, причем сумма весов, вероятностей или частот равна 1.

Средние по времени

Смит утверждает [7], что «время, требуемое для выполнения заданного объема вычислений, является конечной мерой производительности компьютера». Поэтому измерение времени является фундаментальной операцией при моделировании поведения компьютера. При использовании других оценок правомерность измерений обычно может быть проверена путем сведения их к оценке времени. В последующих главах мы рассмотрим и построим модели таких мер, как время запаздывания, время обращения (доступа), время ожидания и время отклика.

Арифметическое среднее. Арифметическое среднее используется для нахождения среднего значения при временных измерениях с равными весами. Арифметическое среднее времени на одно событие находится следующим образом:

$$\text{арифметическое среднее} = \frac{1}{n} \sum_{i=1}^n T_i$$

При вычислении арифметического среднего необходимо, чтобы все данные имели одинаковый вес, а результат вычисления часто называ-

ют просто средним. Например, средняя оценка учащихся в классе x' есть сумма всех оценок x_i , деленная на число студентов n :

$$x' = \frac{1}{n} \sum_{i=1}^n x_i$$

✎ Пример

Конкретное задание выполняется на корпоративном компьютере первого числа каждого месяца. Время выполнения этого задания за период 4 месяца составило 2 ч, 2,2 ч, 1,9 ч и 2,3 ч. Каково среднее время выполнения этого задания?

✎ Решение

Все данные имеют равные веса, поскольку каждый месяц выполняется одно и то же задание. Поэтому для нахождения усредненного значения используется арифметическое среднее. Среднее значение времени выполнения задания за период 4 месяца равно

$$(2,0 + 2,2 + 1,9 + 2,3)/4 = 8,4/4 = 2,1 \text{ ч/задан.}$$

Взвешенное арифметическое среднее. Во многих случаях вычисление арифметического среднего с равными весами даст неверный результат. Следует выделять случаи, когда отдельные события происходят с разными вероятностями и требуют для выполнения различных интервалов времени. Взвешенное время на одно событие и есть взвешенное арифметическое среднее, определяемое как

$$\text{взвешенное арифметическое среднее} = \sum_{i=1}^n W_i T_i$$

Взвешенное арифметическое среднее характеризует усредненное значение времени в пересчете на единицу работы. W_i есть доля операций i среди всех операций, а T_i есть время, затраченное на эту операцию. Обратите внимание на то, что $W_1 + W_2 + \dots + W_n = 1$ и что W_i не есть доля времени, отводимого на операцию i .

✎ Пример

Набор команд процессора включает два класса команд: команды класса А требуют для выполнения 2 такта, а команды класса В — 3 такта. Из всех выполненных команд 75% принадлежали классу А и 25% — классу В. Каково значение CPI процессора?

✍ Решение

Наблюдения осуществляются во времени и являются взвешенными. Поэтому CPI процессора определяется как взвешенное арифметическое среднее:

$$CPI = W_A CPI_A = W_B CPI_B$$

$$CPI = 0,75 \times 2 + 0,25 \times 3 = 1,5 + 0,75 = 2,25 \text{ такта на команду}$$

✓ Замечание

Решая задачу, подобную приведенной выше, сложите вероятности всех событий и удостоверьтесь, что сумма равна 1; если это не так, в решение вкралась ошибка. В таких случаях полезно вместо составления уравнения использовать таблицу, подобную табл. 2.1.

Таблица 2.1. Взвешенные арифметические средние (WAM)

Тип команды	Веса	Такты	Произведение
A	0,75	2	1,50
B	<u>0,25</u>	3	<u>0,75</u>
Сумма = 1,0			WAM = 2,25 такт/команда

Средние по скорости выполнения

Производительность часто измеряется в единицах скорости. Например, автомобиль движется со скоростью 25 км/ч или компьютер работает со скоростью 100 миллионов операций в секунду. Аналогично, компьютер может выполнять 0,5 IPC команд на такт (величина, обратная CPI). Таким образом, для оценки производительности или эффективности модификаций мы можем вместо времени рассматривать скорости. Если наблюдаемыми событиями являются скорости, усредненные значения наблюдений получаются при использовании взвешенного гармонического среднего.

Гармоническое среднее. Гармоническое среднее есть усредненное значение наблюдений, выраженных в скоростях с равными весами; результат обозначает среднее число событий за единицу времени:

$$\text{гармоническое среднее} = \frac{1}{\frac{1}{n} \sum_{i=1}^n \frac{1}{R_i}} = \frac{n}{\sum_{i=1}^n \frac{1}{R_i}}$$

Гармоническое среднее определяется как величина, обратная арифметическому среднему от обратных значений, и характеризует усредненное значение единиц работы за единицу времени.

Пример

Вернемся к рассмотренному выше примеру выполнения двух задач на корпоративном компьютере. Выразим наблюдения в единицах скорости, а именно, как число заданий в час. Конкретные данные составят 0,5, 0,45, 0,53 и 0,43 задан./ч. Каково усредненное значение этих измерений в количестве заданий в час?

Решение

Усредненное значение находится с помощью гармонического среднего:

$$\begin{aligned}\text{число заданий в час} &= \frac{4}{\frac{1}{0,5} + \frac{1}{0,45} + \frac{1}{0,53} + \frac{1}{0,43}} = \frac{4}{2 + 2,2 + 1,9 + 2,3} = \\ &= \frac{4}{8,4} = 0,476\end{aligned}$$

Замечание

Обратите внимание на то, что 0,476 задан./ч есть обратная величина по отношению к найденному ранее арифметическому среднему 2,1 ч/задан.

Взвешенное гармоническое среднее. Если наблюдаемые скорости оказываются взвешенными, среднее становится взвешенным гармоническим средним:

$$\text{взвешенное гармоническое среднее} = \frac{1}{\sum_{i=1}^n \frac{W_i}{R_i}}$$

Здесь W_i есть доля полной работы (не времени), выполняемой со скоростью R_i . Результат представляет собой усредненное среднее, выраженное во взвешенных событиях в единицу времени.

Примеры

1. Программа выполняется в двух режимах: 40% программных команд выполняется со скоростью 10 миллионов команд в секунду, а 60% — со скоростью 5 миллионов команд в секунду. Какова взвешенная скорость выполнения в миллионах команд в секунду?

Решение

Поскольку наблюдения представляют собой скорости и являются взвешенными, для получения усредненного значения следует использовать взвешенное гармоническое среднее, как это показано в табл. 2.2.

Таблица 2.2. Решение с использованием взвешенного гармонического среднего (WHM)

Вес	Скорость	Частное
0,4	10 млн/с	$0,4/10 = 0,04$
0,6	5 млн/с	$0,6/5 = 0,12$
Сумма = 1,0		$WHM = 1/(0,04+0,12) =$ $= 6,25 \text{ млн команд в секунду}$

2. Решите ту же задачу, используя вместо скорости время выполнения команды.

Решение

Поскольку данные представляют собой интервалы времени и являются взвешенными, следует использовать взвешенное арифметическое среднее, как это показано в табл. 2.3.

Таблица 2.3. Решение с использованием взвешенного арифметического среднего (WAM)

Вес	Время	Произведение
0,4	1×10^{-7} секунд на команду	$0,4 \times 10^{-7}$
0,6	2×10^{-7} секунд на команду	$1,2 \times 10^{-7}$
Сумма = 1,0		$WAM = 1,6 \times 10^{-7} \text{ секунд на команду}$

✓ Замечание

Два последних результата, как и следовало ожидать, являются взаимнообратными:

$$\frac{1}{1,6 \times 10^{-7}} = 6,25 \times 10^6$$

Мы можем получить из взвешенного арифметического среднего взвешенное гармоническое среднее, взяв обратную величину от времени, что представляет собой скорость, а затем обратную величину от суммы.

Средние по отношению

Некоторые результаты наблюдений представляют собой отношения либо времен, либо скоростей. Например, мы можем выполнить некоторое количество контрольных задач на двух компьютерах и найти временное отношение для каждой задачи. В таких случаях мерой усредненного значения отношений будет геометрическое среднее или взвешенное геометрическое среднее.

Геометрическое среднее. Геометрическое среднее используется для оценки производительности в тех случаях, когда наблюдения имеют равные веса и являются отношениями времен или скоростей, а также результатом нормализации. Геометрическое среднее есть усредненное значение отношений; оно не является мерой полной производительности, как, например, полное повышение быстродействия системы:

$$\text{геометрическое среднее} = \sqrt[n]{\prod_{i=1}^n \text{отношение}_i}$$

Правильное использование геометрического среднего иллюстрируется последующими примерами.

Примеры

1. Усилитель имеет три ступени с коэффициентами усиления 2, 3 и 6 при полном усилении 36. Коэффициенты усиления ступеней увеличиваются до значений 3, 4 и 7 при полном усилении 84. Каково усредненное значение увеличения коэффициента усиления в пересчете на одну ступень?

Решение

Поскольку мы имеем дело с отношениями, и наблюдения имеют равные веса (потому что ступени усилителя соединены последовательно), усредненное значение увеличения усиления каждой ступени можно найти с помощью геометрического среднего:

$$\begin{aligned} \text{среднее увеличение усиления ступени} &= \sqrt[3]{\frac{3}{2} \times \frac{4}{3} \times \frac{7}{6}} = \\ &= \sqrt[3]{2,333} = 1,326 \end{aligned}$$

✓ Замечание

Среднее увеличение усиления ступени равно 1,326, или 32,6%. Заметьте, что геометрическое среднее не характеризует полное увеличение усиления всего усилителя в целом, которое равно $84/36 = 2,333$. Однако в силу того, что коэффициенты усиления отдельных ступеней умножаются, полное увеличение усиления всего усилителя можно найти, возведя среднее увеличение на ступень в степень, равную числу ступеней. Другими словами, полное увеличение усиления равно $1,326^3 = 2,333$. Эта величина является произведением увеличений усиления для каждой ступени.

2. Проектировщики процессора могут уменьшить тактовый период на 50% (в отношении 0,5) и уменьшить CPI на 25% (в отношении 0,75). Время выполнения команды представляет собой произведение CPI и тактового периода. Каково среднее улучшение от этих изменений в процентах?

✍ Решение

В задаче используются отношения с равными весами. Таким образом, для решения следует использовать геометрическое среднее:

$$\begin{aligned}\text{среднее уменьшение на одну модификацию} &= \sqrt{0,5 \times 0,75} = \\ &= \sqrt{0,375} = 0,612 = 38,7\%\end{aligned}$$

✓ Замечание

Заметьте, что полное улучшение производительности процессора равно $100 \times (1 - 0,375) = 62,5\%$. Среднее уменьшение на одну модификацию не равно среднему от двух уменьшений:

$$(50\% + 25\%)/2 = 37,5\%$$

Вспомним, что задачу такого рода можно проверить, перейдя к использованию времени. Будем считать исходное значение CPI равным 1, т. е. нормализуем его. После изменения нормализованное значение CPI будет $0,5 \times 0,75 = 0,375$, и процентное улучшение составит $100 \times (1 - 0,375) = 62,5\%$.

3. Два компьютера выполняют четыре цикла научной программы за число тактов, показанное в табл. 2.4. Каково усредненное значение ускорения циклов?

Таблица 2.4. Время выполнения циклов (в числе тактов)

Цикл	Компьютер А	Компьютер В	Повышение быстродействия
1	39	20	1,95
2	53	27	1,96
3	27	13	2,08
4	31	13	2,38

 Решение

Наблюдениями являются такты, т. е. мера времени, и события имеют равные веса; в этом случае усредненное значение ускорения для четырех циклов можно найти с помощью геометрического среднего. Вспомним, что повышение быстродействия есть безразмерная величина:

$$\text{среднее повышение быстродействия} = \sqrt[4]{1,95 \times 1,96 \times 2,08 \times 2,38} = \sqrt[4]{18,9} = 2,08$$

✓ **Замечание**

Может возникнуть желание определить среднее полное повышение быстродействия, усреднив ускорения всех циклов. Выполнив вычисления, получим $8,37/4 = 2,09$. Хотя это значение и близко к геометрическому среднему, однако такое вычисление некорректно.

Геометрическое среднее часто (и неверно) используется для представления производительности компьютера одним числом, как это было сделано в последнем замечании. Рассмотрим такой пример. Два компьютера, А и В, выполняют три программы за время, показанное в табл. 2.5.

Таблица 2.5. Времена выполнения программ (с)

Программа	Компьютер А	Компьютер В	Отношение А/В
1	1500	150	10
2	2300	200	11,5
3	24000	2500	9,6
Полное время	27800	2850	

Геометрическое среднее отношений составит

$$\sqrt[3]{10 \times 11,5 \times 9,6} = \sqrt[3]{1104} = 10,33$$

Является ли это число обоснованной мерой повышения быстродействия компьютера В по отношению к компьютеру А для этих трех программ? Нет, не является. 10,33 — это усредненная величина отношений.

Вспомним, что конечной мерой производительности является время выполнения. Полное время для компьютера А составит 27800 с, а для компьютера В — 2850 с; повышение быстродействия будет равно $27800/2850 = 9,75$. Может возникнуть искушение для вычисления повышения быстродействия взять арифметическое среднее трех отношений. Результат этой операции составит $10 + 11,5 + 9,6 = 31,1/3 = 10,37$. Снова результат оказался близок к значению ускорения, полученному из полного времени, однако он неверно отражает в виде одного числа относительную производительность двух процессоров.

Взвешенное геометрическое среднее. В тех случаях, когда наблюдаемые отношения являются взвешенными с учетом доли их использования, для получения усредненного значения следует использовать взвешенное геометрическое среднее:

$$\text{взвешенное геометрическое среднее} = \prod_{i=1}^n \text{отношение}^{\omega_i}$$

✎ Пример

Для приведенного выше примера примем, что для конкретной контрольной задачи цикл 1 выполняется 20 раз, цикл 2 — 30 раз, цикл 3 — 50 раз, а цикл 4 — 100 раз. Каково среднее повышение быстродействия четырех циклов?

✍ Решение

Вес цикла 1 составит $20/200 = 0,1$, цикла 2 — 0,15, цикла 3 — 0,25, а цикла 4 — 0,5. Взвешенное геометрическое среднее дает взвешенное среднее повышение быстродействия для четырех циклов:

$$\begin{aligned} \text{среднее повышение быстродействия цикла} &= \\ &= 1,95^{0,1} \times 1,96^{0,15} \times 2,08^{0,25} \times 2,38^{0,5} = 1,069 \times 1,106 \times 1,201 \times 1,543 = 2,19 \end{aligned}$$

✓ Замечание

Взвешенное среднее повышение быстродействия цикла в значительной степени определяется циклом 4, для которого характерно как максимальное повышение быстродействия (2,38), так и наибольшая доля использования (50%). Как и в других случаях, повышение быстродействия процессора, выполняющего взвешенные

циклы, можно найти с помощью взвешенных полных времен для компьютеров А и В.

$$\begin{aligned}\text{повышение быстродействия} &= \frac{\text{взвешенное время А}}{\text{взвешенное время В}} = \\ &= \frac{(0,1 \times 39) + (0,15 \times 53) + (0,25 \times 27) + (0,5 \times 31)}{(0,1 \times 20) + (0,15 \times 27) + (0,25 \times 13) + (0,5 \times 13)} = \\ &= \frac{3,9 + 7,95 + 6,75 + 15,5}{2 + 4,05 + 3,25 + 6,5} = \frac{34,1}{15,8} = 2,16\end{aligned}$$

Резюме по средним значениям

Все шесть видов средних собраны в табл. 2.6. При выборе формулы среднего для решения конкретной задачи следует определить параметр (время, скорость или отношение), а также выяснить, имеют ли наблюдения равные веса или являются взвешенными. Не забывайте, что среднее, как бы его ни вычислять, является усредненным значением интервалов времени, скоростей или отношений. Только арифметические и гармонические средние дают представление о полной производительности.

Таблица 2.6. Арифметическое, гармоническое и геометрическое средние

Среднее	Равные веса	Взвешенное	Единицы
Арифметическое	$AM = \frac{1}{n} \sum_{i=1}^n T_i$	$WAM = \frac{1}{n} \sum_{i=1}^n W_i T_i$	время/единица работы
Гармоническое	$HM = \frac{n}{\sum_{i=1}^n \frac{1}{R_i}}$	$WHM = \frac{n}{\sum_{i=1}^n \frac{W_i}{R_i}}$	единица работы/время
Геометрическое	$GM = \sqrt[n]{\prod_{i=1}^n r_i}$	$WGM = \sqrt[n]{\prod_{i=1}^n ratio_i}$	отношение

Примеры

1. На компьютере выполняются 200 заданий. Первые 100 заданий выполняются со скоростью 50 заданий в час, вторые 100 заданий – со скоростью 100 заданий в час. Какова средняя скорость выполнения 200 заданий?

Решение

Правильное решение можно найти с помощью гармонического среднего с равными весами. Действительно, исходные данные явля-

ются скоростями выполнения (число заданий в час), а объемы работы одинаковы (100 заданий), поэтому одинаковы и веса:

$$\text{средняя скорость} = \frac{2}{\frac{1}{50} + \frac{1}{100}} = \frac{2}{0,02 + 0,01} = \frac{2}{0,03} = 66,7 \text{ заданий в час}$$

✓ **Замечание**

Эта задача может быть также решена в единицах времени. Первые 100 заданий потребовали 2 ч, в то время как вторые 100 заданий отняли 1 ч. Таким образом, среднее время выполнения 200 заданий составляет 3 ч, а скорость выполнения $200/3 = 66,7$ заданий в час.

2. Компьютер выполняет задание А за 1 с, а задание В за 10 с. На каждое выполнение задания В задание А выполняется 9 раз. Какова скорость выполнения заданий компьютером в числе заданий в секунду?

 Решение

Наблюдения представляют собой единицы времени, а количество выполненных заданий должно быть взвешенным. Однако по условию требуется определить скорость выполнения. Поэтому сначала с помощью взвешенного арифметического определяется усредненное значение времени, которое затем преобразуется в скорость. Вес задания А составляет $9/10 = 0,9$, а вес задания В равен $1/10 = 0,1$. Взвешенное арифметическое среднее времени выполнения равно $0,9 \times 1 + 0,1 \times 10 = 1,9$ с, а взвешенная скорость выполнения составляет $1/1,9 = 0,526$ заданий в секунду.

✓ **Замечание**

Другой способ решения той же задачи состоит в преобразовании наблюдений в скорости выполнения и вычислении усредненного значения с помощью взвешенного гармонического среднего:

$$\text{взвешенная скорость выполнения} = \frac{1}{\frac{0,9}{1} + \frac{0,1}{0,1}} = \frac{1}{0,9 + 1} = \frac{1}{1,9} = 0,526$$

Сложный темп роста

Другим важным для компьютерных инженеров понятием является сложный темп роста. Если каждый год мы получаем процессор, который вдвое быстрее прошлогоднего, то сложный темп роста производительности составит 100% в год. Удвоение производительности каждые 2 года даст значение сложного темпа роста 41% в год.

Сложный темп роста идентичен сложным процентам по денежным вкладам. Каждый год проценты начисляются на сумму капитала и процентов прошлого года.

Формулы для сложного темпа роста:

$$\text{последнее значение} = \text{начальное значение} \left(1 + \frac{\text{темп роста}}{100} \right)^{\text{число времен. интервал.}}$$

$$\text{скорость роста} = 100 \left(\sqrt[\text{числ. времен. интервал.}]{\frac{\text{последнее значение}}{\text{начальное значение}}} - 1 \right)$$

Темп роста 10% в год и начальное значение 1 дадут через 4 года величину 1,464. Обратите внимание на то, что интервал может представлять собой любую единицу времени, совместимую с определением темпа роста.

✎ Примеры

1. В 1995 г. сервер транзакций работал со скоростью 100 транзакций в секунду. От года к году сервер модернизировался и в 2000 г. работает со скоростью 700 транзакций в секунду. Если все транзакции идентичны, каков сложный темп роста в транзакциях в секунду за этот 5-летний период?

✎ Решение

Повышение быстродействия работы составляет $700/100 = 7$. Сложный темп роста равен $100 \times (\sqrt[5]{7} - 1) = 47,5\%$ в год.

2. Какой будет производительность в транзакциях в секунду для аналогичного сервера в 2005 г., если сложный темп роста останется тем же?

✎ Решение

Темп роста этого сервера будет $700 \times 1,474^5 = 4887$ транзакций в секунду.

2.2. Закон Амдала

Закон описывает повышение быстродействия компьютера, когда он обрабатывает два класса задач; один класс может быть ускорен, но другой нет [1]. Такая ситуация часто встречается при разработке компьютерных систем. Решение задачи, подчиняющейся закону Амдала,

нормализуется по времени выполнения, характерному для системы до каких-либо модернизаций.

Предположим, что программа содержит два компонента: t_1 и t_2 . Компонент t_2 может быть ускорен. Каково полное повышение быстродействия системы?

$$\text{повышение быстродействия} = \frac{t_1 + t_2}{t_1 + (t_2/n)}$$

В некоторых случаях значения t_1 и t_2 могут быть неизвестны. Однако мы знаем долю времени выполнения, соответствующую ускоряемой или, наоборот, неускоряемой части программы. Пусть a есть неускоряемая доля времени¹⁰⁾, а доля $1 - a$ может быть ускорена в n раз:

$$t_1 + t_2 = 1,$$

$$a = \frac{t_1}{t_1 + t_2} = t_1,$$

$$1 - a = \frac{t_2}{t_1 + t_2} = t_2,$$

Таким образом, нормализованное повышение быстродействия равно

$$\frac{1}{a + \frac{1-a}{n}}$$

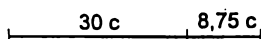
Рассмотрим два крайних случая. Если $a \rightarrow 0$ (другими словами, вся программа может быть ускорена), повышение быстродействия $\rightarrow n$, и если $n \rightarrow \infty$ (другими словами, t_2 уменьшается до 0), повышение быстродействия $\rightarrow 1/a$. Этот второй предел указывает на то обстоятельство, что если 10% времени исходной системы соответствует ее неускоряемому компоненту, то максимально возможное увеличение производительности системы составит 10.

Задачи, которые могут быть решены применением закона Амдала, легче представлять в графической форме, чем подставлять экспериментальные данные в уравнение. Эта методика проиллюстрирована в следующем примере.

✎ Пример

В результате хронометража выполняемой программы выясняется, что последовательная часть программы (компонент, который не может быть ускорен) занимает 30 с, а та часть программы, которая может

¹⁰⁾ Переменная a не есть вес работы, используемый при нахождении арифметического среднего.

**Рис. 2.1.** Закон Амдала.

быть ускорена, занимает 70 с. Вы полагаете, что, используя параллельные процессоры, можно повысить скорость выполнения этой второй части в 8 раз. Насколько повысится быстродействие всей системы?

Решение

Графическое решение этой задачи с использованием закона Амдала приведено на рис. 2.1. Отрезок времени 70 с уменьшается в 8 раз, до 8,75 с. Повышение быстродействия системы можно найти, разделив исходное время (100 с) на уменьшенное время (38,75 с). Таким образом, повышение быстродействия составит 2,58. Тот же результат может быть получен путем подстановки параметров в формулу закона Амдала. Для исходной системы $a = 3$ коэффициент улучшения $n = 8$. Повышение быстродействия определится следующим образом:

$$\text{повышение быстродействия} = \frac{1}{0,3 + (0,7/8)} = \frac{1}{0,3 + 0,0875} = \frac{1}{0,3875} = 2,58$$

Пример

Получим формулу закона Амдала для случая, когда a является долей времени, которая может быть уменьшена.

Решение

Решение можно найти, заменив $1 - a$ на a , а a на $1 - a$:

$$\begin{aligned} \text{повышение быстродействия} &= \frac{t1 + t2}{(1 - a) + (a/n)} = \\ &= \frac{1}{(1 - a) + (a/n)} \end{aligned}$$

2.3. Закон Мура

В 1965 г. Гордон Мур [5], который в то время занимал должность руководителя исследовательского отдела компании Fairchild Semiconductor, заметил, что начиная с 1959 г. плотность схем на кристалле удваивается каждый год. В более поздней работе в 1979 г. [6] он нанес на график новые данные и подтвердил, что плотность схем на кристалле по-прежнему удваивается каждый год, как это показано на рис. 2.2. Мур писал: «Я полагаю, что наклон кривой должен вскоре уменьшиться. Первые 15 лет или около того плотность схем на кристалле удваивалась каждый год, однако теперь намечается тенденция к замедлению этого роста в два раза, так что удвоение плотности будет происходить каждые два года¹¹⁾. Это наблюдение стало известно как закон Мура; оно полезно при оценках будущих возможностей микросхем.

Последние данные, полученные из разных источников (рис. 2.3), подтверждают, что наклон кривой приблизительно равен 1,4, т. е. удвоение числа транзисторов в микропроцессоре происходит каждые 2 года¹¹⁾. Закон Мура ограничивает максимальную достижимую плотность транзисторов на кристалле. Однако наряду с большими кристаллами могут выпускаться и выпускаются микросхемы меньшего размера. Обратите внимание на то, что закон Мура стал формулироваться по-другому: вместо числа транзисторов на кристалле рассматривается число транзисторов в микропроцессоре.

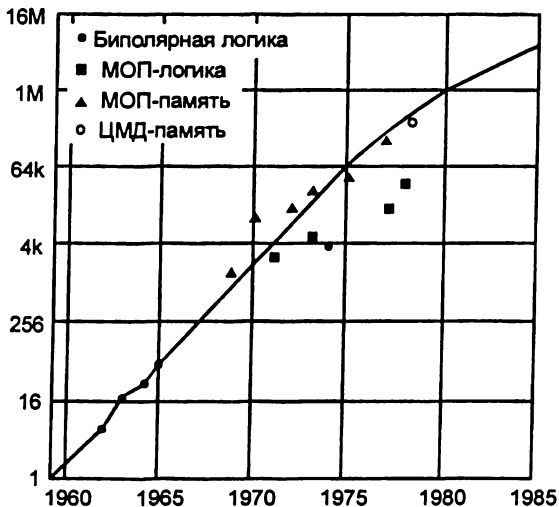


Рис. 2.2. Закон Мура (©[6]). МОП — металл-окисел-полупроводник, ЦМД — цилиндрические магнитные домены.

¹¹⁾ Еще один пример сложного темпа роста.

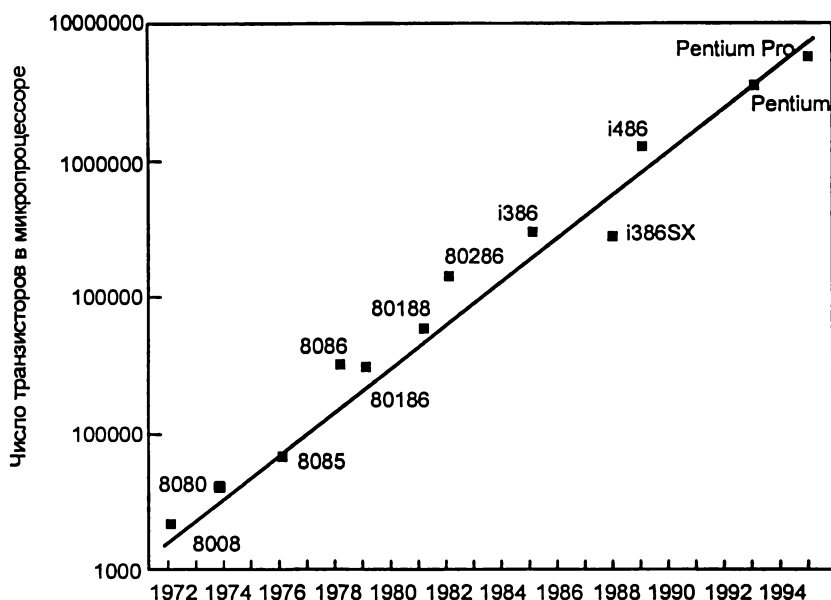


Рис. 2.3. Число транзисторов в микропроцессоре.

Этап проектирования интегральной схемы может занимать от нескольких месяцев для стандартных элементов до нескольких лет для сложных микропроцессоров. В последнем случае проектировщики должны уметь смотреть вперед, чтобы точно предсказать возможности производства проектируемого устройства. Закон Мура помогает проектировщикам получить гарантии того, что новая разработка не устареет еще до ее завершения.

Неизменность выполнения закона Мура объясняется двумя причинами. Во-первых, увеличиваются размеры кристалла по мере того, как появляется возможность наносить изображение на большую площадь; во-вторых, уменьшаются размеры элементов, что позволяет размещать больше устройств на данной площади. Увеличение площади при уменьшении размеров приводит к повышению числа элементов на кристалле. Эти улучшения являются результатом совершенствования литографических методов, применения чистых помещений, а также автоматизации всего процесса. Улучшения не возникают сами по себе; они требуют от производителей полупроводников и всего научного сообщества серьезных вложений в разработку новых методов проектирования, материалов и технологии производства. Современное предприятие по производству полупроводников стоит сегодня более 1 млрд. долларов.

Было замечено, что закон Мура выполняется не только для полупроводниковых устройств, но и для жестких дисков. На рисунке 2.4 показано, что плотность записи битов информации на дисках диамет-

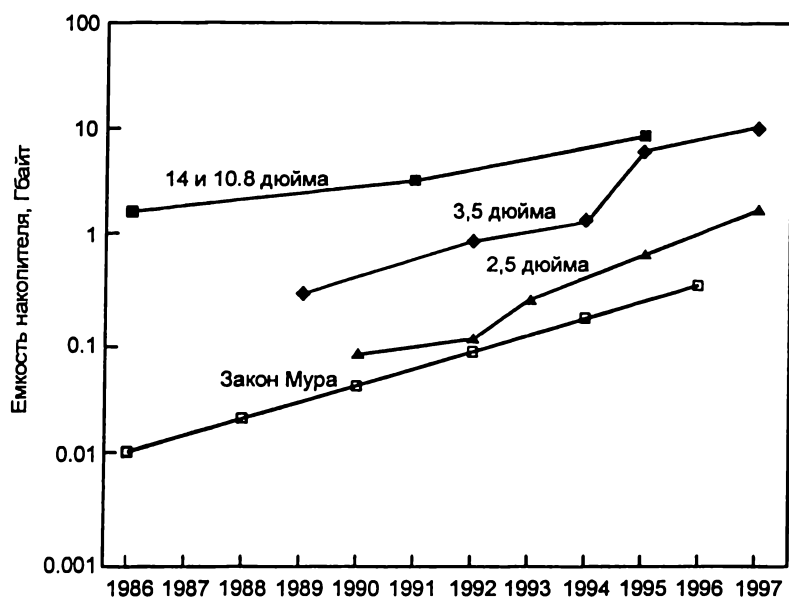


Рис. 2.4. Емкость накопителя на магнитных дисках.

ром 2,5 дюйма растет даже быстрее, чем для микросхем. Предполагается, что к 2030 г. будет достигнута плотность 10^9 Мбайт на квадратный дюйм¹²⁾. Учитывая растущие размеры программного обеспечения, такое увеличение емкости дисков окажется насущно необходимым.

Столь внушительное увеличение плотности информации на дисках является результатом развертывания широкого фронта научно-исследовательских и конструкторских работ в области технологии изготовления дисков. В частности, удалось уменьшить размер головок чтения-записи и приблизить их к поверхности диска; геометрический коэффициент бита изменился от значения 20×1 до 1×1 . В дополнение к этому улучшение электроники позволило читать меньшие по амплитуде сигналы более высокой частоты.

Примеры

1. В 1970 г. полупроводниковые пластины имели диаметр 2 дюйма. К 1997 г. в производство были внедрены 12-дюймовые пластины. Каков сложный темп роста площади пластин в год?

¹²⁾ 1 дюйм равен 2,54 см, 1 кв. дюйм равен 6,45 кв. см. — Прим. ред.

Решение

Площадь пластины в 1970 г. была 3,14 кв. дюйма, а в 1997 г. она увеличилась до 113 кв. дюйма. Рост площади составил $113/3,14 = 36$. При длительности периода 27 лет сложный темп роста составил 14% в год. Можно проверить этот результат, убедившись, что $1,14^{27} = 36$.

2. В 1970 г. геометрический размер составлял 10 мкм, а в 1997 г. он стал равен 0,6 мкм. Каков сложный темп роста плотности устройств на единице площади?

Решение

Уменьшение геометрических размеров составляет $10/0,6 = 16,6$; число устройств возрастает как квадрат этой величины. Это возрастание составит $16,6^2 = 278$ на единицу площади. Рост в 278 раз за период 27 лет дает сложный темп роста 23,2% за год. Можно проверить этот результат, убедившись, что $1,232^{27} = 280$; незначительное расхождение объясняется ошибками округления.

3. Подтверждают ли примеры 1 и 2 закон Мура?

Решение

Не непосредственно; площадь полупроводниковой пластины не равна площади кристалла. Однако исследование этого вопроса показывает, что число кристаллов на пластине остается относительно постоянным для всего обширного семейства архитектуры x86 и составляет приблизительно 80 кристаллов на пластину. Таким образом, площадь пластины является в наших расчетах допустимой заменой площади кристалла.

Произведение роста площади кристалла на рост плотности устройств составляет $36 \times 278 = 10008$. Это дает сложный темп роста 41% за 27 лет. Такой годовой сложный темп роста даст увеличение на 99% за два года; $1,41^2 = 1,99$. Этот результат весьма близок к удвоению каждые два года, что соответствует закону Мура.

2.4. Кривая обучаемости

Каждодневно мы видим примеры быстрого снижения стоимости компьютеров (процессоров и памяти). Причина этого снижения лежит в том, что производители вкладывают значительные средства в оборудование и технологию, что в результате приводит к уменьшению производственных затрат. Для многих продуктов такого рода наблюдается закономерность, названная кривой обучаемости (опытности)⁹⁾.

⁹⁾ Кривая данного вида соответствует закономерности обучаемости в психологии.— *Прим. ред.*

На рисунке 2.5 показана стоимость интегральных схем за период 1964–1966 гг. [2]. Рассматривались интегральные схемы с низким уровнем интеграции (small scale integration, SSI), содержащие от одного до трех вентилях каждая. При каждом удвоении объема выпуска цена снижалась приблизительно на 25%. Например, при суммарном объеме выпуска 5 млн. цена была 10,50 долл., а при выпуске 80 млн. — 3 долл.

Кривая обучаемости устанавливает, что производственные затраты уменьшаются на фиксированное число процентов при каждом удвоении объема производства. На рисунке 2.6 показана эта зависимость в «чистом» виде. Коэффициент обучаемости L определяется как доля затрат, остающаяся после удвоения объема. Другими словами, если стоимость производства снижается на 30% при каждом удвоении объема, коэффициент обучаемости равен 0,7. Заметьте, что затраты должны определяться в неизменной валюте, чтобы убрать из модели влияние инфляции или дефляции.

На основании рис. 2.6 можно получить, что если первый миллион устройств был выпущен при затратах 1 долл. за устройство, то второй миллион потребовал затрат лишь по 0,70 долл.; суммарный объем продукции стал равен 2 млн. При выпуске вторых 2 млн. устройств затраты составят 0,49 долл., если же будут выпущены вторые 4 млн. устройств, затраты снизятся до 0,329 долл.

Кривая обучаемости математически выражается следующей формулой:

$$\text{текущие затраты} = \text{начальные затраты} \times L^{\text{число удвоений}}$$

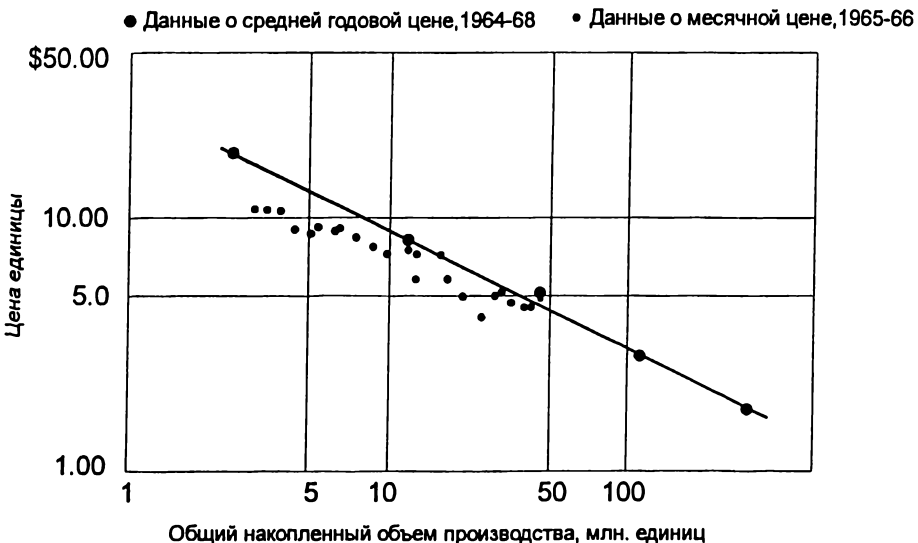


Рис. 2.5. Кривая обучаемости для интегральных схем (©[2]).

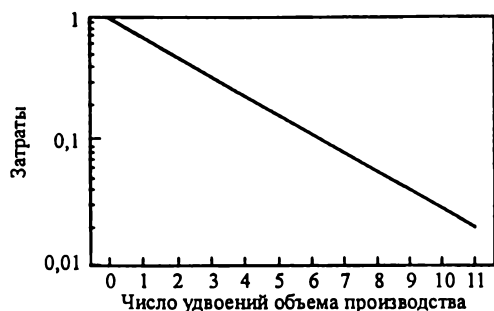


Рис. 2.6. Кривая обучаемости для $L = 0,7$.

Если начальные затраты при выпуске продукта составляли 1000 долл. за единицу, и было выпущено 1024 единицы, то произошло $\log_2 1024$, или 10, удвоений суммарного объема. Если $L = 0,9$, то затраты на последнюю единицу продукта составят $1000 \text{ долл.} \times 0,9^{10} = 348 \text{ долл.}$

Для построения кривой обучаемости необходимо, чтобы выпускаемая продукция оставалась неизменной в течение рассматриваемого периода времени. В качестве примеров таких товаров можно привести модули DRAM, вентили и шоколадки M&M. Кривую обучаемости трудно применить к таким товарам, как автомобили или персональные компьютеры (ПК), потому что для них нет нормализующей единицы. Как правило, мы не будем сегодня покупать такой же компьютер, как и два года назад. Однако для обобщенного компьютера в ценовом диапазоне до 1000 долл. кривая обучаемости будет действовать.

К сожалению, некоторые авторы ошибочно заявляют, что теория кривой обучаемости говорит о ежегодном уменьшении затрат на фиксированный процент. Такое явление действительно наблюдается для модулей DRAM, однако эта закономерность не объясняется какой-либо теорией. Известно также, что суммарный объем модулей DRAM (в битах) приблизительно удваивался ежегодно в течение последних 10 лет, что и привело к ошибочному заключению, будто кривая обучаемости показывает временную зависимость затрат.

✎ Пример

В 1990 г. модули DRAM объемом 4 Мбит стоили приблизительно 40 долл. Предположим, что 1) продажная цена есть функция от производственных затрат, 2) суммарный объем DRAM в битах удваивается каждый год, начиная с 1990 г. и 3) коэффициент обучаемости составляет 0,75. Какой будет предположительно цена 4М-бит модуля DRAM в 1999 г. и в 2005 г.?

Решение

Ожидаемая цена 4М-битового модуля DRAM в 1999 г. составит $40 \text{ долл.} \times 0,75^9 = 3,00 \text{ долл.}$ В 2005 г. ожидаемая цена будет равна $40 \text{ долл.} \times 0,75^{15} = 0,53 \text{ долл.}$

Не забывайте, что кривая обучаемости относится к производственным затратам. Время от времени продажная цена модулей DRAM будет выше или ниже предсказанной. Если запасы исчерпываются, цена товара может возрасти, давая большую прибыль, как это было, например, в 1995 г. Если, однако, запасы велики, продажная цена может упасть, что мы и наблюдали в 1996–1999 гг. Имейте также в виду, что снижение стоимости производственных затрат в соответствии с кривой обучаемости не происходит по мановению волшебной палочки. Этому должны предшествовать значительные инвестиции в научно-исследовательские и опытно-конструкторские работы, а также в новое производственное оборудование.

2.5. Закон Гроша

В 1953 г. Херб Грош [4] сформулировал правило, которое в последние годы часто использовалось некорректным образом. Он заметил, что $C = K \times \sqrt{P}$, где P — вычислительные возможности компьютера, K — константа, а C — стоимость системы.

Это правило было перевернуто и в таком виде утверждало, что вычислительные возможности возрастают как квадрат стоимости компьютера, т. е. $P = KC^2$. Другими словами, за вдвое большие деньги вы можете приобрести вчетверо большие вычислительные возможности.

На рисунке 2.7 показаны цены трех процессоров Pentium II с одинаковой конфигурацией, но с разными тактовыми частотами — 300, 333 и 350 МГц. Предполагается, что тактовая частота полноценно отражает вычислительные возможности. На том же рисунке для сравнения показан график закона Гроша, начинающийся с точки 1200 долл.

Из приведенных данных можно сделать вывод, что между реальными рыночными ценами и законом Гроша для ПК имеется лишь грубое соответствие. Однако, если вам нужно дать прогноз стоимости более быстрого компьютера, закон Гроша дает сносную оценку.

Пример

Розничная цена компьютера на 350 МГц с процессором Pentium II равна 1699 долл. Оцените стоимость модели на 500 МГц той же конфигурации через два года.

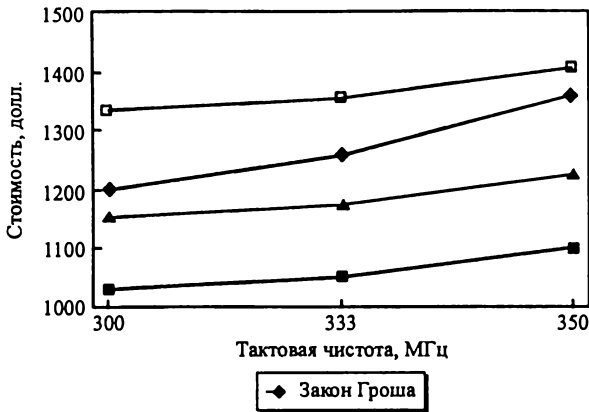


Рис. 2.7. Цены на системы Pentium II.

Решение

Здесь следует учесть две закономерности. Согласно закону Гроша цена более производительной модели составит

$$1699 \sqrt{\frac{500}{350}} = 1699 \times 1,19 = 2021 \text{ долл.}$$

Однако по мере увеличения объема производства компьютеры в течение многих лет демонстрируют подтверждение действия кривой обучаемости при $L = 0,7$. Поэтому ожидаемая цена более быстрого компьютера через 2 года составит

$$2021 \times 0,7^2 = 2021 \times 0,49 = 990 \text{ долл.}$$

Замечание

Учитывая как закон Гроша, так и кривую обучаемости, мы можем прогнозировать, что цена этого более быстрого компьютера через два года будет приблизительно на 709 долл. меньше, чем цена базовой модели. Эта оценка сделана в предположении, что память, диск и система ввода-вывода остаются прежними.

2.6. Стационарная производительность

Стационарную производительность можно измерять различными способами. Одной из наиболее распространенных единиц измерения является MIPS (millions of instructions per second, миллионы команд в

секунду). Производительность в этих единицах можно вычислить, наблюдая выполнение процессором многих команд в течение заданного интервала времени.

Пример

Конкретный процессор выполняет 2 млн. команд за 3 с. Какова производительность процессора в MIPS?

Решение

$$\begin{aligned} \text{MIPS} &= \frac{\text{число выполненных команд}}{\text{число секунд} \times 10^6} = \\ &= \frac{2 \times 10^6}{3 \times 10^6} = 0,66 \text{ MIPS} \end{aligned}$$

Производительность в MIPS лучше всего использовать при сравнении двух процессоров с одинаковыми наборами команд (например, Intel Pentium и Pentium Pro), выполняющих одну и ту же программу. При различающихся наборах команд оценка в MIPS может дать неверные результаты. Если, например, один из процессоров имеет встроенную команду умножения, а другой выполняет умножение с помощью подпрограммы, оценка в MIPS будет сомнительной, так как фактически будут выполняться различные команды. В этом случае лучше воспользоваться другим методом измерения производительности.

Этот другой метод заключается в вычислении времени выполнения задачи. Время определяется следующим образом:

$$\begin{aligned} \text{время выполнения задачи} &= \text{число выполненных команд} \times \text{CPI} \times \text{длительность такта} = \\ &= \frac{\text{число выполненных команд} \times \text{CPI}}{\text{тактовая частота}} \end{aligned}$$

Приведенная формула может использоваться в различных модификациях. Предположим, что вам требуется определить, как повлияет на производительность модификация узла выполнения команд данного процессора, скажем, Intel Pentium, при той же тактовой частоте. В этом случае единственное, что вам нужно учесть, это изменение CPI, поскольку число команд и длительность такта остаются неизменными. Вопросы определения CPI для обычных процессоров будут рассмотрены в гл. 5, а для конвейерных — в гл. 6.

При определении производительности процессора в решении научных задач используется другая мера — MFLOPS (millions of floating-point operations per second, миллионы операций с плавающей запятой в секунду). Эта мера учитывает только выполнение команд с плаваю-

щей запятой; команды загрузки, сохранения, переходов и др. не рассматриваются. Пренебречь этими командами можно потому, что они являются служебными и не дают непосредственного вклада в производительность процессора^{*)}.

🔗 Пример

Процессор выполняет 10 млн. команд с плавающей запятой и 1 млн. служебных команд за 50 мс. Какова производительность этого процессора в единицах MFLOPS?

Решение

$$\text{MFLOPS} = \frac{10 \times 10^6}{50 \times 10^{-3} \times 10^6} = 200$$

✓ Замечание

1 млн служебных команд не вносят вклад в оценку в MFLOPS. Если эти служебные команды учесть, получится производительность 220 MIPS.

2.7. Переходная характеристика

В разделе 2.5 обсуждались две меры производительности компьютера — время выполнения задачи и скорость выполнения задач. Время является полезной мерой для таких задач, как моделирование интегральных схем с помощью программы SPICE или разработка топологии микросхем. Измерения скорости удобны при оценке производительности больших машин и серверов, работающих с потоками заданий. Обратите внимание на то, что измерения и времени, и скорости дают оценки средней производительности в стационарном состоянии.

Однако во многих случаях оценки стационарной производительности компьютера оказываются относительно бесполезными. ПК и клиенты, подключенные к серверам, требуют другой меры скорости работы: времени отклика или переходной характеристики. Работая на ПК, пользователь, щелкая мышью по значку новой задачи, желает

^{*)} Имеется в виду производительность в полезных арифметических операциях, содержащихся в алгоритме решения задачи. — *Прим. ред.*

видеть результат немедленно, или, по крайней мере, с незначительной задержкой. То же самое справедливо по отношению к запросу, посланному на сервер.

В гл. 7 будет рассмотрена модель очередей, которая используется, в частности, при изучении производительности устройств ввода-вывода, когда система находится в стационарном состоянии. Эту модель называют моделью открытых систем. Для закрытых систем, таких, например, как группы терминалов, подключенных к серверу, рассматривается модель закрытых систем.

Для систем с единственным пользователем ресурса, к которым принадлежит ПК, выполняющий только одну задачу, время отклика вычисляется просто как сумма времен, требуемых для выполнения каждой задачи в ответ на соответствующий запрос. Например,

время отклика = время операционной системы + время доступа к диску + время передачи данных

Поскольку время доступа к диску обычно существенно превышает остальные составляющие времени отклика, именно оно определяет время отклика на запрос, использующий диск. По этой причине диски, используемые на однопользовательских рабочих станциях и ПК, должны иметь минимальное время доступа, допускаемое заданной стоимостью системы.

Схожая ситуация наблюдается на системах типа Всемирной паутины (WWW). Когда выставляется запрос сильно загруженному серверу, время отклика сервера составляет большую часть общего времени отклика, независимо от пропускной способности модема и сети. В другое время суток, например в 2 ч ночи, когда сервер практически свободен, основную роль в суммарном времени отклика играет пропускная способность сети. Время отклика WWW описывается открытой моделью с очередями; эта модель будет подробнее рассмотрена в разделе 7.6. Таким образом, время отклика является функцией не только используемой аппаратуры, но и операционной обстановки в момент запроса.

Литература

1. Amdahl, G. M. (1967). "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities", *Proceedings AFIPS Spring Joint Computer Conference*.
2. Conley, P. (1970). "Experience Curves as a Planning Tool", *IEEE Spectrum*, 7:6, pp. 63–68.
3. Croxton, F. E., Cowden, D. J., and Klein, S. K. (1967). *Applied General Statistics*, 3rd ed., Prentice-Hall, Englewood Cliffs, NJ.

4. Grosch, H. R. J. (1953). "High Speed Arithmetic: The Digital Computer as a Research Tool", *Journal of the Optical Society of America*, 43:4, pp. 306–310.

5. Moore, G. (1965). "Cramming More Components onto Integrated Circuits", *Electronics*, April 19, 1965, pp. 114–117.

6. Moore, G. (1979). "VLSI: Some Fundamental Challenges", *IEEE Spectrum*, 16:4, pp.30–37.

7. Smith, J.E. (1988). "Characterizing Computer Performance with a Single Number", *Communications of the ACM*, 31:10, pp. 1202–1206.

Упражнения

2.1. Сходите в библиотеку и просмотрите местные газеты. Найдите рекламируемые цены на диски и модули DRAM, используемые в ПК. Обычно там указываются розничные цены; их можно найти в деловых разделах. Выпишите таблицы этих цен за последние пять лет (старые газеты имеются на микрофильмах) и, построив кривую обучаемости, вычислите коэффициент обучаемости, полагая, что объем выпуска удваивается каждый год. Не забудьте учесть инфляцию.

2.2. Для последних вариантов семиуровневого сетевого протокола было измерено для каждого уровня отдельно повышение производительности (в единицах снижения времени), как это показано в табл. 2.7. Каково среднее значение повышения производительности для каждого уровня? Каково общее повышение производительности протокола?

2.3. На компьютере выполняются две контрольные задачи. Задача 1 выполняется со скоростью 10 MIPS, задача 2 – 15 MIPS. Какой единственной величиной можно воспользоваться, чтобы описать производительность этого процессора?

2.4. Три компьютера выполняют две контрольные задачи за интервалы времени, указанные в табл. 2.8 [7].

Таблица 2.7. Повышение производительности уровней сетевого протокола

Уровень протокола	Повышение производительности (%)
7	18
6	13
5	11
4	8
3	10
2	28
1	5

Таблица 2.8. Время выполнения контрольных задач на трех компьютерах (с)

Задача	Компьютер А	Компьютер В	Компьютер С
1	1	10	20
2	1000	100	20
Полное время	1001	110	40

Представьте свои расчеты для вычисления $x\%$, если:

- 1) А на $x\%$ быстрее В для задачи 1,
- 2) А на $x\%$ быстрее С для задачи 1,
- 3) В на $x\%$ быстрее С для задачи 1,
- 4) В на $x\%$ быстрее А для задачи 2,
- 5) С на $x\%$ быстрее А для задачи 2,
- 6) С на $x\%$ быстрее В для задачи 2,
- 7) В на $x\%$ быстрее А для обеих задач,
- 8) С на $x\%$ быстрее В для обеих задач,
- 9) С на $x\%$ быстрее А для обеих задач.

2.5. Улучшенный компьютер выполняет две программы за 40 с, что составляет 80% времени, отводимого на программу без улучшений. Когда обе программы выполняются на системе без улучшений, время выполнения составляет 200 с. Каково повышение производительности для улучшенной программы?

2.6. Ваша компания способна каждый год удваивать выпуск продукции. На сколько увеличивается каждый год суммарный объем выпущенной продукции? Обоснуйте ваш ответ.

2.7. Были разработаны два микропроцессора. Начальная стоимость изготовления первого процессора составляет 1000 долл., второго — 512 долл. Кривая обучаемости для обоих процессоров характеризуется коэффициентом 0,80. Каково отношение суммарного объема выпуска обоих процессоров, при котором стоимости их изготовления совпадают?

2.8. Формула для определения MIPS процессора имеет следующий вид:

$$\text{MIPS} = \frac{\text{такты́ая частота}}{(\text{CPI} \times 10^6)}$$

Получите эту формулу из выражения:

время выполнения задачи = число выполненных команд $\times$ CPI $\times$ длительность такта

2.9. Объясните природу терминов: арифметическое среднее, гармоническое среднее, геометрическое среднее.

2.10. Одним из наиболее часто используемых приложений геометрического среднего является вычисление сложного темпа роста. Получите формулу для этой величины, данной в разделе 2.1, из геометрического среднего.

ГЛАВА 3

ПРОЕКТИРОВАНИЕ НАБОРА ПРИКЛАДНЫХ КОМАНД

3.0. Введение

В течение последних четырех десятилетий эволюции компьютеров значительное внимание уделялось проектированию системы команд, или архитектуры набора команд (ISA), известной также под названием машинного языка. На начальном этапе, еще со времен фон Неймана, проектировщики ограничивались, главным образом, разработкой команд, которые поддерживали пользовательские, или прикладные программы. Три усовершенствования компьютеров заставили проектировщиков обратить внимание на команды для поддержки операционных систем, систем виртуальной памяти и мультипроцессирования. Все эти вопросы будут рассмотрены в гл. 8. Каждое из перечисленных усовершенствований рассматривалось как невидимое для прикладного программиста. Ради эффективности и логически правильного использования эти команды обычно выполняются в защищенном режиме, недоступном пользователю.

Поскольку компьютеры являются машинами общего назначения, в них с помощью программирования можно предусмотреть функции, отсутствующие в системе команд. В качестве примера можно привести подпрограммы, обеспечивающие операции с плавающей запятой и типами данных. Функции и типы данных, предусмотренные в системе команд, называют архитектурными^{*)}, а функции и типы данных, введенные с помощью программирования — программными.

Наборы команд ранних компьютеров разрабатывались в условиях крайне скудных машинных ресурсов. Память была невелика, что требовало минимального размера командного слова. Ресурсы для выполнения операций тоже были незначительны, в результате чего создавались примитивные системы команд, которые сегодня можно встретить в RISC-компьютерах с ограниченным набором команд. Фон-неймановская система команд содержала всего лишь 21 команду. Адресация была только прямой, и модификация адреса, например индексная, могла выполняться только с помощью команд, воздействующих на саму программу (сагомодифицирующийся код).

^{*)} В отечественной литературе используется термин «аппаратные» (аппаратные функции). — *Прим. ред.*

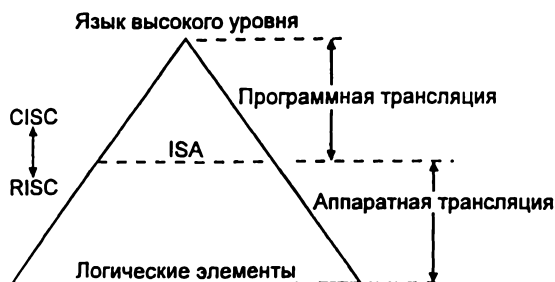


Рис. 3.1. Лексический уровень ISA.

Блау и Брукс отмечают, что различие между машинным и естественным языками заключается в том, что машинный язык является дорогостоящим. Они отмечают четыре аспекта стоимости, которые влияют на разработку системы команд:

1. Желательно, чтобы интерпретация выполнялась быстро, а интерпретаторы были простыми, поэтому машинные языки весьма стеснены в средствах и используют минимум конструкций.

2. Стоимость применительно к пространству и времени требует компактности представления, что отражается на битовом распределении в представлении и на трафике битов при использовании этого представления.

3. Согласно фундаментальным технологическим принципам информации проще всего представлять с помощью устройств с двумя состояниями, поэтому большинство машинных языков используют только два символа — 0 и 1.

4. Для упрощения интерпретации предложения языка строятся однозначными, не зависящими от контекста, и даже семантическая интерпретация может иметь лишь ограниченную контекстную зависимость [4].

3.1. Лексический уровень

Важнейшим этапом проектирования системы команд (ISA) является определение лексического уровня машинного языка. Архитектуру с высоким лексическим уровнем обычно называют сложным набором команд (CISC), архитектуру с более низким лексическим уровнем — ограниченным набором команд (RISC). Эти понятия проиллюстрированы на рис. 3.1.

Программа, написанная на языке высокого уровня, например на Си, компилируется в язык команд исполняющей машины. Программа на языке команд затем интерпретируется аппаратурой компьютера

путем включения и выключения логических элементов процессора и памяти. Несколько строк Си-программы могут вызвать действия, выполняемые миллионами логических элементов, что существенно увеличивает информационный объем программы.

CISC-машины усложняют команды, уменьшая семантический интервал, обслуживаемый компилятором, и увеличивая семантический интервал, обслуживаемый аппаратурой. Были построены экспериментальные компьютеры, выполняющие программы на языке высокого уровня непосредственно, без программной поддержки. С другой стороны, RISC-архитектуры увеличивают программный семантический интервал и уменьшают семантический интервал, обслуживаемый аппаратурой. Доводы в пользу RISC-подхода сводятся к тому, что при более простых командах система будет работать быстрее. Однако при использовании современных микропроцессоров этот довод оказывается не бесспорным. Крайний случай RISC-архитектуры можно найти в работе Вандерполя, который предложил набор команд, состоящий из единственной команды! Мы еще вернемся к архитектурным стилям RISC и CISC в разделе 3.3.

3.2. Архитектура набора команд (система команд)

В архитектуре набора команд (ISA) определяются три понятия: операции, типы данных и адреса. На рисунке 3.2. показаны эти компоненты вместе с существующими между ними связями. Связи описывают взаимозависимость компонентов. Например, если в системе команд предусмотрены типы данных с плавающей запятой, то в ней должны быть и операции с плавающей запятой (и наоборот). Если в системе команд имеются целые одинарной и двойной точности, то должен быть предусмотрен такой механизм адресации, который обеспечивал бы правильное обращение к этим данным. Если при выполнении операции операнды извлекаются из памяти или регистров, в адресном поле команды должен содержаться соответствующий адрес.

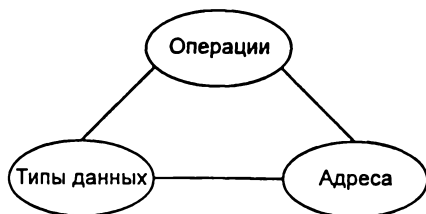


Рис. 3.2. Компоненты ISA.

В некоторых разработках жесткая привязка адресов к типам данных обеспечивается аппаратными средствами, в то время как в других привязка выполняется программно. Например, некоторые процессоры предусматривают команды типа AddFloat (Сложить числа с плавающей запятой) или AddInteger (Сложить целые числа). В этом случае операнды в памяти привязываются к операции с помощью адресов, указываемых в команде. Другими словами, операнды разных типов (с плавающей запятой и целочисленные) хранятся в различных областях памяти и идентифицируются по их адресам. В других случаях операции типа сложения и вычитания являются обобщенными. Типы данных идентифицируются по дополнительным признакам операндов или путем привязки к адресам. Определение взаимосвязей между всеми тремя компонентами является важнейшей задачей проектировщика системы команд.

Компоненты системы команд, показанные на рис. 3.2, входят в состав команды, размещаясь в ней в соответствующих битовых полях. В общем случае команда содержит два типа полей: поле операции и поле (или поля) адреса.

Код операции	Адрес(a)
--------------	----------

Во многих ISA число битов в поле операции n удовлетворяет следующему соотношению: $n \geq \log_2(\text{число операций})$.

Например, 8 бит позволяют однозначно определить до 256 операций или комбинаций операция — тип данного, в зависимости от того, каким образом операции связываются с типами данных. Адреса будут подробнее рассмотрены в разделе 3.2.1, но так или иначе число полей и их размеры являются функциями от метода адресации операндов в памяти или регистрах.

При выполнении любой команды получается непосредственный результат, ради которого она выполняется, но может также присутствовать и побочный результат, или эффект. Побочным называется результат, который неукоснительно получается помимо непосредственного. Пример такого побочного результата можно найти в командах, которые после выполнения, скажем, сложения, устанавливают некоторый бит индикации, если результат сложения равен нулю или отрицателен¹⁴⁾. В общем случае побочные результаты используются для передачи информации от выполняемой команды к следующим. Этот побочный результат может выступать в качестве явного операнда следующей команды. Для того чтобы между разработчиком системы команд и программистом не возникало недопонимание, все побочные результаты должны быть определены в ISA.

¹⁴⁾ Биты индикации, или флаги, фиксируют результат сравнения. Индикаторам будет посвящен раздел 3.2.3.2.



Рис. 3.3. Программный счетчик.

Для того чтобы представить себе, как работает компьютер с хранимой программой, необходимо разобраться в назначении программного счетчика (рис. 3.3). Этот счетчик предоставляет механизм индексации программы, хранимой в программной памяти в виде последовательного списка команд. Программный счетчик, называемый также в некоторых ISA счетчиком адресов или указателем команд, указывает на первую команду из списка, которая должна быть выполнена. В процессе выполнения первой команды содержимое программного счетчика увеличивается на 1, после чего он указывает на вторую команду, затем на третью и т. д. Последовательный доступ к командам списка может быть нарушен с помощью команд условных или безусловных переходов.

Не все компьютеры используют программный счетчик; так, некоторые ранние компьютеры хранили программы в виде связного списка. В этих компьютерах каждая команда содержала в себе адрес следующей. В те времена память не имела средств произвольного доступа, как сейчас, обеспечивая только последовательное обращение к ячейкам.

🔗 Пример

Является ли счетчик частью состояния процесса или состояния процессора?

✍ Решение

Можно указать ряд доводов в пользу того, что программный счетчик является частью состояния процессора: 1) программный счетчик не используется явным образом в программах на языках высокого уровня, 2) программный счетчик переносит информацию от каждой команды к следующей и 3) содержимое программного счетчика сохраняется вместе с остальной информацией о состоянии процессора при переключении контекстов.

3.2.1. Адреса

Начнем рассмотрение трех компонентов системы команд (ISA) с обсуждения концепции адресов. Организация памяти и процедур, с помощью которых процессор получает операнды и выдает результат, является важнейшим элементом ISA. Система команд поддерживает один или оба типа операций, перечисленных в табл. 3.1.

Таблица 3.1. Требуемые адреса

Тип операции	Источники	Приемник	Пример
Двуместная	2	1	Сложение, вычитание, умножение
Одноместная	1	1	Квадратный корень

Таким образом, команда может предоставить два или три адреса, как это требуется для выполнения данной операции. Однако имейте в виду, что адреса могут быть явными или неявными. Кроме того, операнд может быть включен в код команды (он в этом случае называется непосредственным или литеральным) и не требовать адресного поля, хотя и должен быть тем или иным способом привязан к операции.

Пример

Рассмотрим операцию сдвига влево, выполняемую над целым числом. Является ли эта операция одно- или двуместной?

Решение

Все зависит от вашей точки зрения. Поскольку сдвиг влево эквивалентен умножению на 2, операция двуместна. Один операнд адресуется явным образом, а второй операнд включен в код операции. Приемник совпадает с источником; таким образом, команда требует указания единственного адреса.

С другой стороны, операция может рассматриваться как одноместная с одним источником и одним приемником. Приемник совпадает с источником, отчего и требуется лишь один адрес.

Память современных компьютеров имеет иерархическую структуру. Другими словами, имеются два уровня памяти, определенные на уровне ISA. Первый уровень состоит из регистров различной организации; второй уровень называют основной памятью¹⁵⁾. Уровень регистров реализуется в логических узлах процессора, в то время как основная память реализуется с помощью более медленной и дешевой

¹⁵⁾ Начиная с гл. 4 основная память будет называться реальной.

технологии, например в виде МОП-модулей DRAM. Иерархическая память была впервые описана фон Нейманом [16].

Перечислим основные виды организации источников операндов и приемников результата.

1. Источники и приемники хранятся в основной памяти; адреса в командах указывают на основную память. Такой способ адресации называют «память — память».

2. Источники и приемники хранятся в регистровой памяти; адреса в командах указывают на эти регистры. Такую организацию называют загрузкой — запоминанием (load/store)^{*)} или регистровым файлом. Для перемещения данных между памятью и регистровым файлом требуются специальные команды загрузки и запоминания.

3. Одним из источников и приемником является регистр, называемый аккумулятором. В качестве второго источника выступает память; см. рис. 1.4.

4. И источники, и приемники хранятся в локальном стеке. Для перемещения данных между основной памятью и стеком требуются специальные команды загрузки и запоминания, обычно называемые push (проталкивание) и pop (выталкивание).

Обсудим теперь более детально перечисленные выше возможности адресации памяти и регистров. Ниже для каждого вида команды приведены сведения об организации памяти с адресами, а также символическое описание выполняемой операции.

Для каждого способа организации памяти приводится формат команды и его код операции. A_i представляет собой поле прямого адреса памяти, а R_i — поле прямого адреса регистра; Коп обозначает код операции.

- Три адреса, память — память:

Коп	A1	A2	A3
-----	----	----	----

Каждая команда содержит три адреса, два для источников и один для приемника. Адреса указывают на память и могут быть либо полными адресами памяти, либо укороченными смещениями, добавляемыми к содержимому базового регистра длиной в слово:

$$M(A3) \leftarrow (M(A1)) \text{ операция } (M(A2))^{16)}$$

^{*)} В отечественной литературе чаще используется термин «считывание — запись». — *Прим. ред.*

¹⁶⁾ Если имя поля заключено в скобки, как в случае $(A1)$, имеется в виду содержимое поля $A1$. Так, $(M(A1))$ обозначает содержимое ячейки памяти, адресуемой с помощью содержимого поля $A1$.

- Два адреса, память — память:

Коп	A1	A2
-----	----	----

Каждая команда содержит два адреса для двух источников. Приемником является один из источников. Таким образом, один из операндов-источников заменяется результатом. Адреса указывают на память и могут быть либо полными адресами памяти, либо укороченными смещениями, добавляемыми к содержимому базового регистра длиной в слово:

$M(A1) \leftarrow (M(A1))$ операция $(M(A2))$

- Три адреса, регистр — регистр:

Коп	R1	R2	R3
-----	----	----	----

Для архитектур с регистровым файлом и загрузкой — запоминанием и источники, и приемник находятся в регистровом файле. Адреса указывают на регистровый файл и содержат столько битов, сколько требуется для адресации регистров (обычно 4, 5 или 6 бит):

$R(R3) \leftarrow (R(R1))$ операция $(R(R2))$

Из-за того что команды адресуют только регистровый файл, в этой архитектуре для пересылки операндов и результата в память и из нее должны быть предусмотрены команды загрузки и запоминания:

Коп	A	R
-----	---	---

Загрузка: $R(R) \leftarrow (M(A))$

Запоминание: $M(A) \leftarrow (R(R))$

Обратите внимание на то, что команды загрузки и записи имеют по два адреса: адрес памяти и адрес регистра.

- Два адреса, регистр — регистр:

Коп	R1	R2
-----	----	----

Для архитектур с регистровым файлом и источники, и приемник находятся в регистровом файле. Результат записывается на место одного из операндов-источников. Адреса указывают на регистровый

файл и содержат столько битов, сколько требуется для адресации регистров (обычно 4, 5 или 6 бит):

$R(R1) \leftarrow (R(R1))$ операция $(R(R2))$

Из-за того что команды адресуют только регистровый файл, в этой архитектуре для пересылки операндов и результата в память и из нее должны быть предусмотрены команды загрузки и запоминания:

Коп	A	R
-----	---	---

Загрузка: $R(R) \leftarrow (M(A))$

Запоминание: $M(A) \leftarrow (R(R))$

- Один адрес, аккумулятор:

Коп	A
-----	---

Число адресов может быть еще уменьшено, если один из операндов адресуется неявно, причем результат возвращается на то же место. Неявно адресуемый регистр используется как аккумулятор:

Асс $\leftarrow$ (Асс) операция $(M(A))$

Операнд извлекается из памяти; выполняемая операция использует в качестве второго операнда содержимое аккумулятора, и результат возвращается в аккумулятор, как это было показано на рис. 1.4. Аккумулятор может быть очищен с помощью команды Clear Асс или специальной командой, которая очищает аккумулятор и загружает операнд из памяти.

- Отсутствие адреса, или стек:

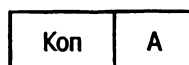
Коп

Дальнейшего уменьшения числа адресов можно достичь, если в качестве внутреннего регистра использовать стек. Все адреса определяются неявно. Для двуместной операции оба операнда-источника извлекаются из вершины стека (top of stack, TOS) и следующей за ней ячейки TOS-1, которую иногда называют следующей вершиной; результат возвращается в стек. Команда содержит только код операции, которая выполняется следующим образом:

$TOS \leftarrow (TOS)$ операция $(TOS-1)$

В зависимости от реализации ISA ячейка TOS-1 может освобождаться или нет. Как и в случае архитектуры с регистровым файлом,

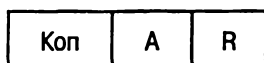
должны быть предусмотрены команды для перемещения данных в стек и из стека. Команда загрузки называется *push* (проталкивание, запись в стек), команда запоминания — *pop* (выталкивание, выборка из стека). Явно указывается только адрес памяти; адрес TOS неявно указан в коде команды:



push: $TOS \leftarrow (M(A))$ (стек проталкивается, чтобы освободить TOS, куда поступает операнд)

pop: $M(A) \leftarrow (TOS)$ (стек выталкивается, чтобы заполнить TOS из TOS-1)

- Множественный аккумулятор:



$R(R) \leftarrow (R(R))$ операция $(M(A))$

Архитектура множественного аккумулятора использует один из регистров регистрового файла в качестве аккумулятора. Поскольку в команду входят адреса и памяти, и регистра, не требуется специальных команд загрузки и запоминания.

Различные компоновки команд (и локальной памяти) никогда не используются «в чистом виде», разве что только в RISC-архитектурах. CISC-архитектуры включают в себя различные варианты адресации. Так, Intel Pentium поддерживает регистровый файл с загрузкой-запоминанием для целых чисел, аккумулятор для целых чисел и стек для чисел с плавающей запятой. Комбинация архитектур *аккумулятор, регистр — регистр и множественный аккумулятор* используется в IBM S360¹⁷⁾.

Битовое распределение

Обратите внимание на то, что все описанные выше регистровые организации имеют различное число битов в формате команды. На ранних этапах производства компьютеров важнейшим условием при разработке системы команд была минимизация числа битов, требуемых для представления программы. Число используемых битов иногда называют *битовым распределением* (бюджетом битов — *bit budget*)¹⁸⁾.

¹⁷⁾ Обозначение S360 используется здесь для указания на все члены этого семейства ISA: S360, S370 и S390. В тех случаях, когда речь будет идти о конкретном члене семейства, будет использоваться детализированное обозначение.

¹⁸⁾ Может измеряться в битах, байтах, словах или командах. Здесь подходит любая разумная единица измерения.

Для иллюстрации того, как вычисляются размеры в битах, предположим, что код операции содержит P бит, адрес памяти — A бит, а адрес регистра — R бит. В таблице 3.2 приведены биты, требуемые для каждого формата команды.

Таблица 3.2. Биты форматов команд

Тип архитектуры	Биты
Три адреса, память — память	$P+3A$
Два адреса, память — память	$P+2A$
Три адреса, регистр — регистр	$P+3R$
загрузка — запоминание	$P+A+R$
Два адреса, регистр — регистр	$P+2R$
загрузка — запоминание	$P+A+R$
Один адрес, аккумулятор	$P+A$
Ноль адресов, стек	P
Проталкивание-выталкивание	$P+A$
Множественный аккумулятор	$P+A+R$

Выбор способов адресации влияет на битовое распределение всей программы. Рассмотрим простое предложение назначения

$$A = B + C$$

Значения B и C находятся в памяти, и результат A возвращается в память. Ни один из источников не изменяется. Результаты анализа размера в битах для этого предложения приведены в табл. 3.3.

Таблица 3.3. Бюджет битов команд для предложения $A = B + C$

Система команд	Программа	Бюджет битов
Три адреса	Сложение: $M(A) \leftarrow M(B) + M(C)$	$P + 3A$
Память — память		
Два адреса	Пересылка: $M(Temp1) \leftarrow M(C)$	$3P + 6A$
Память — память	Сложение: $M(Temp1) \leftarrow M(Temp1) + M(B)$	
	Пересылка: $M(A) \leftarrow (Temp1)$	
Три адреса	Загрузка: $R(1) \leftarrow M(B)$	$4P + 6R + 3A$
Регистр — регистр	Загрузка: $R(2) \leftarrow M(C)$	
	Сложение: $R(3) \leftarrow R(1) + R(2)$	
	Запоминание: $M(A) \leftarrow R(3)$	
Два адреса	Загрузка: $R(1) \leftarrow M(B)$	$4P + 5R + 3A$
Регистр — регистр	Загрузка: $R(2) \leftarrow M(C)$	
	Сложение: $R(1) \leftarrow R(1) + R(2)$	
	Запоминание: $M(A) \leftarrow R(1)$	

Таблица 3.3. Продолжение

Система команд	Программа	Бюджет битов
Один адрес	Очистка и сложение: $\text{Acc} \leftarrow M(B)$	$3P + 3A$
Аккумулятор	Сложение: $\text{Acc} \leftarrow \text{Acc} + M(C)$ Запоминание: $M(A) \leftarrow \text{Acc}$	
Ноль адресов	Проталкивание: $\text{TOS} \leftarrow M(B)$	$4P + 3A$
Стек	Проталкивание: $\text{TOS} \leftarrow M(C)$ Сложение: $\text{TOS} \leftarrow \text{TOS} + \text{TOS} - 1$ Выталкивание: $M(A) \leftarrow \text{TOS}$	

Перед тем как перейти к дальнейшему обсуждению способов адресации, коснемся другого аспекта организации памяти. Память представляет собой линейный массив адресуемых элементов (addressable units, AU). Некоторые виды запоминающих устройств имеют в качестве AU байты, другие — слова. Если слово состоит из 4 байт, при том же числе битов адреса в команде возможна адресация вчетверо большего объема памяти.

С другой стороны, регистры процессора обычно согласуются по размеру с длиной основного типа данных. Так, если основным типом целого числа занимает слово, то и регистровый файл состоит из слов. Если при этом процессор содержит отдельные регистры для операций с плавающей запятой, как это реализовано в IBM S360, Intel x86 и PowerPC, регистры плавающей запятой обычно имеют длину 8 байт.

Переменная длина команд приводит к проблеме выравнивания команд в памяти. Рассмотрим память, состоящую из байтов. Данные, состоящие из 2 байт, должны вынужденно начинаться в четных байтах (0, 2, 4, ...). Данные, состоящие из 4 байт, должны начинаться в байтах, номера которых кратны 4: 0, 4, 8, 12 и т. д. Выравнивание должно привязываться к типу данного, требуемого для данного кода операции. Например, операции над числами с плавающей запятой двойной точности будут игнорировать три младших бита адреса. Взаимоотношение между типом организации памяти (AU) и числом игнорируемых битов в адресах (обозначенных крестиком) показано на рис. 3.4.

Адресные биты				Тип AU			
Старшие биты				Младшие биты			
							Бит
							X
							1/2 слова
					X	X	Слово
				X	X	X	Двойное слово
		X	X	X	X	X	Четверное слово

Рис. 3.4. Привязка байтового адреса к типу памяти.

Порядок байтов

Перейдем к неприятному и запутанному вопросу, касающемуся процессоров, которые используют память с байтовой организацией. Когда многобайтовое слово пересылается из одного места в другое между процессором, памятью и устройствами ввода-вывода, то какой байт пересылается первым, самый старший или самый младший? Различные производители в течение последних десятилетий предлагали разные стандарты для расположения и числа байтов в многобайтовых целых и дробных (с плавающей запятой) типах данных. Расположение само по себе не создает проблемы; сложности возникают только при пересылках между регистрами (как на том же самом компьютере, так и на разных компьютерах). Какой байт пересылается первым, и как адресуются в памяти младшие и старшие байты? Эту проблему, вспомнив «Путешествия Гулливера», назвали «проблемой тупоконечников и остроконечников» [9, 14]. Для 16-битовых (2-байтовых) целых со знаком существуют два способа назначения байтов памяти, как это показано на рис. 3.5.

Формат остроконечников: младший байт назначается меньшему адресу памяти. Младший байт читается из памяти первым.

Формат тупоконечников: старший байт назначается меньшему адресу памяти. Старший байт читается из памяти первым.

В дополнение к расположению байтов в формате остроконечников для 2-байтовых переменных бит с номером 0 является самым младшим битом слова, а бит с номером 15 — знаковым битом слова. В формате тупоконечников назначение битов обратное: бит с номером 0 является знаковым битом слова, а бит с номером 15 — самым старшим битом слова. Дополнительные трудности возникают, если производители назначают битам номера 1...16, а не 0...15.

При использовании формата остроконечников загрузка 2-байтового слова из памяти в регистр осуществляется в следующем порядке: первый байт слова помещается в правый байт регистра, а второй

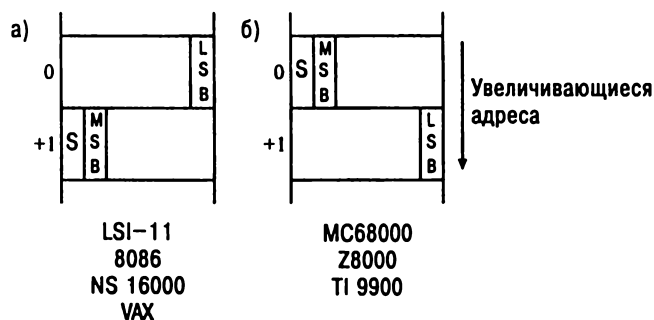


Рис. 3.5. Назначения байтов в целых числах по методу остро- (а) и тупоконечников (б) (© [14]). LSB — младший бит слова, MSB — старший бит слова, S — знаковый бит.

байт слова — в левый байт регистра. В формате тупоконечников загрузка осуществляется в обратном порядке. Примеры использования описанных форматов различными производителями:

- формат остроконечников: PDP-11, VAX, x86, PowerPC (поддерживается);
- формат тупоконечников: IBMS360, MC680x0, PowerPC (по умолчанию).

Режимы адресации памяти

Для фон-неймановской архитектуры характерен единственный режим адресации памяти: адрес в команде непосредственно указывает на ячейку памяти. Очень быстро выяснилось, что этот примитивный способ адресации недостаточен, так как не обеспечивает обращение к элементам сложных структур данных типа массивов по их индексам. При отсутствии аппаратной поддержки индексной адресации слишком большая доля ресурсов процессора требовалась для формирования адреса. Как было показано в табл. 1.6, для образования необходимых адресов приходилось выполнять целую подпрограмму с самомодифицирующимся кодом, содержащую от 8 до 16 команд.

Адрес, получаемый в результате выполнения некоторых операций над адресом, называется эффективным адресом (ЕА)^{*)}. В машинах IBM S370 и S390 используется термин «логический адрес памяти», а в компьютерах Intel Pentium Pro — «смещение».

Необходимым расширением архитектуры команд была признана индексация. В начале 1950-х годов в Манчестерском университете к адресной шине был подсоединен регистр, хранящий целое число, добавляемое к адресу. После каждой операции адресации содержимое регистра увеличивалось на 1. С течением времени к системе команд были добавлены и другие режимы адресации. Некоторые из этих режимов приведены в табл. 3.4.

Процессоры SPARC компании Sun Microsystems обладают любопытным режимом адресации: $EA = A \times B + C$. Этот режим образования эффективного адреса используется для обращения к элементам массивов и матриц. Рассмотрим, например, матрицу $n \times m$ (рис. 3.6) с $m=4$ и $n=5$. Матрица хранится в памяти с разверткой сначала по столбцам, затем по строкам. Ячейка памяти 1 хранит первый (1,1) элемент матрицы. Если программе требуется элемент (i,j) матрицы, эффективный адрес ЕА этого элемента вычисляется по формуле

$$\begin{aligned} EA(i,j) &= \text{адрес}(1,1) + (i-1)n + j - 1 = \\ &= \text{адрес}(1,1) + in - n + j - 1 = \\ &= \text{адрес}(1,1) - n - 1 + in + j \end{aligned}$$

^{*)} Используются также термины «исполнительный адрес», «модифицированный адрес». — *Прим. ред.*

Таблица 3.4. Режимы адресации

Режим	Эффективный адрес операнда
Прямая	Адрес в команде является эффективным адресом.
Индексная	К прямому адресу прибавляется значение индекса.
База+смещение	Адрес в команде является смещением, которое прибавляется к полноразмерному базовому регистру.
База+смещение+индекс	Адрес в команде является смещением, которое прибавляется к полноразмерному базовому регистру и полноразмерному индексному регистру.
Автоинкремент-декремент	В качестве индексного регистра(ов) используется специальный регистр или регистр из регистрового файла. Регистр автоматически модифицируется в соответствии с одним из следующих режимов: прединкремент (перед выполнением операции), постинкремент (после выполнения операции), преддекремент (перед выполнением операции), постдекремент (после выполнения операции).
Косвенная	Эффективный адрес определяется по любому режиму, извлекается операнд, который служит эффективным адресом. Этот режим адресации может быть одноуровневым или многоуровневым.
Непосредственная, или литеральная	Операнды входят в код команды; адреса не требуются.
Регистровая	Операнды находятся в регистровом файле.
Косвенная по регистру	Значение в адресуемом регистре общего назначения является эффективным адресом.

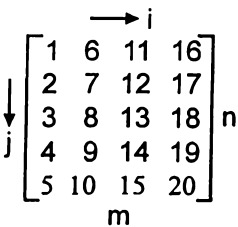


Рис. 3.6. Адресация матрицы.

Формула для эффективного адреса ЕА делится на две части. Часть [адрес (1,0) – $n - 1$] вычисляется компилятором, который знает значение n и адрес первого элемента. Результат этой операции помещается в базовый регистр. Вторая часть, $in + j$, вычисляется во время выполнения с использованием режима адресации $EA = A \times B + C$. Теперь для формирования эффективного адреса элемента (i, j) требуются две команды. Если предположить, что «компиляторная» часть ЕА была вычислена и помещена в базовый регистр, мы получим следующую процедуру:

Компилятор: база = адрес (1,1) – $n - 1$

1-я команда: база' = база + j

2-я команда: ЕА = $i \times n$ + база'

✎ Пример

Матрица размером $m = 4$ и $n = 5$ хранится в памяти, начиная с ячейки 100. Каков эффективный адрес элемента с $i = 3$ и $j = 3$?

✎ Решение

Компилятор вычисляет значение $100 - 5 - 1 = 94$ и помещает его в базовый регистр. Во время выполнения эффективный адрес вычисляется с помощью следующих двух команд:

1. $94 + 3 = 97$

2. $3 \times 5 + 97 = 112$

3.2.2. Операции

Операция, задаваемая в каждой команде в виде кода операции, определяет, что именно требуется выполнить. Для прикладных программ предназначены три группы операций:

- арифметические и логические;
- пересылки данных между ячейками памяти или регистрами;
- управляющие команды типа условных и безусловных переходов, вызовов подпрограмм и возвратов.

Арифметические и логические команды, а также команды управления выполняют операции, обеспечивающие основное назначение процессора; именно эти команды выполняют полезную работу, в то время как команды пересылки можно отнести к издержкам. Программа, написанная на языке высокого уровня, не содержит явных опера-

торов пересылки; команды пересылки добавляются компилятором в соответствии с организацией памяти процессора.

Основным соображением при выборе операций для включения их в систему команд (ISA) является частота их использования. Если операция представляется не нужной в силу низкой частоты ее использования или слишком высокой стоимости ее реализации, в ISA должны быть предусмотрены примитивные операции, которые позволили бы синтезировать эту составную операцию. Если, например, в системе команд нет команды умножения, то вместо нее должны быть включены примитивные команды сдвигов и проверок. Разработчик ISA должен быть уверен, что сложные функции, например трансцендентные, могут быть запрограммированы при разумной эффективности их выполнения.

Арифметические и логические команды

Эти команды образуют ядро системы команд, как оно видится пользователю. Арифметические команды сложения, вычитания, умножения и деления обычно реализуются аппаратно. При использовании сложения и умножения результат может выйти за диапазон представления. Разработчик обязан предусмотреть для таких случаев специальные меры предосторожности. Например, два 16-битовых целых при умножении дают 32-битовый результат. Если число дробное, в результате отбрасываются младшие 16 бит. Если число целое, отбрасываются старшие 16 бит. Преобразование результата таким образом, чтобы он соответствовал размеру числа, требует специальной операции, называемой функцией ограничения (domain function).

Рассматривая логические операции, следует заметить, что хотя всего имеются 16 функций двух булевых переменных, с помощью единственной функции NAND (НЕ И) можно реализовать все остальные 15 функций. Можно также использовать две функции NOT (НЕ) и AND (И).

✎ Пример

Приведем последовательность команд для реализации побитового ИСКЛЮЧАЮЩЕГО ИЛИ (XOR) над двумя словами с использованием одной лишь команды NAND.

✍ Решение

Предположим, что процессор имеет стековую архитектуру. Операция NAND использует операнды TOS и TOS-1, помещая результат в

TOS. Ячейки TOS и TOS-1 вместе с другими ячейками стека показаны (через запятые) в следующей программе:

Команда TOS, TOS-1, ..., TOS-n

Push A	A, x
Push A	A, A, x
NAND	A', x
Push B	B, A', x
NAND	(BA')', x
Push B	B, (BA')', x
Push B	B, B, (BA')', x
NAND	B', (BA')', x
Push A	A, B', (BA')', x
NAND	(AB')', (BA')', x
NAND	((BA')'(AB')')', x
Pop C	x

✓ **Замечание**

Функция ИСКЛЮЧАЮЩЕЕ ИЛИ требует 12 команд. Поскольку эта операция используется довольно часто, она включается в ISA большинства процессоров. В каждой паре команд push вместо второй команды можно использовать команду повторения, чтобы устранить необходимость вторичной выборки операнда из памяти.

Команды пересылки

Команды пересылки в языках высокого уровня не используются в явном виде. Компилятор добавляет различные виды пересылок, чтобы согласовать программу с конкретной организацией памяти, используемой ISA. Кроме этого, пересылки необходимы 1) для перемещения операндов на другие места в памяти ради упрощения адресации, 2) для перемещения операндов между памятью и регистровым файлом в архитектурах с регистровым файлом и загрузкой-запоминанием и 3) для загрузки и запоминания содержимого специальных регистров. Этот третий класс пересылок может присутствовать в явной форме или в виде побочных результатов, как в случае сохранения и загрузки содержимого программного счетчика после переключения контекста. Явные команды пересылки подразделяются на следующие типы:

- из памяти в память,
- из памяти в регистр(ы) общего назначения или наоборот,
- из памяти в выделенный регистр(ы) или наоборот.

В некоторых ISA команды пересылки привязаны к определенным типам данных: байту, двум байтам и т. д. Встречаются также команды пересылки, которые перемещают блок операндов, например, для вы-

полнения такой операции, как загрузка или запоминание всего регистрового файла одной командой.

Пример неявной пересылки можно найти в командах вызова подпрограмм (call) и возврата из них (return). Эти команды перемещают содержимое программного счетчика в пространство временной памяти и назад. Команда call помещает содержимое программного счетчика либо в ячейку памяти, либо в выделенный регистр, например вершину стека. Команда return читает предварительно сохраненное содержимое программного счетчика и возвращает его в программный счетчик (PC):

call: Временная ячейка $\leftarrow$ (PC)

return: PC $\leftarrow$ (временная ячейка)

Команды управления

Регулярный инкремент программного счетчика обеспечивает обычный последовательный ход выполнения программы. При выполнении команды из некоторой ячейки памяти очередная команда находится в следующей ячейке¹⁹⁾. Команды управления позволяют нарушить линейный ход программы, передавая управление на ячейку, отличную от следующей. К командам управления относятся:

- переходы
 условные,
 безусловные;
- вызовы подпрограмм или процедур и возврат из них;
- прерывания и исключения.

При использовании команд переходов предполагается, что ход программы изменяется безвозвратно; состояние программы в точке перехода не сохраняется и не может быть восстановлено. С другой стороны, при использовании команд вызовов предполагается, что изменение хода программы носит временный характер и что через некоторое время управление вернется в точку вызова. Это означает, что состояние вызывающей программы должно быть сохранено, а затем восстановлено. Прерывания схожи с вызовами в том отношении, что чаще всего после обслуживания прерывания прерванная программа должна быть продолжена.

¹⁹⁾ Программный счетчик инкрементируется на длину текущей команды, взятую по модулю AU. Если память организована по байтам, а длина текущей команды составляет 2 байт, программный счетчик инкрементируется на 2.

Переходы. Имеются две формы команд переходов: условные и безусловные. Условные переходы позволяют программе передать управление в зависимости от результата операции или по исчерпанию заданного числа итераций²⁰⁾. В различных процессорах архитектурно обеспечиваются две формы команд условных переходов:

1. Пара ортогональных команд. Первая команда выполняет анализ и устанавливает бит индикации (эти биты будут описаны в разделе 3.2.3.2). Вторая команда анализирует бит индикации и либо передает управление следующей команде, либо помещает в программный счетчик адрес мишени^{*)} и тем самым осуществляет переход. Вторая команда не обязана выполняться непосредственно вслед за первой: между ними могут находиться и другие команды. Примеры такого типа переходов можно найти в системе команд IBM S360 и x86.

2. Единая (неделимая) команда, которая выполняет анализ и в зависимости от результата этого анализа продолжает или не продолжает последовательное выполнение программы. Этот тип команд перехода носит название «анализ и переход». Примеры таких команд можно найти в большинстве ранних ISA, в частности, в фон-неймановской системе команд и в архитектуре MIPS R2000²¹⁾. Эти команды не используют биты индикации; в результате информация не может быть передана последующим командам. Команда процессора R2000 сравнивает содержимое одного из регистров с нулем:

BGEZ: переход по условию «больше или равно нулю».

В дополнение к операциям условного перехода в большинство ISA входят также команды безусловного перехода (jump). Этот класс команд обеспечивает явную передачу управления и используется, например, для возврата из подпрограмм. Другой вид безусловного перехода называется ловушкой (trap). Ловушки — это команды, безусловно изменяющие значение программного счетчика, когда случается событие побочного эффекта, например арифметическая ошибка.

Безусловные переходы не обязательно являются аппаратными. Например, в зарезервированном регистре или ячейке памяти содержится ноль, а команда условного перехода проверяет этот операнд на ноль, что эквивалентно безусловному переходу.

Вызовы подпрограмм и возвраты из них. Подпрограммы подразделяются на две категории: открытые и закрытые. Открытой подпрограммой называется последовательность команд, включаемых в главную программу в процессе ассемблирования. Другими словами,

²⁰⁾ Считается, что этот тип перехода был впервые реализован в машине Bell Labs Mark V.

^{*)} Имеется в виду адрес перехода на другую ветвь программы. — *Прим. ред.*

²¹⁾ Процессор R2000 явился первой реализацией MIPS-архитектуры. С тех пор эволюция MIPS-архитектуры прошла через 5 поколений и большое число различных реализаций. Описание реализации R2000 и соответствующей этому процессору MIPS-архитектуры следует оценивать в историческом контексте, а не как пример той MIPS-архитектуры, которая существует сегодня.

они выступают в виде встроенного кода. В отличие от этого закрытые подпрограммы могут вызываться из любой точки главной программы и возвращать затем управление в главную программу. Закрытые подпрограммы проектируются с учетом их повторного использования, другими словами, к ним можно обращаться многократно. При этом, если подпрограмма изменяет себя в процессе выполнения, как это делают самомодифицирующиеся программы, перед возвратом из подпрограммы она должна быть восстановлена в исходное состояние. Повторное использование подпрограмм может носить различный характер:

Последовательно используемые подпрограммы могут активизироваться лишь один раз на каждый вызов.

Реентерабельные подпрограммы могут активизироваться повторно в процессе своего выполнения.

Рекурсивные подпрограммы могут вызывать сами себя.

Поддерживаемый вид повторного использования отражается на архитектуре, так как каждая активизация подпрограммы требует выделения пространства памяти. При этом пространство памяти может предоставляться системным программным обеспечением или аппаратно. Чаще используется первый способ; в качестве примера процессора с аппаратным выделением памяти можно вспомнить давно уже не выпускаемый Intel 432.

Дж. П. Эккертом [10] были сформулированы минимальные архитектурные требования для закрытых подпрограмм:

- 1) команда для возврата в главную программу,
- 2) числа, необходимые для выполнения подпрограммы,
- 3) команды, сообщающие подпрограмме, где она должна сохранить результат вычислений.

Легко видеть, что фон-неймановская архитектура не может обеспечить эти требования, особенно третье. Содержимое программного счетчика нельзя сохранить, потому что он является регистром, изменяемым программно; но в этом случае нет способа вернуться в главную программу. В дальнейшем, обсуждая подпрограммы, мы будем говорить о вызывающей и вызываемой программах.

Требование 1

Главным предметом при проектировании команд вызова является выделение памяти для адреса возврата. Обычно адрес возврата представляет собой значение программного счетчика плюс 1. Это значение может быть сохранено либо 1) в памяти состояния процесса, либо 2) в памяти состояния процессора (в специальном регистре или стеке).

Сохранение в памяти состояния процесса. В ранних процессорах, поддерживающих самомодифицирующиеся программы, команда call

сохраняла значение возврата для программного счетчика в регистре или ячейке памяти. Команды в вызванной подпрограмме загружали это значение в адресную часть команды безусловного перехода, которая выступала в качестве последней команды подпрограммы.

Впервые подпрограммы были реализованы в Кембриджском университете в машине EDSAC, разработанной под руководством Мори-са Уилкса. Уилкс, Уилер и Джилл написали основополагающую работу по подпрограммам EDSAC [19]. В этой машине использовался следующий подход.

Команда call выполняла два действия:

аккумулятор $\leftarrow (PC) + 1$

PC $\leftarrow$ адрес подпрограммы

Зная длину подпрограммы, первая команда в вызываемой подпрограмме помещала содержимое аккумулятора в адресное поле последней команды, которая выполняла безусловный переход:

адресная часть последней команды $\leftarrow$ (аккумулятор)

Этот метод вызова подпрограммы получил название перехода со связыванием; он широко использовался проектировщиками ранних компьютеров. Когда подпрограмма доходит до своего конца, команда безусловного перехода возвращает управление в вызывающую программу. Обратите внимание на то, что описанный метод можно реализовать лишь в самомодифицирующихся программах, которые в некоторых современных процессорах не поддерживаются.

Другой пример архитектурного программного счетчика, обеспечивающего вызов подпрограммы и возврат из нее, можно найти в машине Atlas 1 (переименованная MU 1). В этой машине была предусмотрена команда STA, сохраняющая адрес очередной команды, т. е. $(PC) + 1$. Этот адрес в дальнейшем загружался в индексный регистр, обеспечивая возврат из подпрограммы, так как все адреса, даже адреса команд, были индексированными [4].

Сохранение в памяти состояния процессора (в специальном регистре). Если процессор не поддерживает самомодифицирующиеся программы, то содержимое программного счетчика необходимо сохранять в архитектурном регистре. Например, в машинах IBM S360 команды перехода со связыванием помещают значение инкрементированного программного счетчика в один из регистров общего назначения вместе с кодом длины команды, битами признаков результата²²⁾ и программной маской. Адрес первой команды подпрограммы либо является обычным эффективным адресом для 16-битового RX-формата, либо извлекается из регистрового файла в 32-битовом RR-формате. Форматы команд IBM S360 перечислены на рис. 8.3 (гл. 8). Для возврата подпрограмма использует команду перехода по усло-

²²⁾ Индикаторы, закодированные в виде 2-битового признака результата.

вию, которая загружает программный счетчик предварительно сохраненным в регистре значением.

Вложение (гнездование) подпрограмм обеспечивалось использованием свободного регистра для каждого уровня вложенности. Очевидно, что при слишком высоком уровне вложенности в процессоре исчерпывались свободные регистры. Такое использование регистров является одним из недостатков архитектуры с регистровым файлом ограниченного размера, состоящим исключительно из регистров общего назначения.

Процессор MIPS R2000 использует другой вариант перехода со связыванием, получившим название перехода через регистр со связыванием (JALR). Эта команда помещает адрес очередной команды ($PC + 1$) в регистр 31. Поскольку команды перехода извлекают адрес мишени из регистра 31, для возврата из подпрограммы может использоваться любая команда перехода. Процессор не использует индикаторы для условных вызовов подпрограмм, поэтому в нем предусмотрена команда, комбинирующая анализ и вызов. Другая команда является безусловной. Эти команды выполняют следующие действия:

- 1) связывание и переход по условию «больше чем или равно нулю»,
- 2) связывание.

Из-за того что для связывания используется выделенный регистр (R31), гнездование подпрограмм возможно только с издержками в виде контекстных переключений. Контекстное переключение перед вызовом подпрограммы помещает значение в R31. Для «раскручивания назад» первого контекстного переключения требуется второе контекстное переключение.

Сохранение в памяти состояния процессора (в стеке). Другой формой памяти для хранения состояния процессора является стек. При выполнении команды call значение программного счетчика проталкивается в стек, а в программный счетчик загружается адрес первой команды подпрограммы.

Для возврата из подпрограммы достаточно вытолкнуть содержимое вершины стека в программный счетчик. Как было описано выше, сохраненное значение представляет собой адрес команды call. Поэтому после возврата программный счетчик должен быть инкрементирован. Описанный метод годится и для таких реализаций процессора, в которых инкремент программного счетчика осуществляется в конце каждой команды.

Преимуществом использования стека для сохранения содержимого программного счетчика является автоматическая поддержка вложенных подпрограмм (когда подпрограмма вызывает другую подпрограмму). При вызове очередной подпрограммы содержимое программного счетчика проталкивается в стек. При каждом возврате

выполняется выталкивание из стека, приводящее к возврату в вызывающую подпрограмму. Некоторые дешевые микрокомпьютеры имеют стек фиксированной глубины, рассчитанный на 4 – 6 уровней, что ограничивает число вложенных вызовов подпрограмм.

На рисунке 3.7 изображен процедурный стек Intel Pentium Pro. Параметры, передаваемые в вызываемую программу, а также локальные переменные для вызывающей программы проталкиваются в стек. Значением указателя команд для команды возврата является увеличенное на 1 значение указателя команд вызывающей программы. Регистры ESP (указатель стека) и EBP (указатель на данные в стеке) входят в регистровый файл общего назначения, как это показано на рис. 3.8.

В процессоре Intel Pentium Pro, где реализованы механизмы сегментной организации памяти и защиты, предусмотрены четыре типа команд вызова подпрограмм с соответствующими командами возврата. К ним относятся:

- ближний вызов: вызов внутри текущего сегмента кода,
- дальний вызов: вызов с переходом в другой сегмент кода,



Рис. 3.7. Процедурный стек процессора Intel Pentium Pro (с любезного согласия корпорации Intel).

Регистры общего назначения

31	16	15	8	7	0	16-бит	32-бит
			AH		AL	AX	EAX
			BH		BL	BX	EBX
			CH		CL	CX	ECX
			DH		DL	DX	EDX
			BR				EBP
			SI				ESI
			DI				EDI
			SP				ESP

Рис. 3.8. Регистры общего назначения Pentium Pro (с любезного согласия корпорации Intel).

- дальний вызов со сменой уровня привилегий: вызов с переходом в другой сегмент, которому присвоен другой уровень привилегий,
- переключение задач: вызов другой задачи.

Требования 2 и 3

Эти требования относятся к пересылке информации между вызывающей и вызываемой программами. В очень простых системах команд ответственность за передачу данных возлагается на саму подпрограмму; в этом случае не требуется какой-либо специальной аппаратной поддержки, за исключением возможности загружать и запоминать программный счетчик.

Передача параметров. Параметры передаются от вызывающей программы к подпрограмме путем записи их в обусловленные заранее ячейки памяти. Другими словами, для передачи параметров резервируется общая область памяти. Результаты работы подпрограммы также помещаются в обусловленные заранее ячейки памяти.

В системе команд со стеком (например, Pentium Pro) вызывающая программа проталкивает параметры в стек. Это удобно, так как при использовании вложенных подпрограмм параметры второго или третьего вызовов также будут помещены в стек. В результате активная подпрограмма найдет свои параметры в верхних ячейках стека. Однако в этом случае требуется соблюдение определенных условий: в подпрограмму следует передавать фиксированное число параметров или вместе с параметрами в стек должен быть помещен их счетчик.

Другой метод передачи параметров заключается в договоренности об использовании одного или нескольких регистров общего назначения. Такой метод хорош тем, что сокращается время, требуемое для передачи параметров, но в компьютерах с регистровыми файлами небольшого объема число регистров, выделенных для передачи параметров, оказывается небольшим, что ограничивает число передаваемых параметров.

Параметры также могут быть сохранены вызывающей программой в памяти, а адрес этой памяти должен быть передан в подпрограмму через регистровый файл в виде одного слова. Ячейки памяти с параметрами носят название списка аргументов, а указатель называется указателем аргументов. При использовании этого метода нет необходимости заранее резервировать память для параметров. Однако система команд должна поддерживать использование адресов из регистрового файла.

Процессор Intel Pentium Pro, например, имеет развитые средства передачи параметров. Этот процессор предусматривает три метода: с помощью регистров общего назначения, с помощью стека, как это показано на рис. 3.7, и с помощью списка аргументов.

Команда `call` сохраняет в стеке только значение программного счетчика; она не сохраняет содержимое регистрового файла. Таким образом, для передачи параметров как в подпрограмму, так и из нее можно использовать до 6 регистров. Параметры можно также передать путем проталкивания их в стек с образованием стекового фрейма. Указатель на базу стекового фрейма используется для фиксации границы фрейма с параметрами. Если используется список аргументов, указатель аргументов передается в вызываемую подпрограмму в оговоренном регистре регистрового файла.

Сохранение состояния процессора. При вызове подпрограммы необходимо сохранить состояние процессора, чтобы подпрограмма могла использовать такие ресурсы, как регистровый файл, указатели стека, биты индикации и прочее. Разработчик должен определить, каким образом и где будет сохраняться состояние процессора.

Здесь имеются две возможности. Ответственность может быть возложена как на вызывающую, так и на вызываемую программы. Вызывающая программа может перед вызовом подпрограммы скопировать все состояние процессора в предназначенную для этого область памяти. Вызываемая подпрограмма должна в этом случае инициализировать состояние процессора перед началом своего выполнения. Когда управление возвращается вызывающей программе, состояние процессора, действовавшее на момент вызова, должно быть восстановлено. Этот метод требует единого интерфейса со всеми подпрограммами.

С другой стороны, задачу сохранения и восстановления состояния процессора можно возложить на вызываемую подпрограмму. Преимуществом такого метода можно считать возможное уменьшение времени на операции сохранения и восстановления, так как подпрограмме могут требоваться не все ресурсы. Если, например, подпрограмма использует только один регистр, зачем сохранять и восстанавливать все остальные? Точно так же, если подпрограмма не использует биты индикации, зачем же их сохранять и восстанавливать?

Прерывания и исключения. Последовательный ход выполнения программы может быть изменен переходами в тексте самой выполня-

емой программы. Однако имеются и другие события, связанные с работой процессора или системы, которые могут потребовать нарушения хода выполнения программы, иногда на значительные периоды времени. Эти события называются прерываниями. Прерывания могут быть трех типов:

- *Синхронные*: события, затребованные программой. Примерами могут служить вызовы средств операционной системы, а также ловушки трассировки программы.
- *Внутренние*: события, возникающие в самом процессоре. Примерами таких событий являются арифметические исключения и отказы страниц^{*)}.
- *Асинхронные*: события, возникающие обычно вне процессора, например, запросы системы ввода-вывода, а также сигналы таймера.

При повторном выполнении одной и той же программы с теми же данными внутренние и синхронные прерывания возникают всегда в одном и том же месте. Асинхронные прерывания могут возникнуть в любое время, поэтому программа не может предугадать момент их возникновения. Внутренние аппаратные отказы считаются асинхронными событиями.

Разработка системы прерываний имеет общие черты с проектированием системы вызова подпрограмм. Основное отличие заключается в том, что недопустимо использование тех из описанных выше методов, которые требуют от прерываемой программы определенных действий, например сохранения состояния или передачи параметров. Действительно, асинхронные события не могут быть предугаданы, поэтому на текущую программу возложить эти задачи нельзя. Прерывания не оповещают о себе заранее. Однако метод сохранения состояний процессора и процесса подпрограммой может быть использован. Для сокращения числа реализованных в ISA процедур проектировщики часто для подпрограмм и прерываний предусматривают схожие алгоритмы. Процессор должен выполнить четыре задачи:

- 1) распознать событие прерывания,
- 2) завершить или приостановить выполняемую программу,
- 3) начать выполнение программы обслуживания прерывания (программы, вызываемой событием прерывания),
- 4) восстановить выполнение прерванной программы.

Процессор, система команд и аппаратура компьютера должны предоставить средства для решения этих задач. Эти средства, как и возможные стратегии, будут описаны ниже. Важнейшим требованием поддержки прерываний в системе команд является сохранение и вос-

^{*)} См. раздел 4.2. — *Прим. ред.*

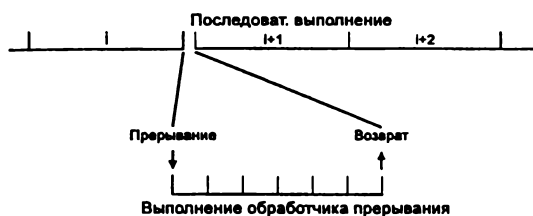


Рис. 3.9. Обслуживание прерывания.

становление содержимого программного счетчика; отсюда следует, что фон-неймановская архитектура в принципе не могла поддерживать прерывания. Важным вопросом при проектировании системы прерываний является место хранения адреса возврата — проблема, аналогичная той, которую приходится решать при реализации вызовов подпрограмм. Адрес возврата обычно представляет собой значение программного счетчика плюс 1. Это значение может быть сохранено либо в памяти состояния процесса, либо в памяти состояния процессора. В процессорах, поддерживающих самомодифицирующиеся программы, адрес возврата может быть сохранен в ячейке памяти.

На рисунке 3.9 показано действие прерывания, когда процессор выполняет последовательный ряд команд²³⁾. Особые проблемы, возникающие в связи с прерываниями в конвейерных процессорах, будут описаны в гл. 6. В верхней части рисунка показана нормальная последовательность выполнения команд. В конце некоторой команды, например, с номером i , возникает сигнал прерывания. Этот сигнал приводит к прерыванию последовательного хода выполнения команд и передаче управления на обработчик прерывания, как показано в нижней части рисунка. После завершения работы обработчика прерывания восстанавливается нормальная последовательность выполнения программы, начиная с команды с номером $i+1$.

Для некоторых прерываний возврат должен осуществляться на команду с номером i , а не $i+1$. Например, если причиной прерывания является ненормальная ситуация в процессоре вроде арифметического исключения в команде i , обработчик прерывания должен выполнить действия по устранению условия исключения, после чего процессор выполняет команду i заново.

В дополнение к отмеченным выше четырем задачам, решаемым при обслуживании прерываний, процессор должен обеспечивать:

- 1) средства сохранения состояний процессора и процесса,
- 2) средства идентификации источника прерывания и загрузки правильного значения программного счетчика,
- 3) средства восстановления состояний процессора и процесса после завершения работы обработчика прерываний.

²³⁾ Выполнение последовательного ряда команд предполагает, что выполнение следующей команды начинается лишь после того, как закончилось выполнение предыдущей.

Сохранение состояния

Как и в случае вызова подпрограмм, минимальной частью сохраняемого состояния процессора является значение программного счетчика, причем он может быть либо сохранен в предназначенном для этого аппаратном стеке процессора или в выделенном регистре, либо помещен в стек в памяти. Использование аппаратного стека является обычной практикой для однокристальных микроконтроллеров, причем глубина стека ограничивает допустимый уровень вложенности прерываний. Выделенный регистр часто использовался в ранних компьютерах, но сейчас так не делают, поскольку в этом случае можно обслуживать лишь одно прерывание. В то же время стек в памяти работает относительно быстро, так как он обычно кэшируется операционной системой.

Идентификация источника прерывания

Источник прерывания распознается одним из двух способов. Простейшим является объединение по ИЛИ всех сигналов прерываний, чтобы любой источник мог вызвать прерывание. Тогда для идентификации источника обработчик прерывания должен в качестве преамбулы выполнить анализ, или опрос (polling), сигнальных линий. Выполнение опроса потребует нескольких команд. Для прерываний, поступающих в случайном порядке, среднее число опросов составляет $1/2$ числа опрашиваемых линий; однако обычно опрос ведется в порядке приоритета прерываний. После того как источник прерывания идентифицирован, программа преамбулы обработчика сверяется с таблицей и находит нужный адрес для загрузки в программный счетчик.

Второй способ идентификации источника прерываний носит название векторных прерываний. В этом случае источник прерывания посылает вместе с сигналом прерывания еще и его идентификатор. Этот идентификатор используется в качестве индекса в таблице, содержащей значения программного счетчика для всех обработчиков прерываний. Векторные прерывания обеспечивают минимальную задержку, но более сложны в реализации.

Восстановление состояния

После того как обработчик прерывания завершил свою работу, выполняется команда возврата из прерывания. Эта команда находит и помещает в программный счетчик его сохраненное в точке прерывания значение, чтобы возобновилось нормальное выполнение прерванной программы. Все остальные сохраненные элементы состояния восстанавливаются либо обработчиком прерывания перед возвратом из него, либо аппаратурой.

Дополнительные соображения

На временной диаграмме на рис. 3.9. обозначено, что прерывание возникло в конце выполнения команды. Для внешних прерываний, поступающих от устройств ввода-вывода, так обычно и происходит. Даже если сигнал прерывания поступает во время выполнения команды, процессор дает возможность команде закончиться. Однако как обстоит дело с прерываниями, которые возникают в процессе выполнения команды, которая не имеет права продолжаться? Примерами таких ситуаций являются неправильный код операции или чтение команды или данного из запрещенной страницы. В таких случаях выполнение команды обычно прекращается. После того как условие, вызвавшее прерывание, скорректировано обработчиком прерывания, прерванная команда выполняется повторно. Эта процедура носит название рестарта. Некоторые процессоры сохраняют состояние частично выполненных команд и затем продолжают их выполнение после устранения причины исключения. Такая процедура называется продолжением.

Система команд может также автоматически сохранять значения регистров, содержащих состояние процессора и процесса; в других случаях эта задача возлагается на преамбулу и заключительную часть обработчика прерывания. Если сохранение состояний осуществляет обработчик, ISA должна предоставить необходимые команды пересылки для сохранения и восстановления этих значений состояния.

На рисунке 3.10 систематизирована техника обслуживания прерываний, которые возникают между и внутри команд, с выделением методов рестарта и продолжения. На рисунке также отмечены реализации машин IBM S360, MC68000 и VAX-11.

Реализация прерываний сопряжена с рядом трудностей. В частности, источникам прерываний желательно назначить приоритеты; некоторые прерывания более важны, чем другие. Приоритеты проявляют себя в двух ситуациях: 1) прерывание возникает в процессе обслуживания другого прерывания и 2) два или более прерываний возникают в одно и то же время.

Если второе прерывание возникает, пока обслуживается предыдущее, определяется приоритет нового прерывания. Если новое прерывание имеет более высокий приоритет, оно прерывает выполнение текущего обработчика прерывания. Если же его приоритет ниже приоритета обрабатываемого прерывания, оно помещается в очередь прерываний для обслуживания позже. Заметьте, что очередь прерываний обслуживается в порядке приоритетов, а не в соответствии с дисциплиной «первый пришел, первый обслужен».

Большинство вычислительных систем должны распознавать и отзываться на прерывания от ряда источников. Кроме того, иногда процессор не должен отзываться на прерывание, как это имеет место в случае обслуживания прерывания с более высоким приоритетом. Для обслуживания множественных прерываний используются три

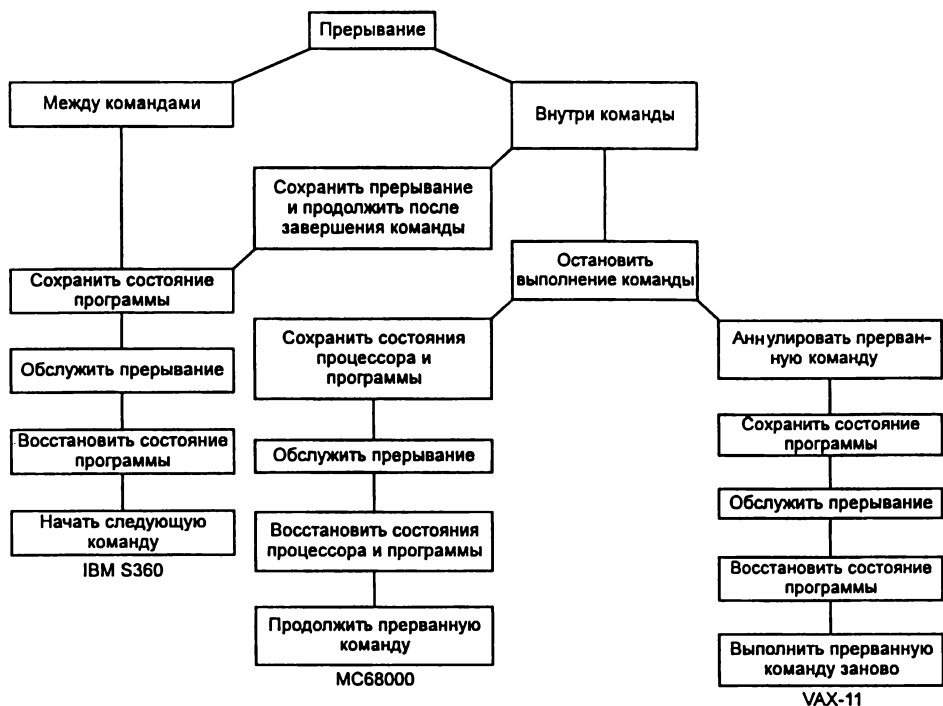


Рис. 3.10. Систематика обслуживания прерываний.

способа: маскируемые прерывания, немаскируемые прерывания и приоритеты. Маскируемое прерывание, как это и следует из его названия, имеет программный переключатель, который позволяет выключить данное прерывание и перенести его обслуживание на более поздний момент. В противоположность этому, немаскируемые прерывания всегда воспринимаются процессором, и в задачу обработчика прерывания входит определение того, должно ли обслуживание данного прерывания выполняться немедленно или может быть отложено. Для обоих этих случаев приоритеты устанавливаются операционной системой (в предположении, что обработчики прерываний являются компонентами операционной системы). Приоритетные прерывания используют аппаратные средства для назначения приоритетов и определения важности их обслуживания.

При использовании маскируемых прерываний процессор должен иметь архитектурный (аппаратный) регистр, в который загружается битовый шаблон, включающий или выключающий каждую из линий прерываний. Тогда, в зависимости от операционной ситуации, одно прерывание можно включить, а остальные выключить. Если, например, операции ввода-вывода не выполняются, но возможен отказ страницы, прерывания ввода-вывода маскируются, а прерывание отказа страницы остается включенным.

Опрос. Некоторые процессоры не имеют архитектурно реализованной системы прерываний. Такие процессоры отзываются на внешние события путем периодического опроса устройств с целью выяснения, не требуется ли тому или иному устройству прервать работу процессора. Если внешнее событие нуждается в обслуживании, вызывается обработчик этого события, который является аналогом обработчика прерывания.

Процессор выполняет цикл опроса периодически. Например, процессор может каждую микросекунду опрашивать все высокоприоритетные источники внешних событий, а опрос таких источников, как цепь питания, выполнять лишь каждые 0,1 с (чтобы зафиксировать потерю питания). Система опроса плохо работает в случае, когда высокоприоритетное прерывание приходит в момент обслуживания прерывания низкого приоритета. Для того чтобы в такой ситуации система работала правильно, обработчики прерываний должны сами содержать циклы опроса высокоприоритетных источников.

Некоторые суперкомпьютеры (CDC 6600, CDC 7600 и TI ASC) представляют собой мультипроцессоры, в состав которых входят процессоры для выполнения опроса. Эти процессоры снимают с центрального процессора нагрузку, связанную с опросом устройств. Сегодня системы с опросом можно найти в микроконтроллерах. В качестве примера укажем на процессор управления клавиатурой, используемый в клавиатурах ПК. Этот процессор непрерывно опрашивает клавиши, чтобы установить их состояние.

3.2.3. Типы данных

Типы данных, включенные в систему команд и поддерживаемые процессором, могут быть двух видов: 1) числовые или логические типы данных и 2) типы данных для управления программой. Типы данных, которые требуются для работы операционной системы, обслуживания прерываний и управления памятью, будут рассмотрены в гл. 8.

Прикладная программа нуждается в целом ряде типов данных, преимущественно целых или с плавающей запятой. Целые могут занимать байт, половину слова, слово или двойное слово, а также быть числами со знаком или без знака. В наши дни для данных с одинарной или двойной точностью почти исключительно используется тип данных с плавающей запятой в формате IEEE (стандарт ANSI/IEEE 754-1985). Другая форма представления целого — десятичный тип данных. Такие данные обычно упаковываются по два в байт и, соответственно, имеют диапазон от 00 до 99. Если десятичное данное со знаком, для него выделяется байт, соответствующий строке байтов с данными десятичного типа.

Суперкомпьютеры используют также данные векторных типов. Обычно вектор состоит из данных с плавающей запятой одинарной

Таблица 3.5. Описатели типов данных

Длина	Название	Точность
Скаляр	Логическое	Фрагментарная
Вектор	Со знаком	Четвертная
Матрица	Десятичное	Половинная
Массив	Восьмеричное	Одинарная
Строка	Символ	Двойная
	Байт	Тройная
	Слог	Учетверенная
	Слово	Множественная
	Целое без знака	Целочисленная
	Целое со знаком	
	Дробь	
	С плавающей запятой	
	Не нормализованное	
	Комплексное	

или двойной точности. Такие данные требуют указания в коде команды их длины в явной или неявной формах.

Некоторые возможные значения трех описателей для прикладных данных (точность, название и длина) приведены в табл. 3.5 [15]. Использование этих описателей иллюстрируется таким примером (в котором подчеркнуты ключевые слова): тип данных может быть **вектором десятичных цифр**, упакованных по две цифры в байте (**фрагментарная точность**).

3.2.3.1. Пользовательские типы данных

Прикладные программы имеют дело со множеством типов данных. Эти данные можно классифицировать следующим образом:

Числа с фиксированной запятой

Целые

Дробы

Смешанные

Числа с плавающей запятой

Логические значения

Целые, дробы и смешанные типы данных могут быть как со знаком, так и без знака. Типы данные с плавающей запятой всегда являются числами со знаком. Арифметические операции с числами с плавающей запятой будут рассмотрены в разделе 3.4.

Целые числа с фиксированной запятой, дроби и смешанные типы данных без знака

Числа могут иметь различное представление. Так, если запятая в позиционном представлении числа находится справа от младшего бита, то такой тип является целым без знака²⁴⁾. Поместив запятую слева от старшего бита, мы получим дробное число без знака. Если запятая находится между этими двумя пределами, то число является смешанным и имеет и целую, и дробную части. Рассмотрим варианты двоичного числа, содержащего n бит.

$$\text{целое без знака} = 2^{n-1} + 2^{n-2} + 2^{n-3} + \dots + 2^{n-n}$$

Двоичная запятая находится здесь справа от числа:

2^{n-1}	2^{n-2}					2^1	2^{n-n}
-----------	-----------	--	--	--	--	-------	-----------

$$\text{дробь без знака} = 2^{-1} + 2^{-2} + 2^{-3} + \dots + 2^{-n}$$

Здесь двоичная запятая расположена слева от числа:

2^{-1}	2^{-2}						2^{-n}
----------	----------	--	--	--	--	--	----------

$$\text{смешанное без знака} = 2^{b-1} + \dots + 2^{b-b} + 2^{-1} + 2^{-2} + \dots + 2^{-(n-b)}$$

Здесь b представляет собой число битов в целой части числа, а $n-b$ — число битов в дробной части числа:

2^{b-1}	2^{b-2}		2^{b-b}		2^{-1}	2^{-2}				$2^{-(b-n)}$
-----------	-----------	--	-----------	--	----------	----------	--	--	--	--------------

Целые и дроби со знаком

Важным элементом проектирования целых и дробных типов данных является способ представления отрицательных значений. Обычно используются три способа представления целых и дробей со знаком: 1) с помощью знака и величины, 2) как дополнение до 1 (называемое также поразрядным дополнением, или обратным кодом) и 3) как дополнение до 2 (называемое также дополнением по основанию, точным дополнением или дополнительным кодом). В таблице 3.6 показаны десятичные представления 3-битового числа каждого из шести типов данных.

²⁴⁾ Также известным как порядковый, или ординальный.

Таблица 3.6. Целые и дробные типы данных

Знак и величина			Дополнение до 1		Дополнение до 2	
Двоичное	Целое	Дробь	Целое	Дробь	Целое	Дробь
000	+0	+0	+0	+0	+0	+0
001	+1	+0,25	+1	+0,25	+1	+0,25
010	+2	+0,50	+2	+0,50	+2	+0,50
011	+3	+0,75	+3	+0,75	+3	+0,75
100	−0	−0	−3	−0,75	−4	−1,00
101	−1	−0,25	−2	−0,50	−3	−0,75
110	−2	−0,50	−1	−0,25	−2	−0,50
111	−3	−0,75	−0	−0	−1	−0,25
Диапазон	от	от	от	от	от	от
	$-(2^{n-1} - 1)$	$-[1 - 2^{-(n-1)}]$	$-(2^{n-1} - 1)$	$-[1 - 2^{-(n-1)}]$	$-2^{(n-1)}$	−1
	до	до	до	до	до	до
	$2^{n-1} - 1$	$1 - 2^{-(n-1)}$	$2^{n-1} - 1$	$1 - 2^{-(n-1)}$	$2^{n-1} - 1$	$1 - 2^{-(n-1)}$

Все эти странные типы данных являются результатом попытки описать нечетное количество чисел x , $x-1$, ..., 0 , ..., $-x-1$, $-x$ с помощью четного количества различающихся чисел 2^n в двоичном слове. Поэтому возникает либо избыточность (представления со знаком и величиной, а также в виде дополнения до 1 имеют по два нуля), либо несимметричное представление (как в случае дополнения до 2). Основным мотивом разработки этих представлений явилось снижение сложности ALU.

Представление со знаком и величиной. Достоинством такого представления является его естественность. Действительно, имеется бит знака, а числа оказываются симметричными относительно нуля. Однако для этой системы характерно наличие двух нулей: -0 и $+0$, что далеко не естественно. В ALU должны быть специальные средства обработки избыточных нулей.

Дополнение до 1. В этом представлении, как и в случае знака и величины, один бит выделяется под знак, но биты величины формируются иначе. Для положительных чисел биты величины трактуются так же, как и для целых чисел без знака (в случае 4-битового числа $2 = 0 \wedge 010$).²⁵⁾ Отрицательные числа представляются в виде побитового дополнения положительного числа (в случае 4-битового числа $-2 = 1 \wedge 101$).

Правила арифметических операций для типов данных, представляемых в виде дополнения до 1, довольно сложны; они требуют использования циклического переноса, который будет описан в разде-

²⁵⁾ Символ $\wedge$ используется для отделения бита знака.

ле 3.4. В результате в современных ISA такой тип данных не используется.

Дополнение до 2. В этом представлении устраняется проблема двух нулей и циклического переноса. Однако представление несимметрично. Как показано в табл. 3.6, максимальное значение составляет +3, а минимальное —4. С другой стороны, преобразование положительного числа в отрицательное требует замены всех битов на обратные значения и прибавления к результату 1

$$(+2 = 0-010, -2 = 1-101 + 1 = 1-110)$$

Прибавление 1 образует дополнительный шаг при создании отрицательного числа по сравнению с представлением в виде дополнения до 1. Однако в целом арифметические операции упрощаются, поскольку устраняется необходимость использования циклического переноса. Арифметика дополнения до 2 обсуждается в разделе 3.4.

Дробные типы данных. В таблицу 3.6 включены дробные представления со знаком и величиной, а также дробные дополнения до 1 и до 2. Как и в случае целых типов данных, либо в составе чисел оказываются два нуля, либо числа несимметричны. Дробные типы данных используются в качестве компонентов типов данных с плавающей запятой, а также как самостоятельный тип в некоторых компьютерах для научных исследований.

Десятичные типы данных без знака

Многие прикладные программы требуют наличия десятичных типов данных. Такое представление естественно для денежных расчетов в долларах и центах, а также для калькуляторов. Другие примеры можно найти в языках программирования электронных таблиц и в языке COBOL. Десятичную нотацию использовали некоторые ранние компьютеры, например Mark I и ENIAC. С другой стороны, двоичная нотация, введенная Цузе и Атанасовым, стала общепринятой при использовании фон-неймановской архитектуры для научных приложений.

Двоичный код. В десятичной нотации десятичные цифры 0—9 выражаются четырьмя битами, которые могут принимать 16 различных состояний. Выбор требуемого подмножества из 16 состояний оставляет 6 состояний неиспользуемыми. Веса битов можно выбирать (одним из примеров выбора является последовательность весов 8, 4, 2, 1, которая используется в так называемых BCD-числах). Выбор значимых состояний и весов выполняется так, чтобы упростить операции сложения, особенно обработку переноса от одного десятичного разряда к следующему. Уайт [19] установил, что существует приблизительно $7,6 \times 10^7$ возможных вариантов выбора, хотя архитектурно реализованы были лишь немногие из них.

Компьютеры IBM S360, S370 и S390 используют для десятичных типов данных кодировку 8, 4, 2, 1. Для правильного выполнения сложения в процессор включается аппаратный сумматор.

Код 2 из 5. В этой системе 2 бита 5-битового числа всегда равны 1, что предоставляет некоторую степень защиты от ошибок. Если 0 изменяется на 1 или 1 на 0 в процессе операции пересылки, то такую ошибку можно обнаружить, поскольку при этом нарушается правило 2/5. Двойные ошибки не обнаруживаются. IBM 650, ранний десятичный компьютер для деловых применений, использовал код 2 из 5, показанный в табл. 3.7.

Таблица 3.7. Десятичные цифры в IBM 650

Десятичное	B4	B3	B2	B1	B0
0	1	1	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	1	0	0	0	1
8	1	0	0	1	0
9	1	0	1	0	0

Код с избытком. В этой нотации 10 десятичных цифр представляются в их двоичном коде с прибавлением 3, как это показано в табл. 3.8.

Таблица 3.8. Десятичные цифры в коде с избытком 3

Десятичное	С избытком 3
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Эта нотация имеет любопытные арифметические свойства. Рассмотрим результат сложения двух чисел: $(a + 3) + (b + 3) = a + b + 6$.

Для того чтобы результат имел избыток 3, из него необходимо вычесть 3, что даст $a + b + 3$. Если $a = 2$ и $b = 3$, то сложение дает

$$0101 + 0110 = 1001 - 0011 = 1000 = 5$$

т. е. правильный результат с избытком 3.

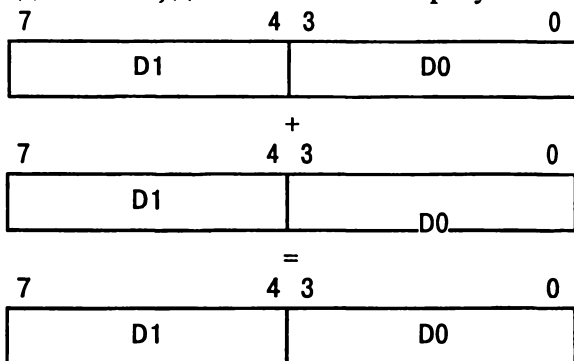
Если при сложении возникает перенос из старшего бита, он одновременно является переносом для десятичного разряда, причем к результату следует *добавить 3*. Рассмотрим операцию сложения $7 + 6 = 13$:

$$1010 + 1001 = 10011 + 0011 = 1\ 0110 = 13$$

Получилась правильная сумма с избытком 3. Однако перенос должен быть представлен правильным образом, как 0100, а не как 0001.

Десятичная корректировка. Семейство микропроцессоров x86 представляет цифры 0–9 в двоичном коде от 0000 до 1001. Далее, поскольку в процессоре 8008 все данные имеют 8 бит, в один байт можно упаковать 2 десятичные цифры. Сложение упакованных десятичных чисел выполняется в 8-битовом ALU, причем предусмотрены средства для восстановления правильного упакованного формата после сложения.

Этот процесс проиллюстрирован ниже. Две десятичные цифры без знака упакованы в 1 байт (D1 и D0). В операции сложения эти два байта складываются, давая 1-байтовый результат:



При формировании суммы может возникнуть перенос как из младшей цифры D0, так и из старшей цифры D1. Значения D0 и D1 после сложения могут быть в диапазоне 0000–1111. Для того чтобы привести их к диапазону 0000–1001, требуется коррекция. Команда десятичной коррекции исправляет результат для каждой десятичной цифры с помощью следующей операции из двух шагов:

Шаг 1. Если $D0 > 9$ или был перенос из бита 3, прибавить 00000110.

Шаг 2. Если $D1 > 9$ или был перенос из бита 7 от первого сложения или от шага 1, прибавить 01100000.

Эти два шага коррекции приведут результат к правильному представлению и образуют в случае необходимости десятичный перенос из D1. Десятичный перенос используется командами сложения с переносом в операциях многократной точности.

✎ Пример

Сложите следующие пары десятичных чисел, используя команды десятичной коррекции:

$$56 + 67, 99 + 99, 81 + 52$$

✎ Решение

56	0101 0110	99	1001 1001	81	1000 0001
+67	<u>0110 0111</u>	+99	<u>1001 1001</u>	+52	<u>0101 0010</u>
	1011 1101		1 0011 0010		1101 0011
	<u>+0000 0110</u>		<u>+0000 0110</u>		<u>+0000 0000</u>
	1100 0011		1 0011 1000		1101 0011
	<u>+0110 0000</u>		<u>+0110 0000</u>		<u>+0110 0000</u>
123	1 0010 0011	198	1 1001 1000	133	1 0011 0011

Десятичные типы данных со знаком

Большинство компьютеров использует полное представление десятичного типа данных в нотации со знаком и величиной. Строке десятичных цифр предшествует или следует за ней байт или полубайт, содержащий знак. Ниже показан тип десятичных 6-разрядных данных, занимающий 3 байта. Цифры могут быть записаны в виде кодов 2 из 5, с избытком 3, с десятичной коррекцией или в любом другом возможном для десятичных цифр виде.

S	D5	D4	D3	D2	D1	D0
---	----	----	----	----	----	----

Поскольку сам тип данных записан со знаком и величиной, сложение и вычитание выполняются по правилам для арифметических операций, описанным в разделе 3.4. Однако отдельные цифры с целью облегчения операции вычитания могут записываться в форме дополнения до 9 или до 10. В архитектуре x86 для выполнения операций вычитания с соответствующей десятичной коррекцией предусмотрена специальная команда DAS.

Типы данных с плавающей запятой

Типы данных с плавающей запятой и соответствующие операции были разработаны с целью устранения необходимости выполнять масштабирование данных в инженерных расчетах. Этот тип данных основан на широко используемой экспоненциальной, или стандартной, нотации, в которой, например, десятичное значение $+18,5$ представляется в виде

$$+1,85 \times 10^1$$

В этой нотации значение показателя степени подбирается таким образом, чтобы перед десятичной запятой была всего одна десятичная цифра. Таким образом, мы имеем дело со смешанным типом данных. Непосредственное преобразование десятичной экспоненциальной нотации в двоичную форму приводит к такому представлению числа $+18,5$:

$$+1,00101000 \times 2^4$$

В некоторых типах данных с основанием 2 и с плавающей запятой число $18,5$ нормализуется, пока в нем не останется лишь дробная часть:

$$+0,100101000 \times 2^5$$

Другими словами, нормализация продолжается либо влево, либо вправо до тех пор, пока не появится 1 в старшем бите дробной части (мантиссы). Полученная последовательность битов затем записывается в слово. Обратите внимание на то, что порядок записан со знаком и величиной; таким образом, знак имеется и у порядка, и у дробной части. С помощью приема, называемого смещением порядка, можно избавиться от знака порядка, как это будет описано позже:

S	Порядок	Дробь
+	+101	0,100101000

Итак, тип данных с плавающей запятой имеет три компонента: знак, порядок и мантиссу. Мантисса может быть дробным, целым или данным смешанного типа. В большинстве форматов чисел с плавающей запятой порядок смещается (выражается с избытком) и, таким образом, не требует указания знака. Комбинация этих трех компонентов дает число с плавающей запятой:

$$\text{число} = \text{знак} \times \text{мантисса} \times \text{основание}^{\text{порядок}}$$

При заданном числе битов в описании данного с плавающей запятой расширение порядка приводит к увеличению диапазона представ-

ляемых чисел при уменьшении их точности. При использовании формата с плавающей запятой диапазон значений оказывается шире, чем диапазон целых, использующих столько же битов.

Проблемы проектирования. Ниже обсуждаются некоторые проблемы, возникшие перед первыми разработчиками, вместе с их решениями для формата IBM S360, широко используемого в современных компьютерах. Мы также достаточно подробно опишем формат IEEE для чисел с плавающей запятой. Типу данных с плавающей запятой выделяется фиксированное число битов (32, 64, 80 и т. д.). Разработчик типа данных должен при этом определиться в выборе целого ряда деталей, некоторые из которых перечислены ниже.

- Система счисления: машинное представление; обычно двоичная для компьютеров, десятичная для калькуляторов.
- Основание: обычно 10, 2 или 16^{26} .
- Число битов в порядке: обычно 7, 8 или 12.
- Содержит ли порядок знак, или используется смещенный порядок?
- Местоположение знакового бита.
- Число битов в мантиссе.
- Является ли мантисса целым, дробным или смешанным типом данных?
- Поскольку число с плавающей запятой является приближенным, как обрабатываются исключительные ситуации?

Последний пункт обсуждается в разделе 3.5.3.1 после описания арифметики с плавающей запятой.

Смещенные порядки. Ранние компьютеры с программируемой плавающей запятой требовали, чтобы тип данных и операции организовывались с помощью подпрограмм. Для этих компьютеров было вполне естественно использовать и для порядка, и для мантиссы первоначальный целый тип данных. В результате и порядок, и мантисса выражались со знаком и величиной, в виде дополнения до 1 или дополнения до 2 (что приводило к наличию в формате числа двух битов знака). Самые ранние компьютеры (Z4, Bell Labs Mark V) с аппаратно реализованной плавающей запятой использовали комбинации порядка и дроби в виде дополнения до 1 или до 2. Например, в компьютере Цузе Z4 порядок записывался в виде дополнения до 2, а дробь — со знаком и смешанной величиной. Такой метод организации чисел с плавающей запятой был перенесен в аппаратную реализацию плавающей запятой на ряде машин (Atlas, IBM Stretch), пока не был вытеснен форматом со смещенным порядком.

Значения данных можно нормировать так, чтобы значения, превышающие середину диапазона, были положительными, а меньшие или равные середине диапазона — отрицательными. Середина диапазона

²⁶⁾ В одном из самых ранних компьютеров, Bell Labs Mark V (1946 г.), использовались архитектурно реализованные десятичные числа с плавающей запятой.

может быть равна либо 2^n , либо $2^n - 1$, в зависимости от деталей реализации. Эта нотация, называемая нормированной, или смещенной, используется для выражения порядка в форматах с плавающей запятой, чтобы избежать необходимости явно указывать знак порядка. Выполнение арифметических операций, сравнения, инкремента или декремента чисел с плавающей запятой приводит к неоправданным сложностям. Таким образом, лучшим решением является использование смещенного (с избытком) порядка, что напоминает тип данных с избытком 3, описанный выше.

Рассмотрим 3-битовый порядок, который может принимать 8 значений от 000 до 111. В таблице 3.9 приведены значения порядка для двух смещений: $2^{n-1} - 1$ и 2^{n-1} . Не забывайте, что нулевой порядок не соответствует нулевому результату. Порядок из всех нулей обозначает основание в степени 0, т. е. 1. Арифметические операции со смещенными порядками обсуждаются ниже в разделе 3.5.3.

Таблица 3.9. Значения смещенного порядка

Двоичное	Смещение = $2^{n-1} - 1$	Смещение = 2^{n-1}
000	-3	-4
001	-2	-3
010	-2	-2
011	0	-1
100	+1	0
101	+2	+1
110	+3	+2
111	+4	+3

Насколько известно автору, первым компьютером, в котором использовался смещенный порядок, был IBM 704 (1955 г.). Аппаратно реализованные команды с плавающей запятой используют регистры и операции реализации; поэтому нет необходимости, чтобы представление порядка было тем же, что и представление мантиссы.

В последующих подразделах описаны два типа данных с плавающей запятой: IBM S360 и стандарт IEEE. Приведенное описание позволит нам сравнить разработки с основанием 16 и 2. Оба этих типа данных с плавающей запятой являются на сегодняшний день наиболее часто используемыми.

IBM S360. Выбор основания для плавающей запятой послужил причиной горячих споров. После выпуска IBM S360 возможности выбора свелись к двум: основание 16 и основание 2, хотя в прошлом широко использовалось и основание 8. Выбор основания тесно связан с распределением битов в слове, которое, в свою очередь, зависит от того, что для нас важнее — диапазон или точность.

Свини [16] проанализировал число сдвигов, требуемых для выравнивания мантисс перед операцией, а также для нормализации, выпол-

няемой после операции.²⁷⁾ Полученные им результаты для 26-битовой дробной мантиссы приведены в табл. 3.10.

После выполнения Свини исчерпывающего анализа научных программ для IBM S360 было выбрано основание 16, на что также повлияло желание иметь максимальный диапазон даже для формата одинарной точности. Диапазон числа с плавающей запятой составляет:

$$\text{диапазон} = \text{основание}^{\text{порядок}} = \text{основание}^{2^{\text{число битов}} - r}$$

Параметр r представляет собой число кодов в порядке, зарезервированных для представления данных особого рода. Для формата плавающей запятой IBM S360 $r = 0$, а для формата IEEE $r = 2$.

Таблица 3.10. Число сдвигов, требуемых для различных оснований чисел с плавающей запятой

Частота сдвигов выравнивания						Частота сдвигов нормализации				
Основа- ние	2	4	8	16	32	2	4	8	16	32
Сдвиги						Нул. рез.→	1,42	1,42	1,42	1,42
						Пере- полн.→	19,65	10,67	6,52	5,50
0	32,61	38,24	45,77	47,32	52,52	59,38	72,11	79,40	82,35	83,86
1	12,11	18,54	19,77	26,02	26,37	6,78	7,96	8,75	7,29	5,99
2	8,61	12,83	11,92	10,47	5,92	3,47	3,35	1,61	1,38	0,87
3	6,72	9,87	6,26	2,24	1,82	2,35	1,49	0,38	1,01	0,88
4	7,17	3,01	1,73	1,31	2,08	1,91	0,34	0,43	1,30	0,41
5	3,88	2,05	1,10	1,70	1,87	1,06	0,14	0,71	0,32	0,88
6	4,39	1,01	0,89	1,24		0,56	0,92	0,25	0,43	
7	4,82	0,72	1,52			0,48	0,18	0,22		
8	1,29	0,63	1,00			0,16	0,13	0,28		
9	1,28	0,94				0,14	0,15			
10	1,31	0,72				0,08	0,18			
11	0,48	0,97				0,09				
12	0,58	0,74				0,32				
13	0,38	0,27				0,55				
14	0,38					0,16				
15	0,32					0,02				
16	0,33					0,04				
17	0,32					0,09				

²⁷⁾ Если результат не нормализован, выравнивание обычно выполняется перед тем, как этот результат используется в качестве входного данного для последующей арифметической операции.

Таблица 3.10. (продолжение)

Основа- ние	Частота сдвигов выравнивания					Частота сдвигов нормализации				
	2	4	8	16	32	2	4	8	16	32
18	0,40					0,08				
19	0,48					0,07				
20	0,36					0,12				
21	0,53					0,07				
22	0,48					0,07				
23	0,33					0,09				
24	0,36					0,11				
25	0,36					0,16				
26	0,19					0,52				
	9,50	9,43	10,04	9,07	9,42	← Избыточный сдвиг				

Для порядка длиной 7 бит диапазон числа при несмещенном порядке составляет от 16^0 до 16^{127} . Если смещение равно $2^{7-1} = 64$, диапазон составит приблизительно от 16^{-65} до 16^{+63} , или, в десятичном выражении, от $5,4 \times 10^{-79}$ до $7,2 \times 10^{+75}$. Ниже в этом разделе смещение порядка обсуждается более детально. В IBM S360 предусмотрены два типа данных с плавающей запятой: с одинарной и двойной точностью. Оба этих формата имеют одинаковый диапазон, но различаются длиной дробной части:

S	7-битовый порядок	24-битовая дробь
---	-------------------	------------------

S	7-битовый порядок	56-битовая дробь
---	-------------------	------------------

Обратите внимание на то, что 1 бит байта порядка и для одинарной, и для двойной точности предназначен для знака всего числа. Оба формата используют одинаковое назначение битов для порядка и, следовательно, одинаковый диапазон.

Под дробную часть отводятся все биты, оставшиеся после выделения битов для порядка и знака. Тип данных IBM S360 использует в дробной части 24 бит для чисел с одинарной точностью и 56 бит для чисел двойной точности. Из-за сложностей в выполнении арифметических операций над мантиссами при использовании целых типов данных в форматах дополнения до 1 или до 2 многие типы данных с плавающей запятой используют дроби со знаком.

В качестве исторической справки заметим, что модели машин IBM S360 для научных применений конкурировали с весьма удачными компьютерами CDC 6600, в которых имелся лишь один тип данных с плавающей запятой длиной 60 бит с основанием 2 и смещенным

порядком длиной 11 бит. Соответственно, диапазон представления составлял от 2^{-1024} до 2^{+1023} , или приблизительно от 10^{-300} до 10^{+300} . Предоставляя столь широкий диапазон в сочетании с 48-битовой дробной частью, компьютеры CDC6600 заслужили уважение пользователей, выполняющих сложные научные расчеты. Используемый формат не вызывает переполнения даже в длинных цепочках вычислений, при этом точность остается вполне удовлетворительной.

🔗 Пример

Определить разницу в числе битов порядка, требуемом для представления одного и того же диапазона при основаниях 2 и 16. Параметр γ положить равным 0.

✍ Решение

$$16^{2^{\text{число битов (16)}}} = 2^{2^{\text{число битов (2)}}}$$

Пусть число битов (16) = 1. Тогда

$$16^{2^1} = 2^8 = 2^{2^{\text{число битов (2)}}$$

$$2^{\text{число битов (2)}} = 8$$

$$\text{число битов (2)} = 3$$

Таким образом, разница в числе битов порядка составит $3 - 1 = 2$.

✓ Замечание

Экономия 2 бит в представлении порядка и передача их дробной части послужила серьезным доводом в пользу выбора основания 16 для типа данных с плавающей запятой в машинах IBM S360.

Представление с основанием 16 требует особых подходов к выравниванию и нормализации. Дробная часть делится на 16-ричные поля по 4 бит каждое. Как правый, так и левый сдвиги нормализации должны выполняться через 4 бит. Таким образом, 24-битовая дробная часть данного одинарной точности в компьютере IBM S360 потребует максимум 6 сдвигов. Каждый сдвиг сопровождается инкрементом или декрементом порядка.

Дробная часть считается нормализованной, если левое, старшее 16-ричное поле не равно 0. Если все 16 состояний 16-ричного поля распределены равномерно, в дробной части старший бит будет недоиспользован на 3/4. В худшем случае потери могут составить 3 бит. Разработчики этого типа данных посчитали, что такую потерю битов можно допустить, учитывая экономию 2 бит порядка (см. приведенный выше пример).

В таблице 3.10 приведены частоты сдвигов для нормализации при различных основаниях. Обратите внимание на то, что частота сдвигов

для нормализации меньше частоты для выравнивания. Например, для входного выравнивания при основании 2 в 32,61% времени требуются 0 сдвигов, и лишь в 1,31% времени требуются 10 сдвигов. При основании 16 в 47,23% случаев требуются 0 сдвигов. В случае нормализации при основании 16 1,42% всех операций дают нулевой результат а 5,5% операций приводят к переполнению.

✎ Пример

Каково среднее взвешенное число сдвигов, требуемых для нормализации при основании 8?

✍ Решение

Число сдвигов основания	Частота	Произведение
Нулевой результат, 0	1,42	0
Переполнение, 0	6,52	0
0	79,40	0
1	8,75	0,085
2	1,61	0,0322
3	0,38	0,0114
4	0,43	0,0172
5	0,71	0,0355
6	0,25	0,0150
7	0,22	0,0154
8	0,28	<u>0,0224</u>
Взвешенное среднее числа сдвигов основания		0,2366

✓ Замечание

Из-за того что 87,34% всех операций не требуют нормализации в силу нулевого результата, переполнения или нулевого сдвига (результат уже нормализован), среднее взвешенное числа сдвигов основания для нормализации оказывается весьма малым (0,2366). Обратите внимание на то, что 89,27% всех операций при основании 16 не требуют нормализации.

Стандарт IEEE. Разработчики стандарта IEEE для чисел с плавающей запятой решили использовать для мантиссы смешанное представление вместо дробного, применив к нему термин *significand* (значащая часть числа, мантисса) [13]. В этом представлении слева от запятой оставлен один бит. Выполнение арифметических операций в этом представлении схоже с привычной нотацией, используемой в ин-

женерных расчетах, с одной десятичной цифрой слева от десятичной запятой. Мантисса определяется в стандарте IEEE как

«компонент двоичного числа с плавающей точкой, который содержит явный или неявный ведущий бит с левой стороны от двоичной запятой, а также поле дробной части с правой стороны».

Тип данных в плавающей запятой в стандарте IEEE назначается двоичному слову, содержащему знак, порядок (смещенное целое) и мантиссу:

S	Порядок	Мантисса
---	---------	----------

В соответствии с определением мантиссы, если число нормализовано, его старший бит равен 1 и является избыточным. Таким образом, ведущая 1 может быть подразумеваемой (неявной); такой бит называют скрытым²⁸⁾. При использовании скрытого бита требуется сохранять лишь дробную часть мантиссы.

При переносе значения с плавающей запятой из арифметического узла в память ведущая 1 отбрасывается (сокрытие скрытого бита), а дробная часть сдвигается на 1 бит влево, увеличивая таким образом точность на 1 бит. Представления в памяти числа 18,5 в формате IEEE и с дробной мантиссой выглядят следующим образом:

IEEE в памяти: $+0,001010 \times 2^4$

IEEE в ALU: $+1,001010 \times 2^4$

Дробная мантисса: $+0,100101 \times 2^5$

При перемещении значения IEEE из памяти в ALU скрытый бит восстанавливается. Использование скрытого бита позволяет арифметическому узлу обрабатывать один дополнительный бит сверх тех, которые хранятся в памяти.

Стандарт IEEE предусматривает три типа данных с плавающей запятой:

одинарная точность: бит знака, 8-битовый порядок, 23-битовая мантисса;

двойная точность: бит знака, 11-битовый порядок, 52-битовая мантисса;

двойная расширенная точность: бит знака, ≥ 15 -битовый порядок, ≥ 63 -битовая мантисса.

Третий тип данных используется в ALU для обеспечения высокой точности и широкого диапазона в сложных вычислениях. При сохранении этот формат должен быть предварительно преобразован в фор-

²⁸⁾ Такой прием был использован Цузе в его Z4 в 1944–1950 гг. Скрытый бит не определен явным образом в спецификации IEEE.

маты одинарной или двойной точности. Значение с двойной точностью может быть снова помещено в ALU в его расширенной форме, чем и достигается повышенная точность для длинной цепочки вычислений.

🔗 Пример

Если для хранения дроби выделено лишь 5 бит, каково будет десятичное значение приведенного выше представления числа 18,5 при помещении в ALU?

✍ Решение

Значение IEEE: $1,00101 \times 2^4 = 1,15625 \times 16 = 18,5$.

Значение с дробной мантиссой: $0,10010 \times 2^5 = 0,56250 \times 32 = 18,0$.

✓ Замечание

Этот простой пример показывает, что формат плавающей запятой IEEE предоставляет дополнительный бит точности.

Порядки типов данных IEEE смещены; характерные значения порядка для одинарной точности приведены в табл. 3.11. Тип данных одинарной точности имеет 8-битовый порядок со смещением на $2^8 - 1 = 127$, а двойной точности на $2^{11} - 1 = 1023$. Два значения порядка, все нули и все единицы, зарезервированы для индикации исключительных результатов; этот вопрос будет рассмотрен в разделе 3.5.3.1.

Таблица 3.11. Значения порядка для одинарной точности IEEE

Порядок	Значение	Смысл
00000000	0	Зарезервировано
00000001	1	2^{-126}
01111111	127	2^0
11111110	254	2^{+127}
11111111	255	Зарезервировано

Сводка типов данных с плавающей точкой

В таблице 3.12 перечислены параметры некоторого количества старых и относительно новых типов данных с плавающей запятой. Тип данных IEEE стал стандартом. Обратите внимание на то, что дроби, или мантиссы, всех включенных в табл. 3.12 представлений плавающей запятой, используют нотацию знака и величины. В результате при

сложении, вычитании, умножении или делении мантисс операции должны следовать обычным арифметическим правилам, обсуждаемым ниже в разделе 3.4.

Таблица 3.12. Типы данных с плавающей запятой

Процессор	Основа- ние	Система счисления	Представле- ние порядка	Мантис- са	Представле- ние мантиссы
Цузе Z4	2	2	Дополн. до 2, 7 бит	Смешан.	3В 22 бит+знак↓
IBM 650	10	10, код 2/5	Смещен. 2 дес. цифры	Дробь	3В 8 дес. цифр+ знак (в дес. цифр.)
IBM S360, одинарн.	16	2	Смещен. 7 бит	Дробь	3В 24 бит+знак
IBM S360, двойн.	16	2	Смещен. 7 бит	Дробь	3В 56 бит+знак
VAX	2	2	Смещен, 8 бит	Дробь	3В 23 бит+знак↓
IEEE, одинарн.	2	2	Смещен. 8 бит	Смешан.	3В 23 бит+знак↓
IEEE, двойн.	2	2	Смещен. 11 бит	Смешан.	3В 52 бит+знак↓
IEEE, двойн. расш.	2	2	Смещен. ≥15 бит	Смешан.	3В≥63бит+знак↓

Примечания: ↓ скрытый бит, 3В — знак и величина.

✎ Пример

Сравнить основания 2 и 256. Сколько битов будет сэкономлено в порядке и потеряно в мантиссе во втором случае? Считать, что скрытый бит не используется.

✍ Решение

При основании 256 мантисса состоит из 8-битовых полей. Таким образом, при нормализации в худшем случае будет потеряно 7 бит. В среднем будет теряться 4 бит. В системе с основанием 2 биты теряться не будут. Следуя решению одного из предыдущих примеров, находим, что для того же диапазона в порядке экономятся 3 бит.

✓ Замечание

С точки зрения экономии или потери битов, основание 256 представляется неудачным выбором: при потере 4 бит в дробной части экономятся 3 бит в порядке.

Логические типы данных

Во многих компьютерах определены логические, или булевы, типы данных, которые представляют собой массивы битов первичных типов данных — байтов, полуслов, слов или двойных слов. Такие битовые массивы используются для записи двоичных событий или состояний. Логические типы данных используются главным образом как вспомогательные внутри программы и редко выступают для программы в качестве входных или выходных.

S360 позволяет присваивать логические значения любому типу данных — байту, полуслову, слову или двойному слову. В процессоре Pentium Pro допустимо выделять битовые поля любой длины, как это показано на рис. 3.11. Адрес указывает на начало поля, а длина определяет число битов в поле.

3.2.3.2. Типы данных для управления программой

Некоторые архитектурные типы данных служат для управления ходом выполнения программы. К этим типам данных относятся программный счетчик и биты индикации.

Программный счетчик

В современных компьютерах программа хранится в последовательных ячейках памяти, доступ к которым осуществляется с помощью программного счетчика. Тип данных программного счетчика представляет собой целое без знака, длина которого во многих архитектурах совпадает с длиной одного из первичных типов данных. Иметь длину типа данных программного счетчика такую же, как и у прикладного типа данных, удобно, так как в этом случае программный счетчик можно сохранять в памяти при контекстном переключении²⁹⁾. Программный счетчик процессора Pentium Pro представляет собой 32-битовый регистр, называемый расширенным указателем команд (EIP).

В компьютерах IBM S360 предложен другой подход. Место для программного счетчика (который носит название адреса команды) предусмотрено в 64-битовом слове состояния программы (program status word, PSW), двойном слове, которое удобно хранить в памяти. Программные счетчики машин S360 и S370 имеют длину 24 бит, в то время как S390 использует 31-битовый программный счетчик.

При реализации переходов эффективный адрес точки перехода должен быть загружен в программный счетчик. Существуют три формы эффективных адресов точек перехода. В приведенном ниже переч-

²⁹⁾ Некоторые однокристалльные микрокомпьютеры не следуют этому соглашению. Длина программного счетчика устанавливается такой, чтобы он мог адресовать встроенное ПЗУ с программой. Специальный аппаратный стек обеспечивает сохранение содержимого программного счетчика (адреса возврата) при вызовах подпрограмм или при обслуживании прерываний.

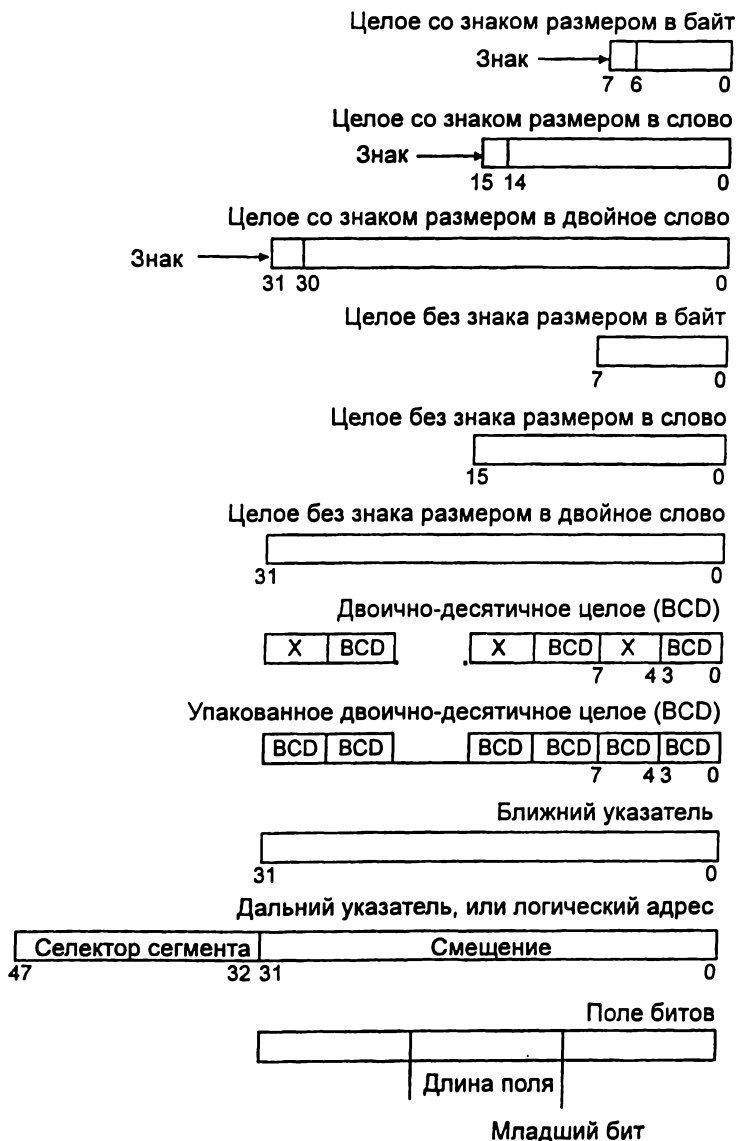


Рис. 3.11. Типы данных процессора Pentium Pro (с любезного согласия корпорации Intel).

не смещение — это специально предусмотренное поле смещения команды перехода, а PC — обозначение программного счетчика.

Прямой переход

PC ← (смещение)

Переход относительно содержимого PC

PC ← (PC) + (смещение)

Переход относительно базы

PC ← (база) + (смещение)

✎ Пример

Что происходит, когда смещение прибавляется к содержимому РС или к значению базы и при этом возникает перенос из старшего разряда?

✎ Решение

Имеется два способа решения этой проблемы на этапе проектирования. Во-первых, можно при обнаружении переноса возбуждать сигнал прерывания, указывающий на ошибку. Во-вторых, можно реализовать свертывание эффективного адреса, и результат свертки использовать как адрес перехода на программу реакции.

✓ Замечание

Разработчик должен обеспечить предсказуемое поведение процессора при всех ситуациях такого рода.

Биты индикации

Если рассматривать биты индикации как тип данных, то это просто биты. Они приобретают характерное значение в силу особенностей их установки и сброса. В разделе 3.2.2 уже обсуждалось использование битов индикации командами условных переходов. Вспомним, что биты индикации позволяют передать некоторое условие следующей команде, которая анализирует эти биты и в зависимости от их состояния осуществляет переход. Биты индикации устанавливаются и сбрасываются двумя способами:

- как побочный результат команды,
- явным образом как первичное действие команды.

Приведем пример записи в качестве побочного эффекта. Во всех арифметических командах выходная шина ALU проверяется на ноль; если на ней установлен ноль, бит индикации устанавливается в 1, в противном случае он устанавливается в 0. Некоторые команды явным образом выполняют запись в бит индикации. Так, команда сравнения Pentium Pro выполняет следующее:

CMR — сравнение двух операндов. Сравнение содержимого двух регистров и установка флага состояния в соответствии с результатом.

В большинстве ранних компьютеров, в частности в архитектуре фон Неймана, команды условных переходов относятся к командам типа «анализ и переход» и не используют биты индикации с целью

упреждающей передачи информации о состоянии³⁰⁾. Насколько известно автору, впервые биты индикации были использованы в IBM 704, установленной в 1955 г. В этой машине входной байт анализируется аппаратно, и его бит четности записывается в бит Р. Этот бит хранит условие четности до следующей операции ввода, которая перезаписывает бит индикации. Таким образом, этот бит должен быть использован до того, как его состояние изменится следующей командой. Все остальные команды переходов в этом компьютере были типа «анализ и переход»; они анализировали состояние различных регистров.

Современный метод использования битов индикации был впервые применен в компьютере IBM Stretch, установленном в 1961 г. В этой машине 64 бит индикации были сгруппированы в аппаратно реализованное двойное слово, которое, помимо возможности его анализа командами условных переходов, могло сохраняться и восстанавливаться при переключении контекста. Семь битов этого слова фиксировали результат целочисленных операций ALU в качестве побочного эффекта: отрицательный, нулевой, положительный, отрицательный со знаком, меньше, равен, больше. Десять битов фиксировали результаты операций с плавающей запятой и исключения.

Признаки результата IBM S360. В этой машине идея архитектурных битов индикации, реализованная еще в IBM Stretch, была использована в урезанном виде. Архитектурное PSW, наряду с другой информацией, содержит признаки результата. Разработчики S360 решили, что индикаторы машины Stretch можно закодировать, сэкономив таким образом биты PSW и освободив место для программного счетчика и другой управляющей информации.

Два бита используются для кодирования четырех признаков; закодированные индикаторы называются признаками результата (condition codes). Результаты записываются как побочные эффекты и являются контекстно-независимыми. Выборочный пример кодирования результатов приведен в табл. 3.13. Команды условных переходов содержат 2-битовое поле, которое сравнивается с 2-битовым признаком результата. Например, если после выполнения команды сложения требуется проанализировать результат на ноль, 2 бита в команде будут равны 00. Если желателен переход по переполнению, биты будут равны 11.

Система IBM S360 легко приводит к программным ошибкам. Пусть, например, выполнена команда сложения; затем сохраняется таймер. Признаки результата выполнения команды сложения уничтожаются. Переход по 00 теперь будет означать, что таймер установлен, в то время как мы предполагали выполнить переход, если результат предыдущего сложения не равен 0.

³⁰⁾ В фон-неймановской архитектуре возможен переход по условию: предварительно вычисленное значение в аккумуляторе ≥ 0 . Аналогично этому релейный калькулятор Bell Model V мог выполнить условный переход по знаку предварительно вычисленного значения.

Таблица 3.13. Признаки результата IBM S360

Команда	Значение признака результата			
	00	01	10	11
Сложение логическое	0, нет переноса	Не 0, нет переноса	0, перенос	—
Вычитание логическое	=	Не 0, нет переноса	0, перенос	Не 0, перенос
Анализ по маске	0	Смесь	—	Единицы
Сложение	0, нет переноса	Не 0, нет	0, перенос	—
Сложение	0	<0	>0	Переполнение
Пересылка длинная	Счетчик равен	Счетчик меньше	Счетчик больше	Перекрытие приемника
Сравнение плавающей запятой	Равно	Меньше	Больше	—
Сложение десятичное	0	<0	>0	Переполнение
Сохранение таймера	Установлен	Не установлен	Ошибка	Неработоспособен

Биты индикации в архитектуре Intel. В архитектуре Intel биты индикации называются флагами; мы будем их называть так же. Первый продукт семейства x86 имел четыре флага реализации, которые устанавливались или сбрасывались как побочный результат других операций; были предусмотрены флаги переноса, нуля, положительного результата, четности. Ввиду того что флаги относились к средствам реализации, их значения нельзя было сохранить и восстановить при контекстном переключении.

Семейство x86 развивалось; были разработаны процессоры i8080 и i8086. В процессоре i286 впервые появились аппаратные флаги. В этой модели под 16 флагов был выделен регистр, который можно было сохранить и восстановить при переключении контекстов. Реально, однако, использовались лишь 12 флагов; 4 бита были оставлены для будущих расширений.

На рисунке 3.12 показаны флаги Intel Pentium Pro. Под них выделено 32-битовое слово. Обратите внимание на то, что имеются три класса флагов: состояния, управления и системные. Флаг вспомогательного переноса используется для выполнения десятичной коррекции после десятичного сложения. Флаг контроля выравнивания появился потому, что в семействе x86 память адресуется по байтам. Если, например, адресуется слово (4 байт), но адрес не выровнен на границу слова (два младших бита адреса $\neq 0$), возникает прерывание,

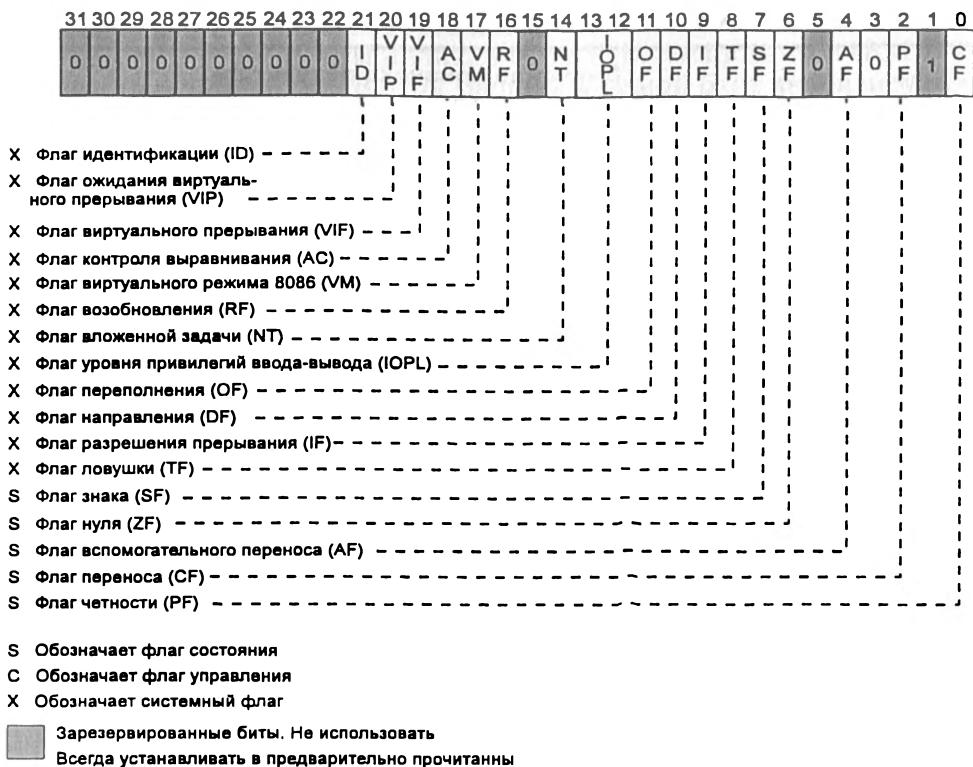


Рис. 3.12. Флаги процессора Pentium Pro (с любезного разрешения Intel Corporation).

и устанавливается флаг AC. Обработчик прерывания анализирует флаги с целью определения причины прерывания.

Все флаги состояния x86 устанавливаются и сбрасываются как результат выполнения команд. Системные флаги смешаны: некоторые, например флаг виртуального режима VM, устанавливаются явным образом операционной системой, другие, например флаг переполнения OF, устанавливаются как побочный результат выполнения прикладных команд.

Пример

Каким образом с помощью флагов Pentium Pro можно проверить условие $(r1) \geq (r2)$?

✍ Решение

Содержимое регистров вычитается одно из другого, и в качестве побочного результата устанавливаются флаги. Следующая команда анализирует состояние флагов. В данном примере нас прежде всего интересует побочный результат команды.

Шаг 1. $(r1) - (r2) =$

- а) если результат отрицателен (так будет, если $r1 < r2$), флаг знака устанавливается в 1;
- б) если результат равен нулю, флаг нуля устанавливается в 1.

Шаг 2. Команды условных и логических переходов могут с помощью этих двух флагов определить, какой из четырех возможных результатов имеет место. Первый флаг указывает на + или —, а второй — на $\neq$ или $=$. После выполнения регистровой операции вычитания два бита индикации могут находиться в одном из следующих четырех состояний:

00	Отрицательно, не 0	$(r1) < (r2), (r2) \neq (r1)$
01	Отрицательно, 0	$(r1) < (r2), (r2) = (r1)$
10	Положительно, не 0	$(r1) > (r2), (r2) \neq (r1)$
11	Положительно, 0	$(r1) > (r2), (r2) = (r1)$

Если $(r1) = 5$, а $(r2) = 3$, вычитание дает положительный и ненулевой результат, поэтому флаги устанавливаются (как побочный результат операции) в состояние 10. Для того чтобы проверить условие $\geq$, следует выполнить над обоими флагами операцию ИЛИ. Для данного примера ИЛИ над битами 10 дает 1, и $5 \geq 2$.

Краткие итоги обсуждения битов индикации

Биты индикации устанавливаются или сбрасываются как побочный результат операции или явным образом. Индикаторы могут быть битами реализации или архитектурными. В таблице 3.14 показаны пять вариантов организации системы битов индикации. Обратите внимание на то, что в системе команд обычно используется сочетание различных вариантов. Очень немногие процессоры ограничиваются лишь одним из них.

3.2.4. Состав команд

В формат команды входят три компонента ISA: операция, адреса и типы данных. Проектирование и размещение в команде кодов операций и адресов будут рассмотрены в следующих разделах. В общем случае типы прикладных данных, за исключением непосредственных целых, не входят в состав команды.

При назначении отдельных полей команды приходится идти на ряд компромиссов. Например, длина команды должна быть кратна разме-

Таблица 3.14. Системы индикаторов

Тип перехода	Индикаторы	Как устанавливаются?	Сохраняются ли при переключении контекста?
Анализ и переход	Отсутствуют	—	Нет
Ортогональный	Реализации	Побочный результат	Нет
Ортогональный	Реализации	Явно	Нет
Ортогональный	Архитектурные	Побочный результат	Да
Ортогональный	Архитектурные	Явно	Да

меру адресуемых элементов. Поле для кода операции должно давать возможность организации богатого набора операций при минимальной длине. Число битов, выделяемое под адреса регистрового файла, определяет его размер. Адреса памяти, которые при прямой адресации могут быть значительной длины, должны быть каким-то образом уменьшены в размерах.

В следующих разделах рассматривается вопрос о назначении битов и полей команды. Целью проектирования должно быть достижение баланса между противоречивыми требованиями.

Коды операций

Выделение в команде места под коды операции может выполняться различными способами. Целью проектирования является уменьшение числа битов в команде (экономное распределение битов) при обеспечении большого числа кодов операций, реализующих богатый набор команд. Для удовлетворения этих требований используются три различных способа.

1. Коды операций фиксированной длины назначаются командам с переменной длиной (IBM S360).
2. Коды операций переменной длины, отвечающие расширениям кодов операций, назначаются командам переменной длины (Intel x86).
3. Коды операций переменной длины, отвечающие расширениям кодов операций, назначаются командам фиксированной длины (MIPS R2000 и IBM Power PC).

Мы рассмотрим все эти три способа путем изучения ISA перечисленных выше процессоров.

IBM S360. Целесообразность разработки команд переменной длины обосновывается возможностью наиболее часто используемые команды сделать короткими, а редко используемые команды — длинными, как в азбуке Морзе. В IBM S360 имеются команды длиной

16, 32 и 48 бит. Однако коды операций для команд любого размера имеют одинаковую длину 1 байт.

Внутри 1-байтового кода операции четыре старших бита определяют тип данных, а четыре младших — операцию. В таблице 3.15 показаны эти соответствия. Например, если четыре старших бита кода операции составляют 0111, подразумевается, что тип данных короткий (одинарной точности) с плавающей запятой. Поскольку такой тип данных занимает 32 бит, а память IBM S360 адресуется по байтам, два младших бита эффективного адреса данного в процессе доступа к памяти игнорируются. Для выполнения операции используется ALU плавающей запятой.

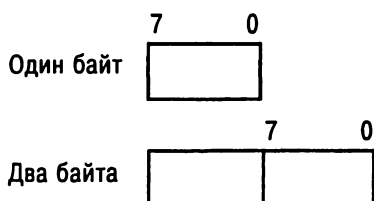
Четыре младших бита определяют выполняемую операцию. Обратите внимание на щедрое резервирование кодов для будущих типов данных. Разработчики предусмотрели декодирование неиспользуемых кодов операций с возбуждением системного прерывания, что предотвращает попытки выполнения неправильных кодов операций.

Таблица 3.15. Классы кодов операций IBM S360

Старшие биты кода операции	Соответствие
0000	Переход и переключение состояния
0001	Фиксированная запятая, полное слово и логический тип
0010	Плавающая запятая длинная
0011	Плавающая запятая короткая
0100	Фиксированная запятая, полслова и переход
0101	Фиксированная запятая, полное слово и логический тип
0110	Плавающая запятая длинная
0111	Плавающая запятая короткая
1000	Установка системной маски
1001	Фиксированная запятая, логический тип и ввод-вывод
1010	Не определено
1011	Не определено
1100	Не определено
1101	Логический тип
1110	Не определено
1111	Десятичный тип

Intel x86. В первом процессоре архитектуры x86, микросхеме 8008, использовались однобайтовые коды операций. Разработка процессора 8080 потребовала расширения набора команд, что привело к включению в код операции дополнительного байта, который вместе с первым байтом образовал так называемый расширенный код опера-

ции. В этом случае зарезервированные сочетания битов в первом байте кода операции приводят к выборке и декодированию второго байта:



1-байт код операции может быть декодирован в 256 различных операций. Однако в процессоре x86 два кода первого байта являются зарезервированными: они требуют выборки и декодирования второго байта. Это коды 0F и 1F (16-ричная система). Код 0F говорит о наличии второго байта команды с целочисленным операндом, в то время как код 1F указывает на наличие второго байта команды с плавающей запятой. Имеются и другие расширения, не описываемые здесь, но суть дела должна быть понятна и без них³¹⁾.

Примеры

1. С учетом вышесказанного, сколько кодов операций можно реализовать с помощью метода, использованного в x86?

Решение

Число кодов операций составляет

$\# \text{ кодов операций} = \# \text{ 1-байтовых кодов} + \# \text{ 2-байтовых кодов} = 256 - 2 + 2 \times 256 = 254 + 512 = 766 \text{ кодов операций,}$

где $\#$ — количество.

Замечание

При назначении коду операции 2 байт полное число кодов операций составит $2^{16} = 65536$. Таким образом, число возможных кодов существенно уменьшено ради того, чтобы часто используемые команды были короткими.

³¹⁾ Когда проектировщики процессора Intel 8080 расширяли набор кодов операций 8008, они выбились из графика работ и оставили некоторые коды операций без назначения. Эта недоработка послужила ключевым моментом при дальнейшем расширении архитектуры Intel.

2. Для кодирования 766 команд вышеприведенного примера требуется (в случае команд фиксированной длины) 10 бит. Какова должна быть доля однобайтовых команд среди всех выполняемых команд, чтобы использование расширенных кодов операций привело к такой же эффективности, как и использование команд с фиксированной длиной 10 бит?

Решение

Обозначим долю 1-байтовых команд через s . Задача решается нахождением такого значения s , которое даст взвешенную среднюю длину команды, равную 10 байт:

$$10 = 8s + (1 - s) \times 16$$

$$-6 = -8s$$

$$s = (6/8) = 0,75$$

✓ Замечание

Если более 75% выполняемых команд имеют длину 1 байт, а менее 25% имеют длину 2 байт, расход битов будет меньше, чем в случае использования 10-битовых кодов операций.

MIPS R2000. При использовании 32-битовых команд фиксированной длины 6 бит были выделены под коды операции. Это дает 64 различных кода операций, что недостаточно для реализации требуемого набора команд.

Проблема была решена путем использования в составе команд с фиксированной длиной кодов операций переменной длины, построенных по методу расширенных кодов операций. Наиболее часто используемые операции закодированы непосредственно в 6-битовом коде операции, в то же время небольшое количество из возможных 64 кодов зарезервировано в качестве «кодов выхода» (escape codes), которые для получения полного кода операции требуют декодирования большего числа битов команды. В этом случае длина команды не увеличивается, как это имеет место в процессорах x86.

Взгляните на рис. 3.13. Под коды операций отведены биты 26–31; исходное кодирование операций приведено в верхней части рисунка. Например, код 000011 соответствует команде перехода JAL. Если код операции составляет 000000, вызывается одна из специальных команд класса SPECIAL, в код операции которой входят биты 0–5 всего поля команд. Эти 6 бит, записанные в формате R, декодируются, как это показано в таблице SPECIAL рис. 3.13. Исходный код BCOND расширяется битами 16–20 поля команд, в то время как COP0 расширяется битами 0–4. Коды COP1, 2 и 3 расширяются битами 16–25.

PowerPC. Другой пример системы расширенных кодов операций можно найти в ISA PowerPC. Этот процессор предусматривает 32-битовые команды, в которых 6 бит (биты 0–5) выделены под коды опе-

28..26		Код операции							
31...29		0	1	2	3	4	5	6	7
0		SPECIAL	BCOND	J	JAL	BEQ	BNE	BLEZ	BGTZ
1		ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2		COP0	COP1	COP2	COP3	⬥	⬥	⬥	⬥
3		⬥	⬥	⬥	⬥	⬥	⬥	⬥	⬥
4		LB	LH	LWL	LW	LBU	LHU	LWR	⬥
5		SB	SH	SWL	SW	⬥	⬥	⬥	⬥
6		LWC0	LWC1	LWC2	LWC3	⬥	⬥	⬥	⬥
7		SWC0	SWC1	SWC2	SWC3	⬥	⬥	⬥	⬥

2..0		SPECIAL							
5...3		0	1	2	3	4	5	6	7
0		SLL	⬥	SRL	SRA	SLLV	⬥	SRLV	SRAV
1		JR	JALR	⬥	⬥	SYSCALL	BREAK	⬥	⬥
2		MFHI	MTHI	MFLO	MTLO	⬥	⬥	⬥	⬥
3		MULT	MULTU	DIV	DIVU	⬥	⬥	⬥	⬥
4		ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5		⬥	⬥	SLT	SLTU	⬥	⬥	⬥	⬥
6		⬥	⬥	⬥	⬥	⬥	⬥	⬥	⬥
7		⬥	⬥	⬥	⬥	⬥	⬥	⬥	⬥

18..16		BCOND							
5..3		0	1	2	3	4	5	6	7
0		BLTZ	BGEZ						
1									
2		BLTZAL	BGEZAL						
3									

25..23		COPz							
22,21,16		0	1	2	3	4	5	6	7
0,0,0		MF	MT	BCF		CO			
0,0,1				BCT					
0,1,0									
0,1,1									
1,0,0		CF	CT						
1,0,1									
1,1,0									
1,1,1									

2..0		COP0							
4..3		0	1	2	3	4	5	6	7
0			TLBR	TLBWI				TLBWR	
1		TLBP							
2		RFE							
3									

⬥ Коды операции, помеченные этим знаком, возбуждают исключения зарезервированной команды и оставлены для будущих вариантов архитектуры.

Рис. 3.13. Карта команд MIPS R2000 (с любезного разрешения MIPS Technologies Inc.).

раций. Часто используемые команды, такие, как `addi` (`add immediate`, непосредственное сложение), кодируются с помощью 6 бит. Команда `addi` имеет код `001110` и содержит 16-битовый непосредственный операнд, размещаемый в битах 16–31. Другой регистр-источник `A`, а также регистр-приемник `D` адресуются двумя 5-битовыми полями, поскольку этот процессор содержит 32-словный регистровый файл:

	0	5	6	10	11	15	16	30
<code>addi</code>	14		D		A			Непосредств. операнд

Редко используемые команды включают расширенный код операции, что увеличивает их длину. Код выхода составляет 31_{10} . Приведенные ниже команды PowerPC `addx`, `addcx` и `addex` иллюстрируют использование расширенных кодов операций:

	0	5	6	10	11	15	16	20	22	30
<code>addx</code>	31		D		A		B			256
<code>addcx</code>	31		D		A		B			10
<code>addex</code>	31		D		A		B			139

Каждая из этих трех команд имеет один и тот же код операции в битах 0–5, но для расшифровки полного кода операции требуется также учесть содержимое битов 22–30.

Адреса регистров

Адреса регистров включаются в формат команды непосредственно. Здесь следует учесть размер регистрового файла, а также количество используемых адресов: 2 или 3. В наши дни обычно используются регистровые файлы, состоящие из 32 слов и требующие 5-битового адреса. Таким образом, трехадресный формат требует для указания адресов регистров 15 бит, а двухадресный – 10.

Адреса памяти

После того как в коде команды выделены поля для кода операции и адресов регистров, обычно остается всего несколько битов для указания адреса(ов) памяти. По этой причине во многих процессорах используются две стратегии, позволяющие уменьшить требуемое число битов для указания адреса памяти. Первая стратегия заключается в использовании либо аккумулятора, либо загрузки-запоминания (т. е. считывания-записи). Эти архитектуры, IBM S360 и x86, нуждаются лишь в одном адресе памяти. Вторая стратегия сводится к выделению в команде только смещения, которое используется для создания бо-

лее длинного эффективного адреса. Эти стратегии могут комбинироваться, как показано в табл. 3.16.

Таблица 3.16. Адреса памяти

Процессор	Биты в команде	Эффективный адрес
IBM S360	12-битовое смещение	Индекс+база+смещение
Intel Pentium	8-, 16-, 32-битовое смещение	Индекс+база+смещение
MIPS R2000	16-битовое смещение	Индекс+смещение
PowerPC	Нет	(RA) + (RB)

Другие поля

В формат команды должны входить и другие поля. В качестве примеров можно указать на счетчик сдвигов, маски и биты режима. Во многих случаях задачей проектировщика является минимизация числа битов, поэтому он должен исследовать взвешенное среднее различных вариантов. Например, компьютер с 32-битовой длиной слова может использовать до 5 бит в качестве счетчика сдвигов. Однако исследования показывают, что в 90% всех случаев сдвиг осуществляется на величину 4 бит. Таким образом, вполне возможно использовать 2-битовый счетчик; для более длинных сдвигов программа может использовать команды многократных сдвигов.

3.3. Архитектурные стили CISC и RISC

В разделе 3.1 мы коснулись лексических уровней ISA. Настоящий раздел посвящен специфике архитектурных стилей CISC и RISC. CISC-архитектура находится на более высоком лексическом уровне по сравнению с RISC-архитектурой и выполняет одной командой более одной операции. RISC-архитектура использует простые команды, перекладывая сложности на программу. В результате CISC-архитектура для выполнения того же функционального алгоритма требует меньшего числа команд, чем RISC-архитектура. Это явление вместе с его экспериментальным подтверждением обсуждается в разделе 3.5.

Имеется ряд важных характеристик, в которых проявляется различие CISC- и RISC-архитектур. Эти различия обсуждаются на примерах процессоров x86, MIPS и R2000:

- 1) размер и функции регистрового файла,
- 2) обращение к памяти для чтения и записи,
- 3) число операций, выполняемых одной командой,
- 4) переменная или фиксированная длина команд,
- 5) число типов данных.

CISC-архитектура

Архитектура Intel может служить убедительным примером CISC-архитектуры с замечательным рыночным успехом. Последующее изложение не претендует на полное описание архитектуры Intel; для получения более детальной информации обращайтесь в Интернет. MMX-расширения этой архитектуры будут описаны в разделе 8.7.

Размер и функции регистрового файла. Регистровая архитектура прежде всего базируется на одноадресном аккумуляторе, который был описан выше. Помимо аккумулятора (регистр A: AL, AH, AX и EAX) имеется еще семь дополнительных выделенных регистров, используемых для формирования эффективного адреса, а также как место для временного хранения. Карта регистров общего назначения процессора Pentium Pro была приведена на рис. 3.8.

Первым представителем архитектуры x86 был процессор 8008, выпущенный в 1971 г.³²⁾ В этом процессоре использовались только целый 8-битовый тип данных и четыре 8-битовых регистра (A, B, C и D). В процессоре i286 эти регистры расширились до 16 бит и появились четыре дополнительных 16-битовых регистра (BP, SI, DI и SP). Процессор i486 увеличил длину регистров до 32 бит, как это показано на рис. 3.8.

Обращение к памяти с целью чтения и записи данных. На рисунке 3.14 показаны некоторые из команд пересылки. Обратите внимание на то, что в составе команд процессора отсутствуют команды пересылки из памяти в память; это действие можно выполнить с помощью двух команд пересылки с использованием регистра общего назначения. Некоторые из арифметических команд неявно используют аккумулятор и требуют размещения другого операнда в памяти. Однако имеются и арифметические команды формата регистр-регистр, которые не требуют обращения к памяти. Такие арифметические команды были предусмотрены в микропроцессоре 8008 для обеспечения адресной арифметики, используемой при вычислении эффективных адресов памяти.

Число операций, выполняемых одной командой. Многие команды выполняют более одной элементарной функции. Например, команда F2XM1 выполняет операцию

$$ST \leftarrow (2^{ST} - 1),$$

где через ST обозначена вершина стека с плавающей запятой. Некоторые из команд пересылки носят условный характер; их выполнение зависит от состояния флагов, установленного предыдущей командой. Флаги Pentium Pro были показаны на рис. 3.12.

³²⁾ Та архитектура, которая стала известна под именем x86, была разработана инженерами Computer Terminal Corporation, г. Сан-Антонио, штат Техас. Компании Intel и Texas Instruments разработали свои микропроцессоры, реализующие эту спецификацию набора команд, причем Texas Instruments сделала это первой.

Тип пересылки данных	Источник → Приемник
Из памяти в регистр	Ячейка памяти → Регистр общего назначения Ячейка памяти → Сегментный регистр
Из регистра в память	Регистр общего назначения → Ячейка памяти Сегментный регистр → Ячейка памяти
Из регистра в регистр	Регистр общего назначения → Регистр общего назначения Регистр общего назначения → Сегментный регистр Сегментный регистр → Регистр общего назначения Регистр общего назначения → Управляющий регистр Управляющий регистр → Регистр общего назначения Регистр общего назначения → Регистр отладки Регистр отладки → Регистр общего назначения
Непосредственные данные в регистр	Непосредственные данные → Регистр общего назначения
Непосредственные данные в память	Непосредственные данные → Ячейка памяти

Рис. 3.14. Команды пересылки Pentium Pro (с любезного разрешения Intel Corporation).

Другие команды содержат условие внутри себя. Так, команда **CMPSCHG** сравнивает два операнда-источника (один из которых должен быть в регистре **AL**, **AX** или **EAX** в зависимости от размера типа данных). Если операнды равны, второй операнд загружается в регистр-приемник; в противном случае содержимое приемника загружается в **AL**, **AX** или **EAX**.

Переменная или фиксированная длина команд. В CISC-архитектуре для сокращения бюджета битов и интенсивности побитовых пересылок (трафика битов) используются команды переменной длины. Это дает возможность кодировать простые команды, не требующие обращения к памяти (типа команды **ADD** формата регистр-регистр), одним байтом. Для более сложных команд, например, требующих три

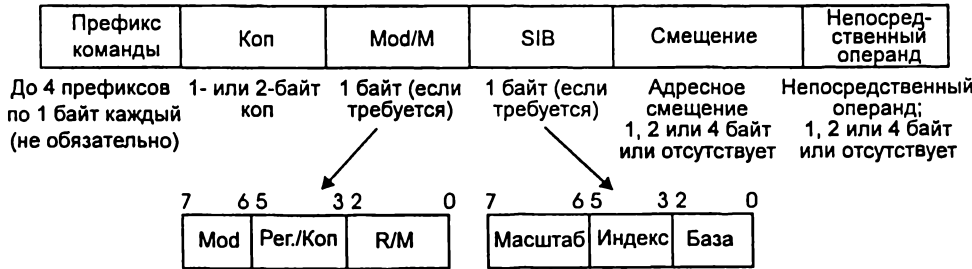


Рис. 3.15. Формат команд Pentium Pro (с любезного разрешения Intel Corporation). Коп – код операции.

адреса памяти, длина команды увеличена, чтобы дать место для этих адресов. Другими словами, длина команды соответствует ее сложности.

На рисунке 3.15 показан формат команд Pentium Pro. Обратите внимание на то, что минимальная длина команды составляет 1 байт. Использование команд переменной длины ведет к ряду сложностей при их реализации, одна из которых заключается в том, что модули памяти обычно имеют логическую ширину, составляющую степень двух, например 4 байт или 8 байт. В потоке команд переменной длины некоторые команды будут размещаться в двух или более словах памяти, что потребует для выборки команды двух или более обращений к памяти. Для устранения этого недостатка в процессоре Pentium Pro предусмотрены специальные аппаратные средства.

✎ Пример

В последовательности команд x86 команды имеют следующую длину: первая команда занимает 3 байта, вторая — 2 байта, третья — 3 байта, четвертая — 1 байт, пятая — 7 байт, шестая — 2 байта, седьмая — 1 байт и восьмая — 8 байт. Модули памяти и шина имеют ширину 4 байта. Каково взвешенное среднее числа циклов памяти в расчете на одну команду, требуемое для извлечения команды из потока команд?

✍ Решение

Ниже приведена карта последовательности команд, на которой для каждого байта памяти указан номер команды, к которой он относится. 8 команд потребуют 7 циклов памяти, что дает $7/8 = 0,875$ циклов памяти на команду.

Слова памяти, 4 байт			
1	1	1	2
2	3	3	3
4	5	5	5
5	5	5	5
6	6	7	8
8	8	8	8
8	8	8	

ки других регистров, предоставления нулевой константы и поддержки адресной арифметики. Два отдельных 32-битовых регистра используются в операциях умножения (для хранения произведения удвоенной длины) и деления (для частного и остатка). В качестве отдельного архитектурного регистра выступает программный счетчик.

Обращение к памяти для чтения и записи. MIPS R2000, как и все RISC-процессоры, используют архитектуры загрузки-запоминания. Это означает, что единственными командами, адресующими память, являются команды загрузки и запоминания (считывания и записи). Все остальные команды получают свои операнды из регистрового файла и туда же отправляют результаты³³⁾. Команды загрузки и запоминания требуют двух обращений к памяти (для выборки команды и данного) и могут использовать в качестве операндов знаковые и беззнаковые байты, слова (2 байт) и двойные слова.

Команды MIPS R2000 используют трехадресный формат регистр-регистр. Другие RISC-процессоры ради экономии адресных битов команды используют двухадресный формат регистр-регистр.

Число операций, выполняемых одной командой. Как и в других RISC-архитектурах, в MIPS R2000 используются простые команды. Форматы команд показаны на рис. 3.17. Обратите внимание на наличие трех видов команд: I-, J- и R-типа.

I-тип: эти команды выполняют загрузку и запоминание, а также арифметические и логические операции с непосредственными операндами.

J-тип: эти команды выполняют операции переходов.

R-тип: эти команды выполняют арифметические и логические операции с двумя регистрами в качестве источников и одним регистром в качестве приемника.

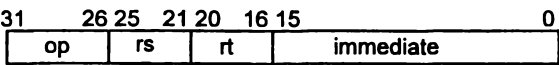
В отличие от архитектуры x86, в ISA MIPS R2000 нет составных команд. Однако команды R-типа в качестве побочного результата устанавливают биты индикации, которые используются последующими командами переходов для управления ходом программы.

Переменная или фиксированная длина команд. Как видно из рис. 3.17, в MIPS R2000 используются 32-битовые команды фиксированной длины. Учитывая, что под код операции отведено 6 бит, можно подумать, что в архитектуре реализованы всего 64 команды. Однако даже для RISC-архитектуры такой сокращенный набор команд недостаточен для обеспечения практического программирования. Для расширения возможностей процессора используется метод расширенных кодов операций; особенности такого метода были рассмотрены ранее.

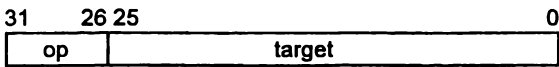
Система команд фиксированной длины обладает рядом преимуществ. Упрощается выборка команд из памяти посредством шины

³³⁾ Первым известным автором воплощением архитектуры загрузки-запоминания явилась машина CDC 6600, разработанная Сеймуром Креем и построенная в 1964 г.

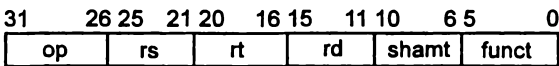
I-тип (непосредственный операнд)



J-тип (переходы)



R-тип (регистры)



Здесь:

op	6-битовый код операции
rs	5-битовый специфик. регистра-источника
rt	5-битовый регистр мишени (источник или приемник) или условие перехода
immediate	16-битовое непосредственное значение, смещение перехода или адрес смещения
target	26-битовый адрес мишени перехода
rd	5-битовый специфик. регистра-приемника
shamt	5-битовое число сдвигов
funct	6-битовое функциональное поле

Рис. 3.17. Форматы команд MIPS R2000 (с любезного разрешения MIPS Technologies Inc.).

фиксированной ширины. Поскольку 32-битовое слово не может оказаться разрезанным границей страниц, в процессе выборки команд не возникает отказов страниц. Декодирование команд и в MIPS R2000, и в x86 выполняется сходным образом, так как обе системы команд требуют предварительного декодирования, чтобы выяснить, нужно ли декодировать следующее поле.

Число типов данных. Число прикладных типов данных в MIPS R2000 невелико по сравнению с типами данных x86. Данные могут быть:

формата IEEE с плавающей запятой одинарной и двойной точности,

8-, 16- и 32-битовые целые со знаком в виде дополнения до 2,

8-, 16- и 32-битовые целые без знака.

Память MIPS R2000 адресуется побайтно. Поэтому в адресе 16-битового целого младший бит игнорируется. При использовании 32-битовых целых данных, а также данных с плавающей запятой одинарной точности игнорируются 2 младших бита адреса. Наконец, при обра-

шении к типам данных с плавающей точкой двойной точности игнорируются 3 младших бита адреса. Привязка адресов к кодам операций выполняется в аппаратуре декодирования команд.

3.4. Статическая и динамическая статистики команд

Выше уже говорилось о битовом распределении (бюджете) и трафике битов. Бюджет битов измеряется числом битов, байтов, слов или команд, требуемых для представления программы, что было проиллюстрировано выше при обсуждении вопроса о составе команды. В ранних компьютерах снижение бюджета битов программы имело решающее значение из-за высокой стоимости основной памяти. Сегодня мы имеем дешевую память, и проектировщик уже не очень озабочен бюджетом битов или статическим учетом количества команд. Напротив, на первое место выходит трафик битов, динамика частоты использования команд. Статическая частота команд является мерой частоты использования команды каждого вида в тексте программы. Динамическая частота команд является мерой частоты использования данной команды в ходе выполнения программы. Программа с циклом, выполняющимся 100 раз, будет иметь существенно разные статическую и динамическую статистики.

Рассмотрим программу, показанную на рис. 3.18 и предназначенную для выполнения умножения матриц [5]. Команда $h+5$ выполняет операцию умножения-сложения и дает вклад в статическую частоту, равный $1/15$, или 6,6%. Однако при размерах матрицы 32×32 команды $h+4$, $h+5$, $h+6$ и $h+7$ выполняются по 32768 раз. Полное число выполненных команд составит 133271, таким образом, эти четыре команды имеют динамический вес 24,6% каждая. Остальные 11 команд вместе представляют лишь 1,6% от всех выполненных команд.

Проектировщик использует статическую и динамическую статистики по-разному. Операцию или функцию, используемую редко, скажем, вычисление квадратного корня, проектировщик может не включать в состав команд, но в то же время позаботиться, чтобы в него входили примитивы, позволяющие программно вычислить квадратный корень. Отказавшись от реализации квадратного корня, проектировщик может упростить процессор, что соответствует духу архитектурного стиля RISC.

Если динамическая статистика показывает, что конкретная команда представляет значительную долю всех выполняемых команд, проектировщик должен принять все меры, чтобы сократить время ее выполнения (вспомните закон Амдала). Проектировщик, заинтере-

КОММЕНТАРИЙ		КОМАНДА	
Подготовка	$h-2$	Загрузка k из k_0	
	$h-1$	Загрузка j_0 из j_{00}	
Начальная установка	h	Загрузка i_0 из i_{00}	
Процедура строки нового произведения	$h+1$	Загрузка i из i_0	←
Процедура произведения нового вектора	$h+2$	Загрузка j из j_0	←
Умножение вектора, внутренний цикл	$h+3$	Обнуление аккумулятора	
	$h+4$	Загрузка кумулятивного множимого из ячейки, заданной i	←
ОПЕРАЦИЯ	$h+5$	Умножение-сложение с операндом, заданным j	
	$h+6$	Инкремент j на p	
Уборка, внутренний цикл	$h+7$	Продвижение i , счет, заполнение заново, когда счетчик достигнет 0, переход на $h+4$, если счетчик не достиг 0	→
	$h+8$	Сохранение кумулятивного произведения в ячейке, заданной k	
Конец процедуры умножения вектора	$h+9$	Инкремент k на 1	
	$h+10$	Продвижение j_0 , счет, заполнение заново, когда счетчик достигнет 0, и переход на $h+2$, если счетчик не достиг 0	→
Конец процедуры строки произведения	$h+11$	Инкремент i на p	
	$h+12$	Уменьшение счетчика k , заполнение заново, когда счетчик достигнет 0, и переход на $h+1$, если счетчик не достиг 0	→

Рис. 3.18. Программа умножения матриц.

сованный в повышении производительности умножения матриц с помощью программы рис. 3.18, прежде всего обратит внимание на четыре команды с номерами $h+4$, $h+5$, $h+6$ и $h+7$.

Стоит отметить любопытный факт, касающийся статистики выполнения команд. Доступность статистических данных для различных компьютеров колеблется от избыточности до почти полного отсутствия. В классе избыточности мы находим MIPS R2000 и SPARC. Вслед за ними идут VAX и IBM S360. В класс почти полного отсутствия данных попадает архитектура x86. Этому можно найти две причины. Во-первых, x86 не привлек внимание академических кругов; во-вторых,

разработчики этой архитектуры собирают статистику для собственного употребления и не видят необходимости в ее публикации. Другими словами, доступность опубликованных статистических данных представляется обратно пропорциональной числу установленных процессоров.

✎ Пример

Вычислим бюджет и трафик битов в числе команд для векторной программы, приведенной на рис. 3.18. Предположим, что как две исходные матрицы, так и результирующая имеют размеры 128×128 .

✍ Решение

Бюджет битов составляет 15 команд. Трафик битов определяется, главным образом, внутренним циклом от $h+4$ до $h+7$, выполняемым 128^3 раз. Следовательно, трафик битов для этой части программы составит $4 \times 128^3 = 8,39 \times 10^6$ команд. Следующий цикл выполняется 128^2 раз. Трафик битов для этой части программы составит $5 \times 128^2 = 8,19 \times 10^4$ команд (команды от $h+2$ до $h+10$, за исключением учтенных выше). Внешний цикл выполняется 128 раз, и трафик битов для этой части составит $3 \times 128 = 3,84 \times 10^2$ команд (команды от $h+1$ до $h+12$, за исключением учтенных выше). Три установочные команды выполняются лишь однажды. Таким образом,

$$\begin{aligned} \text{трафик битов} &\approx 8,39 \times 10^6 + 8,19 \times 10^4 + 3,84 \times 10^2 + 3 \approx \\ &\approx 8,47 \times 10^6 \text{ команд} \end{aligned}$$

✓ Замечание

Проведенные вычисления трафика битов отчетливо показывают, что существенного ускорения этого алгоритма можно добиться только за счет повышения скорости выполнения внутреннего цикла. Именно это и сделано в векторных компьютерах типа TI ASC и CRAY-1. Так, компьютер CRAY-1 сжимает четыре команды в одну, выполняя аппаратное перекрытие команд $h+4$, $h+6$ и $h+7$ с командой $h+5$.

Статистики языков высокого уровня

Вспомним, что программа, написанная на языке высокого уровня, компилируется на язык ISA данного компьютера. Таким образом, перед тем, как заниматься статистикой языка ISA, полезно исследовать частоту использования различных предложений языка высокого уровня. Данные, приводимые в табл. 3.17, взяты из обзора предварительных исследований, направленных на разработку синтетической конт-

Таблица 3.17. Статическая частота использования для языков высокого уровня

Операция	Язык		
	ФОРТРАН (%)	ПАСКАЛЬ (%)	АДА (%)
Присваивание	51	44,0	37,1
Call	12	31,8	26,8
Return	4	—	6,9
If	10	14,8	9,8
Условный цикл	—	3,3	1,3
Цикл for	9	2,8	0,9
With	—	2,1	—
Case			0,4
Exit loop	—	—	1,4
Goto	9	0,3	—
Остальные	5	0,1	15,4

рольной задачи, называемой Drystone [18]. В обзоре приводятся данные и по статической, и по динамической статистикам.

Как уже отмечалось выше, в языках высокого уровня отсутствуют явные предложения пересылки. Эти предложения вносятся компилятором, чтобы сопоставить предложения присваивания с архитектурой памяти: регистрами, аккумулятором, стеком и т. д. Обратите внимание на то, что предложения присваивания и вызовов подпрограмм (call) составляют около 65% всех статических предложений. Можно ожидать, что другие языки высокого уровня, как например, Си и Си++, дадут схожие статические результаты, хотя автору неизвестны публикации такого рода.

Статистики уровня команд

Как и в случае языков высокого уровня, публикации, посвященные статистикам уровня команд, обычно содержат сведения как о статических, так и о динамических частотах. Мы рассмотрим статистики уровня команд для трех CISC-процессоров: IBM S360, DEC VAX и x86. Заодно мы приведем сведения по двум RISC-процессорам, MIPS R2000 и SPARC.

IBM S360. Богатый материал по статическим и динамическим статистикам приведен в ранних работах по изучению программ для IBM S360 [2]. Выборка из этих данных сведена в табл. 3.18, где показаны команды с максимальной частотой повторения. Контрольные задачи, на которых были получены эти данные, не содержали команд с плавающей запятой.

Таблица 3.18. Веса команд IBM S360

Статические			Динамические		
L	Загрузка фиксированного слова	29%	L	Загрузка фиксированного слова	27%
ST	Сохранение	15%	BC	Переход/условие	14%
BC	Переход/условие	10%	ST	Сохранение	10%
LA	Загрузка адреса	7%	C	Сравнение	6%
SR	Вычитание	6%	LA	Загрузка адреса	6%
BAL	Переход со связыванием	5%	SR	Вычитание	5%
SLL	Сдвиг влево, одинарный	4%	IC	Включение символа	4%
IC	Включение символа	3%	A	Сложение	4%
C	Сравнение	3%	SLL	Сдвиг влево одинарный	3%
A	Сложение	2%	N	Логическое И	3%

Рассматривая эти данные, можно заметить, что команды с максимальной частотой использования — это те, которые включаются компилятором и являются издержками по отношению к решаемой задаче в том смысле, что они не выполняют полезной работы, как она понимается программой на языке высокого уровня. В столбце динамической статистики 63% выполняемых команд находятся до арифметических команд. Заметим также, что команды сложения и вычитания составляют всего 9% от всех выполняемых команд.

DEC VAX. Обширный статистический материал по командам DEC VAX был получен в работах Кларка и др. Некоторые данные по VAX ранжируют команды по частоте использования и по времени использования. Второе ранжирование дает представление о командах, интенсивно потребляющих вычислительное время и могущих быть предметом усовершенствования. Пример такого рода данных для нагрузки многопользовательской рабочей станции, приведенный в табл. 3.19, взят из работы [6].

Обратите внимание на то, что 10 команд составляют 47% всех выполняемых команд, в то время как другой набор из 10 команд потребляет 52% времени выполнения. Заметьте также, что одна команда, MOVCS, поглощает 23% времени контрольных задач. Отсюда можно сделать вывод, что более быстрая память повысит производительность этого процессора.

➤ Пример

Каково будет повышение скорости процессора VAX, если время выполнения команды MOVCS сократить в два раза?

Таблица 3.19. Статистика команд VAX

Команда	Частота использования	Команда	Время (%)
MOVL	0,1022	MOVCS	23,30
BEQL	0,0869	BBS	6,44
BBS	0,0525	MOVL	4,48
CMPB	0,0483	BBC	3,82
BNEQ	0,0360	CMPB	2,93
BLBC	0,0332	BEQL	2,47
INCL	0,0307	EDIV	2,43
RSB	0,0306	PROBEW	2,43
BBC	0,0261	RSB	2,03
SUBL3	0,0249	BSBW	2,00

Решение

Решение этой задачи является простым приложением закона Амдала:

$$\begin{aligned}
 \text{повышение быстродействия} &= \frac{77\% + 23\%}{77\% + \frac{23\%}{2}} = \\
 &= \frac{100\%}{88,5\%} = 1,13
 \end{aligned}$$

✓ Замечание

В то время когда разрабатывался процессор VAX, быстрая память была очень дорога. Стоимость быстрой памяти, необходимой для улучшения на 50% скорости выполнения команды MOVCS, возможно, превышала 13% стоимости всей системы, что делало такое улучшение неприемлемым с точки зрения соотношения стоимости и затрат.

Intel x86. Одним из немногих источников статистики использования команд x86 является исследование выполнения программ MS-DOS на процессоре 8086 [1]. 10 команд с максимальной частотой использования показаны в табл. 3.20. Приведенные данные весьма схожи с данными для IBM S360. Обратите внимание на то, что приблизительно 39% всех выполняемых команд — это либо MOV, либо PUSH, т. е. команды, вставляемые компилятором. Соответствующие результаты для IBM S360 составляют 39%, в то время как для VAX — всего 10%.

Таблица 3.20. Динамические веса команд для Intel 8086

Ранг	Команда	Вес	Описание
1	MOV	0,299	Пересылка источника в приемник
2	PUSH	0,092	Проталкивание источника в стек
3	CMP	0,079	Сравнение источника с приемником
4	POP	0,062	Выталкивание вершины стека в приемник
5	JNE	0,045	Переход, если не равно
6	JE	0,036	Переход, если равно
7	ADD	0,034	Сложение
8	CALL	0,033	Вызов подпрограммы
9	RET	0,031	Возврат из подпрограммы
19	JMP	0,023	Безусловный переход

Сравнение MIPS R2000 и Sun SPARC. Процессоры MIPS R2000 и Sun SPARC представляют современные RISC-архитектуры. Оба компьютера схожи во многих отношениях за исключением двух. В MIPS R2000 реализованы элементарные команды анализа и перехода, в то время как SPARC использует биты индикации с ортогональными командами переходов. Второе различие заключается в том, что MIPS R2000 использует для планирования выполнения конвейеров холостые команды, а в SPARC планирование осуществляется аппаратно. Эти вопросы будут обсуждаться в гл. 6.

В печати можно найти данные по всестороннему исследованию числа и частоты использования команд для этих двух компьютеров с помощью нескольких контрольных задач (для целых чисел и с плавающей запятой) Объединенной группы по оценке производительности вычислительных систем SPEC [7]. Пример таких данных приведен в табл. 3.21. Авторы ввели поправки на команды с низкой частотой использования, которые увеличили приблизительно на 100 число выполненных команд для машины SPARC. Тем не менее сравнение данных показывает, что полное число выполненных команд на этих компьютерах приблизительно одинаково, хотя их распределения различаются.

Рассматриваемые процессоры дают наглядный пример того, как архитектурные решения, принимаемые на стадии проектирования, влияют на производительность процессора. В MIPS R2000 используются элементарные команды анализа и перехода, а в SPARC — биты индикации. Приведенные данные показывают, что оба метода реализации команд переходов дают приблизительно равное число выполненных команд. Таким образом, эти методы при измерении трафика битов не имеют существенных преимуществ друг перед другом.

SPARC отличается от MIPS R2000 регистровым файлом большего размера (64 против 32 слов), что увеличивает возможности сохране-

Таблица 3.21. Статистика целочисленных команд для машин MIPS R2000 и Sun SPARC

Класс команды	MIPS R2000		Sun SPARC	
	Число выполненных команд	%	Число выполненных команд	%
Загрузка	220	19,83	214	18,52
Запоминание	129	11,58	94	8,14
Управление	205	18,50	234	20,26
Холостая команда	148	13,34	12	1,02
Целочисленная арифметика	407	36,66	580	50,20
Различные, библиотека	2	0,09	22	1,90
Итого	1111	100	1156	100

ния переменных в регистрах; в результате число команд запоминания на SPARC меньше. Использование более сложных аппаратных средств для планирования выполнения конвейеров приводит также к уменьшению числа выполненных холостых команд.

Сравнение команд RISC и CISC

Данные о выполнении команд, приведенные в предыдущих примерах, не дают возможности сравнения RISC-процессоров (SPARC) с CISC-процессорами (DEC VAX). Как уже отмечалось в разделе 3.1, лексический уровень CISC выше лексического уровня RISC. Естественно ожидать, что CISC требует выполнения меньшего числа команд (трафик битов меньше), чем RISC. Бандаркар и Кларк [3] предприняли проверку этого предположения, откомпилировав и выполнив 9 контрольных задач на процессорах VAX (CISC) и MIPS R2000 (RISC). Отношение числа выполненных на этих процессорах команд показано в табл. 3.22. Отношение > 1 означает, что RISC требует большего числа выполняемых команд или имеет более высокий трафик битов.

Приведенные результаты впечатляющи. Мы видим, что RISC всегда выполняет больше команд, чем CISC. Вспомним (гл. 1), что время выполнения задачи является произведением числа выполненных команд, величины CPI и длительности такта. Из наших данных можно сделать вывод, что при равной тактовой частоте для получения равной производительности значение CPI для RISC должно быть приблизительно вдвое меньше, чем для CISC. В гл. 5 и 6 мы обсудим методы определения величины CPI процессора.

В процессе сложения перенос может возникнуть в любом бите. Если перенос возникает в самом старшем бите, он называется переполнением, потому что полученное значение вышло за пределы допустимого диапазона типа данных. Аналогичная ситуация возникает с займом. Читатель может попытаться ответить на вопрос, в каком случае возникает заем в самом старшем бите при вычитании.

Проектировщик должен решить, стоит ли ему реализовывать в логических схемах операцию вычитания. Как будет показано ниже, вычитание можно заменить сложением с дополнением вычитаемого.

✎ **Пример**

Сложить два значения без знака 17_{10} и 21_{10} . Получить разность $38_{10} - 21_{10}$, используя показанные выше правила вычитания.

✎ **Решение**

Первое слагаемое	17	010001	Уменьшаемое	38	100110
Второе слагаемое	21	010101	Вычитаемое	-21	010101
Перенос		0100010	Заем		100010
Сумма	38	100110	Разность	17	010001

3.5.1. Сложение и вычитание

Обсудим теперь правила сложения и вычитания для трех типов данных: со знаком и величиной, дополнения до 1 и дополнения до 2. Архитектурное вычитание обычно отсутствует в системе команд; функция вычитания выполняется путем предварительного получения дополнения вычитаемого.

Знак и величина. Правила сложения и вычитания для целых и дробей [11] даны в табл. 3.23.

Таблица 3.23. Правила сложения и вычитания со знаком и величиной

Сложение	Знак 1-го слагаемого	Знак 2-го слагаемого	Правило
	+	+	Сложить величины, приписать знак 2-го слагаемого.
	-	-	
	+	-	Получить дополнение 2-го слагаемого, прибавить к величине 1-го слагаемого, если есть циклический перенос, прибавить его
	-	+	

Таблица 3.23. (продолжение)

Сложение	Знак 1-го слагаемого	Знак 2-го слагаемого	Правило
			к результату; если циклического переноса нет, получить дополнение результата, приписать знак 1-го слагаемого.
Вычитание	Знак уменьшаемого	Знак вычитаемого	Правило
	+	+	Получить дополнение битов величины вычитаемого, прибавить к величине уменьшаемого; если есть циклический перенос, прибавить его к результату, приписать знак уменьшаемого.
	—	—	
	+	—	Сложить величины, приписать знак уменьшаемого.
	—	+	

✎ Пример

Рассмотрим 6-битовое слово со знаком и величиной; знак находится в старшем бите.

А. Найти суммы следующих пар чисел: $+13+11$, $-13-11$, $-11+13$, $+11-13$.

Б. Найти разности тех же пар чисел.

✎ Решение

А. Сложение

+13	001101	-13	101101	-11	101011	+11	001011
+11	001011	-11	101011	+13	001101	-13	101101
+24	011000	-24	111000	+2		-2	
					101011		001011
				Сравн.	110010	Сравн.	110010
					11101		11101
					+0		+0
				+2	000010	-2	100010

Б. Вычитание

+13	001101	-13	101101	-11	101011	+11	001011
-(+11)	001011	-(-11)	101011	-(+13)	001101	-(-13)	101101
+2		-2		+24	011000	+24	011000
	001101		001101				
Сравн.	010100	Сравн.	000100				
	000001		000001				
	+1		+1				
+2	0,00010	-2	1,00010				

Дополнение до 1. В таблице 3.24 перечислены правила сложения и вычитания для целых и дробей в нотации дополнения до 1.

Таблица 3.24. Правила сложения и вычитания для дополнения до 1

Сложение	Знак 1-го слагаемого	Знак 2-го слагаемого	Правило
	+	+	Сложить величины; если есть циклический перенос, прибавить его к результату; приписать знак 2-го слагаемого.
	-	-	
	+	-	Сложить числа, включая знак. Если есть циклический перенос, прибавить его к результату, приписать знак -, в противном случае знак +
	-	+	
Вычитание	Знак уменьшаемого	Знак вычитаемого	Правило
	+	+	Получить дополнение битов вычитаемого, прибавить к уменьшаемому; если есть циклический перенос, прибавить его к результату, приписать знак уменьшаемого.
	-	-	
	+	-	
	-	+	

Пример

- Рассмотрим 6-битовое слово в виде дополнения до 1.
- А. Сложить следующие пары чисел: +13+11, -13-11, -11+13, +11-13.
- Б. Найдите разность тех же пар чисел.

 Решение**А. Сложение**

+13	001101	-13	110010	-11	110100	+11	001011
+11	<u>+001011</u>	-11	<u>+110100</u>	+13	<u>+00110</u>	-13	<u>110010</u>
	011000		1100110		1000001		111101
	+0		+1		+1		+0
+24	011000	-24	100111	+2	00011	-2	100010

Б. Вычитание

+13	001101	-13	110010	-11	110100	+11	001011
<u>-(+11)</u>	<u>001011</u>	<u>-(-11)</u>	<u>110100</u>	<u>-(+13)</u>	<u>001101</u>	<u>-(-13)</u>	<u>110010</u>
+13	001101	-13	110010	-11	110100	-11	001011
Сравн.	<u>+110100</u>	Сравн.	<u>+001011</u>	Сравн.	<u>+110010</u>	Сравн.	<u>+101101</u>
	1000001		111101		1100110		0111000
	+1		+0		+1		+0
+2	0,00010	-2	111101	-24	100111	+24	111000

Дополнение до 2. В таблице 3.25 даны арифметические правила для целых и дробей в нотации дополнения до 2.

Таблица 3.25. Правила сложения и вычитания для дополнения до 1

Сложение	Знак 1-го слагаемого	Знак 2-го слагаемого	Правило
	+	+	Сложить величины; игнорировать циклический перенос. Знак результата равен знаку, найденному при сложении.
	-	-	
	+	-	
	-	+	
Вычитание	Знак уменьшаемого	Знак вычитаемого	Правило
	+	+	Получить дополнение всех битов вычитаемого, прибавить к уменьшаемому, игнорировать циклический перенос. Прибавить 1 к результату. Знак результата равен знаку, найденному при сложении.
	-	-	
	+	-	
	-	+	

 Пример

Рассмотрим 6-битовое слово в виде дополнения до 2.

А. Сложить следующие пары чисел: $+13+11$, $-13-11$, $-11+13$, $+11-13$.

Б. Найти разность тех же пар чисел.

Решение

А. Сложение

$+13$	001101	-13	110011	-11	110101	$+11$	001011
$+11$	$+001011$	-11	110101	$+13$	$+001101$	-13	110011
$+24$	011000	-24	101000	$+2$	000010	-2	111110

Б. Вычитание

$+13$	001101	-13	110011	-11	110101	$+11$	001011
$-(-11)$	001011	$-(-11)$	110101	$-(+13)$	001101	$-(-13)$	110011
$+13$	001101	-13	110010	-11	110100	-11	001011
Сравн.	$+110100$	Сравн.	$+001011$	Сравн.	$+110010$	Сравн.	$+001100$
	1000001		111101		1100111		010111
	$+1$		$+1$		$+1$		$+1$
$+2$	000010	-2	111110	-24	101000	$+24$	011000

3.5.2. Умножение и деление

Умножение и деление во многих системах команд относятся к архитектурным элементам. Перед тем как переходить к особенностям проектирования с учетом типов данных, рассмотрим базовые операции. Ниже обсуждаются алгоритмы умножения и деления в различных представлениях, но не аппаратная реализация процесса выполнения этих команд. Правила знака для умножения и деления заключаются в следующем:

Умножение: одинаковые знаки дают положительный знак произведения; разные знаки дают отрицательное произведение.

Деление: величины делятся как целые без знака, одинаковые знаки дают положительное частное и остаток; разные знаки дают отрицательное частное и остаток.

Умножение

Операция умножения выполняется сходно с тем, как мы учили в школе, только основание равно не 10, а 2. Результат умножения битов равен логическому И над битами множимого и множителя. Умножение выполняется в три шага: 1) анализ младшего бита множителя, 2) если младший бит равен 1, прибавление множимого к частичному произведению, 3) сдвиг множителя на один бит вправо, а частичного произведения на один бит влево. Рассмотрим 4-битовое целое без знака в качестве и множимого, и множителя:

$$\begin{array}{r}
 9 \qquad \qquad 1001 \\
 \times 5 \qquad \times 0101 \\
 \hline
 45 \qquad \qquad 1001 \\
 \qquad \qquad 0000- \\
 \qquad \qquad 1001-- \\
 \qquad \underline{0000---} \\
 00101101
 \end{array}$$

Заметим, что 4-битовое данное может иметь максимальное значение $2^4 - 1 = 15$; таким образом, наша операция с очевидностью приведет к переполнению, так как $45 > 15$. Поскольку результат не может быть представлен в формате операндов, применяется функция ограничения, которая преобразует результат в представимую форму. Это означает, что будут сохранены 4 младших бита, что явно неверно. В некоторых ISA запоминаются обе половины произведения, и программист должен решать, что с ними делать.

Если используется дробный тип данных, то приведенный выше пример преобразуется следующим образом:

$$\begin{array}{r}
 0,5625 \qquad \qquad 0,1001 \\
 \times 0,3125 \qquad \times 0,0101 \\
 \hline
 0,17578125 \qquad \qquad 1001 \\
 \qquad \qquad 0000- \\
 \qquad \qquad 1001-- \\
 \qquad \underline{0000---} \\
 0,00101101
 \end{array}$$

В случае использования дробей старшая половина результата сохраняется в качестве произведения, а младшая отбрасывается. Результат дробного умножения, у нас 0,00101101, усекается до 0,0010 или округляется до 0,0011. Вопросы округления в контексте формата IEEE с плавающей запятой будут обсуждаться ниже в разделе 3.5.3.1.

Деление

Деление является более сложной операцией, чем умножение. С другой стороны, она обратна умножению: вместо сложений и сдвигов используются вычитания и сдвиги. Основная идея заключается в том, что делитель d вычитается из делимого D .

Процесс деления десятичных чисел проиллюстрируем на двух алгоритмах деления: с восстановлением и без восстановления. Сначала опишем алгоритм с восстановлением. Этот алгоритм состоит из ряда

Шаги

	1	2	3	4
Частное →	0	01	010	0101
	66			
d'	<u>-96</u>			
	-30			
d'	<u>+96</u>			
	66	66		
d'/2		<u>-48</u>		
		18		
d'/4			<u>-24</u>	
			-6	
d'/4			<u>+24</u>	
			18	
d'/8				<u>-12</u>
				6

Рис. 3.19. Деление с восстановлением, $66 \div 12$.

шагов, перечисленных ниже: на рис. 3.19 показан пример деления $66 \div 12 = 5 + 6/12$.

Инициализация. Умножаем делитель на степень 2, что делает его больше делимого³⁴⁾. Например, 12 умножается на 8, что приводит к операции $66 \div 96$. Назовем 96 начальным делителем d'' .

1. Вычтем d'' (96) из делимого. Если результат, называемый частным остатком P , отрицателен, переходим к шагу 2. Если результат положителен, переходим к шагу 3.

2. Снова прибавляем d'' . Записываем 0 в частное и переходим к шагу 4.

3. Записываем 1 в частное и переходим к шагу 4.

4. Сдвигаем d'' на один бит вправо (делим на 2). Переходим к шагу 1.

Эту процедуру следует повторить число раз, совпадающее с числом битов в операндах. Остатком является последнее значение промежуточного делителя: частное = $0101 + 0110/1100 = 5 + 6/12 = 5,5$.

Вторая форма деления, деление без восстановления, не требует восстановления делимого после получения отрицательного результата. Правила деления приводятся ниже, а сама операция показана на рис. 3.20 для того же примера $66 \div 12$.

³⁴⁾ В CDC 6600 предусмотрена команда, которая помещает в слово ведущую 1, что ускоряет выполнение этой операции перед делением.

	Шаги			
Частное →	1	2	3	4
	0	01	010	0101
	66			
d'	<u>-96</u>			
	-30	-30		
d'/2		<u>+48</u>		
		18		
d'/4			<u>-24</u>	
			-6	
d'/8				<u>+12</u>
				6

Рис. 3.20. Деление без восстановления, $66 + 12$.

Инициализация. Умножаем делитель на степень 2, что делает его больше делимого. Для рассматриваемого примера 12 умножается на 8, что приводит к операции $66 \div 96$.

1. Вычтем промежуточный делитель d'' (96) из делимого. Переходим к шагу 2.

2. Записываем 0 в частное, сдвигаем промежуточный делитель вправо на 1 бит (деление на 2) и прибавляем к предыдущему результату. Если результат отрицателен, переходим к шагу 2. Если результат положителен, переходим к шагу 3.

3. Записываем 1 в частное. Затем сдвигаем промежуточный делитель на один бит вправо (деление на 2) и вычитаем из делимого. Если результат отрицателен, переходим к шагу 2. Если результат положителен, переходим к шагу 3.

Повторяем процедуру до тех пор, пока число шагов не станет равно числу битов операндов. Остатком является последнее значение промежуточного делителя.

При использовании алгоритма с восстановлением выполняется вычитание $D - d$ и, если результат отрицателен, выполняется вычитание «наоборот». Эти дополнительные шаги, проиллюстрированные выше, увеличивают время, требуемое для выполнения алгоритма с восстановлением. Посмотрим, как выполнялись первые два шага обоих алгоритмов. В обоих случаях первое вычитание дало отрицательный результат.

С восстановлением

Шаг 1: $D - d + d = D'$

Шаг 2: $D' - d/2 = D''$

$$D'' = D - d + d - d/2 = D - d/2$$

$$D'' = 66 - 96/2 = 66 - 48 = 18$$

Без восстановления

Шаг 1: $D - d = D'$

Шаг 2: $D' + d/2 = D''$
 $D'' = D - d + d/2 = D - d/2$
 $D'' = 66 - 96/2 = 66 - 48 = 18$

Таким образом, оба алгоритма дают для этих двух шагов тот же промежуточный результат. Поскольку алгоритм без восстановления содержит меньшее число шагов, он является предпочтительным.

 Пример

Разделим целое 45 (101101) на 6 (0000110), используя алгоритм без восстановления и двоичную арифметику.

 Решение

Решение показано на рис. 3.21. Для инициализации умножим делитель на 8, что даст $6 \times 8 = 48$. Результат равен $7 + 3/6$.

Другие типы данных, отличные от целых со знаком и величиной, например дополнение до 1, дополнение до 2, дробь в виде дополнения до 1 и дробь в виде дополнения до 2, также имеют свои алгоритмы деления. Однако эти алгоритмы содержат много сложностей, которые затрудняют их реализацию; рассмотрение этих данных выходит за рамки настоящей книги. Обратите внимание на то, что из-за сложностей алгоритмов деления перечисленных выше типов данных некоторые процессоры, использующие эти типы данных в качестве архитектурных, преобразуют их в тип со знаком и величиной, выполняют деление, а затем преобразуют обратно в архитектурный тип.

Шаги	Делимое D'	Десятичное значение	Частное q
	101101	45	
	<u>-110000</u>	<u>-48</u>	
1	-000011	-3	0
	<u>+110000</u>	<u>+24</u>	
2	0001011	21	01
	<u>-0001100</u>	<u>-12</u>	
3	0001001	9	011
	<u>-0000110</u>	<u>-6</u>	
4	0000011	3	0111

Рис. 3.21. Деление с восстановлением, $45 + 6$.

3.5.3. Арифметика плавающей запятой

Большинство ISA включают архитектурные операции сложения, вычитания, умножения и деления. Все эти операции используют комбинацию приемов, описанных в разделах 3.5.1 и 3.5.2. Мы начнем с обсуждения сложения и вычитания с плавающей запятой.

Сложение и вычитание

Вспомним формат слова типа данных с плавающей запятой стандарта IEEE:

S	Порядок	Мантисса
---	---------	----------

Бит знака S относится к мантиссе; другими словами, этот тип данных имеет представление со знаком и смешанной величиной. В ряде более старых представлений чисел с плавающей запятой имелась дробная мантисса с запятой с левой стороны от старшего бита. Порядок обычно смещен, что устраняет необходимость в явном бите для знака порядка.

Сложение иллюстрируется в десятичном техническом представлении: $500 + 20 = 5,2 \times 10^2$.

Выравнивание

$$\begin{array}{r}
 5,0 \times 10^2 \\
 2,0 \times 10^1 \\
 \hline
 520
 \end{array}
 \qquad
 \begin{array}{r}
 5,0 \times 10^2 \\
 +0,2 \times 10^2 \\
 \hline
 5,2 \times 10^2
 \end{array}$$

Рассмотрим шаги выполнения этого сложения.

1. Сравниваются порядки.
2. Мантисса с меньшим порядком делится на 10, пока порядки не уравниются. Каждое деление на 10 сопровождается увеличением порядка на 1.
3. Выворнутые мантиссы складываются, и к ним присоединяется порядок.

Теперь проиллюстрируем сложение с основанием 2 на примере:

$$4 + 1,5 = 5,5$$

Выравнивание

$$\begin{array}{r}
 1,000 \times 2^2 \\
 +1,100 \times 2^0 \\
 \hline
 \end{array}
 \qquad
 \begin{array}{r}
 1,000 \times 2^2 \\
 +0,011 \times 2^2 \\
 \hline
 1,011 \times 2^2
 \end{array}$$

Вычитание с основанием 2 иллюстрируем на примере:

$$4 - 1,5 = 2,5.$$

	<i>Выравнивание</i>	<i>Нормализация</i>
$1,000 \times 2^2$	$1,000 \times 2^2$	
$-1,100 \times 2^0$	$-0,011 \times 2^2$	
	$0,101 \times 2^2$	$1,010 \times 2^1$

Результат вычитания ($0,101 \times 2^2$) требует нормализации. Другими словами, мантисса сдвигается влево до тех пор, пока старший бит не станет равен 1; при каждом сдвиге порядок уменьшается на 1.

✎ Пример

Что происходит, если при сложении мантисс возникает перенос?

✍ Решение

После сложения выполняется нормализация: результат смещается вправо, а порядок инкрементируется. Например, $4 + 4 = 8$:

$$\begin{array}{r}
 1,000 \times 2^2 \\
 +1,000 \times 2^2 \\
 \hline
 10,000 \times 2^2 \\
 1,000 \times 2^3
 \end{array}$$

Умножение и деление

Типы данных с плавающей запятой в формате IEEE могут участвовать в операциях умножения и деления. В некоторых отношениях правила для этих операций проще, чем для сложения и вычитания. Вспомним еще раз, что порядок смещен, а мантисса является смешанным числом со знаком.

Умножение. Правила умножения чисел с плавающей запятой заключаются в следующем:

1. Складываем порядки.
2. Умножаем мантиссы со знаком.
3. Присоединяем к результату полученный на шаге 1 порядок.

Например, $4 \times 1,5 = 6$:

$$\begin{array}{r}
 1,00 \times 2^2 \\
 \times 1,10 \times 2^0 \\
 \hline
 1,1000 \times 2^2
 \end{array}$$

Деление. Правила деления чисел с плавающей запятой заключаются в следующем:

1. Вычитаем порядки.

2. Делим мантиссы со знаком.
3. Присоединяем к результату полученный на шаге 1 порядок.

Например, $6 \div 3 = 2$:

$$\begin{array}{r} 1,100 \times 2^2 \\ \div 1,100 \times 2^1 \\ \hline 1,000 \times 2^1 \end{array}$$

Операции со смещенными порядками

В типе данных с плавающей запятой над порядком допустимо выполнять лишь ограниченный набор операций, что делает его специальным типом данных. Из приведенных выше примеров видно, что над порядком выполняются четыре операции:

- 1) сравнение двух порядков, выбор меньшего и запоминание разности,
- 2) сложение двух порядков,
- 3) вычитание двух порядков,
- 4) инкремент и декремент порядка.

В современных процессорах, где операции с плавающей запятой обеспечиваются на уровне архитектуры, порядок относится к типу данных реализации; он не вычисляется непосредственно самой командой. Операции с порядком осуществляются в контексте полного типа данных с помощью аппаратно реализованного конечного автомата или микропрограммы. Таким образом, выбор типа данных для представления порядка является в общем случае исключительно вопросом реализации, при условии, что проектировщик обеспечивает точность и диапазон, необходимые для данных с плавающей запятой.

Какое представление должно быть у порядка? Вспомним, что дополнение до 1 включает два нуля и нуждается в циклическом переносе. Дополнение до 2 тоже не очень удачно, так как при вычитании требует образования дополнения вычитаемого и добавления 1 к результату. Использование смещенного порядка позволяет относительно просто выполнять операции сравнения без дополнительных сложностей со знаком. Сложение и вычитание при отсутствии знака тоже оказываются более простыми.

Поскольку смещенный порядок является, в сущности, представлением с избытком (рассмотренным ранее применительно к десятичным данным), правила выполнения операций схожи. Разница заключается, главным образом, в причинах использования такого представления. Для десятичных типов данных нотация с избытком используется ради обеспечения десятичного представления и десятичного переноса. Для порядков нотация с избытком позволяет упростить реализацию требуемых арифметических операций.

✎ Пример

Учитывая, что порядок чисел с плавающей запятой смещен (нотация с избытком), должна ли выполняться коррекция результата после сложения или вычитания порядков? Какая именно? Возьмем для примера 3-битовый порядок, $n = 3$, и смещение $2^{n-1}-1 = 2^{3-1}-1 = 3 = 011$; см. табл. 3.9.

✎ Решение

Для 3-битового порядка со смещением 011 мы имеем следующие значения:

<i>Двоичное</i>	<i>Десятичное</i>
000	-3
001	-2
010	-1
011	0
100	+1
101	+2
110	+3
111	+4

В случае сложения $(-2) + (+1) = -1$ или, в представлении смещенного двоичного, $001+100 = 101$, что неверно. Из результата следует вычесть 3: $101 - 011 = 010$, что дает правильный результат (-1 в смещенной нотации).

В случае вычитания $(+3) - (+1) = +2$ или, в представлении смещенного двоичного, $110 - 100 = 010$, что опять неверно. К результату следует прибавить 3: $010 + 011 = 101$, что дает правильный результат ($+2$ в смещенной нотации).

✓ Замечание

В случае сложения $(a + \text{смещение}) + (b + \text{смещение}) = a + b + 2 \times \text{смещение}$, и требуется вычитание смещения. В случае вычитания $(a + \text{смещение}) - (b + \text{смещение}) = a - b$, и требуется прибавление смещения. Выполнение аппаратной коррекции как с плюсом, так и с минусом существенно облегчается, если смещение является степенью 2, т. е. 2^{n-1} . Таким образом, в рассмотренном примере использование смещения 3 не очень удачно; лучше было использовать смещение 4.

3.5.3.1. Точность

Типы данных с плавающей запятой являются приближениями; поэтому арифметическая операция может дать результат, который нуждается в интерпретации и корректировке. Эти действия должны

рассматриваться с точки зрения получения результата, годного в качестве входного данного для дальнейших операций. В последующих абзацах обсуждаются проблемы и процедуры коррекции типов данных с плавающей запятой в формате IEEE.

Операции над числами с плавающей запятой могут дать результат, который имеет большую длину, чем операнды-источники. Дополнительная длина образуется младшими битами мантиссы. В дальнейшем эти биты будут называться избыточными младшими битами (excess low-order bits, ELB); их также называют битами остатка [4]. Биты остатка используются двояко: для поддержки постнормализации результата со сторожевыми битами, а также для поддержки округления с битами округления.

Сторожевые биты

Нормализация IEEE-типов данных с плавающей запятой помещает 1 в бит слева от двоичной запятой мантиссы. Биты ELB возникают в результате арифметических операций и постнормализации:

S	Порядок	Мантисса	ELBs
---	---------	----------	------

Умножение двух мантисс приведет к появлению битов ELB. Рассмотрим умножение мантисс 1,125 и 1,625; пусть мантисса имеет 4 бит:

$$\begin{array}{r}
 1,125 \\
 \times 1,625 \\
 \hline
 1,828125
 \end{array}
 \qquad
 \begin{array}{r}
 1,001 \times 2^0 \\
 \times 1,101 \times 2^0 \\
 \hline
 1001 \\
 1001 \\
 1001 \\
 \hline
 1,110101 \times 2^0 \\
 1,75
 \end{array}$$

3 бит ELB, выделенные на рисунке жирным шрифтом, возникают из-за того, что тип данных содержит лишь 4 бита, в то время как результат умножения содержит 7 бит. Если эти биты игнорировать, результат операции окажется равен 1,75, что является плохой аппроксимацией точного результата 1,828125. Как будет показано в дальнейшем, биты ELB сохраняются в качестве сторожевых и используются затем для округления с целью повышения точности результата.

При вычитании двух чисел с плавающей запятой и разными значениями порядков возникает вероятность получить результат с существенной погрешностью. Рассмотрим пример вычитания мантисс: $3 - 1,25 = 1,75$. Пусть у нас будут 3-битовое ALU и 3-битовые мантиссы. Знаменатель мантиссы выравнивается перед вычитанием, так как порядки должны быть одинаковы, и в результате возникает один бит

ELB, который игнорируется 3-битовым арифметическим узлом. Вычисленная разность составляет 2, а точная разность в двоичном коде равна $1,11 \times 2^0$, т. е. 1,75; таким образом, два бита результата и порядок ошибочны:

$$\begin{array}{r} 1,10 \times 2^1 \\ -1,01 \times 2^0 \\ \hline \end{array} \quad \begin{array}{r} 1,10 \times 2^1 \\ -0,101 \times 2^1 \\ \hline 1,00 \times 2^1 \end{array}$$

Потеря точности в этом примере уменьшается использованием сторожевого бита, который расширяет арифметический узел на один разряд³⁵⁾. Использование сторожевого бита иллюстрируется на том же самом примере. После сложения сторожевой бит служит самым младшим битом нормализованной мантиссы. Полученный результат 1,75 является точной разностью:

$$\begin{array}{r} 1,10 \times 2^1 \\ -1,01 \times 2^0 \\ \hline \end{array} \quad \begin{array}{r} 1,100 \times 2^1 \\ -0,101 \times 2^1 \\ \hline 0,111 \times 2^1 \\ \text{Нормализация} \quad 1,11 \times 2^0 \end{array}$$

При сложении и вычитании число битов в ELB равно разности порядков. Как показано в табл. 3.10, для систем с основанием 2 в 6,78% случаев нормализация приводит к появлению 1 бита, а в 3,47% появляются 2 бита. В 59,34% случаев сдвиги нормализации не требуются. Рассмотренный выше пример повторен с отличием в порядке вычитаемого (−1 вместо 0):

$$\begin{array}{r} 1,10 \times 2^1 \\ -1,01 \times 2^{-1} \\ \hline \end{array} \quad \begin{array}{r} 1,1000 \times 2^1 \\ -0,0101 \times 2^1 \\ \hline 1,0011 \times 2^1 \\ \text{Усечение} \quad 1,00 \times 2^1 \end{array}$$

Ограничивающая функция усечения дает разность 2, в то время как точная разность составляет $3,0 - 0,625 = 2,375 = 1,0011 \times 2^1$. Усечение просто отбрасывает все биты ELB. При усечении два бита ELB теряются, однако эти биты можно сохранить и повысить тем самым точность округленного результата.

Другой случай образования битов ELB относится к сложению двух чисел, имеющих одинаковые порядки, когда за сложением следует операция нормализации суммы. В приводимом ниже примере суммирования $3 + 3,5 = 6,5 = 1,110 \times 2^2$ показано, как создаются биты ELB. Нормализованные исходные данные, как и ALU, имеют по 3 бит. Биты

³⁵⁾ Формат IEEE предусматривает сторожевой бит; формат IBM S360 имеет сторожевой 16-ричный разряд для основания 16.

ELB создаются в силу переноса при сложении, который требует для нормализации сдвига вправо:

$$\begin{array}{r}
 1,10 \times 2^1 \\
 +1,11 \times 2^1 \\
 \hline
 11,01 \times 2^1 \\
 \text{Нормализация} \quad 1,101 \times 2^2 \\
 \text{Усечение} \quad 1,10 \times 2^2
 \end{array}$$

После усечения получаем, что результат этого сложения равен 6,0 вместо правильного 6,5. Вопрос заключается в том, что делать с 1 в ELB. ELB сохраняется и используется затем для округления результата.

Изготовители применяют различные подходы к использованию сторожевых битов. В IBM S360 предусмотрен сторожевой 16-ричный разряд. Процессор PowerPC, реализующий стандарт IEEE, использует 64-битовые регистры плавающей запятой для типов данных как одинарной, так и двойной точности. Арифметический узел дополнен и сторожевыми битами, и битами округления. Intel, начиная с процессора 8087, реализует 80-битовые регистры плавающей запятой в виде стека. Тип данных длиной 80 бит, называемый промежуточной памятью расширенной точности, предоставляет достаточно битов, которые можно использовать в качестве сторожевых. Motorola для серии MC68000 также использует 80-битовые внутренние регистры плавающей точки.

Округление

Сложение, умножение или деление мантиссы может привести к результату с битами ELB, который выходит за пределы допустимого представления и требует применения функции ограничения. После постнормализации биты ELB сохраняются в качестве резервных (сторожевых) и используются в дальнейшем при округлении.

Мы знакомы с простым округлением до ближайшего, используемым с десятичными числами. Если избыточные младшие цифры (excess low-order digits, ELD) меньше $1/2$ младшей цифры результата (least significant digit, LSD), выполняется округление в сторону уменьшения (вниз). В противном случае округление выполняется вверх. При использовании этого алгоритма дробные ELD 0,0—0,4 округляются до 0, в то время как ELD 0,5—0,9 округляются вверх добавлением 1 к LSD. В результате 84,3 округляется до 84, а 84,5 до 85, и 84,7 тоже до 85.

Каким образом округление можно применить к результатам с плавающей запятой? Для этого используются различные алгоритмы, дающие различную степень точности. В некоторых процессорах округление определено разработчиками и зафиксировано аппаратно; например, арифметический узел IBM S360 использует усечение. В других случаях пользователю предоставляется несколько возможностей,

из которых он может программно выбрать удобную для себя. В стандарте IEEE плавающей запятой описаны четыре алгоритма округления.

Округление к ближайшему. Эта операция округляет до значения, ближайшего к точному значению. Из двух ближайших репрезентативных значений, которые являются равно близкими, берется то, в котором младший бит мантиссы равен 0.

Округление к 0. Эта операция округляет в направлении нуля и является усечением.

Округление к $+\infty$. Эта операция для положительных чисел округляет вверх до ближайшего к точному значению, но не меньшего его.

Округление к $-\infty$. Эта операция для отрицательных чисел округляет вниз до ближайшего к точному значению, но не большего его.

Округление к ближайшему является умолчанием для стандарта IEEE, и в табл. 3.26 приведены правила реализации. Эти правила применяются после нормализации результата арифметической операции. Всего имеются 3 случая, и для их однозначной идентификации требуются 2 бит; эти биты называют первым и вторым битами промежуточного округления.

Таблица 3.26. Округление к ближайшему в стандарте IEEE плавающей запятой

Биты ELB	Округляющее действие
Меньше $1/2$ младшего бита мантиссы	Оставить мантиссу без изменений.
Равны $1/2$ младшего бита мантиссы	Из двух ближайших репрезентативных значений, которые являются равно близкими, берется то, в котором младший бит равен 0.
Больше $1/2$ младшего бита мантиссы	Прибавить к мантиссе 1.

Первым битом промежуточного округления служит самый старший бит ELB после нормализации. Если этот бит равен 0, значение битов $ELB < 1/2$ младшего бита мантиссы; мантисса в этом случае не изменяется. Если бит округления равен 1, значение $ELB \geq 1/2$ младшего бита мантиссы. Для разрешения двух условий $=$ и $>$ требуется второй бит.

Второй бит промежуточного округления устанавливается в 1, если справа от первого бита округления есть хотя бы одна 1. Если и первый, и второй биты округления равны 1, значение битов $ELB > 1/2$ младшего бита мантиссы. Если первый бит округления равен 1, а второй 0,

значение битов ELB составляет точно $1/2$ младшего бита мантиссы³⁶. Таблица 3.27 детализирует сведения из табл. 3.26 для пяти битов ELB. Алгоритм коррекции мантиссы показан для случая, когда биты ELB равны $1/2$ младшего бита мантиссы.

Таблица 3.27. Первый и второй биты промежуточного округления в стандарте IEEE плавающей запятой

Биты ELB	1-й бит	2-й бит	Округляющее действие
0xxxx	0	x	Оставить мантиссу без изменений.
10000	1	0	Прибавить младший бит мантиссы к мантиссе ($0+0 = 0$, $1+1 = 10$).
10010	1	1	Прибавить к мантиссе 1.

В таблице 3.28 показан пример применения описанных правил к двум 5-битовым мантиссам 1,1000 и 1,0001. Например, если биты промежуточного округления составляют число 10, к мантиссе, у которой младший бит равен 1, прибавляется 1, а мантисса с младшим битом, равным 0, не изменяется.

Таблица 3.28. Округление к ближайшему

Мантисса	1-й и 2-й биты промежуточного округления		
	0x	10	11
1,1000	1,1000	1,1000	1,1001
1,0001	1,0001	1,0010	1,0010

🔗 Пример

Показать, что округляющее действие «прибавить младший бит мантиссы к мантиссе ($0+0 = 0$, $1+1 = 10$)» удовлетворяет спецификации округления в том случае, когда два репрезентативных значения являются равно близкими.

³⁶ Второй бит промежуточного округления называют битом залипания (sticky bit). Это название возникло из наблюдения образования битов ELB. Было замечено, что иногда они образуются путем сдвига. Если обнаруживается 1, триггер устанавливается в 1 и остается «залипшим» в этом состоянии до приложения функции ограничения. Бит залипания является также результатом операции логического ИЛИ над всеми битами ELB справа от первого бита округления.

 Решение

Предположим, что неокругленная мантисса равна 1,0001, первый бит округления равен 1, а второй — 0. Следующее меньшее бесконечно точное значение будет 1,00010, а следующее большее — 1,00100. В результате выбирается округленная мантисса 1,0010, так как ее младший бит равен 0. Выбранная мантисса формируется путем прибавления к ней ее младшего бита ($1,0001 + 1 = 1,0010$):

Следующее меньшее	1,0001 0
Мантисса	1,0001 1 0
Следующее большее	1,0010 0

Второй случай, когда младший бит мантиссы равен 0, показан ниже:

Следующее меньшее	1,1000 0
Мантисса	1,1000 1 0
Следующее большее	1,1001 0

Для этого случая округленная мантисса равна 1,1000. Она формируется путем прибавления к ней ее младшего бита ($1,1000 + 0 = 1,1000$). Таким образом, если биты ELB равны 10, то все, что надо сделать — это прибавить к мантиссе ее младший бит.

Сложение двух нормализованных чисел, $3 + 3,5 = 6,5$, будет служить иллюстрацией алгоритма округления к ближайшему. Бит ELB, равный 1, был сохранен как сторожевой и становится битом округления:

	$1,10 \times 2^1$
	$+ 1,11 \times 2^1$
	$11,01 \times 2^1$
Нормализация	$1,101 \times 2^2$
Округление	$1,10 \times 2^2$

Справа от сторожевого бита нет 1, т. е. бит залипания равен 0. Прибавление 0 к мантиссе дает правильно округленный результат. Округление до ближайшего дает результат 6,0, это другая аппроксимация бесконечно точного результата, равного 6,5.

Рассмотренная выше операция умножения дает другой пример использования битов округления и залипания. Если биты ELB равны 101, это интерпретируется так, что бит округления = 1, и бит залипания = 1.

Таким образом, значение $ELB > 1/2$ младшего бита мантииссы, и к мантииссе добавляется 1; округленный результат составляет 1,875:

$$\begin{array}{r}
 1,125 \\
 \times 1,625 \\
 \hline
 1,828125
 \end{array}
 \qquad
 \begin{array}{r}
 1,001 \times 2^0 \\
 \times 1,101 \times 2^0 \\
 \hline
 1001 \\
 1001 \\
 \hline
 1001 \\
 1,110101 \times 2^0 \\
 +1 \\
 \hline
 1,111 \\
 \hline
 1,875
 \end{array}$$

Функция ограничения «округление до ближайшего» гарантирует, что абсолютное значение ошибки будет $\leq 1/2$ младшего бита мантииссы. Округленный результат этого умножения определен с ошибкой $1,828125 - 1,875 = -0,046875$. Такая величина ошибки лежит в пределах $\pm 1/2$ младшего бита 4-битовой мантииссы, значение которой составляет $0,125/2 = 0,0625$.

✎ Пример

Какова максимальная ошибка для числа с плавающей запятой одинарной точности в формате IEEE при округлении до ближайшего?

✎ Решение

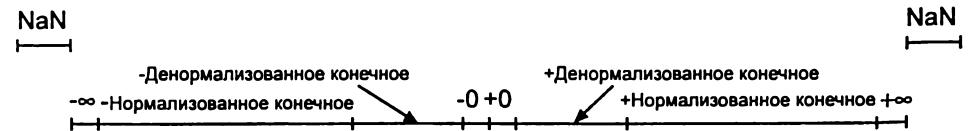
Для формата одинарной точности с 23-битовой мантииссой максимальная ошибка составит

$$\leq \frac{2^{-23}}{2} = 2^{-24} = 5,96 \times 10^{-8}.$$

Кодирование типов данных с плавающей запятой

На рисунке 3.22 изображены различные значения данных в стандарте IEEE плавающей запятой. В верхней части рисунка обозначены обычные действительные значения, образующиеся в результате арифметических операций. Можно выделить 8 областей действительных чисел: $\pm\infty$, $\pm$ нормализованное конечное, $\pm$ денормализованное конечное и ± 0 . Вспомним, что этот тип данных относится к данным со знаком и величиной и двумя нулями.

Представление типов данных показано на рисунке справа и слева для положительных и отрицательных значений. Эти типы данных со-



Кодирование действительных чисел и NaN для 32-битового формата плавающей запятой

S	E	F			S	E	F
1	0	0	-0		0	0	+0
1	0	0.XXX ²	-Денормализованное конечное		0	0	+Денормализованное конечное
1	1...254	Люб. знач.	-Нормализованное конечное		0	1...254	+Нормализованное конечное
1	255	0	-∞		0	255	+∞
X ¹	255	1.0XX ²	-SNaN		X ¹	255	+SNaN
X ¹	255	1.1XX	-QNaN		X ¹	255	+QNaN

Примечание:

1. Знаковый бит игнорируется
2. Дробь должна быть не 0

Рис. 3.22. Значения чисел с плавающей точкой в формате IEEE (с любезного разрешения Intel Corporation). Обозначения: S – знак, E – порядок, F – мантисса.

ставляют $\pm$ ноль, $\pm$ денормализованное конечное, $\pm$ нормализованное конечное, $\pm\infty$, SNAN и QNaN.

Обозначения SNAN и QNaN требуют объяснений. NAN обозначает «не число» (от not a number). Некоторые арифметические операции, например $\sqrt{-1}$, приводят к результату, который не является обычным действительным числом. Как таковые, эти значения кодируются, как показано на рисунке, с числом 255 в качестве порядка. Буквы S и Q обозначают «сигнальный» (signal) и «тихий» (quiet), соответственно.

Число SNAN возбудит прерывание неправильной операции, если его значение используется в последующей арифметической операции. С другой стороны, число QNaN «проскочит» последующую арифметическую операцию без возбуждения прерывания. Из всех 2^8 возможных значений порядка значения 0 и 255 (вместе со специальными значениями мантисс) зарезервированы для однозначного кодирования $\pm\infty$, SNAN и QNaN.

Литература

1. Adams, T. L. and Zimmerman, R. E. (1989). «An Analysis of 8086 Instructions Set Usage in MSDOS Programs», *ASPLOS III*, pp. 152–160.
2. Alexander, W. G. and Wortman D. B. (1975). «Static and Dynamic Characteristics of XPL Programs», *Computer*, 8:11.
3. Bhandarkar, D. and Clark, D. W. (1991). «Performance from Architecture: Comparing a RISC and a CISC with Similar Hardware Organization», *ACM SIGARCH Computer Architecture News*, 19:2, pp. 310–319.
4. Blaauw, G. A. and Brooks, F. P. Jr. (1997). *Computer Architecture, Concepts and Evaluation*, Addison-Wesley Longman, Reading, MA.
5. Buckholz, W., ed. (1962). *Planning a Computer System*, McGraw-Hill, New York.
6. Clark D. W. and Levy H. M. (1982). «Measurement and Analysis of Instruction Use in the VAX-11/780», *ACM Transactions on Computer Systems*, 1:1.
7. Cmelic, R. F., Kong, S. I., Ditzel, D. R. and Kelly E. J. (1991). «An Analysis of MIPS and SPARC Instruction Set Utilization on the SPEC Benchmarks», *ACM SIGARCH Computer Architecture News*, 19:2, pp. 290–302.
8. Cody, W. J., Coone, J. T., Gay, D. M., Hanson, K., Hough, D. Kahan, W., Karpinsik, R., Palmer, J., Ris, F.N. and Stevenson, D. (1984). «A Proposed Radix- and Word-Lenght-Independent Standard for Floating-Point Arithmetic», *IEEE Micro*, 4:4, pp 86–100.
9. Cohen, D. (1981). «On Holy Wars and a Plea for Peace», *Computer*, 14:10, pp. 48–54.
10. Eckert, J. P. (1946). *Theory and Techniques for Design of Electronics Digital Computers*, Moore School Special Course in July and August 1946. MIT Press, Cambridge, MA. Reprinted as Volume 9, The Charles Babbage Institute Reprint Series for the History of Computing, 1985.
11. Flores, I. (1963). *The Logic of Computer Arithmetic*, Prentice-Hall, Englewood Cliffs, N.J.
12. IBM. (1970). «IBM S/360 Principles of Operation», Order GA22-6821-8, IEEE, ANSI/IEEE Standard 754-1985, The Institute of Electrical and Electronics Engineers, New York, 1985.
13. IEEE, *ANSI/IEEE Standard 754–1985*. The Institute of Electrical and Electronics Engineers, New York, 1985.
14. Kirrmann, H. (1983). «Data Format and Bus Compatibility in Microprocessors», *IEEE Micro*, 3:4, pp.32–47.
15. Siewiorek, D. P., Bell, C. G. and Newell, A. (1982). *Computer Structures: Principles and Examples*, McGraw-Hill, New York.
16. Sweeney, D.W. (1965), «An Analysis of Floating-Point Addition», *IBM Systems Journal*, 4,1.
17. von Neumann, J. (1945). «First Draft of a Report on the EDVAC», Contract W-670-ORD-4926. Moore School of Electrical Engineering, University of Pennsylvania.

18. Wiecker, R. P. (1984). «Dhrystone: A Synthetic Systems Programming Benchmark», *Communications of the ACM*, 27:10.

19. White, G. S. (1953), «Coded Decimal Number Systems for Digital Computers», *Proc. IRE*, 41, 10, pp. 1450–1452.

20. Wilkes, M. V., Wheeler, D. J. and Gill, S. (1951). *The Preparation of Programs for an Electronic Digital Computer*, Addison-Wesley, Cambridge, MA. Reprinted in the Reprint Series for the History of Computing, Vol. 1, Charles Babbage Institute, Tomash Publishers, Los Angeles.

Упражнения

3.1. Выполните заново пример на смещенный порядок в разделе 3.5.3, используя смещение 4 вместо 3. Обсудите преимущества такой замены.

3.2. Из таблицы 3.12 определите смещение для каждого из перечисленных форматов плавающей запятой. Считайте, что смещение должно быть четной степенью 2.

3.3. Посмотрите документацию Intel и покажите, как Intel Pentium с MMX использует расширенные коды операций.

3.4. Каков диапазон в байтах длины команд Pentium Pro? Другими словами, каковы длины самой короткой и самой длинной команды?

3.5. Предположите следующее: 40% всех выполненных команд переходов относятся к условным переходам и 76% от всех переходов выполнены. Какова доля выполненных переходов от всех переходов?

3.6. Вы проектируете ISA и выделяете биты в команде для адресации непосредственных операндов. Изучение 399 непосредственных операндов дало следующее распределение:

<i>Биты адреса</i>	<i>Полное число</i>	<i>Доля</i>	
1	41	0,103	
2	89	0,223	
3	47	0,118	
4	90	0,226	
5	46	0,115	
6	57	0,143	
7	18	0,045	
8	4	0,010	
9	5	0,012	
10	<u>2</u>	<u>0,005</u>	
Всего		399	1,00

Вы остановились на двух вариантах операндов:

вариант А: 4-битовый короткий непосредственный и 10-битовый длинный непосредственный,

вариант Б: 8-битовый короткий непосредственный и 10-битовый длинный непосредственный.

Какой из этих вариантов даст минимальное взвешенное среднее числа битов для непосредственных операндов?

3.7. Выведите правила для десятичной коррекции команд вычитания процессора x86.

3.8. Вы проектируете тип данных с избытком, который должен представлять числа 0–99 в одном байте. Каким будет значение избытка?

3.9. Покажите 10 десятичных цифр для 4-битового кода с весами +6, +2, –2, +1.

3.10. В обсуждении формата плавающей точки IBM S360 было выдвинуто следующее утверждение: если 16 состояний 16-ричного поля распределены равномерно, в дроби не используется приблизительно $3/4$ одного бита. В худшем случае не используется 3 бит. Объясните и обоснуйте это утверждение.

3.11. С помощью данных [16] из табл. 3.9 определите, каково среднее взвешенное числа сдвигов, требуемых для нормализации типа данных с плавающей запятой и основанием 2.

3.12. ISA x86 предусматривает тип данных BCD. Не является ли этот тип избыточным при наличии команды десятичной коррекции? Обоснуйте ваш ответ.

3.13. Проблема 2000 года возникла из-за того, что в байте с данными BCD-типа кодируются лишь две младшие цифры года. Как вы должны знать, при таком способе кодирования остаются неиспользуемые состояния. Спроектируйте 1-байтовое кодирование, которое будет указывать год в двадцатом и двадцать первом столетии. В 8 бит можно закодировать 256 состояний, так что кодирование 200 состояний не составит труда.

3.14. В книге говорится, что смещение порядка в формате плавающей запятой IBM S360 равно 64. Каково двоичное значение порядка?

3.15. Смещение порядка для IBM S360 составляет 2^{n-1} , а в IEEE-формате $2^{n-1} - 1$, где n — число битов порядка. Объясните, почему эти смещения отличаются, и перечислите преимущества и недостатки каждого решения.

3.16. Рассмотрите младшие 4 бит кода операции с плавающей запятой IBM S360. Как организуется привязка типа данного к операции?

3.17. Объясните, почему скрытый бит можно использовать только в формате плавающей запятой с основанием 2.

3.18. Опишите правила IEEE округления до ближайшего для данных с плавающей запятой.

3.19. Пусть дробная часть мантииссы формата IEEE для плавающей запятой занимает 4 бит и предусмотрены оба бита округления. Начните с значения 0,010000 и выполняйте инкремент бита залипания вплоть до значения 0,101111. Воспользуйтесь правилами округления в соответствии с таблицей 3.6; постройте график округленных значений 4-битовой мантииссы вместе с точными значениями (6 бит) для каждого из возможных точных значений.

3.20. Компьютер Атанасова — Берри, описанный в гл. 1, использует для решения линейных алгебраических уравнений метод гауссовых исключений (см. рис. 1.9). Покажите, как эта машина решает систему уравнений $3x + 2y = 12$, $4x + y = 11$ при использовании двоичных типов данных.

3.21. Нельзя ли объединить в один шаг два шага операции десятичной коррекции x86? Если нет, объясните, почему.

3.22. В разделе 3.3 говорится, что одним из преимуществ команд фиксированной длины является отсутствие случаев перехода команды через границу страниц. Объясните, почему это так.

3.23. Посмотрите документацию Intel по любому процессору семейства x86 и объясните, как выполняются арифметические операции над десятичными числами со знаком.

3.24. В разделе 3.2.4 обсуждаются комбинации команд и кодов операций фиксированной и переменной длин. Почему в список вариантов не включена комбинация с командой фиксированной длины и кодом операции фиксированной длины?

3.25. Программный счетчик процессора VAX назначен регистру 15 регистрового файла. Обсудите преимущества и недостатки такого решения.

3.26. Посмотрите документацию IBM S360 и постройте таблицу, эквивалентную табл. 3.13 для классов кодов операций.

3.27. Пример в разделе 3.2.3 показывает, что при основании 16 требуется на 2 бита меньше, чем при основании 2 при том же диапазоне. Уравнение было решено в предположении, что при основании 16 имеется всего 1 бит порядка. Приведите другое решение уравнения, которое не требует такого предположения, и покажите, что полученный результат имеет общий характер.

3.28. В разделе 3.2.3 отмечалось, что средняя длина команды процессора 8086 составляет 2,25 байта. Это среднее было получено путем испытаний в 1989 г. Как вы думаете, сейчас число байтов на команду в процессоре Pentium Pro стало меньше или больше этой величины? Объясните свой ответ и дайте оценку, которую можно использовать при моделировании.

3.29. Дайте обоснование функции ограничения при операции округления до ближайшего в стандарте IEEE плавающей запятой для случая, когда бит округления равен 1, а бит залипания — 0.

3.30. На рисунке 3.9 изображены три класса методов обслуживания прерываний. Однако из текста следует, что таких методов четыре: комбинации «между командами» и «внутри команды», а также продолжения или рестарта. Поясните, как эти четыре метода описаны в табл. 3.9.

3.31. Выведите общее правило выполнения коррекции смещенного порядка после сложения или вычитания.

3.32. Относятся ли целые типы данных Intel к одному из шести типов данных, описанных в табл. 3.6? Обоснуйте свой ответ.

ГЛАВА 4

СИСТЕМЫ ПАМЯТИ

4.0. Введение

В главе 1 мы описывали компьютер как систему, центральным звеном которой является память. Состояние процесса сохраняется в памяти, затем это состояние модифицируется процессором и возвращается в память. Ввиду важности памяти мы прежде всего должны рассмотреть различные аспекты ее архитектуры и реализации и лишь после этого перейти к вопросам проектирования процессора, которым посвящены гл. 5 и 6. В главе 3 обсуждались типы данных и было подчеркнуто, что они различаются длиной: 1 байт, 2 байт и т. д. В этой главе, описывая организацию памяти, мы будем опираться на понятие адресуемого элемента (addressable unit, AU). Для таких процессоров, как IBM S360, адресуемым элементом является байт. Для других процессоров AU может быть 2 или 4 байт.

Состояние процесса

Процессом называется выполняемая программа. Программа определяется как последовательность команд, обрабатывающих данные и меняющих состояние процесса. Таким образом, состояние процесса состоит из программы, исходных данных и результатов. Выполнение программы приводит к преобразованию данных; память должна предоставлять команды и данные и принимать результаты.

Отсюда следует, что память должна быть достаточно велика, чтобы в ней поместилось состояние процесса, и память может быть фактором, ограничивающим производительность компьютера. Состояние процесса изменяется от команды к команде. В некоторых программных системах, примерами которых являются MS DOS и Windows 3.1, команды могут рассматриваться как данные и модифицироваться программой, которая в этом случае носит название самомодифицирующегося кода; эта концепция рассматривалась в гл. 1 применительно к примеру фон-неймановской программы. В таких системах при проектировании системы памяти должно учитываться то обстоятельство, что часть состояния процесса, хранящая команды программы, может модифицироваться по ходу выполнения программы.

Желательные характеристики памяти

Система памяти имеет три характеристики, которые в идеальном случае должны быть следующими:

1. **Размер:** бесконечно большой, отсутствие ограничений на размер программы или данных.
2. **Скорость:** бесконечно большая, чтобы время ожидания было минимально и соответствовало новейшим технологиям построения запоминающих устройств.
3. **Стоимость:** в расчете на бит стоимость должна быть минимально возможной.

Очевидно, что перечисленные требования не могут быть достигнуты одновременно, так как являются взаимоисключающими. Однако возможности современных технологий полупроводников и магнитной памяти позволяют приблизиться к идеальному соотношению характеристик.

Системы памяти характеризуются двумя важными параметрами [2].

«Время запаздывания, или время отклика, представляет собой полное время, требуемое для обработки (от возбуждения до ответа) конкретного элемента данных в определенной фазе вычислительного процесса».

«Пропускная способность характеризует скорость событий. В частности, вычислительная, или исполнительная, пропускная способность представляет собой число команд, выполняемых в секунду, а пропускная способность памяти определяется как скорость извлечения из памяти операндов и других операционных слов памяти (слово в секунду)».

Эти параметры независимы и во многих случаях проектировщик может обеспечить их разумное сочетание. Для некоторых процессоров и систем памяти тот или иной из этих параметров может иметь особое значение.

Иллюстрацию времени отклика и пропускной способности вне сферы компьютеров можно найти в системах удаленной связи. Гео-синхронные спутники оказываются идеальными для межкомпьютерной связи, где большое время запаздывания (приблизительно 250 мс) поглощается большой длительностью передачи данных. В таких приложениях на первый план выходит пропускная способность. Однако при таких временах запаздывания телефонный разговор между двумя абонентами становится весьма неприятным. В результате телефонная связь через спутники заменяется оптоволоконными подводными кабелями, которые характеризуются практически нулевым временем запаздывания. Волоконная оптика с течением времени может быть заменена низкоорбитальными спутниками, которые обеспечивают сравнительно малые времена запаздывания.

4.1. Иерархическая память

В этом разделе будет показано, как проектировщики реализуют системы памяти, сочетающие разумную цену с производительностью, приближающейся к идеальной. Такие системы памяти имеют иерархическую многоуровневую структуру. Память, ближайшая к процессору, делается относительно небольшой и быстрой, но отличается высокой стоимостью в расчете на бит. Этот уровень получил название кэш-памяти. Реальная память, иногда называемая основной или оперативной, имеет больший объем, работает медленнее и характеризуется меньшей стоимостью бита. Самый низкий уровень в иерархии памяти обычно представлен магнитным диском, который отличается самым большим временем отклика и самой низкой пропускной способностью; однако эта память может быть очень велика и дешева в расчете на бит. Иерархия памяти показана на рис. 4.1.

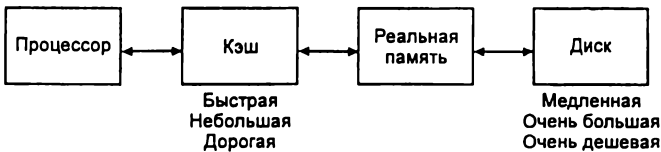


Рис. 4.1. Иерархия памяти.

Обратите внимание на то, что на рис. 4.1 не показан регистровый файл процессора. Регистровый файл представляет собой программно-управляемую кэш-память; обычно он не включается в иерархию системной памяти. Следует также добавить, что в системе может быть более одного кэша.

4.2. Страничная виртуальная память

Страничная виртуальная память является воплощением первого сформулированного выше пожелания, именно, чтобы процессору^{*)} был доступен очень большой объем памяти. Ввиду важности этого требования мы прежде всего обсудим взаимоотношение между виртуальной и реальной памятью. Виртуальная память представляет собой механизм, с помощью которого процессор адресует диск, — большую и медленную память, находящуюся на нижнем уровне иерархии. Устройства памяти между процессором и диском служат для повышения произ-

^{*)} Спорное утверждение, что виртуальная память обеспечивает доступ к памяти очень большого объема прежде всего процессору. — *Прим. ред.*

водительности (уменьшения времени запаздывания и увеличения пропускной способности) большого, медленного диска. Если бы каждая ссылка на команду или данные требовала обращения к диску, производительность процессора была бы чрезвычайно низкой.

Однако почему виртуальная память так важна? Память большого объема нужна для больших программ и наборов данных. Ранние компьютеры с небольшой реальной памятью требовали, чтобы передача данных между реальной памятью и диском осуществлялась непосредственно операционной системой или пользователем. Виртуальная память предоставляет возможность автоматического управления этой частью иерархии памяти с помощью комбинации аппаратных и программных средств.

Интерфейс виртуальной памяти изображен на рис. 4.2. Реальная память объемом 16 Мбайт и виртуальная объемом 2 Гбайт показаны лишь в качестве иллюстрации; многие современные системы памяти имеют гораздо больший объем. Пространство виртуальной памяти разделено на участки равного размера, называемые страницами. Страница в современных компьютерах составляет 1, 2 или 4 Кбайт. Реальная память также делится на равные участки того же размера, называемые страничными кадрами. Обмен информацией между пространствами виртуальной и реальной памяти осуществляется только целыми страницами.

Из-за того что реальная память меньше виртуальной, страницы при перемещении на более высокие уровни, ближе к процессору, теряют идентификацию по виртуальным адресам. Поэтому эффективный адрес, формируемый процессором и называемый теперь виртуальным адресом, должен быть так или иначе преобразован, транслирован перед тем, как он будет представлять адрес реальной памяти. Между про-

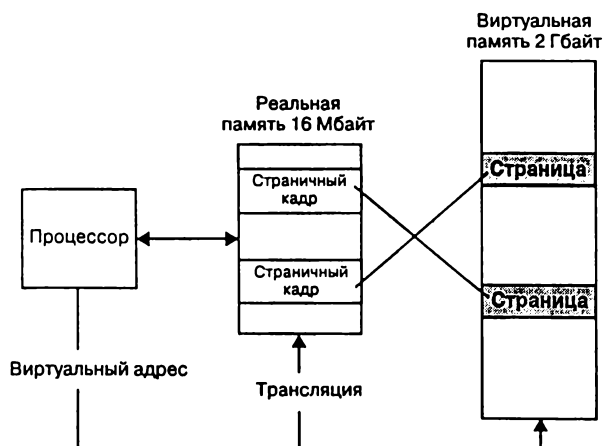


Рис. 4.2. Выделение страниц и трансляция адресов.



Рис. 4.3. Интенсивность отказа страниц без локализации и с локализацией.

чим, архитектура, изображенная на рис. 4.2, повторяет, за исключением размеров устройств памяти, структуру компьютера ATLAS, построенного в начале 1960-х годов в Манчестерском университете [6].

Хотя основной целью построения системы виртуальной памяти является получение для процессора большого и прозрачного адресного пространства, однако нельзя не учитывать производительность памяти. На рисунке 4.2 показано, как процессор обращается к реальной памяти и к диску. Если адреса, формируемые процессором, распределены равномерно, как показано на рис. 4.3, доля ссылок, не найденных в реальной памяти (называемая интенсивностью отказа страниц)³⁷⁾, будет равна:

интенсивность отказа страниц (равномерное распределение) =

$$= 1 - \frac{\text{размер реальной памяти}}{\text{размер виртуальной памяти}}$$

В случае равномерного распределения и конкретных данных, показанных на рис. 4.2 (16 Мбайт реальной памяти и 2 Гбайт виртуальной), 0,008 от всех ссылок будут найдены в реальной памяти и 0,992 — на диске. Зачем же стараться использовать более быструю и более дорогую реальную память, если большинство обращений к памяти должны обслуживаться очень медленным диском?

³⁷⁾ Отказ страницы возникает, когда при адресации реальной памяти страница, содержащая адресуемый элемент, отсутствует в реальной памяти. Отказ страницы схож с кэш-промахом, который будет обсуждаться позже.

Ответом на этот вопрос служит тот факт, что обращения к памяти распределены неравномерно. В силу естественных свойств программы и структур данных формируемые процессором адреса имеют две формы локализации.

Пространственная локализация (spatial locality): в ближайшем будущем вероятны ссылки на области адресного пространства, близкие к текущему адресуемому месту.

Временная локализация (temporal locality): команды и данные, на которые программа ссылалась за последнее время, вероятно, будут адресоваться и в ближайшем будущем.

Ввиду наличия локализации интенсивность отказа страниц оказывается заметно меньше, что показано на рис. 4.3. Интенсивность отказа страниц³⁸⁾ теперь можно определить как

$$\text{интенсивность отказа страниц} = \frac{\text{число ссылок, найденных в реальной памяти}}{\text{полное число ссылок}}$$

Локализация существенно снижает интенсивность отказа страниц даже для небольшой (относительно объема виртуальной памяти) реальной памяти, поэтому большинство ссылок обслуживаются в быстрой реальной памяти.

Для потока адресов команд характерна пространственная локализация в силу того, что программный счетчик по ходу программы инкрементируется, и, за исключением переходов, следующая команда находится по следующему большему адресу относительно текущей. По этой причине размер страницы выбирается больше длины одной команды, и как только ссылка на команду вызывает отказ страницы, в реальную память загружается страница целиком. После того как страница выбрана в реальную память, очередная команда с большой вероятностью окажется именно там, и в качестве времени запаздывания будет выступать время запаздывания реальной памяти, а не диска. Пространственную локализацию иногда называют предварительным просмотром (look-ahead), потому что механизм выборки как бы заранее просматривает поток команд или данных.

Временную локализацию иногда называют просмотром назад (look behind). Это значит, что страница, после того как она выбрана с диска в реальную память, будет там находиться в ожидании будущего использования. Только когда возникнет необходимость в свободном страничном кадре или в переключении контекстов, страница будет возвращена на диск. Для потока адресов данных характерна временная локализация, а пространственная локализация более присуща потоку команд. После того как данное было использовано, к нему с большой вероятностью будут повторные обращения. Помимо этого временная локализация характерна для программных циклов, поскольку команды, входящие в цикл, в ближайшем будущем будут выполняться повторно.

³⁸⁾ Называется также интенсивностью ошибок отсутствия страниц.

При обращении к реальной памяти аппаратное и программное обеспечение компьютера должно выполнить ряд шагов. В дальнейшем рассматривается операция чтения из памяти. Шаги, из которых состоит обращение к памяти, таковы:

1. Определить, находится ли в реальной памяти страница, содержащая затребованный адресуемый элемент (AU).

2. Если эта страница находится в памяти, транслировать ее имя^{*)}, сцепить со смещением, прочитать реальную память, передать AU процессору и продолжить выполнение.

3. Если страницы в памяти нет, вызвать обработчик отказа страницы:

а) если имеется свободный страничный кадр, выбрать с диска в реальную память страницу, содержащую AU, обновить таблицу трансляции имен и продолжить выполнение;

б) если в реальной памяти нет свободного страничного кадра, выгрузить страницу на диск; вернуться к шагу 3а.

Существенным моментом при выполнении описанных шагов является определение наличия или отсутствия в реальной памяти страницы, содержащей запрошенный AU, что должно выполняться до того, как произойдет обращение к реальной памяти. Этот процесс носит название раннего выбора (early select). Страничная память для нахождения соответствия виртуальных адресов и реальной памяти использует конгруэнтное отображение³⁹⁾. Это означает, что виртуальные адреса делятся на две части, имя страницы и смещение внутри стра-

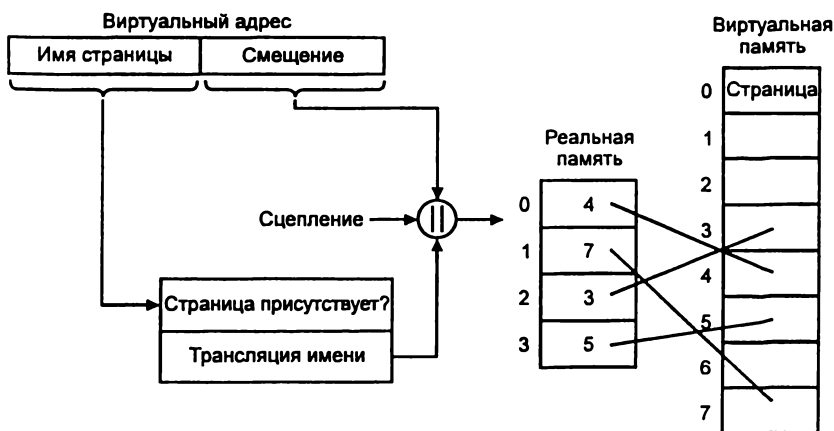


Рис. 4.4. Конгруэнтное отображение (с любезного согласия Jones and Bartlett Publishers).

^{*)} Под именем страницы здесь и в дальнейшем автор понимает ее номер. — Прим. перев.

³⁹⁾ Два целых числа называются конгруэнтными по модулю m , если после деления на модуль m оба имеют одинаковый остаток r .

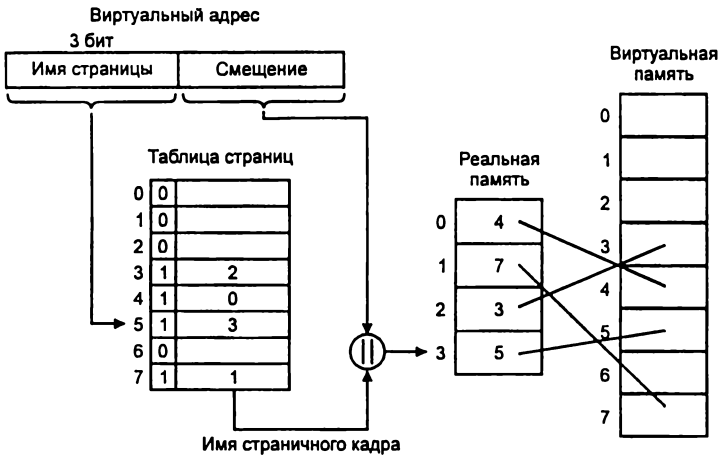


Рис. 4.5. Прямая трансляция имени (с любезного согласия Jones and Bartlett Publishers).

ницы, как это показано на рис. 4.4. С математической точки зрения конгруэнтное отображение заключается в том, что адрес делится на размер страницы; частное является именем страницы, а остаток — смещением.

Упрощенная система, показанная на рис. 4.4, имеет виртуальное адресное пространство размером в 8 страниц и 4 страничных кадра в реальной памяти. Из-за того что размер страницы составляет целую степень 2 от размера АУ, скажем, 2 Кбайт, виртуальный адрес делится на две части: имя страницы и смещение внутри страницы. Для двухкилобайтовых страниц смещение должно содержать 11 бит, и для нашей простой системы имя страницы поместится в 3 битах.

Линии на рисунке показывают текущее отображение страничных кадров реальной памяти: страница 3 в кадре 2, страница 7 в кадре 1 и т. д. Это отображение приведено в табл. 4.1. Имя страницы транслируется таблицей (например, 4 преобразуется в 0), что дает адрес страничного кадра в реальной памяти. Между именем страницы и адресом страничного кадра не существует прямого соответствия, потому что страница может быть размещена в любом свободном страничном кадре. Транслированное имя страницы сцепляется со смещением, образуя адрес затребованного АУ в реальной памяти.

Существуют два способа трансляции имени страницы в адрес страничного кадра: 1) прямая трансляция и 2) инверсная трансляция. Оба метода используются в современных системах памяти.

Прямая трансляция

На рисунке 4.5 показано то же отображение памяти, что и на рис. 4.4, с указанием элементов, входящих в систему прямой трансляции. Таб-

лица страниц индексируется именем страницы⁴⁰⁾. Элементы таблицы страниц включают два поля; 1-битовое поле указывает, присутствует ли запрашиваемая страница в реальной памяти, а второе поле содержит начальный, или базовый адрес страничного кадра реальной памяти с данной страницей. Если, например, имя запрашиваемой страницы есть число 5, читается элемент номер 5 таблицы страниц, по 1 в бите присутствия выясняется, что страница присутствует, и ее страничный кадр в реальной памяти имеет начальный адрес 3. Значение 3 сцепляется со смещением с образованием адреса реальной памяти.

Таблица 4.1. Карта трансляции имен страниц

Имя страницы	Адрес страничного кадра
4	0
7	1
3	2
5	3

Если запрашивается страница 2, то по значению 0 бита присутствия элемента 2 таблицы страниц выясняется, что эта страница отсутствует в реальной памяти и должна быть загружена из виртуальной памяти. Однако реальная память заполнена, так как в ней заняты все страничные кадры. Для того чтобы освободить место для запрашиваемой страницы, одна из присутствующих страниц должна быть сброшена на диск. Имеются различные алгоритмы выбора сбрасываемой страницы; эти алгоритмы будут описаны ниже.

Характерной чертой прямой трансляции имени является наличие для каждой страницы виртуальной памяти своего элемента в таблице страниц. Для небольших виртуальных адресов такой способ оказывается весьма эффективным. Таблица страниц реализуется аппаратно, и поэтому время запаздывания реальной памяти лишь немного увеличивается. Однако для больших виртуальных адресов таблица страниц становится столь большой, что ее приходится размещать в реальной памяти, что значительно увеличивает время запаздывания.

✎ Пример

Рассмотрим компьютер с 32-битовым виртуальным адресом и 4-килобайтовой страницей. Из скольких элементов должна состоять таблица страниц прямой трансляции? Если каждый элемент таблицы

⁴⁰⁾ Термин «индекс» используется в тех случаях, когда выбирается элемент таблицы. Термин «адрес» используется при обращении к ячейке памяти.

страниц имеет размер 4 байт, каков будет полный размер таблицы страниц?

Решение

Число страниц в виртуальной памяти определяется, как число адресуемых единиц в виртуальном адресном пространстве, деленное на размер страницы:

$$\text{число страниц} = 2^{32} / 2^{12} = 2^{20}$$

Поскольку для каждой виртуальной страницы в таблице страниц должен существовать отдельный элемент, таблица страниц будет содержать 2^{20} элементов. При размере каждого элемента 4 байт размер таблицы составит 2^{22} байт, или 4 Мбайт памяти.

Замечание

Для больших виртуальных адресов, с которыми работают современные процессоры, размер таблицы страниц прямой трансляции может оказаться значительно больше реальной памяти.

Современные процессоры характеризуются даже большими виртуальными адресами, чем указаны в рассмотренном выше примере; например, в IBM S370 используются 42-битовые виртуальные адреса, а Intel Pentium Pro и IBM PowerPC имеют 52-битовые виртуальные, или линейные, адреса. Поэтому в Intel Pentium Pro реализована многоуровневая прямая система трансляции имен страниц с частичным заполнением, показанная на рис. 4.6. Примерно так же выглядит и система трансляции IBM S370; она будет рассмотрена в разделе 4.6.

Линейный адрес, имеющий размер 32 бит, расширяется за счет двадцати битов управляющего регистра CR3, что дает в сумме 52-би-

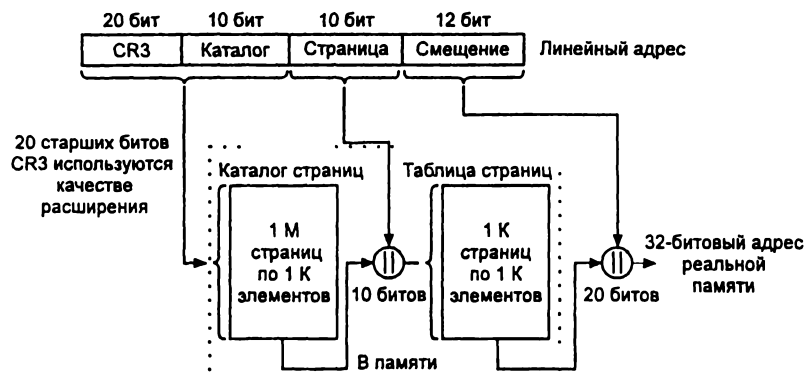


Рис. 4.6. Система трансляция имен страниц Intel 486 и Pentium Pro (с любезного согласия Jones and Bartlett Publishers).

товый виртуальный адрес. Старшие 30 бит линейного адреса служат индексом элемента в каталоге страниц, что обеспечивает в целом 1М страниц по 1К страничных элементов в каждой. 10-битовый выход каталога страниц сцепляется со следующими 10 битами виртуального адреса для образования индекса в таблице страниц, содержащей 1К таблиц по 1К элементов каждая. 20-битовый выход таблицы страниц сцепляется с 12-битовым смещением, образуя 32-битовый адрес реальной памяти. Поскольку все эти таблицы находятся в реальной памяти, трансляция имени таблицы требует некоторого числа циклов памяти, что увеличивает время запаздывания реальной памяти — это серьезный недостаток системы. Возможности преодоления этого недостатка с помощью специальной кэш-памяти будут рассмотрены ниже.

Многоуровневая таблица страниц существенно уменьшает размер таблицы страниц прямой трансляции, хранящейся в памяти. Однако полностью таблица страниц находится в виртуальной памяти, и ее активная часть сама должна отображаться с помощью страничной трансляции на свободные страничные кадры. Страничная трансляция таблиц страниц создает возможность отказа страницы, содержащей таблицу страниц, в процессе трансляции имени страницы. Такой отказ страницы может привести к блокировке или полному отказу системы. Эта проблема решается, главным образом, путем размещения таблиц страниц в защищенной области реальной памяти, на которую не распространяется механизм трансляции адресов.

🔗 Пример

Многоуровневая система трансляции имен страниц с частичным заполнением, применяемая для процессоров Pentium Pro, существенно уменьшает объем реальной памяти, выделяемой для таблиц прямой трансляции. Какая часть полной таблицы страниц активна в каждый момент для системы, изображенной на рис. 4.6?

✍ Решение

Число элементов полной таблицы страниц при 52-битовом виртуальном адресе и 4-килобайтовых страницах равно:

$$\text{число элементов} = 2^{52} / 2^{12} = 2^{40}$$

Число элементов в частично заполненной таблице страниц обоих уровней равно:

$$\begin{aligned} \text{число элементов} &= \text{элементы каталога страниц} + \text{элементы таблицы страниц} = \\ &= 2^{30} + 2^{20} \cdot 2^{30} \end{aligned}$$

Таким образом, активная часть полной таблицы страниц составит:
активная часть = $100 \times 2^{30} / 2^{40} = 0,097\%$.

✓ Замечание

При использовании частично заполненной многоуровневой таблицы страниц интенсивность отказов страниц трансляции будет ненулевой. Однако на практике интенсивность отказов при трансляции имени приближается к нулю, за исключением моментов контекстных переключений. Причина этого заключается в том, что, поскольку для страниц характерна временная локализация, информация из таблицы трансляции имен также локализована во времени. Поддержка таблиц страниц архитектурой ISA будет рассмотрена в гл. 8.

Инверсная трансляция

Для современных компьютеров характерны 52-битовые виртуальные адреса. При использовании прямой трансляции таблицы трансляции либо оказываются очень велики, либо должны использовать иерархическую структуру с частичным заполнением, что, как отмечалось выше, ведет к отказам страниц с таблицами страниц. Решение этой проблемы было найдено в использовании инверсной страничной трансляции, которую реализовала IBM в машинах RS/6000 и PowerPC, а также Hewlett-Packard в своей архитектуре. В дальнейшем обсуждении мы будем ссылаться на рис. 4.4, в котором нас будет интересовать блок «Страница присутствует?, Трансляция имени». Этот блок может быть реализован с помощью инверсной таблицы страниц.

Инверсная таблица страниц имеет по одному элементу для каждого страничного кадра в реальной памяти, что существенно уменьшает ее размер. Например, 32-Мбайтовая реальная память при 4-Кбайтовых страницах будет содержать 8К страничных кадров, и в инверсной таблице страниц должно быть всего 8К элементов с 13-битовыми индексами. В прямой таблице страниц для 52-битовых виртуальных адресов было бы 2^{40} элементов с 40-битовыми индексами. Обратите внимание на то, что размер инверсной таблицы страниц зависит не от размера виртуального адреса, а только от размера установленной реальной памяти.

52-битовый виртуальный адрес 4-Кбайтовой страницы должен содержать 40 бит для имени страницы. В этом случае возникает вопрос, каким образом длинное имя страницы преобразуется в короткий (13 бит для 8К страничных кадров) адрес для индексирования внутри таблицы страниц. Решение этой проблемы найдено в использовании хеширования. Хеширование преобразует длинный, но слабо заполненный адрес в короткий адрес с равномерным заполнением. Обзор хеширования можно найти в работе [7].

Хеширование

Чтобы показать, как работает хеширование, воспользуемся следующим примером. Мы хотим спроектировать систему для доступа к записям класса, включающего 66 студентов; каждый студент имеет уникальный номер страховой карточки. Этот номер содержит 9 десятичных цифр, что дает приблизительно 10^9 уникальных адресов, в то время как в нашем классе имеется всего 66 уникальных студентов. Таким образом, пространство страховых адресов заполнено очень слабо. Нам хотелось бы иметь для записи информации о студентах небольшую таблицу из 66 элементов, а не огромную таблицу из 10^9 элементов.

Для доступа к таблице мы берем страховой номер студента и преобразуем его («хешируем») в маленькое число, которое будет служить индексом в маленькой таблице. Простая хеш-функция в качестве индекса в таблице страниц использует две младшие цифры страхового номера студента. Двух цифр в данном случае достаточно, потому что размер класса меньше 100. Сама таблица содержит указатели на папки с информацией о каждом из студентов аналогично именам страничных кадров на рис. 4.5.

Для иллюстрации этого процесса в табл. 4.2 приведен список страховых номеров студентов без первых трех цифр. Распределение хешированных значений приведено в табл. 4.3. Например, первый страховой номер оканчивается на 50, и в 50-й позиции таблицы 4.3 записана 1.

Таблица 4.2. Страховые номера студенческого класса

xxx-24-9750	xxx-06-5204	xxx-30-7338	xxx-33-5575	xxx-79-1528	xxx-24-4547
xxx-15-5434	xxx-92-3719	xxx-16-0336	xxx-07-9694	xxx-78-3559	xxx-71-3545
xxx-32-4770	xxx-79-2257	xxx-15-8675	xxx-14-2174	xxx-66-2744	xxx-49-1053
xxx-74-2329	xxx-80-3192	xxx-33-7633	xxx-65-4266	xxx-64-7567	xxx-31-9175
xxx-51-7706	xxx-23-2470	xxx-37-8075	xxx-24-7287	xxx-33-4790	xxx-25-8960
xxx-47-9713	xxx-58-9175	xxx-35-5887	xxx-67-0236	xxx-81-6899	xxx-45-5440
xxx-07-8151	xxx-19-6783	xxx-31-7123	xxx-43-5165	xxx-33-5602	xxx-92-2685
xxx-15-1499	xxx-33-7364	xxx-14-5397	xxx-15-8572	xxx-67-6292	xxx-21-2918
xxx-09-3439	xxx-94-9000	xxx-70-8804	xxx-79-1364	xxx-98-6396	xxx-27-3156
xxx-20-8363	xxx-25-7900	xxx-16-6251	xxx-54-9597	xxx-25-4877	xxx-14-2758
xxx-24-6094	xxx-03-7826	xxx-03-3545	xxx-07-7529	xxx-72-6559	xxx-72-3616

Следует обратить внимание на то, что некоторые из 100 возможных чисел на выходе хеш-функции появились 0, 1, 2 или 4 раза, потому что выходные данные хеш-функции распределены неравномерно. Что происходит, когда возникает более одного попадания в один элемент таблицы (например, элемент 75 получил 4 попадания)? Такое событие, называемое столкновением, или конфликтом, может быть

Таблица 4.3. Распределение хешированных страховых номеров

		Единицы									
		0	1	2	3	4	5	6	7	8	9
Десятки	0	2		1		2		1			
	1				1			1		1	1
	2				1			1		1	2
	3				1	1		2		1	1
	4	1				2	1		1		
	5	1	2		1		1	1	1	1	2
	6	1			1	2	1	2			
	7	2		1		1	4		1		
	8				1		1		2		
	9	1		2		2		1	2		2

разрешено разными способами. Один из них заключается в связывании элементов таблицы, имеющих одинаковый хеш-индекс, в связанный список.

На рисунке 4.7 показан связанный список из трех элементов. Доступ к первому элементу списка осуществляется по имени элемента в хешированной таблице. Элемент 1 имеет адрес связи с элементом 2, который, в свою очередь, имеет адрес связи с элементом 3. Последний элемент связанного списка вместо адреса следующего элемента имеет в качестве терминатора символ Т. Для четырех страховых номеров, которые в результате хеширования попали в один (75-й) элемент таблицы, потребуется связанный список из четырех элементов.

Связные списки могут храниться в реальной памяти, причем элементы и их указатели могут занимать любые ячейки в таблице страниц. Например, столкновения для хеш-индекса 75 могут храниться в ячейках 75, 20, 21 и 22 таблицы страниц. Добавление или удаление элементов из связанного списка осуществляется путем коррекции связей.

Одноуровневая система инверсных таблиц страниц

Рассмотрим теперь систему инверсных таблиц страниц, предназначенную для трансляции имен страниц. Блок-схема простой системы такого рода изображена на рис. 4.8. Имя страницы хешируется в

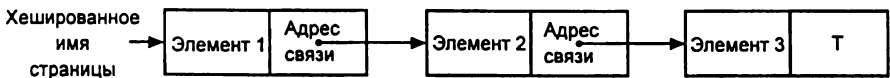


Рис. 4.7. Связанный список.

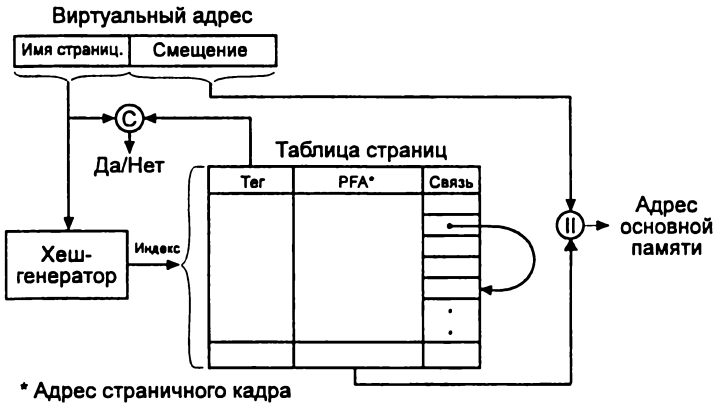


Рис. 4.8. Одноуровневые инверсные таблицы страниц (с любезного согласия Jones and Bartlett Publishers).

меньшее значение, которое индексирует таблицу страниц. Одно из полей каждого элемента таблицы страниц выделено для тега, содержащего имя той страницы, которая является резидентной в адресуемом страничном кадре.

Процесс поиска требуемой информации в инверсной таблице страниц и получения адреса страничного кадра называется пробой (зондированием).

Схема процесса такова:

1. Хеширование имени страницы.
2. Получение индекса в таблице страниц. Если тег соответствует имени страницы, переход на шаг 4; в противном случае переход на шаг 3.
3. Находится ли в поле связи символ-терминатор?
Да: Отказ страницы; вызов обработчика отказа страниц.
Нет: Использование связи как индекса; переход на шаг 2.
4. Адрес страничного кадра читается и сцепляется со смещением с образованием адреса реальной памяти.

На практике желательно иметь таблицу страничных кадров, вдвое превышающую по размеру число страничных кадров в реальной памяти. Это нужно для того, чтобы разбросать по большей площади получаемые хешированием индексы и уменьшить тем самым вероятность конфликтов. При этом как теоретические расчеты, так и эксперименты показывают, что взвешенное среднее число проб для доступа к инверсной таблице страниц составляет для аккуратно разработанного алгоритма хеширования всего 1,5. Такое число проб требует значительно меньше циклов памяти, чем доступ к многоуровневой таблице страниц прямой трансляции.

✎ Пример

Используя данные из табл. 4.2 и результаты хеширования для двух младших цифр, приведенные в табл. 4.3, определить, каково среднее число проб, требуемое для доступа к элементу.

✎ Решение

Это задача на простое взвешенное среднее. 48 элементов могут быть найдены одной пробой, 16 двумя пробами, 1 тремя пробами и 1 четырьмя пробами. Таким образом, среднее взвешенное число проб составит

$$\frac{(48 \times 1) + (16 \times 2) + (1 \times 3) + (1 \times 4)}{66} = \frac{87}{66} = 1,32 \text{ пробы}$$

✓ Замечание

Среднее значение 1,32 пробы весьма близко к приведенному выше значению 1,5.

Производительность виртуальной памяти

Виртуальная память, как показано на рис. 4.2, является иерархической. Каково эффективное время запаздывания реальной памяти с системой виртуальной памяти с точки зрения процессора? Рассмотрим стационарную модель; переходные процессы, например контекстное переключение, изменят производительность по сравнению с вычисленной для стационарного случая. Модель имеет следующие параметры:

t_{eff} = эффективное время запаздывания реальной памяти

t_i = время трансляции имени

P_f = интенсивность отказов страниц

t_{rm} = время запаздывания реальной памяти

T_{drm} = транспортное время, необходимое на перемещение страницы с диска в реальную память или наоборот (время запаздывания + время пересылки)

t_{os} = время, требуемое операционной системе для обслуживания отказа страницы

Производительность вычисляется с помощью табулирования параметров, как это было описано в гл. 2 и показано в табл. 4.4. Имеются два события: отказ страницы и попадание на страницу. Этим событиям соответствуют определенные времена.

Таблица 4.4. Вычисление времени запаздывания виртуальной памяти

Событие	Вероятность	Время	Произведение
Попадание	$1 - P_f$	$t_i + t_{rm}$	$(1 - P_f)(t_i + t_{rm})$
Промех	P_f	$t_i + T_{drm} + t_{os}$	$P_f(t_i + T_{drm} + t_{os})$
Сумма = 1		Сумма = $(1 - P_f)(t_i + t_{rm}) + P_f(t_i + T_{drm} + t_{os})$	

Упростив сумму из табл. 4.4, получим:

$$t_{elrm} = t_i + (1 - P_f)t_{rm} + P_f(T_{drm} + t_{os}) \quad 41)$$

Локализация существенно повышает производительность системы памяти по сравнению с системой, получающей адреса, равномерно распределенные по всему виртуальному адресному пространству. Интенсивность отказа страниц, о которой уже шла речь выше, является ключевым параметром, используемым при анализе производительности. Хотя литературных данных по интенсивности отказов страниц мало, можно считать, что в современных системах стационарная интенсивность отказов страниц составляет порядок от 10^{-5} до 10^{-6} .

✎ Пример

Рассмотрим систему виртуальной памяти со следующими параметрами. Транспортное время системы диск — реальная память равно 20 мс, время запаздывания реальной памяти 200 нс, взвешенное среднее число проб трансляции имени 1,5, интенсивность отказа страниц 1×10^{-5} . Обработчик отказов страниц требует 2 мс компьютерного времени. Каково эффективное время запаздывания реальной памяти?

✎ Решение

Используя выведенную выше формулу, получим:

$$t_{elrm} = 1,5 \times 200 \times 10^{-9} + (1 - 1 \times 10^{-5}) \times 200 \times 10^{-9} + 1 \times 10^{-5} \times (20 \times 10^{-3} + 2 \times 10^{-3}) \approx 7,2 \times 10^{-7} = 720 \text{ нс}$$

✓ Замечание

Хотя 720 нс значительно больше, чем базовое время запаздывания реальной памяти, составляющее около 200 нс, однако по сравнению с дисковой системой без реальной памяти мы имеем ускорение, равное 28×10^3 .

⁴¹⁾ Эффективное время запаздывания можно было бы назвать взвешенным арифметическим средним временем запаздывания.

Резидентная страница в реальной памяти, модифицированная выполняемой программой, должна быть возвращена на диск перед тем, как страничный кадр будет использован для загрузки новой страницы. Это приводит к включению в t_{elrm} дополнительного члена P_m , который представляет собой вероятность модификации страницы. Считая, что все страничные кадры в реальной памяти заняты, взвешенное среднее время запаздывания реальной памяти изменится следующим образом:

$$t_{elrm} = t_t + (1 - P_t)t_{rm} + P_f(T_{drm} + t_{as})(1 + P_m)$$

Если распределение памяти позволяет иметь страницы, зарезервированные для команд, для этих страниц P_m всегда равно нулю. Некоторые страницы с данными могут претерпеть модификацию; эти страницы должны быть возвращены на диск перед использованием их страничных кадров для загрузки других страниц. Простой обработчик отказов страниц может исходить из предположения, что все страницы были модифицированы; в этом случае $P_m = 1$.

Многозадачность

Поскольку отношение T_{drm} к t_{rm} находится между 1×10^5 и 1×10^6 , многие системы не ожидают обслуживания отказа страницы, а на время ожидания запускают на выполнение новую программу. Эффективность такого решения подтверждается следующим примером. Процессор со скоростью 200 MIPS за время ожидания обслуживания отказа страницы (30 мс) может выполнить 6×10^6 команд. Таким образом, в многозадачном режиме ресурсы системы используются более эффективно. Однако однопользовательская система при отсутствии фоновой программы должна ожидать завершения обслуживания отказа страницы. В этом случае единственным решением будет применение более быстрых дисков или использование дискового кэша. Этот вопрос будет затронут в разделе 4.3.

Буфер ассоциативной трансляции

Как уже отмечалось выше, оба метода трансляции имени страницы — прямой и инверсный — требуют нескольких циклов памяти, что увеличивает время запаздывания при каждом обращении к памяти. В приведенном выше примере трансляция имени страницы добавляет к времени запаздывания реальной памяти 300 нс. Даже если произошло попадание в реальную память, время запаздывания остается около 500 нс. Что можно предпринять, чтобы устранить или хотя бы снизить эти затраты?

Решение состоит в использовании буфера ассоциативной трансляции (translation lookaside buffer, TLB). TLB представляет собой не-

большую быструю реализованную на регистрах таблицу страниц, которая хранит последние по времени использования транслированные адреса. Эксперименты показывают, что если эта быстрая память содержит совсем немного элементов, скажем, 10 или 16, высока вероятность того, что трансляция будет выполнена с помощью TLB без обращения к таблицам страниц в памяти.

TLB, реализованный на быстрых регистрах, может иметь время запаздывания приблизительно 10 нс. С таким TLB эффективное время запаздывания памяти для вышеприведенного примера уменьшится с 720 нс приблизительно до 430 нс, что даст повышение быстродействия 1,67. TLB схож с кэш-памятью, поэтому более подробно работа TLB будет описана в разделе 4.3.

4.3. Кэш-память

В разделе 4.2 мы обсуждали, как виртуальная память расширяет адресное пространство процессора. Однако время запаздывания реальной памяти слишком велико, чтобы удовлетворить требованиям высокопроизводительных процессоров. Даже при установке высокоскоростных модулей DRAM, которые используются сегодня в качестве реальной памяти, что-то еще надо предпринять для уменьшения времени запаздывания.

Пример

Рассмотрим процессор, который требует двух обращений к памяти на одну команду и работает в условиях последовательного выполнения команд. Пусть цикл памяти вместе с временем запаздывания составляет 70 нс. Если сам процессор работает бесконечно быстро, каково будет значение MIPS системы?

Решение

Время выполнения команды складывается из времени памяти и времени процессора:

$$\text{время выполнения команды} = 2 \times 70 \times 10^{-9} + 0 = 140 \text{ нс},$$

$$\text{MIPS} = \frac{1}{140 \times 10^{-9} \times 10^6} = 7,1$$

✓ **Замечание**

Современные микропроцессоры работают в диапазоне 100 – 500 MIPS. Производительность нашего бесконечно быстрого процессора ограничивается временем запаздывания памяти.

Повышение производительности достигается путем добавления еще одного уровня к нашей иерархии памяти, который называется кэш-памятью, или просто кэшем (рис. 4.9). Как уже отмечалось в разделе 4.2, виртуальное адресное пространство подразделяется на страницы равного размера. Эти страницы размещаются в страничных кадрах реальной памяти того же размера. Поскольку страницу можно разместить в любом свободном страничном кадре, в назначении страничных кадров страницам нет никакого особого порядка. При добавлении кэш-памяти из страничных кадров в реальной памяти берутся блоки размером 16–32 байт и помещаются в блоки кэш-памяти, к которым имеет доступ процессор. В современных процессорах кэш-память обычно имеет время запаздывания, равное одному процессорному такту, поэтому и команды, и данные могут выбираться без запаздывания, за исключением тех случаев, когда требуемый элемент программы не находится в кэш-памяти.

Перед тем как приступить к обсуждению вопросов проектирования кэш-памяти, полезно остановиться на требованиях, предъявляемых к памяти процессором. Память и вся система памяти должны поддерживать три или более потоков обращений процессора к памяти:

- 1) выборка процессором команд,
- 2) чтение процессором данных,

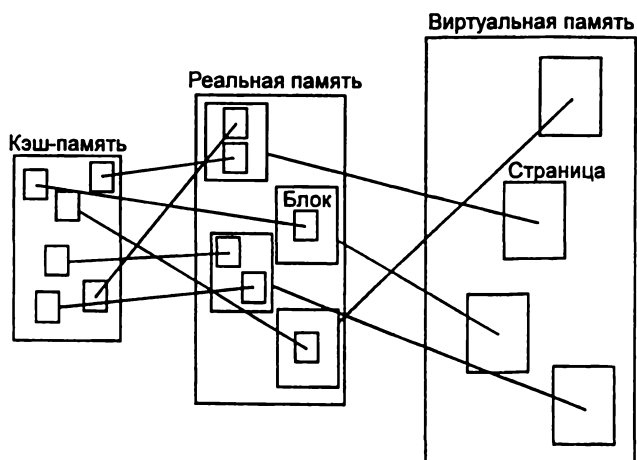


Рис. 4.9. Реальная память и кэш-память.

- 3) запись процессором данных⁴²⁾,
- 4) ввод-вывод.

Статистика обращений к памяти для различных процессоров без учета ввода-вывода приведена в табл. 4.5. Приблизительно 71% всех обращений к памяти составляют выборки команд, 22% — чтение данных, и только 7% — запись данных. Отсюда неумолимо следует, что проектировщики системы памяти должны обеспечивать малое время запаздывания прежде всего для выборки команд, затем для чтения данных и только в третью очередь для записи данных. Вспомните закон Амдала: делайте быстрым то, что наиболее часто встречается.

Таблица 4.5. Статистика обращений к памяти

	SPARC целые	SPARC плавающ. запятая	MIPS целые	MIPS I плавающ. запятая	BM S360	VAX	Среднее
Команды	0,79	0,80	0,76	0,77	0,50	0,65	0,71
Чтение данных	0,15	0,17	0,15	0,19	0,35	0,28	0,22
Запись данных	0,06	0,03	0,09	0,04	0,15	0,07	0,07

Обратите внимание на то, что архитектуры с регистровыми файлами (SPARC и MIPS) демонстрируют меньшее число обращений к памяти, чем архитектуры, ориентированные на работу с памятью (IBM S360 и VAX). Причина этого заключается в том, что в компьютерах SPARC и MIPS регистровые файлы имеют больший размер, и для большинства команд операнды находятся в регистровом файле; к памяти фактически обращаются лишь команды загрузки и запоминания. В архитектуре с регистровым файлом на команду приходится приблизительно 0,25 обращений к памяти, в то же время для компьютера S360 эта величина доходит до 1,0, а для VAX — до 0,41. Отсюда можно сделать вывод, что система памяти, разработанная для VAX, окажется не идеальной для MIPS. Проектировщик, разрабатывая систему памяти, должен ознакомиться с динамической статистикой конкретной ISA.

✎ Примеры

1. Объяснить, почему процессор S360 выполняет в среднем одно обращение к памяти на команду. Используйте статистику из табл. 4.5.

⁴²⁾ Если процессор поддерживает самомодифицирующиеся программы, в память также осуществляется запись модифицируемых команд.

Решение

Доля операций чтения и записи данных составляет $0,35 + 0,15 = 0,5$. Доля операций выборки команд тоже равна $0,5$. Таким образом, число обращений к памяти на одну команду составляет $1,0$.

2. Время запаздывания при выборке команды составляет 100 нс; время запаздывания при чтении данных составляет 150 нс.

А. Каково эффективное время запаздывания для памяти t_{el} , если задержка при записи данных составляет 200 нс и 250 нс? Используйте усредненные данные из табл. 4.5.

Б. Каково в процентах изменение производительности при переходе от 200 нс к 250 нс?

Решение

Здесь следует использовать простое взвешенное среднее.

$$A. t_{el} (200 \text{ нс}) = 0,71 \times 100 + 0,22 \times 150 + 0,07 \times 200 = 118 \text{ нс},$$

$$t_{el} (250 \text{ нс}) = 0,71 \times 100 + 0,22 \times 150 + 0,07 \times 250 = 121,5 \text{ нс}.$$

$$B. \% \text{ изменения производительности} = 100 \times \frac{S-1}{S} = 100 \times \frac{118/121,5-1}{118/121,5} = \\ = 100(1-1,029) = -2,9\%$$

Топология кэш-памяти

Топология кэш-памяти описывает число и взаимосвязи кэш-буферов, расположенных между процессором и реальной памятью. На рисунке 4.10 показаны возможные топологии.

Из-за того что большая доля обращений к памяти падает на выборку команд, ранние компьютеры содержали одноуровневый кэш команд; данные извлекались непосредственно из памяти. Более поздние топологии использовали одноуровневый общий кэш, содержа-

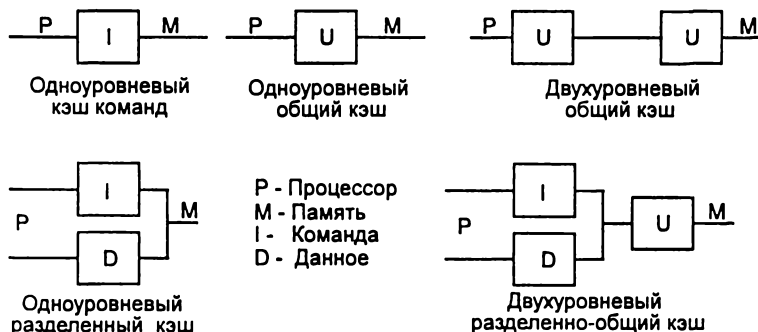


Рис. 4.10. Топологии кэш-памяти.

щий и команды, и данные. Двухуровневый общий кэш вполне возможен, однако сведений об использовании такой топологии на каком-либо компьютере нет. Наиболее обычной топологией для ранних микропроцессоров был одноуровневый разделенный кэш: один кэш для команд, другой для данных. В этих микропроцессорах оба кэша реализовывались на кристалле процессора. Системный разработчик мог, используя такой микропроцессор, установить общий кэш второго уровня между встроенными в процессор разделенными кэшами и реальной памятью. Примером двухуровневого разделенного кэша является процессор Pentium Pro, в котором общий кэш второго уровня находится в том же корпусе, что и процессор с встроенным в него разделенным кэшем.

Структура кэш-памяти и ее организация

Рассмотрим теперь общую организацию кэш-памяти, после чего можно будет перейти к различиям между кэшем команд и кэшем данных. На рисунке 4.11 показаны взаимосвязи системы *процессор — кэш-память — реальная память*. Блоки реальной памяти назначаются кэшу при обнаружении кэш-промаха, так же как в системе виртуальной памяти страницы дисковой памяти назначались страницам реальной памяти. Поскольку размер кэша меньше размера реальной памяти, между адресами реальной памяти и адресами кэш-памяти нет прямого соответствия. Кэш-память доступна как со стороны нетранслированных виртуальных адресов, так и со стороны транслированных реальных адресов. Следствия, вытекающие из такой схемы кэш-памяти, обсуждаются в разделе 4.5. Пока просто учтем, что обращение к кэш-памяти осуществляется по адресам.

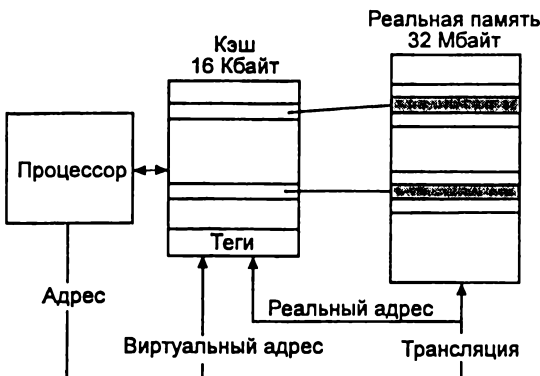


Рис. 4.11. Структура и адресация кэш-памяти.

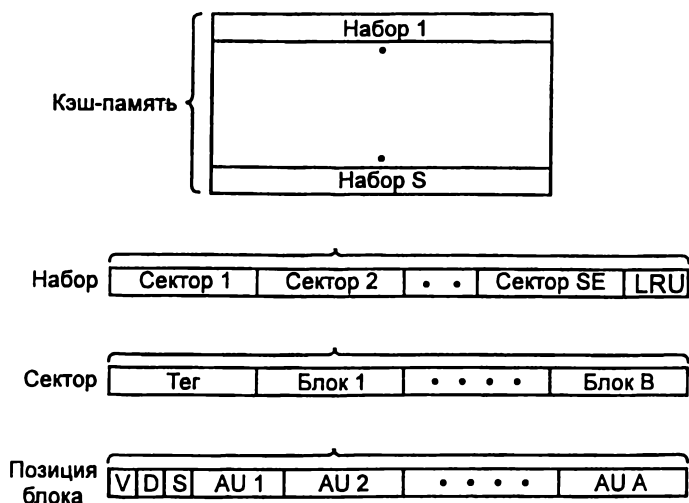


Рис. 4.12. Секционирование кэш-памяти (с любезного согласия Jones and Bartlett Publishers).

При проектировании кэш-памяти следует учитывать больше факторов, чем при разработке системы виртуальной памяти. На рисунке 4.12 показана структура кэш-памяти.

- Кэш-память разделена на наборы с номерами 1 – S. Управляющие биты каждого набора, обозначенные на рисунке как LRU⁴³⁾, помогают выбрать замещаемый блок, когда возникает необходимость переместить новый блок в реальную память.
- Набор разделен на секторы с номерами 1 – SE. В большинстве современных кэшей SE=1. Каждый сектор содержит поле тега, содержащее имя блока, а также блоки данных.
- Сектор разделен на позиции блоков с номерами 1 – B⁴⁴⁾. В большинстве современных кэшей B=1.
- Блок и позиция блока разделены на адресуемые элементы AU с номерами 1 – A. В каждой позиции блока имеются биты достоверности V (valid – достоверный), модификации D (dirty – «грязный») и совместного использования S (shared – разделяемый). Бит V означает, что информация в блоке достоверна, бит D в некоторых системах указывает, что блок был модифицирован командой записи, а бит S используется только в мультипроцессорных системах.

⁴³⁾ Least recently used, наиболее давно использовавшийся.

⁴⁴⁾ Блок представляет собой данные, а ячейки в кэш-памяти, содержащие блок, называются позицией блока (block slot). Это аналогично странице и страничному кадру в системе виртуальной памяти со страничной трансляцией.



Рис. 4.13. Каноническая кэш-память (с любезного согласия Jones and Bartlett Publishers).

Каноническая структура кэш-памяти изображена на рис. 4.13. Этот кэш имеет 8 наборов, по два сектора в наборе, по два блока в секторе и по два AU в блоке. Биты совместного использования не показаны в блоке.

Адресация кэш-памяти, как и адресация страницы виртуальной памяти, использует конгруэнтное отображение. Однако адреса, передаваемые кэш-памяти, должны содержать адресные поля для набора, сектора, блока и AU внутри блока. Ниже приведен канонический кэш-адрес.

Имя блока	Индекс набора	Индекс блока	Смещение
-----------	---------------	--------------	----------

Индекс набора адресует набор, индекс блока адресует блок, а смещение адресует AU внутри блока.

Большинство современных кэшей содержат в секторе один блок, и поле индекса блока имеет нулевую длину; полностью ассоциативная кэш-память содержит один набор, поэтому поле индекса набора тоже имеет нулевую длину. Если в секторе содержится только одна позиция блока, термин «сектор» обычно не используется. В дальнейшем обсуждении организации кэш-памяти мы будем рассматривать случай одного сектора с одним блоком и, соответственно, говорить о позиции блока, а не о секторе.

Проверка блока на его наличие в кэш-памяти выполняется не так, как в виртуальной. Вспомним, что системы виртуальной памяти перед обращением к реальной памяти сначала исследуют таблицу стра-

ниц (прямую или инверсную) и выясняют, присутствует ли страница с запрошенным АУ. Такая процедура носит название раннего выбора. В противоположность этому в кэш-памяти имя блока хранится в поле тега вместе с адресуемыми элементами АУ, как это показано на рис. 4.13. Обращение к блокам (секторам) и тегам выполняется в одном цикле. Затем теги проверяются, чтобы убедиться, что обращение выполнено к достоверному блоку. Такая процедура носит название позднего выбора.

Ниже рассматриваются операции чтения; операции записи будут описаны позже в этом же разделе. Элементы структуры кэш-памяти, изображенные на рис. 4.12 и 4.13, могут быть реализованы одним из трех способов: ассоциативным, частично ассоциативным и с прямым отображением.

Ассоциативная кэш-память

Ассоциативная кэш-память содержит один набор; ее организация изображена на рис. 4.14. В этом примере кэш содержит восемь позиций блоков, каждый из которых включает два АУ. Адрес, поступающий в кэш-память, разделен на имя блока и смещение. Поля индексов набора и блока имеют нулевую длину.

Каждый из восьми блоков содержит тег, в котором хранится имя резидентного блока, два АУ, бит достоверности и бит модификации. Бит достоверности необходим потому, что совпадение имени блока с

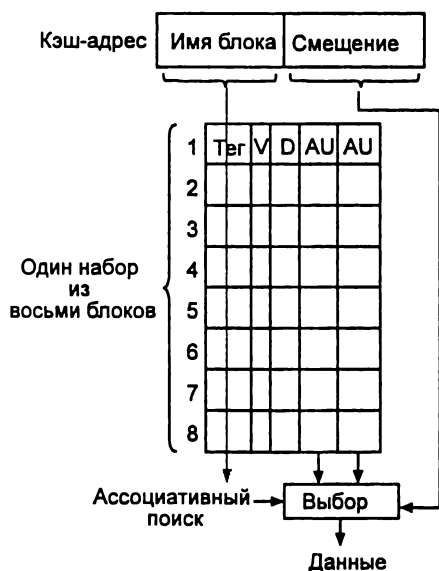


Рис. 4.14. Ассоциативная кэш-память (с любезного согласия Jones and Bartlett Publishers).

тегом может произойти и при достоверных данных. Этот бит устанавливается в состояние «достоверно» только после того, как после кэш-промаха в позицию блока загружаются достоверные данные. Бит модификации указывает на то, что некоторая часть блока была модифицирована.

Имя блока последовательно сравнивается с тегами, и если найдено совпадение и бит достоверности установлен, читаются адресуемые элементы блока. Совпадение тега с запрашиваемым именем блока при установленном бите достоверности называется кэш-попаданием (cache hit); это означает, что запрошенный блок находится в кэше. Требуемый AU выбирается с помощью поля смещения и передается в процессор. Кэш-промах (cache miss) возникает, если либо ни один из тегов не соответствует имени блока в адресе, либо бит достоверности сброшен; это означает, что запрошенный блок не находится в кэше и должен быть выбран из реальной памяти. Ассоциативная организация обладает тем привлекательным свойством, что блок может быть помещен в любую позицию блока в кэше; в этом отношении какие-либо ограничения отсутствуют.

Частично-ассоциативная кэш-память

Второй способ организации кэш-памяти, частично-ассоциативный, требует наличия двух или более наборов. На рисунке 4.15 показана организация 2-входовой частично-ассоциативной кэш-памяти того же объема, что и кэш на рис. 4.14. Два младших бита имени блока служат индексом считываемой строки кэша. Теги, содержащие имена блоков, сравниваются с именем блока в адресе. Если тег соответствует

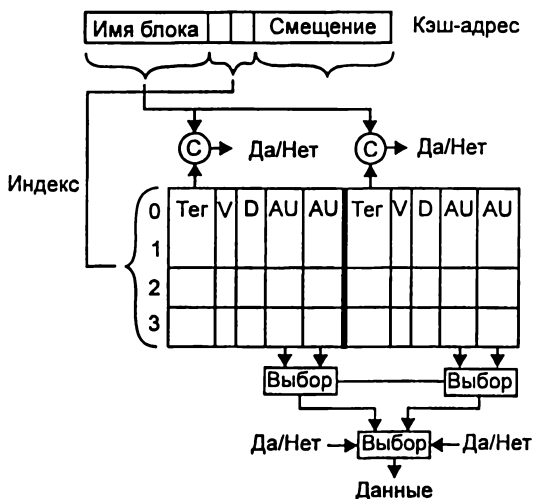


Рис. 4.15. Двухвходовая частично-ассоциативная кэш-память (с любезного согласия Jones and Bartlett Publishers).

имени блока и бит достоверности установлен, фиксируется кэш-падение, и по смещению выбирается и направляется в процессор требуемый AU. Кэш-промах требует выборки из реальной памяти.

Поскольку рассматриваемая память является 2-входовой частично-ассоциативной, запрашиваемый блок может размещаться только в одной из двух позиций блоков в наборе. Например, блоки с именами, два младших бита которых образуют сочетание 01, могут размещаться только в позициях блоков с индексом 1. Даже если в кэше есть свободные позиции блоков, размещение указанных блоков ограничено этим набором. Если позиции блоков заполнены и поступает запрос с тем же значением индекса набора, фиксируется кэш-промах по конфликту наборов.

Описанная организация содержит только два компаратора, чем выгодно отличается от полностью ассоциативного кэша (рис. 4.14), требующего восемь компараторов. Можно также спроектировать 4-входовую частично-ассоциативную кэш-память с четырьмя компараторами или даже 5-входовую с пятью компараторами. Для кэшей с одним сектором в наборе число компараторов и степень ассоциативности равны числу блоков в наборе.

Кэш-память с прямым отображением

Третья организация, называемая кэш-памятью с прямым отображением, уменьшает число компараторов до одного; на рис. 4.16 изобра-

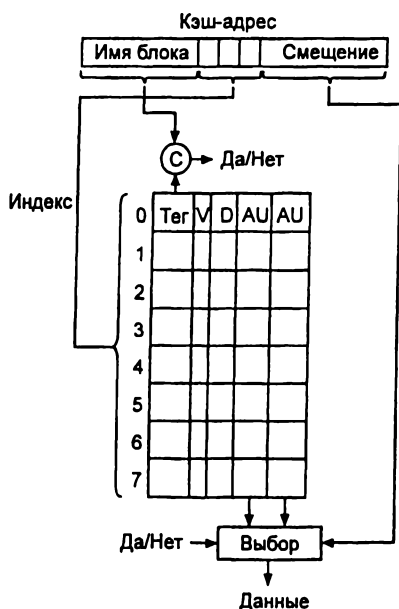


Рис. 4.16. Кэш-память с прямым отображением (с любезного согласия Jones and Bartlett Publishers).

ражена такая кэш-память с восемью наборами. В качестве индекса набора используются три младших бита имени блока.

Прямая кэш-память является простейшим способом организации, требующим всего один компаратор тегов. Однако в каждом наборе имеется только одна позиция блока, таким образом, размещение блоков однозначно определяется. Блоки с одинаковыми значениями индекса набора (независимо от имени блока) могут быть помещены в одну единственную позицию блока. Например, блоки с именами, три младших бита которых образуют сочетание 010, могут быть размещены только в позиции блока набора 2, даже если все до единого позиции блоков в кэше свободны. В результате увеличивается вероятность вынужденной выгрузки блока из позиции блока для того, чтобы освободить место для нового блока. Это увеличивает интенсивность кэш-промахов (эквивалентную обсуждавшейся ранее интенсивности отказов страниц).

✎ Примеры

1. Рассмотреть 2-входовую частично-ассоциативную кэш-память. Какие наборы могут содержать блоки, имена которых оканчиваются комбинацией битов 011?

✎ Решение

Эти блоки могут храниться только в левых или правых позициях блоков набора 3.

2. Для трех способов организации кэш-памяти, показанных на рис. 4.14 – 4.16, создать таблицу, заполнив ее числами наборов, секторов и блоков, а также числом адресуемых элементов AU в блоке. Вычислить также емкость кэшей в числе A и привести формулу для вычисления этой величины.

✎ Решение

Ответ на этот вопрос содержится в табл. 4.6.

число AU в кэше = число наборов $\times$ (секторы/набор) $\times$ (блоки/сектор) $\times$ (число AU/блок)

✓ Замечание

Все рассмотренные варианты организации имели одинаковый размер. Таблицы 4.8, 4.9 и 4.10 показывают, что выбор организационных параметров может иметь заметное влияние на производительность кэш-памяти.

Таблица 4.6. Организация и емкость кэш-памяти

Организация	Наборы	Секторы/ набор	Блоки/ сектор	Число AU/ блок	Емкость
Ассоциативная	1	1	8	2	16 AU
2-входовая	4	1	2	2	16 AU
частично-ассоциативная					
Прямая	8	1	1	2	16 AU

Стратегии чтения и записи

Когда процессор обращается к кэш-памяти с целью чтения или записи, аппаратура должна обеспечивать определенную стратегию выполнения этого действия. Рассмотрим сначала стратегию чтения. Вспомним, что кэш-попадание предполагает соответствие тега имени блока при установленном бите достоверности.

Кэш-попадание при чтении:

- Адресуемый AU находится в кэше; AU переносится в процессор.

Кэш-промах при чтении:

- Если в кэш-памяти имеется свободная позиция блока, в которую можно поместить новый блок, блок выбирается из реальной памяти и помещается в нее; адресуемый AU переносится в процессор. Тег и бит достоверности обновляются.

- Если в кэш-памяти нет свободной позиции блока, которая могла бы принять требуемый блок (кэш полон или произошел конфликт наборов), то некоторый блок должен быть возвращен в реальную память с освобождением занимаемой им позиции. Схемы управления кэшем повторяют операцию чтения, которая приводит к кэш-промаху чтения при наличии в кэше свободной позиции блока, что соответствует шагу 1.

- Если кэш-память использует бит модификации и выбранный для замещения блок не был модифицирован, требуемый блок может быть записан в кэш без возвращения в реальную память замещаемого блока. Для выбора блока, подлежащего замещению, имеется ряд алгоритмов, в частности случайный поиск или выбор наиболее давно использовавшегося блока.

Стратегии записи, важные для кэш-данных, но не используемые с кэшами команд, оказываются более сложными. Как и в случае чтения, команда, выполняющая запись в кэш, может привести к кэш-попаданию или к кэш-промаху. Ниже обсуждаются альтернативы обработки этих событий.

Кэш-попадание при записи:

- Выполняется запись в кэш и в реальную память. Эта стратегия называется сквозной записью (write through). Поскольку блок в кэше и его копия в реальной памяти в этом случае согласованы⁴⁵⁾, при последующем промахе чтения этого блока не будет необходимости возвращать его в реальную память, поскольку допустимо его затирание выбираемым блоком. При такой стратегии отпадает необходимость использования бита модификации, так как кэш-память и реальная память всегда согласованы.

- Запись выполняется только в кэш-память. После завершения операции записи устанавливается бит модификации, что говорит о выполненной модификации блока или его части. Поскольку блок в кэше и в реальной памяти теперь не согласован, при возникновении кэш-промаха чтения этот блок должен быть возвращен в реальную память, и лишь после этого можно выбирать из памяти новый блок. Эта стратегия носит название отложенной записи (write back).

Кэш-промах при записи:

- Отсутствующий блок выбирается в кэш-память, после чего выполняется запись (либо сквозная, либо обратная, в зависимости от принятой стратегии кэш-попадания при записи). Эта стратегия носит название записи с размещением (write allocate), так как требуемый блок к моменту записи уже размещен в кэш-памяти.

- Запись выполняется непосредственно в реальную память в обход кэш-памяти; такая стратегия называется записью без размещения (no-write allocate) или прямой записью.

В большинстве процессоров стратегия выбирается на этапе проектирования и аппаратно реализуется в процессоре, как стратегия умолчания. Некоторые процессоры предоставляют программисту возможность выбирать стратегию записи. Эти возможности будут рассмотрены ниже, в разделе 8.5.

Модель времени запаздывания кэш-памяти

Модель взвешенного эффективного времени запаздывания $t_{\text{еф}}$ для кэш-памяти схожа с моделью интерфейса реальной памяти с диском. Мы рассмотрим здесь стационарную модель; переходные процессы, например контекстное переключение, изменят производительность по сравнению с вычисленной для стационарного случая. Модель имеет следующие параметры: время запаздывания кэш-памяти t_c , транспортное время перемещения из реальной памяти в кэш-память $T_{\text{тмс}}$, а так-

⁴⁵⁾ Про память говорят, что она согласована (coherent), если адреса указывают на одно и то же значение, даже если в различных ячейках памяти имеются две или более копий этого значения.

же интенсивность кэш-промахов P_{miss} , которая определяется следующим образом:

$$P_{miss} = 1 - \frac{\text{число обращений, обслуженных кэш-памятью}}{\text{полное число обращений к кэш-памяти}}$$

Взвешенное эффективное время запаздывания вычисляется с помощью табулирования параметров, как это было описано в гл. 2 и показано в табл. 4.7. Возможны два события: кэш-промах и кэш-попадание, причем каждому событию соответствует определенное время:

$$\begin{aligned} t_{elc} &= (1 - P_{miss})t_c + P_{miss}(t_c + T_{rmc}) = \\ &= t_c - P_{miss}t_c + P_{miss}t_c + P_{miss}T_{rmc} = \\ &= t_c + P_{miss}T_{rmc} \end{aligned}$$

Таблица 4.7. Эффективное время запаздывания кэш-памяти

Событие	Вероятность	Время	Произведение
Попадание	$1 - P_{miss}$	t_c	$(1 - P_{miss})t_c$
Промах	P_{miss}	$(t_c + T_{rmc})$	$P_{miss}(t_c + T_{rmc})$
	Сумма = 1		Сумма = $(1 - P_{miss})t_c + P_{miss}(t_c + T_{rmc})$

Причина, по которой время промаха является суммой времени запаздывания кэш-памяти и транспортного времени, заключается в том, что перед обращением к реальной памяти осуществляется обращение к кэш-памяти и проверка тега. Такая последовательность носит название позднего выбора (late select). Полученный результат может быть нормализован путем приведения к базовому времени запаздывания кэш-памяти, поскольку это время для синхронных систем обычно составляет один процессорный такт. В синхронных системах T_{rmc} всегда является целым числом.

$$t_{elc} \text{ (в тактах)} = 1 + P_{miss}T_{rmc}$$

🔗 **Пример**

При проектировании кэш-памяти нужно оценить взаимосвязь P_{miss} и T_{rmc} . Начертим график зависимости эффективного времени запаздывания от T_{rmc} при изменении T_{rmc} от 7 до 14 для значений $P_{miss} = 0,1$ и 0,08.

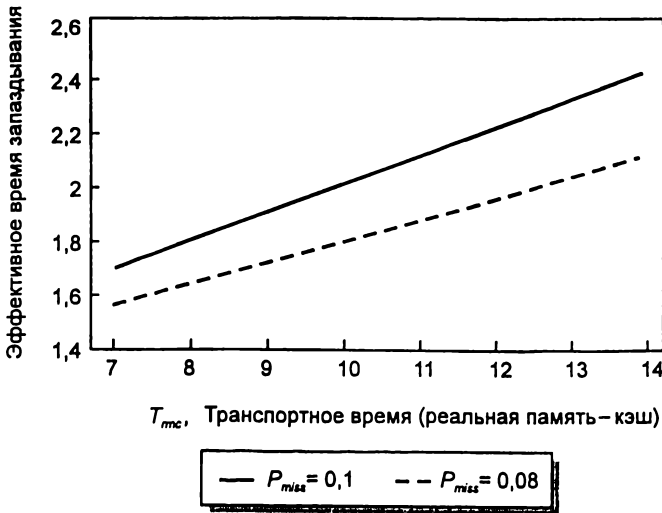


Рис. 4.17. Эффективное время запаздывания для различных значений T_{mc} и P_{miss} .

Решение

На рисунке 4.17 приведен график для t_{elc} (в тактах) для указанных значений параметров.

✓ Замечание

Обратите внимание на возможность выбора между T_{mc} и P_{miss} . Эффективная задержка 1,7 такта может быть получена как при сочетании $T_{mc} = 7$ и $P_{miss} = 0,1$, так и при сочетании $T_{mc} = 13$ и $P_{miss} = 0,08$.

Кэш-память данных

Рассматривая выборку команд, мы можем учитывать пространственную локализацию, так как программный счетчик монотонно инкрементируется от команды к команде. Пространственная локализация нарушается только при выполнении команд переходов. Пространственная локализация используется в стратегиях кэш-памяти следующим образом. Если обращение к команде вызывает кэш-промах, блок, содержащий требуемую команду, выбирается в кэш-память. В результате и адресуемая команда, и некоторое количество следующих за ней команд помещаются в кэш. Теперь при инкременте программного счетчика и переходе к следующей команде требуемая команда, скорее всего, уже будет в кэше.

Данные по кэш-попаданиям и кэш-промахам можно найти в различных источниках [3]. Абстрактный набор данных по промахам для контрольных задач SPEC и 16-байтовых блоков приведен в табл. 4.8.

Таблица 4.8. Значения P_{miss} для кэша команд при выполнении контрольных задач SPEC, суммарно

n-входов	Число наборов						
	64	128	256	512	1К	2К	4К
1, прямой кэш	0,051	0,039	0,031	0,020	0,012	0,007	0,003
2	0,034	0,024	0,017	0,010	0,008	0,003	0,0002
4	0,023	0,016	0,009	0,008	0,001	0,0002	0,0001
8	0,016	0,009	0,008	0,0007	0,0002	0,000	0,000
16-байтовые блоки, 1 позиция блока в наборе							

Эта таблица организована таким образом, что кэши равного размера располагаются по диагонали (от левого нижнего угла к правому верхнему). Например, 8-Кбайтовый кэш, выделенный курсивом, может быть организован как 8-входовый частично-ассоциативный с 64 наборами. Однако этот кэш может быть организован и как прямой кэш с 512 наборами. Заметьте, что в целом организация кэш-памяти слабо влияет на P_{miss} ; главным фактором является сам размер. Для небольших кэшей команд можно обнаружить незначительное уменьшение значения P_{miss} по мере роста степени ассоциативности и снижения числа наборов.

Кэши команд также поддерживают временную локализацию и просмотр назад. Кэш должен быть достаточно велик, чтобы в него поместился цикл, что особенно важно для научных программ. После первой выборки цикла он остается в кэш-памяти и не будет требовать повторных выборов.

По мере развития аппаратных средств реализации компьютеров было введено использование очередей для выполнения функции предварительного просмотра и буферизация команд для просмотра назад [8]. В этой работе предлагается, что кэш выполняет обе функции. Кэши с очередями используются в ряде современных процессоров, в частности, в Pentium Pro и PowerPC.

Кэши данных

Обращения к данным обычно не обнаруживают заметной пространственной локализации⁴⁶⁾, однако после первого использования конкретное данное часто используется снова; т. е. имеет место времен-

⁴⁶⁾ За исключением научных программ с массивами данных. Для них пространственная локализация настолько сильна, что некоторые суперкомпьютеры не используют кэши данных.

ная локализация. Кэш-память данных сохраняет данные, к которым было обращение, в ожидании их использования в будущем. В таблице 4.9 приведены результаты выполнения контрольных задач SPEC по кэш-промахам для 16-байтовых блоков. Кэши равного размера располагаются по диагонали.

Таблица 4.9. Значения P_{miss} для кэша данных при выполнении контрольных задач SPEC, суммарно

п-входов	Число наборов						
	64	128	256	512	1К	2К	4К
1, прямой кэш	0,255	0,216	0,170	0,114	0,087	0,069	0,057
2	0,188	0,155	0,104	0,074	0,059	0,048	0,039
4	0,134	0,098	0,071	0,058	0,047	0,038	0,028
8	0,091	0,069	0,057	0,047	0,038	0,028	0,017
16-байтовые блоки, 1 позиция блока в наборе							

Как и в случае кэш-команд, главным фактором, влияющим на P_{miss} , является размер кэша. Для кэш-команд небольшого размера интенсивность промахов уменьшается с ростом частичной ассоциативности. Однако для прямого кэша характерна более высокая интенсивность промахов, чем для 2-, 4- или 8-входовых частично-ассоциативных кэш-команд того же размера. В прямом кэше реализуется более короткий (на несколько логических элементов) логический тракт, чем в других видах кэш-команд. Поэтому возможно, хотя это и не было доказано, что при уменьшении длительности такта благодаря более короткому логическому тракту кэш с меньшим временем запаздывания может оказаться эффективнее, несмотря на увеличение вероятности промаха P_{miss} . При наличии более быстрого кэша процессор может иметь более высокую тактовую частоту, что увеличит общую производительность компьютера.

Общая кэш-память

Общий (unified) кэш, сохраняющий как команды, так и данные, используется в настоящее время в качестве кэша второго уровня в дополнение к кэшам команд и данных, выполняемым на кристалле процессора. Как и следует ожидать, параметр P_{miss} для общего кэша лежит между значениями для кэша команд и кэша данных. В таблице 4.10 приведены значения P_{miss} для тех же контрольных задач SPEC и 16-байтовых блоков.

Как и в случае кэш-команд и данных, важнейшим параметром, позволяющим снизить значение P_{miss} , является размер общего кэша. Однако для общего кэша данного размера значение P_{miss} прямого кэша приблизительно на 20-25% больше, чем для двухвходового частично-

ассоциативного кэша. Поэтому общие кэши второго уровня обычно реализуются как частично-ассоциативные с двумя или восемью входами.

Таблица 4.10. Значения P_{miss} для общего кэша при выполнении контрольных задач SPEC, суммарно

п-входов	Число наборов						
	64	128	256	512	1К	2К	4К
1, прямой кэш	0,151	0,114	0,086	0,056	0,039	0,029	0,020
2	0,092	0,069	0,047	0,031	0,023	0,016	0,011
4	0,062	0,044	0,029	0,022	0,015	0,010	0,007
8	0,041	0,028	0,021	0,015	0,010	0,008	0,005
16-байтовые блоки, 1 позиция блока в наборе							

Размер блока

Данные по значениям P_{miss} , представленные в табл. 4.8 — 4.10, получены для 16-байтовых блоков. Однако чем определяется размер блока на этапе проектирования? Здесь следует принимать в расчет два соображения: 1) как размер блока влияет на значение P_{miss} и 2) как размер блока влияет на транспортное время перемещения из реальной памяти в кэш.

Исследуя связь P_{miss} и размера блока, можно рассмотреть две крайности: блок в 1 АУ, с одной стороны, и блок размером в весь кэш, с другой. Для блоков небольшого размера не будет предварительного просмотра, а для очень больших блоков будет мало просмотров назад. В обоих случаях P_{miss} будет велико. На рисунке 4.18 приведены значения P_{miss} для общих кэшей с различными размерами всей кэш-памяти и отдельных блоков [3]. Эти данные получены на общем кэше с 4-входной ассоциативной организацией при использовании комбинированных контрольных задач SPEC для целых чисел и плавающей запятой.

Обратите внимание на то, что для кэшей небольшого размера имеется отчетливый минимум значений P_{miss} при некотором значении размера блока. По мере увеличения размера кэша минимум перемещается в сторону больших размеров блоков. Для кэшей большого размера, 64 Кбайт и 128 Кбайт, наблюдается относительно постоянное уменьшение P_{miss} при каждом удвоении размера блока, что напоминает кривую обучаемости, описанную в гл. 2:

$$P_{miss}(\text{конечное}) = P_{miss}(\text{начальное}) L^{\log_2 \frac{\text{конечный размер блока}}{\text{начальный размер блока}}}$$

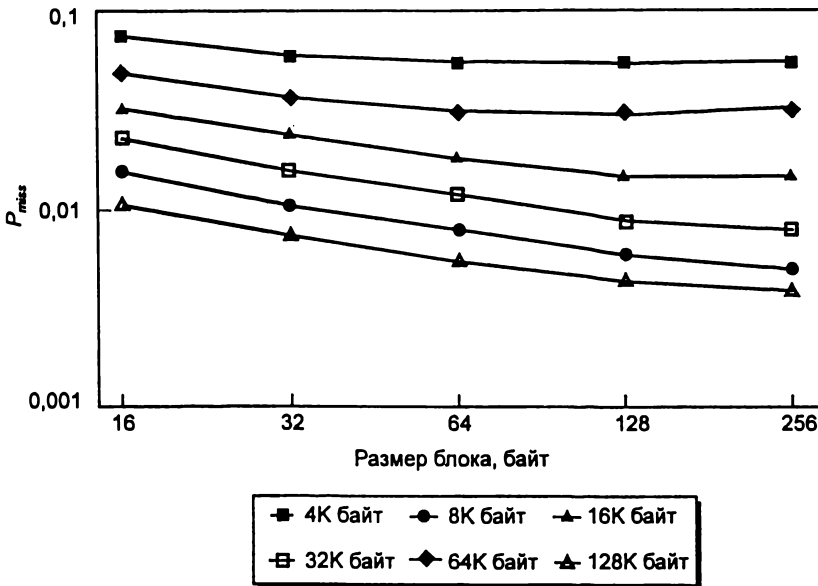


Рис. 4.18. Значения P_{miss} для различных значений размеров блока и кэша.

64-Кбайтовый кэш (см. рис. 4.18) имеет значение $L \approx 0,7$. Другими словами, при удвоении размеров блока P_{miss} уменьшается приблизительно до 0,7 от его первоначального значения. Влияние размеров блока на транспортное время обсуждается в разделе 4.4, где рассматривается модель интерфейсной шины между кэшем-памятью и реальной памятью.

Многоуровневая кэш-память

Многие современные процессоры имеют кэш-память команд и кэш-память данных непосредственно на кристалле процессора вместе с кэш-памятью второго уровня на отдельном кристалле. Блок-схема такой двухуровневой кэш-памяти приведена на рис. 4.19. Два показанных кэша обычно называют кэш L1 и кэш L2^{*)}.

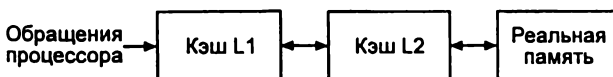


Рис. 4.19. Двухуровневая кэш-память.

^{*)} От Level – уровень. – Прим. перев.

Для одноуровневого кэша отношение транспортного времени T_{rc} к времени запаздывания кэша L1 может находиться в диапазоне 10–50. Такое большое значение отношения объясняется рядом причин. Реальная память адресуется посредством шины, а для шин необходимо время синхронизации; передача блока по шине требует нескольких тактов, так как шина может иметь ширину всего несколько байтов. Для уменьшения влияния больших значений T_{rc} и используется кэш второго уровня L2.

Модель для вычисления эффективного времени запаздывания двухуровневого кэша t_{elc2} строится следующим образом. Кэш L1 имеет время запаздывания 1 такт. Обозначим транспортное время перемещения из L2 в L1 через T_{c2} , а транспортное время перемещения из реальной памяти в кэш L2 через T_{rc3} . Транспортные времена T_{c2} и T_{rc3} выражаются в числе кэш-циклов. Пусть P_{miss1} и P_{miss2} представляют интенсивность кэш-промахов в кэшах L1 и L2 соответственно. Предположим, что все обращения удовлетворяются на одном из трех уровней памяти.

Если в кэше L1 происходит попадание, кэш L2 и реальная память (RM) находятся в безразличном состоянии. Аналогично, при промахе в кэше L1 и попадании в кэше L2 в безразличном состоянии находится реальная память. В таблице 4.11 показаны составляющие модели эффективного времени запаздывания.

$$\begin{aligned} t_{elc2} &= 1 - P_{miss1} + P_{miss1}(1 - P_{miss2})(1 + T_{c2}) + P_{miss1}P_{miss2}(1 + T_{c2} + T_{rc3}) = \\ &= 1 + P_{miss1}T_{c2} + P_{miss1}P_{miss2}T_{rc3} \end{aligned}$$

Таблица 4.11. Эффективное время запаздывания многоуровневой кэш-памяти

L1	L2	RM	Вероятность	Время	Произведение
Попад.	х	х	$1 - P_{miss1}$	1	$1 - P_{miss1}$
Прوماх	Попад.	х	$P_{miss1}(1 - P_{miss2})$	$1 + T_{c2}$	$P_{miss1}(1 - P_{miss2})(1 + T_{c2})$
Прوماх	Прوماх	Попад.	$P_{miss1}P_{miss2}$	$1 + T_{c2} + T_{rc3}$	$P_{miss1}P_{miss2}(1 + T_{c2} + T_{rc3})$
			Сумма = 1		Сумма = t_{elc2}

Пример

Одноуровневая кэш-система характеризуется значениями $P_{miss} = 0,1$ и $T_{rc} = 10$. Рассмотрим двухуровневую кэш-память с характерными параметрами $P_{miss1} = 0,1$, $P_{miss2} = 0,1$, $T_{c2} = 3$ и $T_{rc3} = 10$. Каким будет снижение эффективного времени запаздывания при установке кэша L2?

 Решение

Эффективное время запаздывания одноуровневой кэш-системы составит

$$t_{\text{elc}} = 1 + 0,1 \times 10 = 2 \text{ такта}$$

Для двухуровневой кэш-памяти

$$t_{\text{elc}} = 1 + 0,1 \times 3 + 0,1 \times 0,1 \times 10 = 1,4 \text{ такта}$$

$$\text{повышение быстродействия} = 2/1,4 = 1,43$$

 Замечание

Установкой кэша L2 со значением T_c меньшим, чем T_{pc} одноуровневого кэша, мы существенно (на 30%) снизили запаздывание кэш-памяти. Кэш L2 будет полезен, только если t_{c2} заметно меньше транспортного времени перемещения между одноуровневым кэшем и реальной памятью.

Модель двухуровневой кэш-памяти, описанная выше, может быть расширена на любое число уровней кэш-памяти или усовершенствована включением уровня виртуальной памяти. Значения P_{miss} , используемые в этой модели, называются вероятностями локальных промахов (local miss probability); другими словами, вероятностями промаха запроса к кэшу определенного уровня. Другим подходом к моделированию многоуровневой кэш-памяти является использование глобальных вероятностей попадания. Глобальная вероятность попадания (global hit probability) представляет собой долю от всех обращений, которые обслуживаются на данном уровне; все глобальные вероятности попадания должны в сумме давать 1. Глобальные вероятности попадания для вышеприведенного примера составят

$$P_{\text{hit1g}} = 0,9$$

$$P_{\text{hit2g}} = 0,1 \times 0,9 = 0,09$$

$$P_{\text{hit3g}} = 1 - 0,9 - 0,09 = 0,01$$

 Пример

Решим заново вышеприведенный пример для двухуровневой кэш-памяти, используя глобальные интенсивности попаданий.

 Решение

$$t_{\text{elc}} = 0,9 \times 1 + 0,1 \times (1 + 10) = 2$$

$$\begin{aligned} t_{\text{elc2}} &= 0,9 \times 1 + 0,09 \times (1+3) + 0,01 \times (1 + 3 + 10) = \\ &= 0,9 + 0,36 + 0,14 = 1,4 \end{aligned}$$

$$\text{повышение быстродействия} = 2/1,4 = 1,43$$

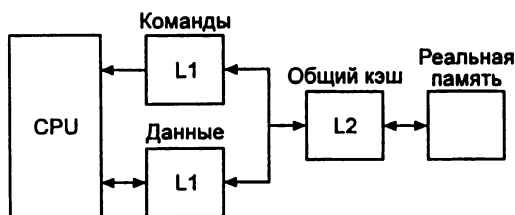


Рис. 4.20. Многоуровневая кэш-система.

На рисунке 4.20 показана блок-схема современной системы памяти. Эта память включает кэши L1 команд и данных на кристалле и отдельный общий кэш L2.

Из-за очень близкого расположения процессора и кэш-памяти второго уровня транспортное время перемещения между кэшами команд и данных и общим кэшем может быть небольшим, в диапазоне от 2 до 4 циклов. Однако транспортное время перемещения между общим кэшем и реальной памятью может оказаться в диапазоне от 10 до 50 циклов из-за требований синхронизации шины.

✎ Пример

Кэши команд и данных на рис. 4.20 имеют размер 8 Кбайт и являются двухходовыми частично-ассоциативными. Общий кэш имеет размер 32 Кбайт и также является двухходовым частично-ассоциативным. Будем считать, что кэши состоят из 16-байтовых блоков, а локальное значение P_{miss} для реальной памяти равно 0, т. е. виртуальная память не используется.

1. Какая доля обращений к командам и данным будет удовлетворяться в реальной памяти?

2. Если каждая команда выполняет 0,3 обращения к данным, каково взвешенное среднее число обращений к памяти в расчете на одну команду, которые удовлетворяются в реальной памяти?

3. Ответим на предыдущий вопрос в предположении, что кэш второго уровня отсутствует.

✎ Решение

1. Из таблиц 4.8, 4.9 и 4.10 можно найти, что для кэша команд $P_{\text{miss}} = 0,017$, для кэша данных $P_{\text{miss}} = 0,104$, и для общего кэша $P_{\text{miss}} = 0,023$. Доля обращений к командам, удовлетворенных в реальной памяти, составит

$$0,017 \times 0,023 = 0,00039$$

Доля обращений к данным, удовлетворенных в реальной памяти, составит

$$0,104 \times 0,023 = 0,00239$$

2. Взвешенное среднее число обращений к памяти на одну команду, удовлетворенных в реальной памяти, составит

$$1 \times 0,00039 + 0,3 \times 0,00239 = 0,0011$$

3. Число обращений к реальной памяти на одну команду без кэша второго уровня составит

$$1 \times 0,017 + 0,3 \times 0,104 = 0,048$$

✓ Замечание

Использование кэша второго уровня снижает долю обращений к реальной памяти приблизительно от 0,05 до 0,001 на команду. Это снижает нагрузку на шину и позволяет более эффективно выполняться параллельным операциям ввода-вывода.

Специальная кэш-память

В процессоре имеется некоторое количество буферов, которые по своей организации и использованию похожи на кэш-память. В целом эти специальные кэши так или иначе связаны с явлением временной локализации.

Одним из таких специальных кэшей является кратко упомянутый в разделе 4.2 буфер ассоциативной трансляции, или TLB. Этот буфер хранит транслированные имена страниц, которые были транслированы с помощью таблиц страниц в предыдущих обращениях. TLB процессора Intel Pentium Pro показан на рис. 4.21. Всего имеются два TLB, один для команд, другой для данных.

Данный TLB является 4-входовым частично-ассоциативным и содержит всего 8 наборов; каждый набор включает бит достоверности. В этом кэше вместо данных или команд сохраняются транслированные адреса. Опубликованных данных по интенсивности промахов для TLB мало; те же, что имеются [9], указывают, что $P_{miss} = 0,01 - 0,02$. Впрочем, такой процент промахов не так плох, как кажется.

Обращения к командам и данным адресуют кэши, а TLB обычно подключается параллельно кэш-памяти, как это показано на рис. 4.22. В этом случае, когда обращение к кэшу дает промах, транслированный адрес уже имеется в TLB, если, конечно, не было TLB-промаха. При значении P_{miss} для кэша 0,1 число обращений к памяти, требующих трансляции, составляет 10%. Для TLB $P_{miss} = 0,01$, и число обращений, которые должны транслироваться с помощью таблиц страниц, составляет $0,1 \times 0,01 = 0,001$, т. е. 1 на 1000. Это отношение становится еще меньше, если используется кэш второго уровня.

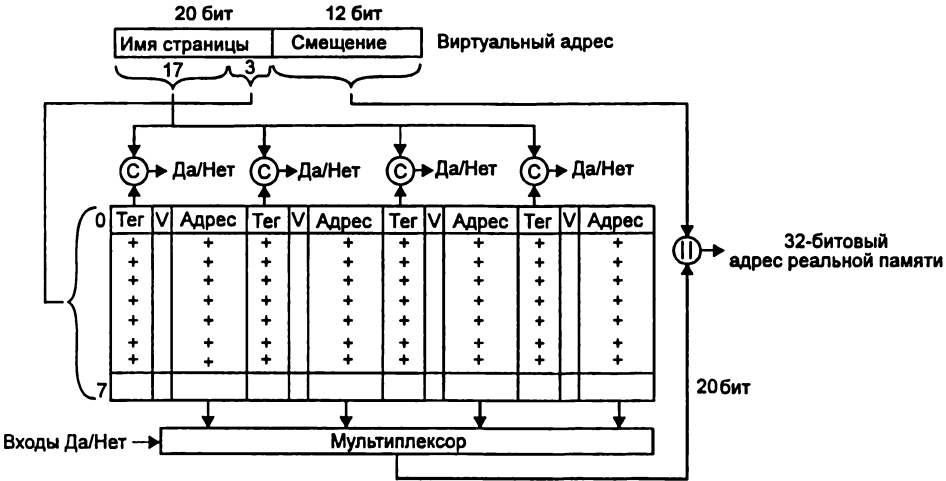


Рис. 4.21. TLB процессора Intel Pentium Pro (с любезного согласия Jones and Bartlett Publishers).

Пример

Для кэша $P_{missC} = 0,1$, а для TLB $P_{missTLB} = 0,01$. Время запаздывания кэша составляет 1, время запаздывания TLB равно 5% от времени цикла реальной памяти. Таблица трансляции требует двух циклов памяти, а цикл памяти эквивалентен четырем циклам кэша. Каково эффективное время запаздывания реальной памяти?

Решение

Поиск в TLB осуществляется параллельно с поиском в кэше; его содержимое используется, только если зафиксирован кэш-промах, причем TLB быстрее кэша. Таким образом, в случае TLB-попадания запаздывание TLB не дает вклада в время запаздывания. В таблице 4.12 приведены события, вероятности и времена, требуемые для решения этой задачи.

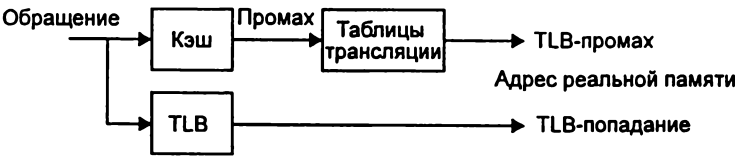


Рис. 4.22. Обращение к кэшу и TLB.

Таблица 4.12. Эффективное запаздывание реальной памяти

Событие	Вероятность	Время	Произведение
Кэш-попадание	$1 - P_{missC}$	1	$1 - P_{missC}$
Кэш-промах, TLB-попадание	$P_{missC} \times (1 - P_{missTLB})$	$0,05 \times 4 + 4$	$4,2 P_{missC} \times (1 - P_{missTLB})$
Кэш-промах, TLB-промах	$P_{missC} \times P_{missTLB}$	$1 + (2+1)4$	$13 P_{missC} \times P_{missTLB}$
			Сумма = эффективное запаздывание реальной памяти

Эффективное запаздывание реальной памяти равно

$$1 - P_{missC} + 4,2 \times P_{missC} \times (1 - P_{missTLB}) + 13 \times P_{missC} \times P_{missTLB} =$$

$$= 1 - 0,1 + 4,2 \times 0,1 \times 0,99 + 13 \times 0,1 \times 0,01 = 1,33 \text{ цикла}$$

✓ Замечание

Этот пример показывает, что, если используется TLB, многоуровневая трансляция с помощью таблицы страниц лишь незначительно увеличивает запаздывание реальной памяти.

В процессоре Intel Pentium Pro TLB организован как 8-входовый частично-ассоциативный; другие процессоры используют другую организацию. Например, в процессоре IBM S370 используется инверсный TLB, обращение к которому осуществляется путем хеширования имени страницы (аналогично инверсным таблицам страниц, обсуждавшимся в разделе 4.2).

Стратегия чтения TLB аналогична стратегии кэша команд. Если возникает TLB-промах, то после выполнения трансляции в таблицах страниц TLB обновляется. Поскольку в TLB, в отличие от кэша данных, процессор ничего не записывает, новый результат трансляции может быть немедленно помещен в TLB⁴⁷⁾. Нет необходимости доказывать, что для TLB не может быть стратегии записи. В некоторых системах TLB переключается вместе с переключением контекста. Переключение экономит время, предотвращая выполнение заново трансляции имени страницы при активизации старой программы.

Некоторые TLB имеют несколько ячеек с аппаратно установленными адресами страничных кадров, обычно для адресов страниц начала адресного пространства. Это делается для того, чтобы страницы

⁴⁷⁾ TLB обновляется в качестве побочного результата трансляции имени страницы с помощью таблиц страниц.

операционной системы всегда имели одинаковое отображение. Было бы очень неприятно, если бы система зафиксировала отказ страницы в тот момент, когда она пытается обслужить отказ страницы.

4.4. Реальная память с перекрытием

Реальная память, располагаемая между кэш-памятью и диском, должна иметь особые качества. Обращение к этой памяти происходит только при возникновении кэш-промаха по чтению или записи. Если происходит такое событие, необходимо переместить блок в кэш или из кэша. В настоящее время реальная память составляется из модулей DRAM с временем запаздывания 10–50 кэш-циклов. Каждый блок состоит из нескольких AU; поэтому пропускная способность реальной памяти оказывается столь же важной, как и ее запаздывание. Каким образом можно обеспечить высокую пропускную способность при таких относительно медленных компонентах памяти? Решением проблемы стало использование параллельных модулей памяти, называемое перекрытием (interleaving).

На рисунке 4.23 изображена 4-входовая память с перекрытием. Четыре модуля памяти с длиной слова W_d адресуемых элементов подключены к шине, имеющей ту же ширину; шина подключена к кэш-памяти. Тактовый период шины равен циклу кэш-памяти; он задает тактовый период системы. В нашем примере время цикла модулей памяти разделено на четыре такта. Сначала происходит обращение к первому модулю M_0 , затем, одним тактом позже, ко второму модулю M_1 . Так последовательно происходит обращение ко всем модулям, после чего процесс может быть повторен. Выход (например, результат чтения) модуля M_0 доступен в конце такта t_4 , модуля M_1 — в конце t_5 и т. д. Такая конфигурация иногда называется C-доступом, потому что доступ к модулям осуществляется одновременно (concurrently), хотя и со сдвигом во времени.

Транспортное время T_{π} — это время перемещения блока из реальной памяти в кэш или наоборот. Для памяти с перекрытием T_{π} в единицах тактов шины равно

$$T_{\pi} = a + b(A/W_d - 1)$$

Здесь a — это длительность цикла модулей памяти в тактах шины, время образования первого выхода памяти, а также фактор перекрытия. Число тактов, требуемое для получения каждого последующего выхода, обозначено b ; обычно $b=1$. A — это число AU в блоке, а W_d — ширина шины в единицах AU.

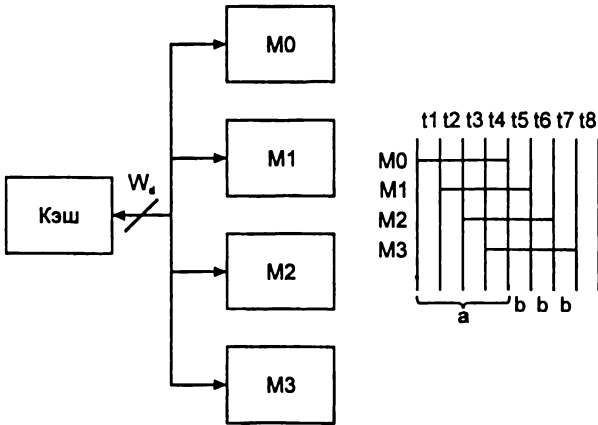


Рис. 4.23. Четырехвходовая память с перекрытием (с любезного согласия Jones and Bartlett Publishers).

Модель транспортного времени следует модифицировать, включив в него время синхронизации шины, к которому мы еще вернемся в гл. 7. Будем считать, что синхронизация шины требует s циклов. Тогда транспортное время будет равно

$$T_{rc} = s + a + b(A/W_d - 1) \text{ тактов}$$

✎ Пример

Рассмотрим память с перекрытием, имеющую следующие характеристики. Фактор перекрытия равен 4, $s = 3$, $A = 32$ байт, а $W_d = 4$ байт. Каким будет транспортное время блока?

✎ Решение

$$T_{rc} = 3 + 4 + (32/4 - 1) = 15 \text{ тактов}$$

✓ Замечание

Если $A \rightarrow \infty$, то $T_{rc} \rightarrow A/W_d$. Другими словами, суммарное время синхронизации шины и чтения первого модуля распределяется по большому числу пересылаемых байтов.

Вернемся теперь к размеру блока. Модель эффективного запаздывания кэш-памяти можно получить, используя выражения для T_{rc} и P_{miss} базовой модели эффективного запаздывания кэш-памяти t_{elc} :

$$t_{elc} \text{ (в тактах)} = 1 + P_{miss} \times T_{rc}$$

$$P_{miss}(\text{конечное}) = P_{miss}(\text{начальное}) L^{\log_2 \frac{\text{конечный размер блока}}{\text{начальный размер блока}}}$$

$$T_{rc} = s + a + b \times (A/W_d - 1) \text{ такт}$$

$$t_{elc} \text{ (в тактах)} = 1 + P_{miss}(\text{начальное}) L^{\log_2 \frac{\text{конечный размер блока}}{\text{начальный размер блока}}} [s + a + b (A/W_d - 1)]$$

Кривая на рис. 4.24, соответствующая нашей модели, показывает, как эффективное запаздывание кэш-памяти изменяется по мере роста размера блока от 16 до 256 байт. График построен для параметров модели $s = 4$, $a = 4$, $b = 1$ и $W = 8$. Параметры кэш-памяти (см. рис. 4.18) соответствуют 8-Кбайтовой общей 4-входовой частично-ассоциативной кэш-памяти. Для 16-байтового блока начальное значение $P_{miss} = 0,015$ и $L = 0,7$.

Из рисунка 4.24 следует важный результат: при фиксированных параметрах системы памяти для размера блока имеется оптимальное значение. Для нашего примера минимум достигается при размере блока 64 байт. Для блоков большего размера транспортное время компенсирует снижение P_{miss} , и эффективное запаздывание кэш-памяти начинает расти. Размытый характер минимума позволяет при проектировании системы несколько изменять размер блока (от 32 до 128 байт) относительно оптимального значения, чтобы удовлетворить другим проектным соображениям.

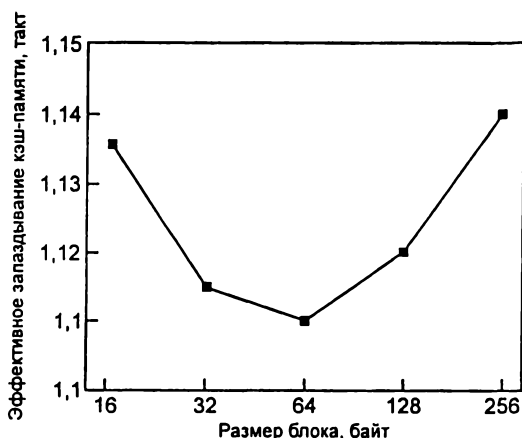


Рис. 4.24. Зависимость эффективного запаздывания кэш-памяти от размера блока.

Адресация памяти с перекрытием

На рисунке 4.25 показан метод адресации памяти с 4-входным перекрытием. Адрес разделяется на два поля; два младших бита адреса декодируются и дают начало сигналам разрешения кристалла для каждого из четырех модулей. По мере инкремента младших байтов адреса в процессе чтения блока последовательных АУ выходы модулей памяти, объединенные по ИЛИ, формируют выходные данные, поступающие на шину.

Обратите внимание на распределение адресов памяти. Модуль M0 содержит АУ с адресами 0, 4, 8 и т. д. В модуле M1 находятся АУ с адресами 1, 5, 9 и т. д. Если инкремент адреса составляет 1, скорость вывода данных будет составлять 1 АУ на такт. Если, однако, адреса инкрементируются на 2, скорость вывода данных уменьшится до 1 АУ за два такта. При инкременте 4 скорость вывода будет 1 АУ на четыре такта. Другими словами, повышение быстродействия системы памяти с перекрытием адресов по отношению к одиночному модулю памяти лежит в следующих границах:

$$1 \leq \text{повышение быстродействия} \leq \text{фактор перекрытия}$$

Пример

Обращение к 4-входовой системе памяти с перекрытием осуществляется с помощью потока адресов с инкрементом 2. Каково будет повышение быстродействия такой системы по сравнению с одиночным модулем памяти?

Решение

Временная диаграмма для указанных условий, схожая с рис. 4.23, приведена на рис. 4.26. Для чтения первой пары АУ из модулей M0 и

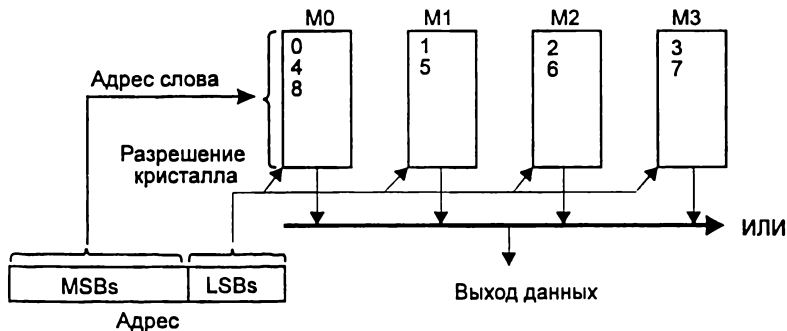


Рис. 4.25. Перекрытие с помощью младших битов адреса (с любезного согласия Jones and Bartlett Publishers). MSBs – старшие биты, LSBs – младшие биты.



Рис. 4.26. Пример перекрывтия.

M2 требуются 5 тактов. Для следующего чтения из M0 и M2 потребуются 4 дополнительных такта и т. д. В итоге для данного примера число тактов на AU составит $2n + 1$, где n — число AU, считываемое из памяти (четное).

При одном модуле памяти число тактов на 1 AU составляет na , где a — фактор перекрывтия. Таким образом, повышение быстродействия памяти с перекрывтием по сравнению с одиночным модулем составит

$$\text{повышение быстродействия} = na / (2n + 1)$$

Для больших n повышение быстродействия равно $(4/2) = 2$.

Память с длинными словами

Другим способом увеличения пропускной способности системы памяти является использование памяти с длинными словами (wide word). В этом случае один цикл памяти помещает длинное слово, состоящее из нескольких AU, в буфер, который затем передается на шину. Описываемый способ проиллюстрирован на рис. 4.27. Старшие биты адреса поступают во все модули памяти одновременно. Выходы модулей одновременно помещаются в буфер. После этого с помощью младших битов адреса данные последовательно передаются из буфера на шину.

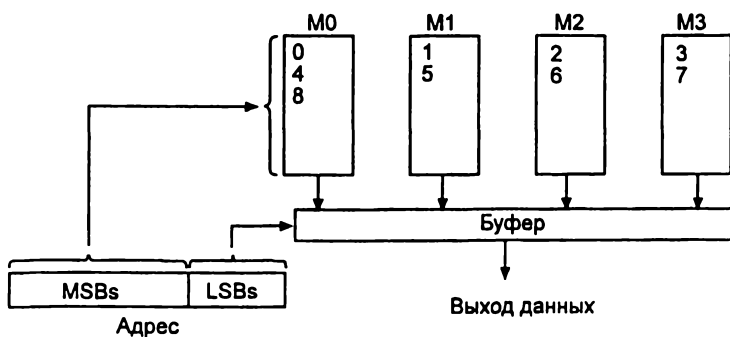


Рис. 4.27. Память с длинными словами (с любезного согласия Jones and Bartlett Publishers). MSBs — старшие биты, LSBs — младшие биты.

Такая конфигурация памяти с перекрытием иногда называется S-доступом, поскольку обращение к модулям выполняется одновременно (simultaneously). Повышение быстродействия в этом случае такое же, как и для конфигурации С, приведенной на рис. 4.23. Недостатком С-доступа является повышенный уровень шума, поскольку все модули памяти переключаются в один и тот же момент.

Модули MOS DRAM

Изобретение модулей MOS DRAM (динамическое ОЗУ на МОП-структурах) знаменовало революцию в производстве компьютерной памяти. Возможная организация DRAM-памяти показана на рис. 4.28. Обычно память представляет собой квадратную матрицу из одностранзисторных ячеек. Старшие биты адреса выступают в качестве адреса строки, а младшие биты адресуют столбец. Операции чтения или записи передают на контакт ввода-вывода (I/O) один бит.

На рисунке 4.29 показана упрощенная временная диаграмма базовой организации DRAM. Операция чтения строки с заданным адресом начинается по стробу адреса строки RAS (row address strobe). Обращение к столбцу осуществляется после обращения к строке по стробу адреса столбца CAS (column address strobe). В общем случае доступ к строке занимает большее время, чем доступ к столбцу. Для $t_{row} > t_{col}$ вре-

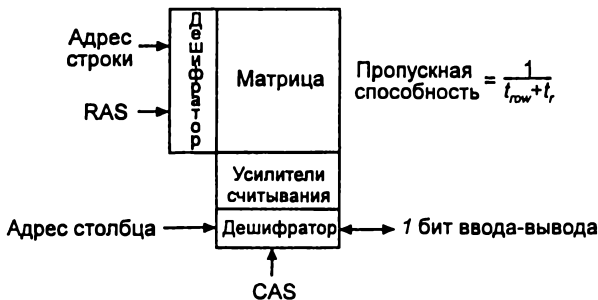


Рис. 4.28. Базовая организация DRAM.

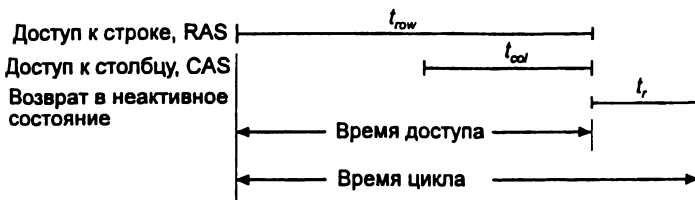


Рис. 4.29. Упрощенная временная диаграмма базового DRAM.



Рис. 4.30. DRAM с полубайтовым режимом.

мя доступа $\approx t_{\text{row}}$. После завершения обращения к матрице требуется дополнительное время t_r для возврата различных сигналов в неактивное состояние. В результате полный цикл памяти равен $t_{\text{row}} + t_r^{48)}$. Пропускная способность при такой организации составляет 1 бит на цикл.

В работе Крейгона [1] показано, что поскольку число адресуемых битов в строке равно квадратному корню из полного числа битов матрицы, значительная часть пропускной способности памяти пропадает. Например, матрица объемом 64 Кбит имеет в одной строке 256 бит, однако при обращении к строке используется фактически лишь один бит. Для преодоления этих потерь в пропускной способности были предложены различные усовершенствования, которые позволяют использовать более полно скрытую пропускную способность памяти.

Далее мы рассмотрим только один из многочисленных существующих вариантов организации DRAM. Все эти варианты ставили своей целью уменьшение времени запаздывания (времени произвольного доступа) и увеличение пропускной способности. Имейте в виду, что различные производители используют разные термины для описания своих вариантов DRAM.

Память DRAM работает асинхронно. Другими словами, контроллер посылает в микросхемы DRAM различные синхронизирующие сигналы, задающие последовательность доступа к строкам и столбцам. Выходные данные появляются на контакте I/O в некоторый гарантированный момент, и в обязанности проектировщика системы памяти входит обеспечение того, чтобы выход памяти был зафиксирован в регистре. Далее содержимое этого регистра может быть прочитано синхронно по системному такту.

DRAM с полубайтовым режимом. К ранним конфигурациям DRAM относится память DRAM с полубайтовым режимом, показанная на рис. 4.30. Здесь сначала выполняется обращение к строке, затем четы-

⁴⁸⁾ В описании работы DRAM используется большое количество других временных параметров, однако наша простая модель позволяет выполнить оценку относительной производительности различных вариантов организации DRAM.

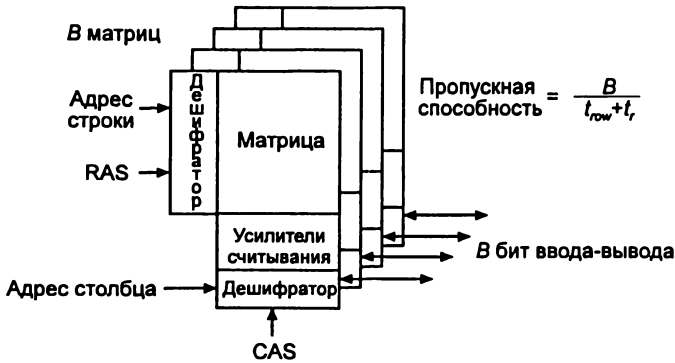


Рис. 4.31. DRAM со страничным режимом.

ре обращения к столбцам, в результате чего последовательность из 4 битов посылается на контакт I/O. Первый бит требует t_{row} . Каждый последующий бит требует цикла доступа к столбцу t_{col} ; затем для завершения операции должно пройти время t_r . Полубайтовый режим позволяет получить повышение пропускной способности по сравнению с базовой организацией приблизительно на 90%.

DRAM со страничным режимом. За период с 1978 по 1996 г. число битов в DRAM удваивалась приблизительно каждые 1,5 года, что дало рост больший, чем предсказывался по закону Мура. По мере увеличения числа битов в матрице растет длина строк и столбцов, что приводит к росту t_{row} и t_{col} , который, впрочем, несколько снижен из-за уменьшения геометрических размеров DRAM. Тем не менее время цикла произвольного доступа оказывается приблизительно тем же, что и для базовых DRAM. Чтобы ускорить работу памяти, проектировщики DRAM приняли схему, схожую с принципом памяти с длинными словами, обсуждавшимся выше и продемонстрированным на рис. 4.27.

DRAM со страничным режимом показана на рис. 4.31. При такой организации одновременно выполняется обращение к одноименным строкам B матриц (так же, как и в базовой DRAM, рис. 4.28). По одному биту из каждой плоскости параллельно поступает на контакты I/O; приблизительное значение пропускной способности показано на рисунке. В силу разделения памяти на плоскости каждая матрица может быть меньше, что уменьшает время доступа по сравнению с единой матрицей того же объема. Если $B = 4$, пропускная способность при такой организации приблизительно вдвое выше, чем для DRAM с полубайтовым режимом.

DRAM со страничным режимом получили практическое распространение одновременно с улучшением технологии изготовления очень больших интегральных схем и появлением корпусов с большим числом выводов. Например, память 1 Мбит $\times$ 16 требует 20 адресных выводов и 16 выводов I/O, не считая различных управляющих сигналов.

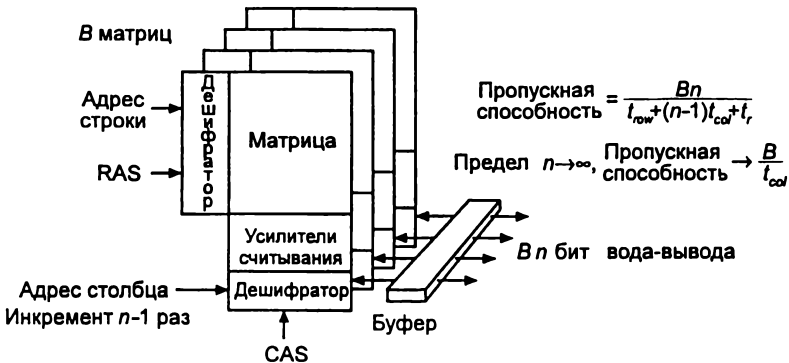


Рис. 4.32. EDO DRAM.

EDO DRAM. Эти модули памяти (EDO — extended data out, расширенный вывод данных), известные также, как гиперстраничные DRAM, отличаются от DRAM со страничным режимом наличием выходного буфера, как это показано на рис.4.32, и используют метод инкремента, как и DRAM с полубайтовым режимом. Устройство выполняет пакетную передачу n слов по B бит каждое. Первое слово из B бит, прочитанное из матриц, запоминается в буфере; эта операция занимает полное время доступа t_{row} . Затем адреса столбцов инкрементируются, и читаются следующие B бит, в то время как содержимое буфера передается на контакты I/O. Для больших значений n потоковая пропускная способность приближается к B/t_{col} .

При сравнении с DRAM страничного режима оказывается, что время произвольного доступа ($n = 1$) остается тем же, однако потоковая интенсивность существенно растет. Это может быть важно для таких приложений, как обработка изображений, а также для загрузки кэш-памяти. Очевидно, что максимальное значение n равно числу столбцов в одной матрице. Если адрес в результате оборачивания переходит к другой строке, выполняется новый цикл обращения к строке.

✎ Пример

Рассмотрим 4-Мбитовую EDO RAM с 8 контактами I/O. Скорость доступа к столбцам для матрицы составляет 33 МГц. Какова максимальная потоковая пропускная способность такого устройства?

✍ Решение

Поскольку при каждом обращении к столбцу передается один байт, потоковая пропускная способность устройства составит 33 Мбайт/с.

DRAM с синхронной связью

Типичная организация DRAM с синхронной связью (synchronous-link DRAM, SLD RAM) схожа с организацией EDO на рис. 4.32. Отличие заключается в том, что система предоставляет модулям DRAM тактовые сигналы, которыми синхронизируется все управление памятью. Ввод и вывод данных также осуществляются синхронно по системным тактам. Такая организация обеспечивает более устойчивую работу памяти, так как устраняются минимальное и максимальное значения времени доступа. Спецификация SLD RAM была разработана рабочей группой стандартов IEEE; детали организации SLD RAM описаны в [4].

Ядром синхронной DRAM по-прежнему является асинхронная схема, приведенная на рис. 4.28. Основное отличие заключается в том, что большая часть синхронизирующих сигналов создается в микросхеме DRAM. Многочисленные сигналы управления теперь не должны генерироваться и разводиться на системной плате.

✎ Пример

Пусть DRAM характеризуется следующими параметрами: $t_{\text{row}} = 70$ нс, $t_{\text{col}} = 25$ нс и $t_r = 6$ нс. Какова пропускная способность (BW) в битах за наносекунду для всех описанных конфигураций DRAM: базовой, с полубайтовым режимом, со страничным режимом и четырьмя матрицами и EDO DRAM с четырьмя матрицами, если $n = 8$?

✎ Решение

базовая $BW = 1/(70 + 5) = 1/75 = 0,013$ бит/нс

полубайтовая $BW = 4/(70 + 3 \times 25 + 5) = 4/150 = 0,026$ бит/нс

страничная $BW = 4/(70+5) = 4/75 = 0,053$ бит/нс

EDO $BW = (4 \times 8)/(70 + 7 \times 25 + 5) = 32/250 = 0,128$ бит/нс

✓ Замечание

Каждый шаг усложнения организации приблизительно удваивает пропускную способность DRAM. Однако страничная DRAM и EDO имеют три дополнительные вывода I/O. Заметьте, что предел пропускной способности пакетного режима для EDO составляет 0,16 бит/нс для очень больших значений n .

4.5. Кэш-память виртуальных и реальных адресов

На рисунке 4.11 было показано, что адреса, поступающие в кэш-память, могут быть как виртуальными адресами от процессора, так и транслированными реальными адресами. В современных процессорах используется и тот, и другой метод. Преимущества и недостатки этих методов собраны в табл. 4.13. Как следует из этой таблицы, популярные процессоры используют для обращения к кэш-памяти реальный адрес.

Таблица 4.13. Кэши виртуальных и реальных адресов

Метод адресации	Преимущества	Недостатки
Виртуальные адреса. Используются на Intel i860, SPUR	Трансляция адресов не сопровождается дополнительным временем запаздывания. Отсутствуют ограничения размера кэш-памяти (можно использовать столько битов виртуального адреса, сколько требуется).	Если для адресации кэш-памяти используется только часть виртуального адреса (для уменьшения числа битов тега), разные виртуальные адреса могут указывать на один и тот же адрес кэша (синоним или псевдоним).
Реальные адреса. Используются на VAX 11/780, Pentium Pro ^{*)} , IBM RS/6000 ^{*)} , IBM S370	Нет проблемы синонима, или псевдонима.	Добавляется время запаздывания, поскольку необходима трансляция адреса. При использовании небольшого специального TLB запаздывание может быть небольшим.

^{*)} С конвейером

Кэши реального адреса могут содержать конвейеры для преодоления потери производительности этого метода. При использовании конвейера, как показано на рис. 4.33, индекс кэш-набора и смещение образуются непосредственно из смещения виртуального адреса, которое не подвергается трансляции. Пока выполняется обращение к кэшу и его тегам, имя страницы транслируется и затем сравнивается с реальными тегами в кэше.

Конвейеризация подробно описывается в гл. 6. Однако уже из рис. 4.33 должно быть ясно, что при наличии стационарного потока достоверных адресов, поступающих в этот кэш, в каждом такте будет происходить одно обращение к кэшу. На рисунке 4.33 изображен пря-

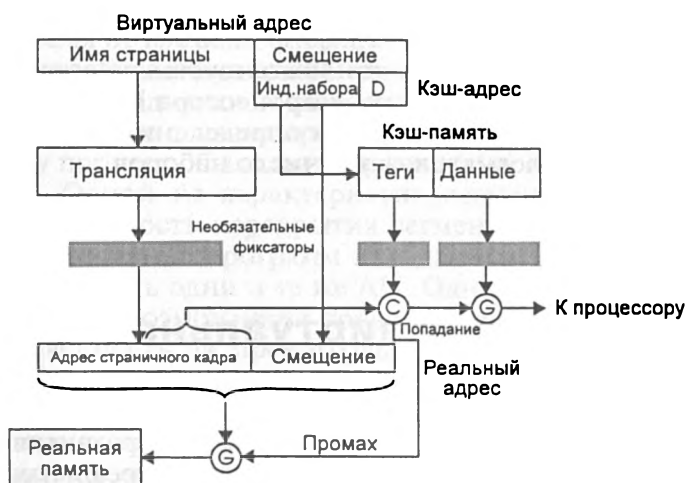


Рис. 4.33. Кэш-память реального адреса с конвейером (с любезного согласия Jones and Bartlett Publishers).

мой кэш; однако для увеличения размеров кэш-памяти можно использовать также и n -входовый частично-ассоциативный кэш. Для кэша реальных адресов с конвейером максимальный размер кэша определяется как

$$\text{размер страницы} \times \text{степень частичной ассоциативности}$$

✎ Пример

Процессор с побайтовой адресацией использует 2-Кбайтовые страницы и 16-байтовые блоки. Какая степень частичной ассоциативности требуется при использовании 8-Кбайтового кэша реальных адресов с конвейеризацией?

✎ Решение

Виртуальный адрес содержит 11-битовое поле смещения. 4 бит из виртуального смещения используются в качестве кэш-смещения в 16-байтовом блоке. Для индекса набора остается 7 бит. Требуемая степень ассоциативности составит

$$\frac{\text{желательный размер кэш-памяти}}{\text{размер страницы}} = \frac{2^{13}}{2^{11}} = 4$$

✓ **Замечание**

Рассмотренная конфигурация кэш-памяти используется во встроенных в кристалл кэшах команд и данных процессора Intel i486. Любопытно задаться вопросом, почему при определении максимального размера кэша не рассматривается число наборов.

4.6. Сегментированная виртуальная память

Все рассуждения насчет виртуальной памяти в ранние годы развития вычислительной техники основывались на том факте, что реальная память была весьма дорогой. Например, 1Мбайт памяти на магнитных сердечниках для IBM S360 в 1965 г. стоил 1,3 млн долл.! Из-за высокой стоимости памяти идея использования страниц фиксированной длины рассматривалась многими проектировщиками как неэкономичная, потому что для каждого выделенного рабочего набора⁴⁹⁾ в среднем 1/2 страницы пространства реальной памяти оказывалась неиспользуемой. Считалось, что если единица выделения памяти составляет не страницу, а один AU, то каждому рабочему набору можно выделить в точности столько реальной памяти, сколько нужно для его функционирования.

Такое направление рассуждений привело к концепции сегмента, который представляет собой страницу переменной длины с дробностью^{*)} 1 AU. При фиксации отказа сегмента в реальную память из дисковой виртуальной следует загрузить требуемый сегмент. Однако сегментация ведет к массе осложнений. Например, если системе требуется загрузить сегмент размером 3456 байт, то в реальной памяти необходимо найти или создать свободный связный участок, размер которого равен или превышает 3456 байт. В отличие от страничных систем, в которых для указания на занятость страничного фрейма используется простой вектор битов, в сегментированной системе свободные области памяти должны идентифицироваться по стартовым адресам и длинам. Эта информация обычно сохраняется в виде списка, который приходится просматривать при поиске свободного связного участка, поскольку сегмент не может быть разделен на части.

Свободное пространство реальной памяти (т. е. пространство, не назначенное активным сегментам) нуждается в управлении; обычно оно собирается в одну большую свободную область. Это означает, что

⁴⁹⁾ Рабочий набор — это набор страниц, выделенный программе для обеспечения ее эффективного выполнения.

^{*)} Объемом неделимого фрагмента. — *Прим. ред.*

время от времени операционная система должна перемещать сегменты, собирая их в одно место, объявлять ранее занимаемые ими области свободными и переопределять эти области, создавая из них свободное связное пространство. Этому вспомогательному служебному процессу было дано оригинальное название «сборка мусора».

Одной из характеристик сегментированной системы является возможность перекрытия сегментов, допускающего совместное использование программ или данных. Два или более сегментов могут содержать одни и те же АУ. Однако совместное использование приводит к возможности появления в программах ошибок и сбоев; ответственность за предотвращение сбоев возлагается на программное обеспечение.

После большой исследовательской и конструкторской работы, целью которой было заставить сегментированную систему функционировать в чистом виде, эта идея была отброшена. Такая система требовала слишком больших аппаратных и программных накладных расходов, а цена памяти, главное основание для использования сегментации, неуклонно падала. Тем не менее системы с сегментами используются сегодня в двух формах: страничная сегментация и виртуальная сегментация.

Страничная сегментация

Страничная сегментация используется сегодня в большинстве компьютеров в системах управления программами. Вместо выделения памяти адресуемыми элементами (АУ), как это делалось в чисто сегментированных системах, при страничной сегментации память выделяется на уровне страниц. Управление сегментами (сбор страниц) выполняется операционной системой при некоторой аппаратной поддержке, причем аппаратура и программное обеспечение управляют страницами так, как это было описано в разделе 4.2.

В системе со страничной сегментацией эффективный адрес расширяется путем сцепления с полем, находящимся в управляющем регистре или таблице, индексируемой частью битов эффективного адреса, как это показано на рис. 4.34. Детали организации таблиц трансляции виртуального адреса были показаны в рис. 4.6 и 4.8.

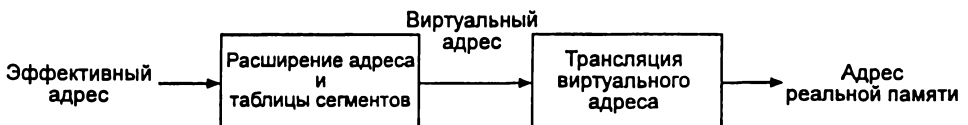


Рис. 4.34. Трансляция адреса в системе со страничной сегментацией.

IBM S370

На рисунке 4.35 показано взаимодействие таблиц сегментов и страниц в системе виртуальной памяти IBM S370. 24-битовый логический (эффективный) адрес расширяется на 18 бит содержимым управляющего регистра 1, давая в сумме 42-битовый виртуальный адрес. 24-битовый логический адрес S370 затем подразделяется на индекс сегмента, индекс страницы и индекс байта, или смещение. Длины полей в логическом адресе могут варьироваться в зависимости от заданной конфигурации.

Предусмотрены следующие конфигурации:

64-Кбайтовые сегменты, 4-Кбайтовые страницы

64-Кбайтовые сегменты, 2-Кбайтовые страницы

1-Мбайтовые сегменты, 4-Кбайтовые страницы

1-Мбайтовые сегменты, 2-Кбайтовые страницы

Биты управляющего регистра 1 добавляются к индексу сегмента, давая в итоге индекс таблицы сегментов. Биты из элемента таблицы сегментов добавляются к индексу страницы с образованием индекса таблицы страниц. Биты из таблицы страниц сцепляются с индексом байта, образуя 24-битовый адрес реальной памяти. Содержимое управляющего регистра 1, а также элементов таблиц сегментов и страниц будут рассмотрены в разделе 8.2.

Поскольку и сегментные, и табличные страницы находятся в реальной памяти, эти таблицы сканируются буфером TLB, содержащим данные предтрансляции.

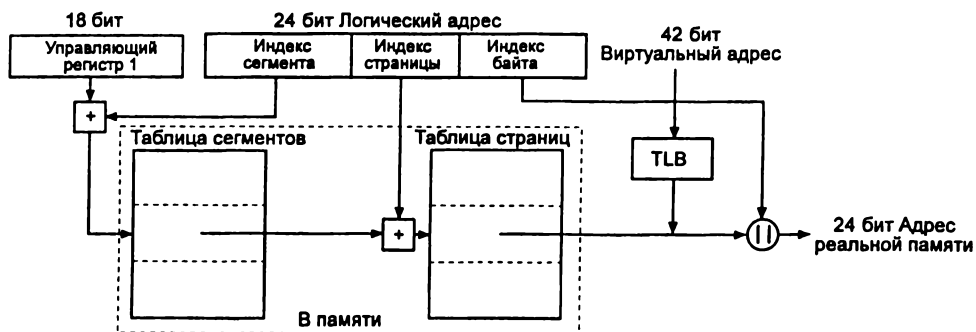


Рис. 4.35. Трансляция виртуальных адресов в IBM S370.

Intel Pentium Pro

Схема сегментации процессора Intel Pentium Pro показана на рис. 4.36. В функции таблицы дескрипторов входит задание базовых линейных адресов сегментов, а также их длин.

14 бит из 16-битового сегментного селектора, находящегося в аппаратном регистре, служат индексом таблицы дескрипторов, размещаемой в реальной памяти. Всего имеются шесть сегментных регистров: один для сегмента кода, один для сегмента стека и четыре для сегментов данных. Одно из полей дескриптора сегмента содержит базовый адрес, который после сложения с 32-битовым смещением образует 32-битовый линейный адрес. Линейный адрес расширяется 20-ю битами регистра CR3 и выбираемое из него имя страницы транслируется двухуровневой таблицей, показанной на рис. 4.6.

Таблица дескрипторов, каталог страниц и таблица страниц находятся в реальной памяти, поэтому буфер TLB и скрытые дескрипторы сегментов, хранящиеся в аппаратных регистрах, выпадают из тракта трансляции. Скрытые регистры дескрипторов сегментов хранят те же дескрипторы сегментов, которые находятся в таблице дескрипторов. Каждому из шести селекторов сегментов соответствует свой скрытый

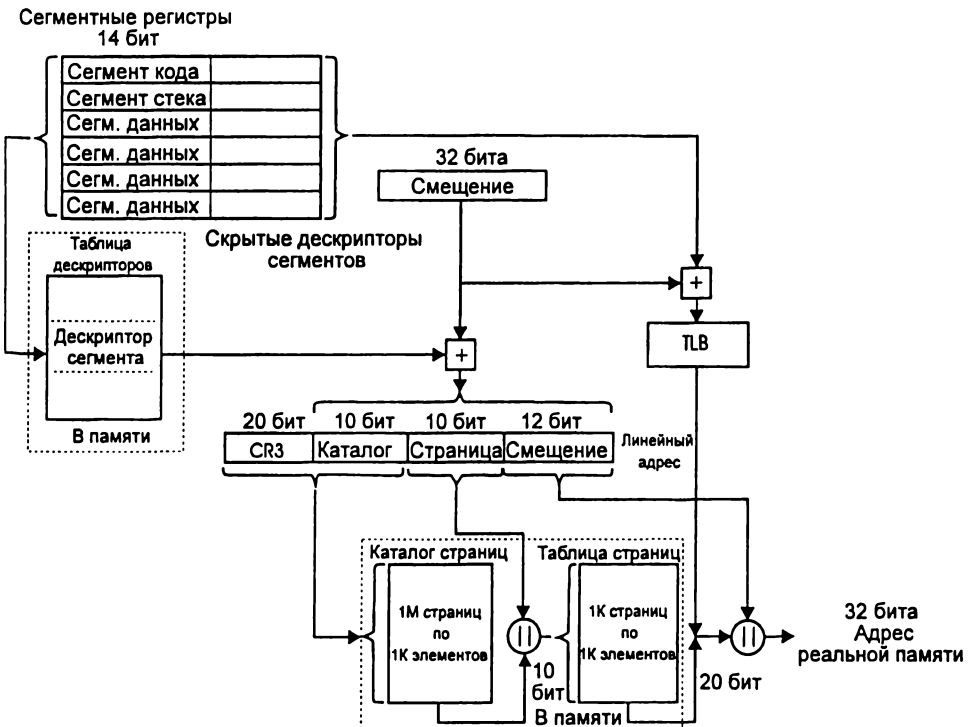


Рис. 4.36. Сегментация в Pentium Pro.

регистр. Таким образом, образуется быстрый тракт для получения 20-битового базового адреса^{*)} для обращения к реальной памяти.

Состав селектора, дескриптора сегмента, а также элементов таблиц сегментов и страниц будет рассмотрен в разделе 8.2.

Виртуальная сегментация

Системы с виртуальной сегментацией сегодня можно найти в некоторых интерактивных программных системах, таких как Lotus 1-2-3. Эта система отличается высокой степенью динамичности, для нее характерно частое выделение и освобождение ячеек. Число выделенных и освобожденных ячеек непостоянно, и система страничной сегментации оказывается недостаточно гибкой. Поэтому для автоматического выделения и освобождения ячеек, а также для сбора мусора реализуется примитивная чисто сегментная система. На ранних компьютерах типа PC, в которых использовались процессоры i286 и i386, издержки для такого рода задач были слишком заметны. Вариант этой системы для MS DOS даже включал предупреждающий сигнал, который говорил пользователю, что идет выделение, освобождение или сбор мусора.

4.7. Дисковая память

На самой нижней ступени иерархии памяти стоит жесткий диск. Под диском мы будем понимать жесткий диск в терминологии машин PC; другие формы дисков, например, гибкие дискеты, не подсоединены к системе постоянно и могут рассматриваться, как системы архивации. Это различие будет обсуждено в дальнейшем в гл. 7.

Организация диска включает несколько уровней, как это показано на рис. 4.37. Диск состоит из нескольких пластин («тарелок»), причем на каждой пластине используется одна или обе стороны. Каждая поверхность содержит некоторое количество дорожек, разделенных на секторы; число байтов в секторе составляет целую степень двух. В некоторых конструкциях дисков все дорожки имеют равное число секторов. В результате плотность битов убывает при приближении к периферии диска. В других конструкциях плотность битов на дорожке остается постоянной, но при приближении к периферии диска увеличивается число секторов на дорожке. Такая конструкция диска

^{*)} 4-Кбайтовой страницы в реальной памяти. — *Прим. перев.*



Рис. 4.37. Организация диска (с любезного разрешения Jones and Bartlett Publishers).

называется зонированной. Обычно адресуемой единицей диска является сектор; сектор диска не следует путать с сектором кэш-памяти.

Каждой поверхности соответствует один позиционер несущих головок чтения и записи или одной комбинированной головки. Головки записи преобразуют цифровые сигналы в магнитные, которые поляризуют магнитную поверхность, создавая на ней последовательность нулей и единиц. Головки чтения по мере вращения диска воспринимают изменения магнитного потока и генерируют последовательность нулей и единиц. Заметьте, что области битов на диске не образуют заранее созданные объекты (как это имеет место в модулях DRAM-памяти), а определяются воздушным зазором головки записи, когда поверхность пластины в процессе ее вращения проходит под головкой.

Транспортное время записи или чтения диска представляет собой сумму нескольких временных компонентов. Во-первых, имеется запаздывание вращения. Многие диски имеют только одну головку чтения-записи для каждой поверхности, и для того, чтобы требуемый сектор подошел к головке, диск должен повернуться на некоторый угол. При скоростях вращения от 3600 до 10000 оборотов в минуту максимальное время запаздывания вращения равно длительности полного оборота, что составляет от 16,6 до 6 мс. Минимальное запаздывание вращения равно нулю, а среднее соответствует половине оборота, или для скорости вращения 5200 оборотов в минуту 5,75 мс⁵⁰.

Вторым компонентом запаздывания является запаздывание поиска. Это запаздывание определяется временем позиционирования головки на адресуемую дорожку. Позиционер с головкой должен стронуться с исходной позиции, ускориться и установиться над требуемой дорожкой. Минимальное запаздывание равно нулю, если головка уже позиционирована, но максимальное может достигать 20 мс и более, в зависимости от числа дорожек на поверхности. Для снижения времени запаздывания поиска в многозадачных компьютерах разработаны различные приемы.

⁵⁰) Переносные компьютеры, которые ради сохранения энергии батарейного питания останавливают вращение дисков, имеют еще большее запаздывание, поскольку для раскрутки диска требуется значительное время (5–10 с).

Следующий компонент запаздывания связан с работой операционной системы. Он может достигать 1–2 мс, в зависимости от сложности обработчика отказов страниц и скорости процессора. Операционная система или обработчик отказов страниц должны выполнить трансляцию двоичного виртуального адреса в физические адреса дисковой системы: диск, пластина, поверхность, дорожка и сектор.

Помимо запаздывания диска в транспортное время следует еще включить время передачи данных на диск или с диска. Время, требуемое для фактического чтения или записи данных, определяется размером сектора, числом непрерывно следующих секторов, пересылаемых за раз, а также скоростью поступления битов. Для современных дисков скорость передачи находится в диапазоне 5 – 15 Мбит/с для одной головки. Если для передачи параллельного потока битов используется набор головок, полная скорость передачи умножается на число головок в наборе.

Полное транспортное время для чтения или записи включает три компонента запаздывания и время передачи:

$$\begin{aligned} \text{транспортное время} = & \text{запаздывание вращения} + \text{запаздывание поиска} + \\ & + \text{запаздывание операционной системы} + \text{время передачи} \end{aligned}$$

✎ Примеры

1. Диск вращается со скоростью 5200 оборотов в минуту, скорость передачи данных составляет 5 Мбайт/с, частота сети – 50 Гц. Среднее время поиска равно 25 мс, размер сектора 1 Кбайт. Компьютер со скоростью 100 MIPS имеет 4-Кбайтовые страницы, а обработчик отказа страниц состоит из 750000 команд.

А. Каково транспортное время для одной страницы при частоте сети 60 Гц?

Б. Каково транспортное время для одной страницы при частоте сети 50 Гц?

✎ Решение

А. Время запаздывания составит

$$5,75 \times 10^{-3} + 25 \times 10^{-3} + 7,5 \times 10^5 / (100 \times 10^6) \times 10^{-3} = 38,25 \text{ мс.}$$

$$\text{Время передачи составит } 4 \times 1024 / (5 \times 10^6) = 0,819 \text{ мс.}$$

Транспортное время будет равно

$$38,25 \times 10^{-3} + 0,819 \times 10^{-3} = 39,07 \text{ мс.}$$

Б. При частоте сети 50 Гц среднее запаздывание вращения равно 6,9 мс, а скорость передачи данных 4,16 Мбайт/с. Запаздывание будет равно $6,9 \times 10^{-3} + 25 \times 10^{-3} + 7,5 \times 10^{-3} = 39,49 \text{ мс.}$

Время передачи составит

$$0,819 \times 10^{-3} \times 60 / 50 = 0,819 \times 10^{-3} \times 1,2 = 0,983 \text{ мс.}$$

Транспортное время будет равно

$$39,49 \times 10^{-3} + 0,983 \times 10^{-3} = 40,47 \text{ мс.}$$

2. Диск имеет 128 секторов на дорожке и 512 байт в секторе. Скорость вращения равна 5200 оборотов в минуту. Каково время передачи в единицах «байт/с»?

Решение

Время оборота составит $60/5200 = 11,5$ мс. Время передачи равно $(128 \times 512 / 11,5) \times 10^{-3} = 5,7$ Мбайт/с.

Дисковая кэш-память

Приведенный выше пример определения транспортного времени для диска иллюстрирует одну из проблем иерархии памяти. Даже если интенсивность отказа страниц очень низка, когда отказ все-таки возникает, происходит ощутимая приостановка выполнения программы на время разрешения отказа страницы. Одно из решений этой проблемы состоит в использовании кэш-памяти между реальной памятью и диском, как это показано на рис. 4.38.

Имеются две возможности реализации дискового кэша: 1) реализовать аппаратный кэш или 2) реализовать виртуальный кэш диска в реальной памяти.

Казалось бы, первый вариант весьма прост: проектировщик должен только выбрать размер и способ организации. Однако здесь возникает сложность с выбором организации тегов кэша. Виртуальный адрес, формируемый процессором, является двоичным числом, дисковый же адрес не двоичный; в него входят диск, пластина, дорожка и сектор. Если эти поля должны быть назначены тегам кэша, следует предусмотреть возможность увеличения емкости диска или установку нового диска, который может быть организован по-иному. Преодолеть это затруднение можно следующим образом: в тегах использовать виртуальный адрес, а отображение виртуального адреса на дисковый выполнять программно при каждом промахе дискового кэша.

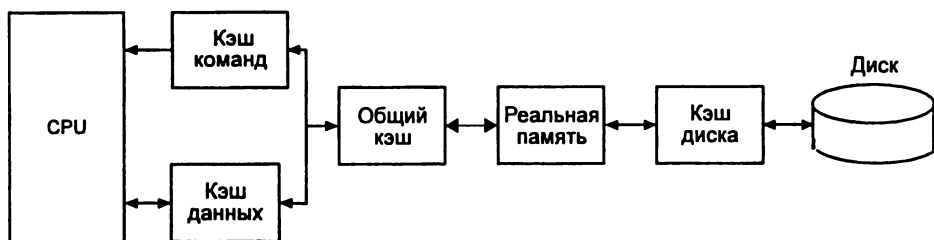


Рис. 4.38. Дисковая кэш-память.

В втором варианте место для виртуального дискового кэша выделяется в реальной памяти, а все требуемые функции выполняются программным обеспечением. Даже если эти функции потребуют нескольких сотен тысяч команд, при использовании процессора на 100 MIPS затраченное время будет невелико по сравнению с транспортным временем диска — микросекунды, а не миллисекунды.

Операционная система MS DOS предоставляет виртуальный дисковый кэш. В зависимости от объема памяти в системе дисковый кэш (реализуемый программой SMARTDrive) может достигать до 2 Мбайт. Размер страницы выбирает пользователь; типичным является значение 8 Кбайт. Пользователь может также установить размер кэш-памяти, причем значение интенсивности попаданий от 0,7 до 0,8 вполне достижимо. SMARTDrive поддерживает чтение и запись только при обмене данными с файловой системой; виртуальная память не поддерживается.

Windows 95 имеет два виртуальных кэша: один для системы виртуальной памяти (программа VCACHE), другой для компакт-дисков (программа CDFS). В отличие от SMARTDrive для MS DOS, размер обоих кэшей Windows устанавливается динамически операционной системой. Эти размеры выбираются максимально возможными исходя из свободной памяти в конкретный момент. Размер страницы программы VCACHE фиксирован и составляет 4 Кбайт.

✎ Пример

Система памяти с транспортным временем 50 мс для 4-Кбайтовых страниц модернизируется установкой дискового кэша. Вероятность попадания $P_{\text{hit}} = 0,75$, а транспортное время передачи страницы из дискового кэша в реальную память равно 50 мс. Во сколько раз установка кэша уменьшит транспортное время?

✍ Решение

$$\text{взвешенное среднее транспортное время} = 0,75 \times 50 \times 10^{-6} + 0,25 \times 50 \times 10^{-3} = 12,5 \times 10^{-3} \text{ с}$$

$$\text{повышение быстродействия} = 50 \times 10^{-3} / (12,5 \times 10^{-3}) = 4$$

✓ Замечание

Мы получили хороший пример для закона Амдала. Три четверти времени уменьшились почти в бесконечное число раз (быстродействие RAM по отношению к быстродействию диска), а одна четверть времени не претерпела изменений. Полное повышение производительности легко определяется следующим образом:

$$\text{повышение быстродействия} = 1/(1 - \alpha) = 1/(1 - 0,75) = 1/0,25 = 4$$

4.8. Действие системы

Мы рассмотрели компоненты системы виртуальной памяти, использующей кэш-память. Теперь мы можем объединить все эти компоненты и посмотреть, что же происходит с адресом и данными по мере того, как они проходят сквозь систему. Для наглядности рассмотрим операцию чтения в системе с перечисленными ниже компонентами.

- Кэш данных L1
 Попадание при записи: сквозная запись
 Промах при записи: запись без размещения
- Общий кэш уровня L2
 Попадание при записи: сквозная запись
 Промах при записи: запись без размещения
- Реальная память с перекрытием (RM): сквозная запись
- Буфер ассоциативной трансляции
- Инверсная таблица страниц
- Дисковый кэш (DC)
 Попадание при записи: обратная запись
 Промах при записи: запись без размещения
- Диск

Первичная таблица событий приведена ниже (табл. 4.14). Таблица названа первичной, потому что каждое из действий может потребовать составления собственной таблицы событий. Поскольку в качестве стратегии кэш-памяти принята сквозная запись, все составляющие памяти — кэши, реальная память и дисковый кэш — считаются согласованными.

Ввиду принятия стратегии сквозной записи все кэши могут затираться при перемещении блока на более высокий уровень. Для некоторых событий действия могут выполняться параллельно, что уменьшает время обслуживания события. Проектировщик всегда должен стараться вводить параллельные действия, так как они позволяют повысить производительность системы.

Пусть, например, мы имеем промахи в кэшах L1 и L2; как только данные попадут в процессор, вычисления могут быть продолжены. Передачи данных в два кэша могут выполняться параллельно с передачей в процессор. Если, однако, до завершения передач произойдет следующее обращение к кэшу, это обращение может привести к останову. Другой пример — TLB, который можно опрашивать параллельно с опросом таблицы трансляции адреса.

Таблица 4.14. События, возникающие при операции чтения в системе памяти

L1	L2	RM	DC	Диск	Действие
Попадание	х	х	х	х	Чтение данного из L1 в процессор.
Промех	Попадание	х	х	х	Чтение данного из L2 в процессор. Запись с затиранием блока из L2 в L1.
Промех	Промех	Попадание	х	х	Трансляция адреса в TLB или таблицах. Чтение данного из RM в процессор. Запись с затиранием блока из реальной памяти в L2. Запись с затиранием блока из реальной памяти в L1.
Промех	Промех	Промех	Попадание	х	Трансляция адреса в адреса дискового кэша. Чтение данного из дискового кэша в процессор. Запись страницы с затиранием в RM. Запись блока с затиранием в L2. Запись блока с затиранием в L1. Обновление TLB.
Промех	Промех	Промех	Промех	Попадание	Если страница дискового кэша была модифицирована, отправить ее на диски и прочитать страницу в дисковый кэш. Если страница в RM была модифицирована, отправить ее на диск и прочитать страницу в RM. Рестарт операции чтения.

Литература

1. Cragon, H. G. (1980). «The Elements of Single-Chip Microcomputer Architecture», *Computer*, 13:10, pp. 27–41.
2. Flynn, M. J. (1966). «Very High-Speed Computing Systems», *Proceedings of the IEEE*, 54:12.
3. Gee, J. D., Hill, M. D., Pnevmatikatos, D. N. and Smith A. J. (1993). «Cache Performance of the SPEC92 Benchmark Suit», *IEEE Micro*, 13:4, pp.17–27.
4. Gillingham, P. and Vogley, B. (1997). «SLDRAM: High-Performance, Open-Standard Memory», *IEEE Micro*, 17:6, pp. 29–39.
5. IBM (1976). «IBM System/370 Principles of Operation», Order GA22-7000-5.

6. Kilburn, T., Edwards, D. B. G., Lanigan, M.J. and Summer, F.H. (1962). «One-Level Storage System», *IRE Transactions on Electronic Computers*, 7:4.
7. Lewis, T. G. and Cook, C. R. (1988). «Hashing for Dynamic and Static Internal Tables», *Computer*, 21:10, pp. 45–56.
8. Milligan, M. K. and Cragon, H. G. (1995). «Processor Implementations Using Queues», *IEEE Micro*, 15:4.
9. Saavedra, R. H. and Smith, A. J. (1993). «Measuring Cache and TLB Performance and Their Effect on Benchmark Run Times», Report USCB/CSD 93/767, University of California, Berkeley.
10. Smith, A. J. (1987). «Line (Block) Size Choice for CPU Cache Memories», *IEEE Transactions on Computers*, C-36:9.

Упражнения

- 4.1. Существуют две стратегии обслуживания попадания при записи и две стратегии обслуживания промаха при записи, всего четыре возможных варианта. Перечислите эти четыре варианта и обсудите достоинства каждого.
- 4.2. Кэш команд достаточно велик, чтобы хранить цикл. Каково будет значение P_{miss} для этого кэша для цикла, который выполняется 50, 100 или 200 раз?
- 4.3. Выведите формулу для оценки эффективного запаздывания кэш-памяти по данным из рис. 4.24. Возьмите первую производную, приравняйте нулю и найдите минимальное значение функции.
- 4.4. Рассмотрите производительность памяти с перекрытием, если фактор перекрытия не является целой степенью двух, а представляет собой простое число: 3, 5, 7 и т. д.
- 4.5. Обсудите проблему синонимов, или псевдонимов, характерную для кэшей виртуальных адресов.
- 4.6. Выше обсуждались последствия увеличения размера кэш-памяти реальных адресов с конвейером. Используя данные по интенсивности промахов для контрольных задач SPEC, определите изменение величины P_{miss} кэша i486 для следующих значений степени ассоциативности: 1, 2, 4 и 8.
- 4.7. Вспомните обсуждение зонированных дисков в разделе 4.7. Подумайте, как повлияет на скорость передачи способ распределения секторов по дорожкам.
- 4.8. Обсудите проблемы использования команды перехода со связыванием, которая помещает в память ссылку, применительно к системе памяти с кэшем.
- 4.9. Измените формулу для вычисления t_{clrm} , соответствующую табл. 4.4, применительно к TLB.
- 4.10. Объясните, почему для фиксации кэш-попадания требуется использование бита V. Разве не достаточно сравнения тегов?
- 4.11. На рисунках, изображающих прямые и инверсные таблицы страниц, не показаны биты V. Нужны ли они? Обоснуйте свой ответ.
- 4.12. На рисунке 4.26. показан тракт трансляции адреса в процессоре S370. Допустим, что обращение к таблице сегментов и таблице страниц в реальной памяти занимает по 4 цикла памяти, обращение к TLB требует 0,1 цикла памя-

ти, а TLB характеризуется интенсивностью попаданий 0,95. Определите эффективное время трансляции адреса для этого процессора.

4.13. На рисунке 4.36 показано, что TLB в процессоре Intel Pentium Pro использует в качестве входных данных виртуальные, или линейные, адреса. Каким образом в этой архитектуре преодолевается запаздывание, возникающее при обращении к таблице дескрипторов?

4.14. Объясните, каким образом в архитектуре процессора Intel таблица каталога страниц индексируется 30 битами, если адреса реальной памяти содержат 32 бита.

4.15. Повторите вычисления взвешенного среднего числа проб для инверсного файла с информацией о студенческом классе по табл. 4.2 и 4.3. Воспользуйтесь двумя схемами: хешируются три младших разряда и один младший разряд. Вспомните, что при использовании двух младших разрядов взвешенное среднее число проб составило 1,32.

4.16. Вычислите и нанесите на график эффективное запаздывание кэш-памяти для шины шириной 1, 2, 4, 8, 16 и 32 байт, используя параметры, относящиеся к рис. 4.24. Считайте, что размер блока равен 64 байт.

4.17. В таблице 4.8 даны значения P_{miss} для кэша команд, а в табл. 4.9 — для кэша данных. Возможно ли, используя данные из табл. 4.8 и 4.9, получить разумные оценки данных для табл. 4.10? Если да, составьте таблицу с этими данными; если нет, объясните, почему.

4.18. Выше упоминалось, что можно построить 5-входовый частично-ассоциативный кэш. Приведите достоинства и недостатки такой конструкции. Является ли выбор простого числа в качестве фактора перекрытия выгодным или невыгодным?

4.19. Составьте таблицу событий для записи данных, схожую с табл. 4.14.

4.20. Примите какие-либо значения для интенсивности промахов и времени выполнения действий по табл. 4.14 и вычислите взвешенное среднее время записи в память.

4.21. Перерисуйте рис. 4.12 и 4.13 применительно к современным схемам кэш-памяти, использующим 1 позицию блока на сектор.

ГЛАВА 5

ПРОЕКТИРОВАНИЕ УПРАВЛЯЮЩИХ ЦЕПЕЙ ПРОЦЕССОРА

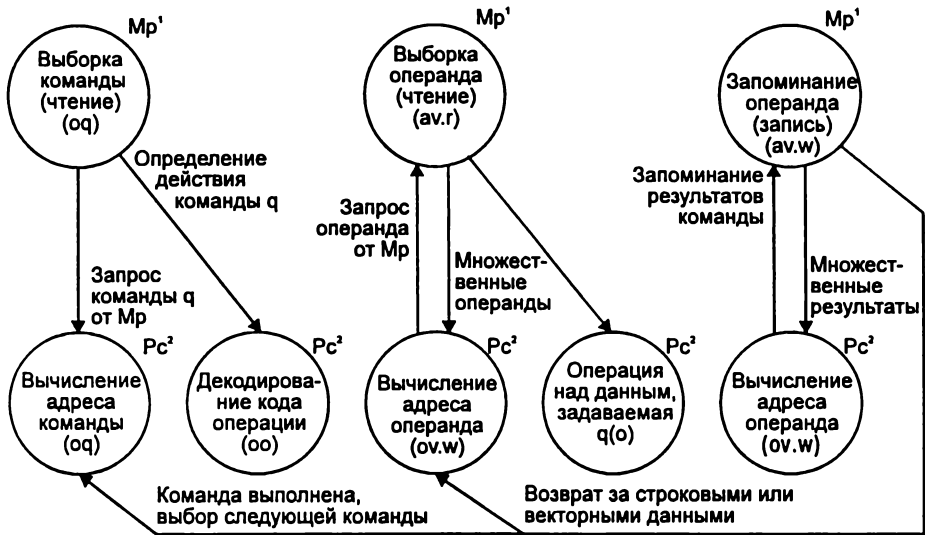
5.0. Введение

Вспомним рис. 3.1, на котором показан интерфейс между программной трансляцией и аппаратной интерпретацией для программы, написанной на языке высокого уровня. В настоящей главе рассматриваются два метода аппаратной трансляции: аппаратное и микропрограммное управление. Сегодня принято изображать последовательность событий, возникающих при интерпретации команды, в виде диаграммы состояний.

Диаграмма состояний для интерпретации команды показана на рис. 5.1 [1]. Эта диаграмма состояний представляет одну из самых ранних систем команд (ISA), в которой использовался цикл интерпретации с последовательным выполнением; для того чтобы началось выполнение следующей команды, должно завершиться выполнение предыдущей. Три верхних состояния управляются памятью; пять нижних состояний управляются процессором. Любое состояние может быть нулевым, что указывает на интерпретацию другой команды.

В качестве базы для описания аппаратного и микропрограммного управления мы будем использовать простую машину с регистровым файлом загрузки-запоминания. Процессор этой машины обеспечивает три класса команд: регистровые арифметико-логические команды ALU, команды загрузки-запоминания и переходы. Формат команд этих трех классов приведен ниже.

Регистровые команды ALU	Кон	RA _{s1}	RA _{s2}	Rdest
Команды загрузки-запоминания	Кон	Rbase	Rs/d	Смещение
Переходы	Кон	Смещение		



¹ Состояние управляется памятью (Mr)

² Состояние управляется процессором (Pc)

Замечание: любое состояние может быть нулевым

Название состояния	Время в состоянии	Значение
soq/oq	toq	Операция определения команды q
saq/aq	taq	Обращение (к памяти) за командой q
so.o/o.o	to.o	Операция декодирования действия q
sov.r/ov.r	tov.r	Операция определения адреса переменной v
sav.r/av.r	tav.r	Обращение (к памяти) для чтения переменной v
so/o	to	Действие, заданное q
sov.w/ov.w	tov.w	Операция определения адреса переменной v
sav.w/av.w	tav.w	Обращения (к памяти) для записи переменной v

Рис. 5.1. Диаграмма состояний процессора (с любезного согласия McGraw-Hill Companies Inc.).

Регистровые команды ALU имеют два операнда-источника и один операнд-приемник, все в регистровом файле. Режим адресации для команд загрузки-запоминания включает базу Rbase и смещение; значение базы хранится в регистровом файле. Команда перехода анализирует знаковый бит R1 в точности так же, как фон-неймановская система команд анализировала содержимое аккумулятора. Адрес перехода команды перехода вычисляется как содержимое программного счетчика плюс смещение. В нашем изложении типы команд ALU, как и размеры полей команд, будут игнорироваться, чтобы не затемнять идеи, лежащие в основе узла управления.

Перед тем как приступить к рассмотрению деталей управления, мы должны рассмотреть то, чем мы будем управлять. В современных реализациях процессоров регистры процессора помещены в тракт дан-

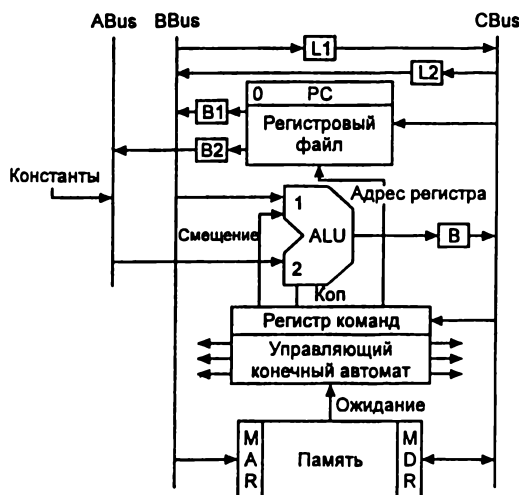


Рис. 5.2. Тракт данных. PC — программный счетчик, ALU — арифметико-логическое устройство, Коп — код операции, MAR — Регистр адреса памяти, MDR — Регистр данных памяти.

ных. Тракт данных, описываемый ниже, предоставляет ресурсы для интерпретации трех перечисленных выше классов команд.

Тракт данных

Как уже отмечалось в гл. 1 и 3, в процессоре содержится большое количество регистров. Эти регистры можно разделить на две категории: архитектурные регистры, доступные из программы (например, регистровый файл) и регистры реализации, используемые управляющей аппаратурой при выполнении команд для побочных результатов (примером является регистр адреса памяти). Управляющий конечный автомат должен иметь возможность читать и запоминать информацию в регистрах обеих категорий.

В ранних компьютерах все регистры процессора были соединены друг с другом в некотором порядке, определяемом конкретными командами. На современном уровне техники, где главенствует идея упорядоченных межсоединений, эти регистры соединяются с помощью одной или нескольких шин. Простой пример тракта данных показан на рис. 5.2; в последующих обсуждениях аппаратного и микропрограммного управления мы будем возвращаться к этому рисунку.

Тракт данных включает три шины, что позволяет двуместным операциям выполняться за один такт. В дополнение к этим трем шинам предусмотрены еще выделенные пути передачи данных, не покрываемые линиями шин. Эти выделенные пути прежде всего связаны с ре-

гистром команд (который хранит команду все время, пока она выполняется). Различные поля команды непосредственно соединены с соответствующими приемниками. Например, поле кода операции соединено с управляющим конечным автоматом и с ALU. Поле адреса регистра выступает в качестве входа регистрового файла.

Для того чтобы шины Bbus и Cbus могли взаимодействовать, предусмотрены два тракта связи L1 и L2. Заметьте также, что программный счетчик расположен в позиции 0 регистрового файла, а шина Abus имеет входы констант, таких, как +1, 0, -1. На выходе ALU расположен буфер В, в который загружаются результаты каждой операции ALU. Предусмотрен также специальный тракт из памяти в управляющий конечный автомат, по которому поступает сигнал, указывающий на необходимость ожидания. Управляющий конечный автомат организует последовательный переход между состояниями и генерирует логические управляющие сигналы, разрешающие пересылки данных по шине.

Тракты данных низкой производительности могут быть реализованы одной единственной шиной. Двухшинные тракты данных увеличивают степень параллельности операций, что приводит к росту производительности. Такие одношинные и двухшинные тракты данных вполне можно использовать в тех случаях, когда стоимость важнее производительности, например, в некоторых встроенных микропроцессорах.

5.1. Аппаратное управление

Сначала мы рассмотрим аппаратное управление. В этом случае проектировщик возлагает все управление на конечный автомат. Основным преимуществом такого подхода является высокая скорость работы. Обычно удается сконструировать конечный автомат, который управляется тактовым генератором более высокой частоты и содержит меньше состояний, чем микропрограммный контроллер, который описан ниже, в разделе 5.2. Характерной чертой аппаратного управления является то, что стоимость аппаратуры управляющего конечного автомата приблизительно пропорциональна числу команд в системе макрокоманд.

Основным недостатком аппаратного управления является отсутствие гибкости. После того, как разработка управляющего конечного автомата завершена, изменения, имеющие своей целью устранение ошибки проектирования или добавление к системе команд, могут потребовать полной повторной разработки управляющего конечного автомата. Если при этом управляющий конечный автомат реализован в виде микросхемы, потребуется еще и полное повторное проектирова-

ние этой микросхемы. На первых порах развития вычислительной техники разработчики часто, внося изменения в управляющий конечный автомат, пользовались желтыми проводами. По мере возрастания объема желтого монтажа можно было наглядно судить о сложностях такого рода проектирования. В следующем разделе будут описаны тракт данных и методика проектирования аппаратного управляющего конечного автомата.

Суть аппаратного управления заключается в том, что конечный автомат проходит через состояния диаграммы состояний, выдавая для каждого состояния в тракт данных соответствующие управляющие сигналы. Диаграмма состояния является основой проектирования управляющего конечного автомата. Это проектирование состоит из пяти шагов:

1. Построение диаграммы состояний.
2. Выделение в тракте данных путей передачи данных.
3. Выполнение вторичных назначений состояний.
4. Построение таблицы истинности переходов между состояниями.
5. Составление входных уравнений для триггеров.

Построение диаграммы состояний

Архитектуре загрузки-запоминания соответствует простая диаграмма состояний, изображенная на рис. 5.3. Первый шаг интерпретации заключается в выборке команды, на которую показывает программный счетчик. Команда декодируется, и если она требует регистровых значений, выполняется обращение к регистрам. Следующее состояние либо вычисляет эффективный адрес ЕА для операций с памятью, либо выполняет функцию, вызываемую в соответствии с кодом операции. Состояние «Память» выполняет для команд загрузки-запоминания операцию чтения (R) или записи (W). Состояние записи в регистр записывает результат операции или данные, записи в регистровый файл записывает результат операции или данные,

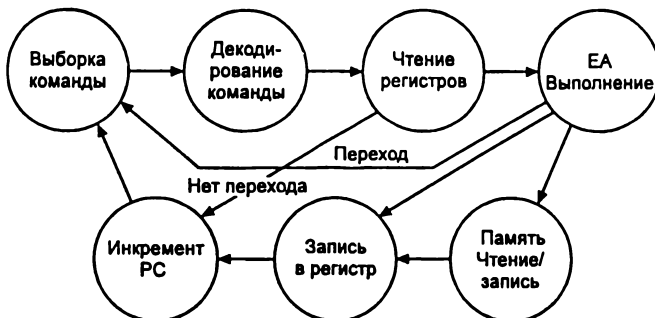


Рис. 5.3. Диаграмма состояний процессора. РС – программный счетчик.

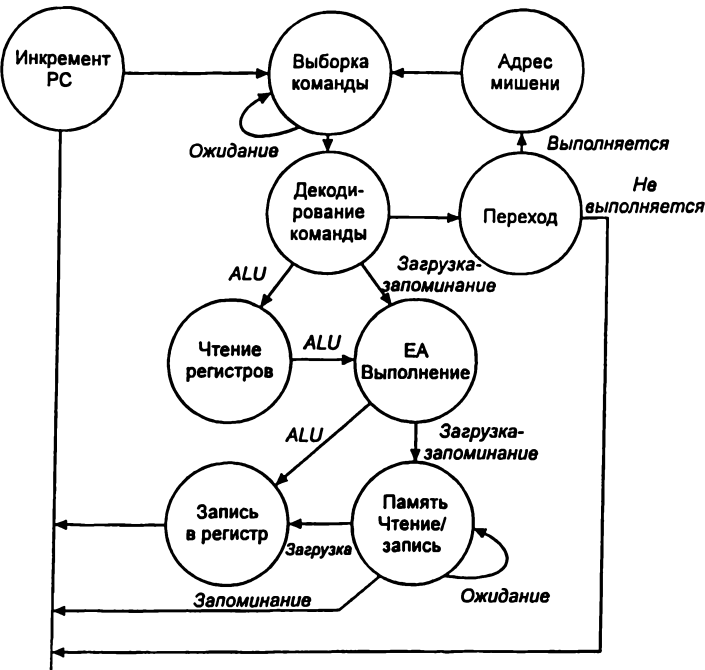


Рис. 5.4. Диаграмма состояний процесса интерпретации.

полученное командой загрузки. Наконец, выполняется инкремент программного счетчика, после чего весь процесс повторяется. В таблице 5.1 перечислены состояния из рис. 5.3 и классы команд, использующие эти состояния.

Таблица 5.1. Классы команд и использование состояний

	Выборка команды	Декоди- рование команды	Чтение регис- тров	ЕА выпол- нение	Память R/W	Запись в регистр	Инкремент РС
ALU	X	X	X	EX		X	X
Загрузка	X	X		EA	R	X	X
Запоминание	X	X	X	EA	W		X
Переход	X	X	X	EA			
Нет пере- хода	X	X	X				X

Диаграмма состояний рис. 5.3 повторена на рис. 5.4 с большими деталями фактического прохождения команд разных типов через диаграмму состояния. Обратите внимание на то, что все команды используют состояния выборки и декодирования команды. После состояния

декодирования пути команд расходятся в зависимости от их классов. Из-за того что выборка команды и чтение-запись памяти требуют обращения к памяти, при использовании медленной памяти могут потребоваться состояния ожидания. Если в процессоре имеется кэш-память, ожидание также требуется в случае кэш-промаха.

Команды переходов в тех случаях, когда переход выполняется, вычисляют адрес мишени (адрес перехода) и возвращаются к состоянию выборки команды. Если переход не выполняется, осуществляется инкремент программного счетчика и возврат к состоянию выборки команды.

Выделение в тракте данных путей передачи данных

Приведенные ниже 9 рисунков (рис. 5.5а – 5.5и) показывают пути, разрешенные в течение любого из состояний рис. 5.4. Пересылки между регистрами требуют одного такта. Длительность цикла памяти равна одному такту плюс требуемое число состояний ожидания. В некоторых случаях объединение состояний является произвольным; пересылка могла быть объединена с другой функцией. Пути пересылок показаны серым цветом; для каждой из 9 функций приведено число тактов.

Детализация диаграммы состояний и выполнение вторичных назначений состояний

Имея пути передачи данных, выделенные на рис. 5.5а – 5.5и, а также число тактов, требуемое для каждой функции, можно детализировать диаграмму состояний рис. 5.4 и представить ее в виде расширенной диаграммы, изображенной на рис. 5.6. Каждое состояние расширенной диаграммы занимает один такт. Состояние может потребовать большего числа шагов, только если из памяти приходит сигнал ожидания.

Получив диаграмму состояний, изображенную на рис. 5.6, мы можем выполнить вторичные назначения состояний. При 23 состояниях для однозначного определения каждого состояния требуется 5 триггеров⁵¹⁾. Вторичные назначения состояний показаны на рис. 5.6

⁵¹⁾ Некоторые проектировщики предпочитают резервировать несколько триггеров сверх необходимого минимума для обеспечения возможных расширений ISA. Допустимым вариантом системы будет также конечный автомат, в котором для каждого возможного состояния выделен отдельный триггер. Из этих триггеров в каждый момент времени будет установлен только один.

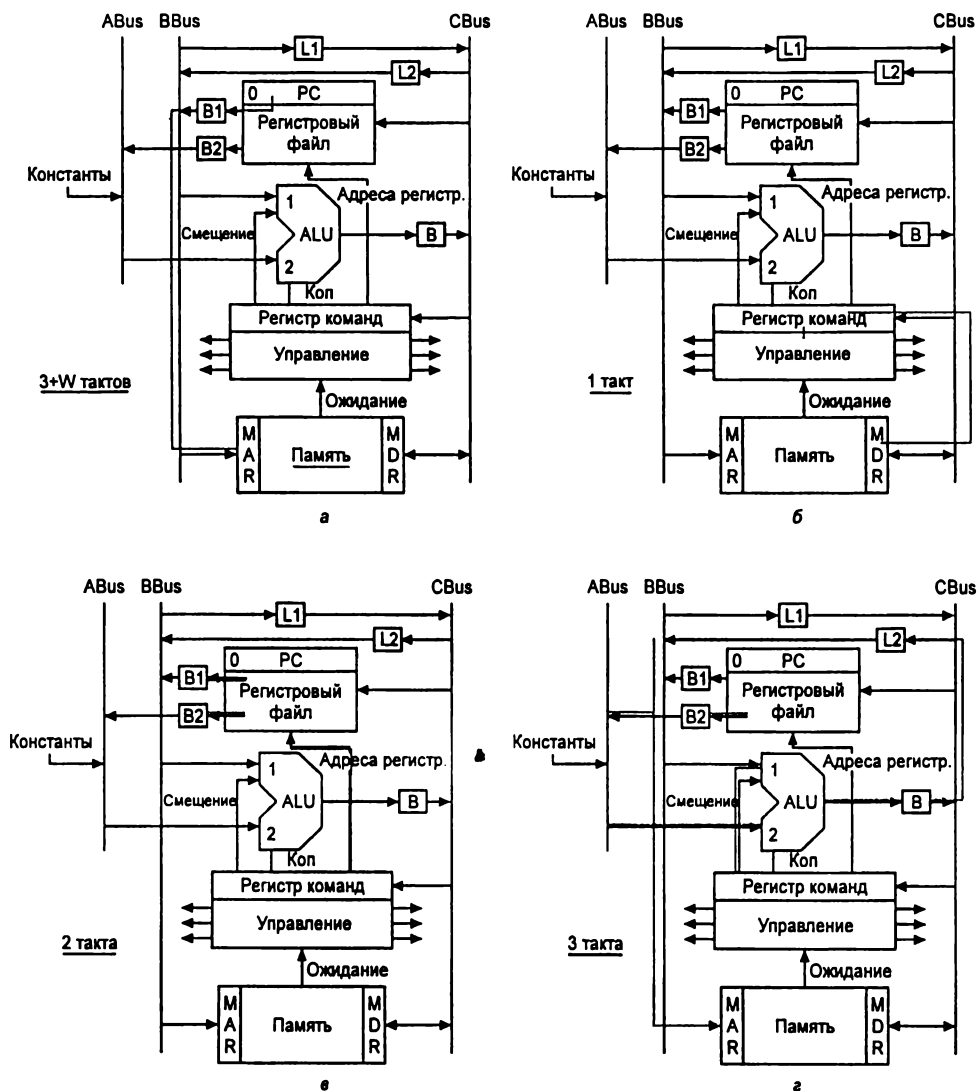
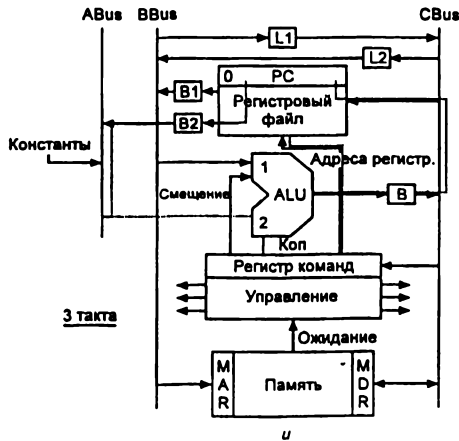
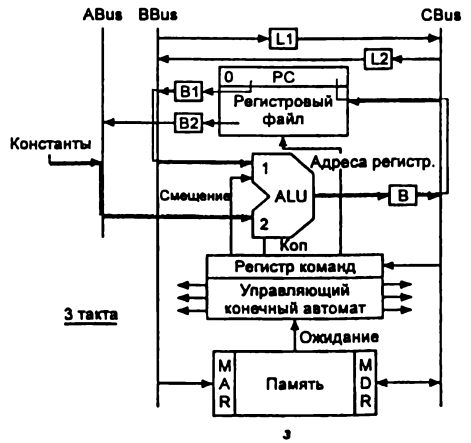
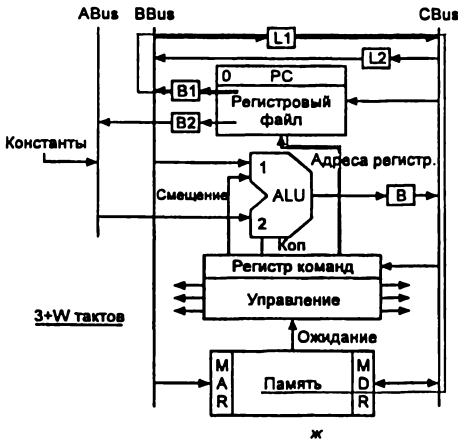
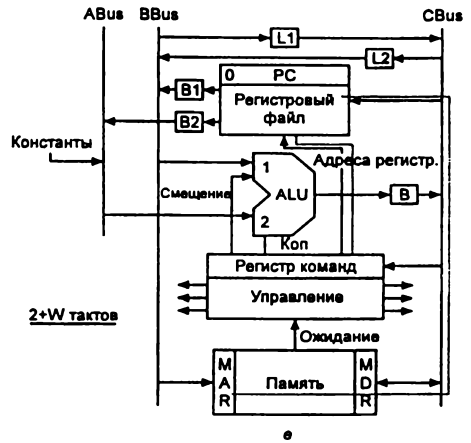
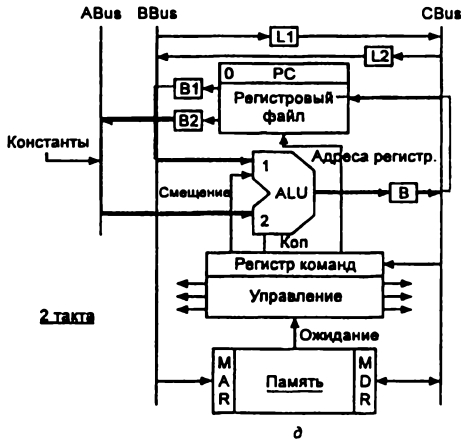


Рис. 5.5. *а* – выборка команды, *б* – декодирование команды, *в* – выбор регистра, *г* – эффективный адрес, *д* – выполнение, *е* – чтение памяти, загрузка, *ж* – запись памяти, запоминание, *з* – память, инкремент PC, *и* – адрес мишен-ни перехода.



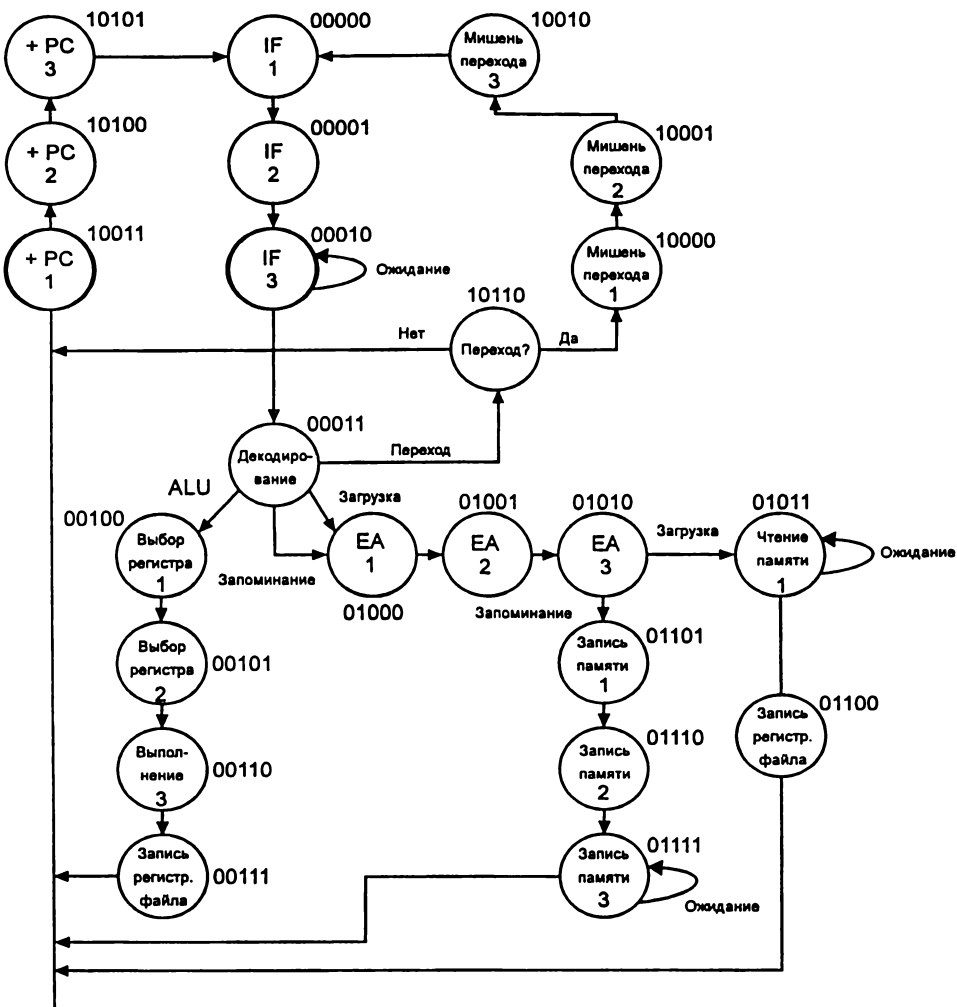


Рис. 5.6. Расширенная диаграмма состояний.

около каждого символа состояния. Проектировщик конечного автомата может обнаружить, что это конкретное вторичное назначение после его реализации не даст оптимальную по количеству логических элементов схему. Оно было выбрано безотносительно к сложности реализации.

Построение таблицы истинности переходов между состояниями

Таблица 5.2 представляет собой таблицу истинности переходов между состояниями. Предположим, что дешифратор кода операции

генерирует для различных типов команд четыре управляющих сигнала **A**, **L**, **S** и **B** помимо управляющих сигналов, посылаемых в ALU для выбора операции. Память генерирует сигнал ожидания **W**. Сигнал **?** генерируется знаковым битом регистра **R1**; если переход должен быть выполнен, **?** = 1. Безразличные состояния отмечены в таблице крестиком.

Таблица 5.2. Таблица истинности переходов между состояниями

Текущее состояние	Переход ?	Ожидание W	ALU A	Загрузка L	Запоминание S	Переход B	Следующее состояние
00000	x	x	x	x	x	x	00001
00001	x	x	x	x	x	x	00010
00010	x	1	x	x	x	x	00010
00010	x	0	x	x	x	x	00011
00011	x	x	1	x	x	x	00100
00011	x	1	x	1	x	x	01000
00011	x	x	x	x	1	x	01000
00011	x	x	x	x	x	1	10110
00100	x	x	x	x	x	x	00101
00101	x	x	1	x	x	x	00110
00011	x	x	x	x	x	x	00111
00111	x	x	x	x	x	x	10011
01000	x	x	x	x	x	x	01001
01001	x	x	x	x	x	x	01010
01010	x	x	x	1	x	x	01011
01010	x	x	x	x	1	x	01101
01011	x	1	x	x	x	x	01011
01010	x	0	x	x	1	x	01100
01100	x	x	x	x	x	x	10011
01101	x	x	x	x	x	x	01110
01110	x	x	x	x	x	x	01111
01111	x	1	x	x	x	x	01111
01111	x	0	x	x	x	x	10011
10000	x	x	x	x	x	x	10001
10001	x	x	x	x	x	x	10010
10010	x	x	x	x	x	x	00000
10011	x	x	x	x	x	x	10100
10100	x	x	x	x	x	x	10101
10101	x	x	x	x	x	x	00000
10110	1	x	x	x	x	x	10000
10110	0	x	x	x	x	x	10011

Составление уравнений для триггеров

На рисунке 5.7 изображен управляющий конечный автомат, схожий с тем, что был показан на рис. 1.1. Память автомата представляет собой 5-битовый регистр с входами ?, W, A, L, S и B вместе с текущим состоянием битов регистра. Автомат переходит из состояния в состояние, генерируя управляющие выходные сигналы, которые открывают логические элементы, стоящие в тракте данных.

Пятибитовый регистр состояний реализован на SR-триггерах, именуемых a, b, c, d и e и закрепленных за битами вторичных назначений состояний по порядку слева направо. Ниже приведено уравнение для триггера a в форме минтерма, как оно будет реализовано в программируемой логической матрице (PLA).

$$\begin{aligned} S_a &= a' b' c' d e B + a' b' c d e + a' b c d' e' + a' b c d e W', \\ R_a &= a b' c' d e' + a b' c d' e. \end{aligned}$$

Пример

На рисунке 5.6 показаны 23 состояния, в то время как таблица истинности переходов содержит 31 строку. Чем объясняется эта разница?

Решение

Таблица истинности переходов должна содержать одну строку для каждой выходящей стрелки на диаграмме состояний. Пересчитайте их: имеется 31 стрелка.

Мы видим, что реализация управляющего конечного автомата с помощью тракта данных приводит к тому, что команды разных классов занимают разное число тактов. Подсчитывая число тактов при движении по расширенной диаграмме состояний (рис. 5.6), необходимо

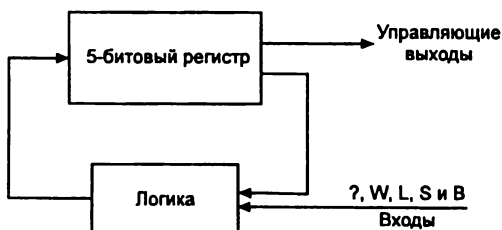


Рис. 5.7. Управляющий конечный автомат.

прибавить число тактов ожидания памяти. Заметьте, что логические пути для команд переходов одинаковы при наличии или отсутствии перехода.

ALU:	11 + W тактов
Переход:	8 + W тактов
Загрузка:	12 + 2W тактов
Запоминание:	13 + 2W тактов

✎ Пример

Пусть веса классов команд таковы: ALU — 0,30, переход — 0,30, загрузка — 0,25 и запоминание — 0,15. Пусть процессор имеет кэш, причем кэш-попадание не требует состояний ожидания, но кэш-промах требует 10 состояний ожидания. Значение P_{miss} для кэша составляет 0,1. Какова величина CPI этого процессора?

✍ Решение

Величина CPI процессора представляет собой взвешенное среднее число тактов для каждого класса команд. Число состояний ожидания определяется таким образом:

$$W = (1 - P_{miss}) \times 0 + P_{miss} \times 10 = 10P_{miss} = 10 \times 0,1 = 1 \text{ такт}$$

$$\begin{aligned} \text{CPI} &= \text{вес ALU} \times \text{такты ALU} + \text{вес перехода} \times \text{такты перехода} + \text{вес загрузки} \times \\ &\quad \times \text{такты загрузки} + \text{вес запоминания} \times \text{такты запоминания} = \\ &= 0,3 \times (11 + 1) + 0,3 \times (8 + 1) + 0,25 \times (12 + 2) + 0,15 \times (13 + 2) = \\ &= 3,6 + 2,7 + 3,5 + 2,25 = 12,05 \end{aligned}$$

✓ Замечание

Компьютеры с аппаратными управлением и трактом данных вроде изображенного на рис. 5.4 имеют значения CPI в диапазоне от 10 до 15. Наш пример оказался типичным для компьютеров такого рода. Заметьте, что уменьшение P_{miss} вдвое повысит CPI данного процессора приблизительно на 10%. Другой способ повышения производительности заключается в уменьшении числа тактов в логических путях. Так, удаление одного такта из состояния выборки команды повысит производительность приблизительно на 8%.

5.2. Микропрограммное управление

М. В. Уилкс первым высказал мысль о том, что наилучшим способом построения компьютера была бы замена аппаратно реализованного конечного автомата простым управляющим компьютером [4]. Этот управляющий компьютер имел бы очень простую систему команд (ISA), и его легко спроектировать и построить с минимальным числом ошибок. После этого система макрокоманд (макро-ISA) будет интерпретироваться программой, которую можно назвать микропрограммой, выполняемой на управляющем компьютере с системой микрокоманд (микро-ISA)⁵²⁾. На рисунке 5.8 воспроизведен рис. 3.1 с указанием двух уровней программной трансляции, используемых в системе с микропрограммным управлением⁵³⁾.

Исправление ошибок и внесение добавлений в макро-ISA выполнялось бы путем простых программных изменений вместо перекоммутации с помощью желтых проводов, как это практиковалось для аппаратных управляющих средств. Из-за высокой стоимости ПЗУ микропрограммирование не использовалось до выпуска в 1960-х годах семейства IBM S360 [3]. Все члены этого семейства, кроме наиболее производительных, содержали микропрограммные средства.

На рисунке 5.2 был показан тракт данных машины с управляющим конечным автоматом, реализованным с помощью аппаратных средств. На рисунке 5.9 показана та же машина, но с микропрограммным управлением. Относительно этого рисунка следует сделать ряд важных замечаний. В большинстве случаев процессор, содержащий микропрограммное управление, нетрудно распознать по наличию в нем двух программных счетчиков: микро-РС и макро-РС. Правда, в некоторых

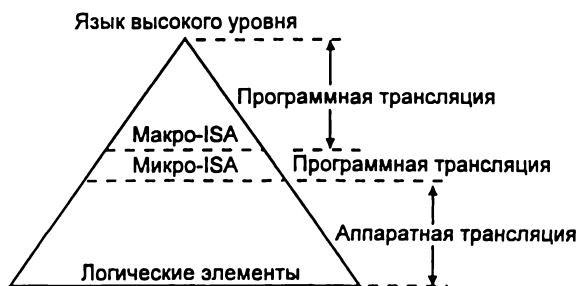


Рис. 5.8. Микропрограммные лексические уровни.

⁵²⁾ Термины макро-ISA и микро-ISA позволяют провести различие между двумя системами команд.

⁵³⁾ Программы, выполняемые на микро-ISA, иногда называются «защитными» программами (firmware), потому что они хранятся в ПЗУ и находятся как бы между программными и аппаратными средствами; отсюда слово firm (твердый).

Выборка командыТакт 1: $(R0) \rightarrow B1$ Такт 2: $(B1) \rightarrow MAR$ Такт 3 + ожидание: $(Mem(MAR)) \rightarrow MDR$ **Декодирование команды** $(MDR) \rightarrow IR$

Case op-code; Add: go to Add

Load: на загрузку

Store: на запоминание

Branch: на переход

Выбор регистра для операций ALU⁵⁴⁾Такт 1: $R(IR, RAs1) \rightarrow B1$ Такт 2: $R(IR, RAs2) \rightarrow B2$ **Эффективный адрес [база+смещение]**Такт 1: $R(IR, Rbase) \rightarrow B2$ Такт 2: $(B2) \rightarrow ALU2$ $(IR, \text{смещение}) \rightarrow ALU1$ Такт 3: $(B) \rightarrow CBus$ $(Cbus) \rightarrow BBus$ $(Bbus) \rightarrow MAR$ **Выполнение⁵⁵⁾**Такт 1: $(B1) \rightarrow ALU1$ $(B2) \rightarrow ALU2$ Такт 2: $(B) \rightarrow R(IR, Rdest)$ **Чтение памяти**Такт 1 + ожидание: $(Mem(MAR)) \rightarrow MDR$ Такт 2: $(MDR) \rightarrow R(IR, Rdest)$ **Запись в память**Такт 1: $(R(IR, RAs1)) \rightarrow B1$ Такт 2: $(B1) MDR \rightarrow BBus$ $(BBus) \rightarrow CBus$ $(CBus) \rightarrow MDR$ Такт 3 + ожидание: $(MDR) \rightarrow Mem(MAR)$ **Инкремент PC**Такт 1: $(R0) \rightarrow B1$ Такт 2: $(B1) \rightarrow ALU1$ $+1 ALU2$ Такт 3: $(B) \rightarrow R0$

⁵⁴⁾ Обозначение $(IR, RAs1)$ относится к полю RA_{s1} регистра IR .

⁵⁵⁾ Выходной буфер ALU неявно загружается каждой микрокомандой, использующей ALU.

Адрес мишени перехода (относительно PC)

Такт 1: (R0) → B2

Такт 2: (B2) → ABus

(ABus) → ALU2

(IR, смещение) → ALU1

Такт 3: (B) → R0

Определив пути пересылок, мы должны выбрать один из двух вариантов проектирования микрокоманд(ы) move:

1. Можно составить таблицу различных микрокоманд пересылок в соответствии с приведенным выше списком, например (R0) → B2, и предусмотреть для каждой отдельный код операции. Поскольку различных операций у нас 19, для кода операции хватит 5 бит.

2. Можно использовать однобитовый код операции для двух микрокоманд move (пересылка) и branch (переход), и предусмотреть в команде поля источника и приемника. Такой ортогональный подход потребует включения в микрокоманду большего числа битов, но предоставит максимум гибкости для расширений и исправлений.

В настоящем примере мы используем второй, ортогональный подход. Перечислим источники и приемники, требуемые для команд пересылки (см. табл. 5.3). Согласно схеме тракта данных, для шин не предусмотрены регистры хранения; пересылка по шине осуществляется путем инициализации портов шины для приемника и источника.

Таблица 5.3. Источники и приемники

Источники	Приемники
R0	R0
B1	B1
B2	B2
B	MAR
R(IR, RAs1)	ALU1
R(IR, RAs2)	ALU2
R(IR, база)	R(IR, Rdest)
Mem(MAR)	Mem(MAR)
MDR	MDR
IR, смещение	IR
Поле константы	

Например, для пересылки содержимого B1 по шине BBus в ALU1 используется микрокоманда Move, B1, ALU1. Заметьте, что отдельные поля регистра IR задаются явным образом в тех случаях, когда поле используется в качестве источника, например, микрокоманда Move, IR, RAs2, B1 пересылает поле RAs2 регистра макрокоманды в B1. Поле

кода операции макрокоманды аппаратно подсоединено к ALU и активизируется вместе с ALU, когда на входе ALU действуют достоверные сигналы.

Микрокоманды переходов (branch). Микропрограмма должна иметь средства осуществления переходов. Посмотрим еще раз на рис. 5.6. Некоторые состояния, например состояние декодирования, имеют множественные выходы. Поэтому микро-ISA должна включать команды переходов, как условных, так и безусловных. В нашем примере предусмотрены две команды условного перехода и одна команда безусловного перехода, а также специальный переход по сигналу ожидания из памяти:

1. Условный переход анализирует знак регистра R1 и осуществляет переход в случае положительного содержимого.

2. Микрокоманда имеет также поле константы, в которое поступают коды операций макро-ISA. Это поле сравнивается с кодом операции макро-ISA, чтобы реализовать предложение выбора case для функции декодирования.

3. Безусловный переход необходим для возврата в начало последовательности микрокоманд.

4. Когда память генерирует сигнал ожидания, микропрограммный счетчик останавливается; он остается в своем текущем состоянии еще на один такт. Другими словами, микропрограмма осуществляет переход на текущий адрес. Символически этот переход можно описать так:

(PC)W → PC

Расширенные коды операций используются для однозначного определения трех типов переходов. Адрес мишени перехода представляет собой прямой адрес, назначенный микрокоманде. Прямая адресация выбрана для того, чтобы сэкономить время, которое потребовалось бы для прибавления к PC относительного смещения перехода, а также ввиду относительно малого числа битов, требуемых для описания адреса мишени перехода.

Формат микрокоманды. Каков формат микрокоманды? Поскольку мы решили реализовать лишь два типа команд (пересылки и перехода), код операции занимает 1 бит. Поля источника, приемника и константы уже обсуждались. Поле ALU/Мет содержит 2 бита, которые разрешают ALU выполнять операции, задаваемые кодом операции макро-ISA, и, кроме того, переключают циклы чтения и записи в памяти:

Коп	Источник	Приемник	Константа	ALU/Мет	Управление	Адрес перехода
-----	----------	----------	-----------	---------	------------	----------------

Поле управления задает различные функции, как, например, указание на то, что микрокоманда перехода является безусловной. Коды

в поле управления определяют три типа условных переходов, а также безусловный переход.

✎ Пример

Каким образом три функции в поле ALU/Мет могут быть заданы с помощью 2 бит?

✎ Решение

Задание функции можно выполнить разными способами; одно из возможных решений таково:

- 00 Разрешить ALU
- 01 Чтение памяти
- 10 Запись памяти

Микропрограмма. Теперь мы можем запрограммировать интерпретатор макро-ISA на языке микро-ISA. На рисунке 5.10 приведена программа для макрокоманд ALU и перехода. Эта микропрограмма соответствует диаграмме состояний рис. 5.6. Вспомним, что первым делом следует выбрать макрокоманду. Эта задача выполняется микрокомандами в ячейках 0–3.

Вслед за выборкой макрокоманды с помощью последовательности из четырех команд микропереходов выполняется декодирование кода макрооперации. Адреса переходов этих четырех команд определяют наборы микрокоманд для четырех типов макрокоманд. После полной отладки микропрограммы она может быть оптимизирована с целью сокращения объема занимаемой ею памяти.

Макрокоманды типа ALU реализованы в ячейках 10–14; микрокомандами в ячейках 15–18 выполняется инкремент макропрограммного счетчика. Безусловный переход возвращает микропрограммный счетчик к ячейке 0, с которой начинается выборка следующей макрокоманды.

Макрокоманда перехода начинается в ячейке 40, где анализируется знаковый бит регистра R1. Если знак отрицателен, микрокомандами в ячейках 41–44 инкрементируется программный счетчик. Микрокоманда в ячейке 45 осуществляет безусловный переход на ячейку 0, с которой начинается выборка следующей макрокоманды.

Из рисунка 5.10 можно определить CPI двух реализованных типов макрокоманд. Команда ALU требует $15 + W$ тактов; команды переходов требуют $14 + W$ тактов как в случае выполнения перехода, так и в случае его пропуска. Производительность процессора оказывается

#	Коп	Источник	Приемник	Константа	ALU/Мет	Управление	Адрес перехода	Комментарий
0	1	RO	B1					Выборка команды
1	1	B1	MAR					"
2	1	Mem/(MAR)	MDR		М чтение			"
3	1	MDR	IR					"
4	0			ALU		Сравнение	10	Декодирование
5	0			Загрузка		Сравнение	20	"
6	0			Хранение		Сравнение	30	"
7	0			Переход		Сравнение	40	"
8								
9								
10	1	R(IR, RAs1)	B1					ALU
11	1	R(IR, RAs2)	B2					"
12	1	B1	ALU1					"
13	1	B2	ALU2		ALU			"
14	1	B	R(IR, RAdest)					"
15	1	RO	B1					Инкремент PC
16	1	B1	ALU1					"
17			ALU2	+1				"
18	1	B	RO					"
19	0					Безусловн.	0	Возврат на выборку команды
.								
40	0					Анализ знака	46	Переход
41	1	RO	B1					Инкремент PC
42	1	B1	ALU1					"
43	1		ALU2	+1				"
44	1	B	RO					"
45	0					Безусловн.	0	Возврат на выборку команды
46	1	RO	B1					Мишень перехода
47	1	B1	ALU2					"
48	1	IR, RAdisp	ALU1					"
49	1	B	MAR					"
50	0					Безусловн.	0	Возврат на выборку команды
51								

Рис. 5.10. Микропрограммы ALU и перехода.

ниже, чем в случае аппаратной реализации, которая давала значения CPI для команд ALU 11 + W тактов, а для команд переходов 8 + W тактов.

Проанализируем причины потери производительности. Этот анализ покажет, какие именно элементы нашей разработки можно изменить с целью увеличения производительности микропрограммного управления.

1. Функция декодирования при аппаратной реализации конечного автомата выполняется за один такт. Если микрокоманда ISA реализу-

ется с помощью 4-входовой команды перехода, декодирование может быть выполнено тоже за один такт.

Переход	Услов. 1	Миш. 1	Услов. 2	Миш. 2	Услов.3	Миш. 3	Услов. 4	Миш. 4
---------	----------	--------	----------	--------	---------	--------	----------	--------

2. Для загрузки MAR в микропрограммной последовательности выборки требуются два такта (две микрокоманды). Линия передачи непосредственно из регистрового файла к регистру MAR устранил этот добавочный такт.

3. Для загрузки буферов В1 и В2 регистрового файла для макрокоманд ALU требуются два такта (две микрокоманды). В аппаратно реализованном конечном автомате эта задача выполняется за один такт, поскольку используется двухпортовый регистровый файл. Для того чтобы сэкономить один такт в микропрограммном контроллере, требуется двухпортовый регистровый файл, а также дополнительные поля источника и приемника в формате микрокоманды.

4. В конце каждой последовательности требуется явный безусловный переход для возврата на начало последовательности макрокоманды выборки. В аппаратно реализованном автомате в этом такте нет необходимости. Устранить такт возврата можно путем придания каждой команде, в дополнение к ее основной функции, также и средств перехода⁵⁶⁾.

Приведенный выше формат микрокоманды допускает параллельное выполнение нескольких простых функций. Увеличение степени параллелизма в тракте данных, при котором большее число функций имеет возможность выполняться одновременно, будет способствовать росту производительности процессора. При увеличении степени параллелизма в тракте данных возрастает и длина микрокоманды. Не является чем-то необычным макрокоманда длиной 100 бит и более.

Про микропрограммируемые процессоры с короткими микрокомандами говорят, что в них реализовано вертикальное микропрограммирование, потому что для выполнения функции требуется последовательность микрокоманд. Микропрограммируемые процессоры с длинными микрокомандами относятся к категории процессоров с горизонтальным микропрограммированием, так как в них одна микрокоманда может выполнять за один такт несколько функций.

✎ Пример

Частота использования макрокоманд по классам такова: ALU — 0,30, переходы — 0,30, загрузка — 0,25, запоминание — 0,15. Каково

⁵⁶⁾ Популярный микропроцессор ARM использует этот способ на уровне макро-ISA.

взвешенное среднее число микрокоманд, требуемых для выполнения функции декодирования?

Решение

Из рисунка 5.10 видно, что вслед за проверкой команды на принадлежность ее классу ALU следует проверка на классы загрузки, запоминания и перехода. Таким образом, взвешенное среднее число микрокоманд равно

$$\begin{aligned}\text{взвешенное среднее} &= 0,3 \times 1 + 0,25 \times 2 + 0,15 \times 3 + 0,3 \times 4 = \\ &= 0,3 + 0,5 + 0,45 + 1,2 = \\ &= 2,45 \text{ микрокоманды}\end{aligned}$$

✓ Замечание

Использование четырехходовой микрокоманды перехода снизит среднее взвешенное CPI процессора почти на 1,5 такта, или приблизительно на 10%.

Размер микропрограммы

Здесь уместно сделать несколько замечаний по поводу размера микропрограммы. Реализация ISA типа, например, IBM S360, требует описания для каждой макрокоманды несколько шагов (микрокоманд). Сумма этих шагов дает число микрокоманд, которые необходимо записать в ПЗУ для данного варианта ISA. Анализ показывает, что при реализации того или иного варианта ISA число требуемых битов (а не число требуемых шагов) остается постоянным. Это означает, что

$$\text{биты} \times \text{слова} [\text{горизонтальная организация}] \approx \text{биты} \times \text{слова} [\text{вертикальная организация}]$$

В таблице 5.4 приведены данные по организации микропрограммной памяти для четырех моделей IBM S370 [2]. Средний объем памяти для всех этих моделей составляет 400 Кбит. Из этих данных следует, что проектировщик может искать компромисс между сложностью тракта данных (что более или менее соответствует одновременности операций) и производительностью, не боясь существенного изменения числа битов в микропрограмме. Организация ПЗУ, разумеется, изменится, но число битов в нем останется приблизительно тем же.

Пример

Мы хотим оценить затраты на микропрограммирование по сравнению с аппаратной реализацией управляющего узла. Микропрограммирование имеет постоянную составляющую затрат в 1000 единиц площади кристалла и переменную составляющую в 1 единицу площа-

Таблица 5.4. Микропрограммная память IBM S370

Модель	Слова	Бит/слово	Полное число битов
135	24 К	16	384 К
145	12 К	32	384 К
155	6 К	69	414 К
165-II	4 К	105	420 К
		Среднее	400 К

ди кристалла для сохранения в ПЗУ одной микрокоманды. Аппаратное управление требует затрат в 5 единиц площади кристалла на каждую команду. При каком числе команд затраты совпадут?

Решение

Стоимость микропрограммного управления при реализации n команд равна

$$\text{стоимость (M)} = 1000 + n$$

Стоимость аппаратной реализации равна

$$\text{стоимость (H)} = 5n$$

Приравняем эти две величины и решим относительно n :

$$\text{стоимость (M)} = \text{стоимость (H)}$$

$$1000 + n = 5n$$

$$n = 1000/4 = 250 \text{ команд}$$

✓ Замечание

Эти расчеты позволяют предположить, что выбор микропрограммного управления для процессоров типа CISC с богатым набором команд, например, семейства IBM S360, определяется лишь вкусами проектировщиков. Однако простые ISA с малым числом команд дешевле реализовывать аппаратно.

Литература

1. Bell, C. G. and Newell, A. (1971). *Computer Structures: Readings and Examples*, McGraw-Hill, New York.
2. Case, R.P. and Padegs, A. (1978). «Architecture of the IBM System/370», *Communication of the ACM*, 21:1, p.p. 73–96.
3. Hellerman, H. (1967). *Digital Computer System Principles*, McGraw-Hill, New York.
4. Wilkes, M. V. (1951). «The Best Way to Design an Automatic Calculating Machine,» *Manchester University Computer Inaugural Conference*, published by Ferranti Ltd., London.

Упражнения

5.1. Для приведенного выше примера аппаратной реализации управляющего конечного автомата напишите входные уравнения для схем, использующих Т- и JK-триггеры.

5.2. Для того же примера примите, что при удлинении такта на 10% состояния инкремента РС можно включить в состояния выборки команды, а состояние адреса мишени — в состояние «ЕА Выполнение».

А. Изобразите диаграмму переходов состояний.

Б. Получите входные уравнения для JK-триггеров.

В. Каково повышение производительности этой реализации по сравнению со схемой с восемью состояниями?

5.3. Объясните, почему в самой быстрой модели IBM S360 было использовано аппаратное управление, а в менее производительных моделях — микропрограммное.

5.4. Альтернативным способом аппаратной реализации конечного автомата является использование по триггеру на каждое состояние вместо кодирования состояний, как это было сделано в рассмотренном выше примере. Напишите входные уравнения для такой реализации, использующей Т-триггеры.

5.5. Поработайте в библиотеке и найдите доказательства того, что число битов микропрограммы для заданной ISA приблизительно постоянно и не зависит от скорости процессора. Исходная информация для машин IBM S360 и DEC VAX имеется в технической литературе, ее только надо найти.

5.6. Напишите микропрограмму для макрокоманды загрузки регистра.

5.7. Большинство макрокоманд требует для своей реализации несколько микрокоманд. Что произойдет с микропроцессором, если в середине процесса выполнения команды возникнет внешнее прерывание? Имеются ли варианты реакции на такое событие?

5.8. Некоторые микропрограммные управляющие узлы включают средства вызова микроподпрограмм. Какие изменения следует внести в рис. 5.9, чтобы предоставить такую возможность? Вспомните наше обсуждение условий использования подпрограмм в гл. 1 и 3.

5.9. Для диаграммы состояний, изображенной на рис. 5.1, предложите внутреннюю регистровую организацию процессора (аккумулятор, регистровый файл, стек и т. д.).

5.10. Таблица 5.3 — это таблица истинности переходов между состояниями для диаграммы состояний, изображенной на рис. 5.6. Вместо того чтобы кодировать состояния с помощью пяти триггеров, используйте по одному JK-триггеру на состояние (всего 23 триггера). Напишите уравнения для J- и K-входов триггеров такого управляющего конечного автомата.

5.11. Напишите логические уравнения для триггеров b, c, d и e с учетом таблицы истинности переходов между состояниями (табл. 5.2).

5.12. Что представляют собой операции, типы данных и адреса для микро-ISA?

5.13. Покажите, как можно преобразовать микропрограмму рис. 5.10, используя общий код для инкрементирования программного счетчика.

5.14. Спроектируйте конечный автомат для управления фон-неймановской ISA, описанной в гл. 1.

5.15. Объясните, каким образом можно спроектировать микропрограммный процессор без макропрограммного счетчика или без микропрограммных счетчиков, используя связанные списки.

5.16. Отвечает ли узел микропрограммного управления, изображенный на рис. 5.9, фон-неймановской или гарвардской архитектуре? Обоснуйте ваш ответ.

5.17. Может ли узел микропрограммного управления, изображенный на рис. 5.9, выполнять подпрограммы? Обоснуйте ваш ответ.

ГЛАВА 6

КОНВЕЙЕРНЫЕ ПРОЦЕССОРЫ

6.0. Введение

В главе 5 описывалось проектирование процессоров с последовательным выполнением команд и либо аппаратным, либо микропрограммным управлением. Настоящая глава будет посвящена конвейеризации обработки данных. Конвейеризация обязана своим появлением желанию усовершенствовать технологию производства. Обычно идею внедрения в производство конвейера приписывают пионеру автостроения Генри Форду, который организовал конвейерную сборку автомобилей Model T. Говорят, что Форд посетил мясоперерабатывающую фабрику в Чикаго и наблюдал там разделку коровьих туш. Он будто бы заметил, что если что-то можно последовательно разбирать на части, то можно делать это и в обратном порядке. Так зародилась идея сборочного конвейера.

Конвейерный процессор содержит функциональные узлы для каждой операции, выполняемой в процессе интерпретации команды. В некоторых случаях такие функциональные узлы выполняют более одной операции. На конвейер подаются команды, а на выходе появляются результаты. Транзитное время (запаздывание) для конвейера может быть значительным, однако в большинстве случаев более важной характеристикой конвейера является скорость получения результатов на выходе. Это, кстати, справедливо для конвейерной системы подачи в ваш дом питьевой воды. Для вас не имеет значения, сколько миль или дней понадобилось воде, чтобы от озера дойти до вашего дома; единственное, что вас интересует, это скорость потока воды.

Базовый процессор с конвейеризацией

Как уже обсуждалось в гл. 5 и было показано на рис. 5.4, для интерпретации команды ISA типа загрузки-запоминания требуется выполнить следующие шаги:

- 1) выбрать команду,
- 2) декодировать команду,
- 3) прочитать регистры,
- 4) выполнить операцию,
- 5) вычислить эффективные адреса данных и мишени перехода,
- 6) обратиться к памяти по записи или считыванию,

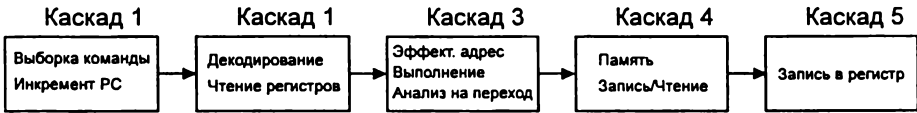


Рис. 6.1. Базовый конвейер.

- 7) выполнить запись в регистр,
- 8) выполнить анализ на переход,
- 9) инкрементировать программный счетчик.

Эти функции можно сгруппировать по конвейерным ступеням, или каскадам, как это показано на рис. 6.1, где изображен конвейер, схожий с используемым в процессоре MIPS R2000. Мы выбрали этот конкретный конвейер благодаря его простоте; более современные конвейеры, используемые в процессорах Intel Pentium Pro или IBM S390, весьма сложны и не так удобны для начального изучения предмета. Будем называть приведенный на рис. 6.1 конвейер базовым.

Каскад 1 выполняет выборку команды и инкремент программного счетчика, подготавливая его для выборки следующей команды. Каскад 2 также выполняет две функции: декодирование команды и выборку операндов-источников из регистрового файла. Каскад 3 выполняет одну из трех функций: вычисление эффективного адреса памяти для команд загрузки, запоминания или перехода; выполнение целочисленной операции; выполнение анализа на переход. Каскад 4 выполняет чтение из памяти или запись в нее, если команда принадлежит к числу команд чтения или запоминания. Каскад 5 записывает в регистровый файл результаты целочисленной операции или выход команды загрузки.

Обратите внимание на то, что 9 функций разделяются на 5 каскадов. Такое разделение возможно потому, что некоторые функции не являются взаимозависимыми (они могут выполняться в одно и то же время); например, вполне можно выбрать команду и в то же время инкрементировать программный счетчик. Кроме того, некоторые функции могут не использоваться в конкретных командах. Другой пример: вычисление эффективного адреса не требуется для команды сложения регистров, поэтому эти функции могут использовать один и тот же арифметический узел.

6.1. Модели производительности

Как уже отмечалось во введении, скорость потока команд, проходящих через процессор, определяет скорость выполнения программы. Скорость выполнения обычно измеряется числом команд в единицу



Рис. 6.2. Таблица резервирования по каскадам (с любезного согласия Jones and Bartlett Publishers).



Рис. 6.3. Таблица резервирования по командам (с любезного согласия Jones and Bartlett Publishers).

времени. Поскольку такты процессора следуют друг за другом с фиксированной частотой, скорость выполнения может также измеряться числом команд на такт (IPC) или обратной величиной, числом тактов на команду (CPI). Если команды проходят через конвейер без задержки, то и IPC, и CPI равны 1⁵⁷⁾. Однако имеются события, которые могут вызвать задержки в работе конвейера, и величина CPI определяется следующим образом:

$$CPI = 1 + \text{задержки}$$

Важнейшей задачей проектировщика конвейерного процессора является максимально возможное снижение задержек. В разделе 6.3 этой главы рассматриваются источники задержек и методы борьбы с ними.

Составление модели производительности конвейера осложняется тем, что действия конвейера происходят во времени и пространстве. Анализ производительности таких систем удобнее всего выполнять с помощью таблиц резервирования [2].

Две наиболее употребительные формы таблиц резервирования приведены на рис. 6.2 и 6.3. На рисунке 6.2 столбцы отображают такты времени, а строки — такты конвейера. Вы можете проследить за движением команды по конвейеру. Например, команда I1 проходит каскад 1 в момент t1 и доходит до последнего каскада S4 в момент t4. В момент t4 в конвейере находятся четыре команды: I1, I2, I3 и I4. Ва-

⁵⁷⁾ Суперскалярные процессоры, обсуждаемые в разделе 6.5, могут иметь в стационарном режиме CPI меньше 1.

		Время →							
		1	2	3	4	5	6	7	8
Каскады	1	I1	I2	I3	I4				
	2		I1	I2	I3	I4			
	3			I1	I2	I3	I4		
	4				I1	I2	I3	I4	
		s				$k - 1$			

Рис. 6.4. Таблица резервирования модели конвейера.

риацию такой таблицы можно найти в технических описаниях, где она используется для иллюстрации активности интерфейсов памяти, шины и проч.; вместо каскадов в таблице могут быть изображены различные сигнальные линии или характеристики аппаратуры.

В другой форме таблицы резервирования столбцы также соответствуют тактам времени, но строки — командам, как это показано на рис. 6.3. Здесь для того, чтобы проследить ход выполнения команды, надо двигаться по горизонтали. В момент времени t_3 команда I3 проходит каскад 1, затем в момент t_4 каскад 2 и т. д. Такая форма таблицы резервирования используется в технических описаниях для показа прохождения команд по конвейру.

Первая форма таблицы резервирования удобнее по ряду причин. Она привычнее для инженеров-электронщиков, поскольку напоминает изображение на экране логического анализатора. Кроме того, при использовании второй формы возникают трудности графического характера: по мере увеличения числа команд вторая таблица растет в двух измерениях, в то время как первая — только в одном.

На основе таблиц резервирования мы можем построить базовую модель производительности конвейера (рис. 6.4). Последовательность из четырех команд прерывается некоторым событием, например переходом или задержкой иного рода. Для общности положим, что длина последовательности равна k , а число каскадов в конвейере — s . Тогда число тактов, необходимых для выполнения этой последовательности, равно

$$s + k - 1,$$

откуда

$$CPI = \frac{s + k - 1}{k} = 1 + \frac{s - 1}{k} = 1 + \frac{1}{k}(s - 1).$$

Обратите внимание на то, что полученный нами ответ согласуется с приведенной ранее формулой: $CPI = 1 + \text{задержки}$. Проанали-

зировав предельные значения модели, мы получим представление о возможном диапазоне производительности конвейера:

При $k \rightarrow \infty$ $CPI \rightarrow 1$

При $k \rightarrow 1$ $CPI \rightarrow s$

Отсюда $1 \leq CPI \leq s$

Таким образом, значение CPI в худшем случае равно s (что может говорить о недостаточной длине конвейеров), а в лучшем случае 1, если непрерывная последовательность команд имеет достаточную длину.

Процессору, соответствующему последовательной модели выполнения (будем называть его последовательным процессором), требуется несколько шагов для завершения одной команды, после чего он может приступить к следующей. Конвейер из s каскадов функционально эквивалентен последовательному процессору, требующему s шагов⁵⁸). Таким образом, повышение быстродействия конвейерного процессора по сравнению с последовательным составляет

$$\text{повышение быстродействия} = \frac{\text{число тактов (последовательный)}}{\text{число тактов (конвейерный)}} = \frac{sk}{s+k-1} = \frac{s}{1+\frac{s-1}{k}}$$

При $k \rightarrow \infty$ повышение быстродействия $\rightarrow s$

При $k \rightarrow 1$ повышение быстродействия $\rightarrow 1$

Отсюда $1 \leq \text{повышение быстродействия} \leq s$

Из этой модели повышения быстродействия мы можем заключить, что эффективнее работает конвейер с большим числом каскадов. Однако ранее мы получили противоположный результат. Так какова же должна быть длина конвейера?

6.2. Разбиение конвейера

Предложенные выше модели, похоже, дают противоположные результаты в отношении оптимального числа каскадов в конвейере. Чтобы разрешить это противоречие, мы будем делить конвейер на все большее и большее число каскадов и смотреть, как изменяется его производительность. Если нам удастся найти оптимальное число каскадов, мы решим поставленную задачу.

⁵⁸) Легко видеть, что среднее число состояний, обозначенных в табл. 5.1, составляет 5,2. Это близко к числу каскадов в базовом конвейере, изображенном на рис. 6.1.

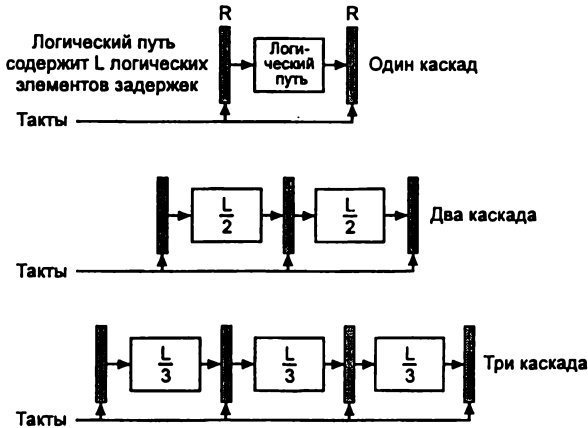


Рис. 6.5. Разбиение конвейера.

Начнем с однокаскадного конвейера, как показано на рис. 6.5. Если между двумя регистрами расположено достаточное количество логических уровней, полная интерпретация команды может быть выполнена за один такт. Тактовый период будет большим, однако интерпретация не потребует более одного такта. Предположим теперь, что логический путь может быть разделен на два каскада. Поскольку длина логического пути уменьшилась теперь до $L/2$ логических элементов, тактовый период может быть уменьшен в два раза, и мы получаем двухкаскадный конвейер с дополнительным регистром. Далее конвейер можно разделять на большее число каскадов (на рис. 6.5 показаны три каскада). В общем случае разделим конвейер на s каскадов. Каково будет значение s , дающее максимальную производительность конвейера?

Время выполнения последовательности команд определяется, как число тактов, умноженное на тактовый период, и равно $(s + k - 1)$ тактовых периодов. Какова величина тактового периода? В первом приближении тактовый период (в задержках логических элементов) равен L/s , поскольку один каскад длиной L был разделен на s каскадов⁵⁹. Однако дело обстоит не так просто.

Ради повышения производительности в качестве регистров используются защелки-фиксаторы, а не запоминающие элементы на базе JK-триггеров. Использование фиксаторов приводит к ряду ограничений. Во-первых, все логические пути в каскаде должны быть равной длины (с равной задержкой), чтобы логические входы поступали на все фиксаторы регистра в одно и то же время. Во-вторых, тактовые сигналы

⁵⁹ Разделение L на равные части не всегда может оказаться возможным; в этих случаях, чтобы обеспечить правильную синхронизацию, к коротким каскадам придется добавлять холостые логические каскады.

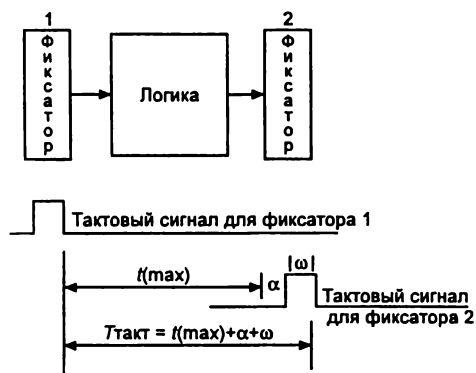


Рис. 6.6. Тактовые сигналы с перекосом и временем установления.

тактовые сигналы должны подаваться на все фиксаторы регистров. Из-за сложности разводки монтажа на кристалле линии, подводящие сигналы к фиксаторам регистров, могут давать разную задержку, выражающуюся в перекосе сигнала [4]⁶⁰. Наконец, фиксаторам свойственно определенное время установления состояния, которое должно учитываться при выборе периода тактовых сигналов.

На рисунке 6.6 показан логический путь между двумя фиксаторами. Тактовые сигналы распределяются от одного источника, расположенного близко от фиксатора 1. Задержка логики L/s вместе с задержкой выходного устройства составляет $t(\max)$. Длительность тактового сигнала для фиксатора 2 должна быть достаточно большой, чтобы перекрыть время установления фиксатора; это время обозначим ω . В силу задержек в линиях, подводящих тактовый сигналы, может возникнуть время перекоса α между сигналами на фиксаторах 1 и 2. Поэтому минимальный период тактовых сигналов определится как

$$T(\text{такт}, \min) = t(\max) + \alpha + \omega,$$

$$\text{где } \omega = L/s$$

Сумму перекоса сигнала α и времени установления фиксатора ω обозначим τ . Тогда время выполнения для многокаскадного конвейера будет равно

$$(s + k - 1) \left(\frac{L}{s} + \tau \right)$$

⁶⁰) Насколько известно автору, впервые проблема перекоса тактовых сигналов возникла при разработке компьютера Mark I в 1942 г. Электромеханические шаговые переключатели приводились в движение от общего вала с независимыми зубчатыми передачами. Деление одного оборота вала на такты должно было учитывать люфты в передающих шестернях и задержки в активизации сцепляющих муфт.

Чтобы найти оптимальное число каскадов конвейера, вычислим производную от времени выполнения по переменной s , приравняем ее нулю и решим полученное уравнение. Решение и будет представлять собой оптимальное значение s .

$$s_{opt} = \sqrt{\frac{L(k-1)}{\tau}}$$

Если значение может быть снижено, оптимальное число каскадов конвейера увеличится, что приведет к росту производительности из-за снижения времени выполнения для такого конвейерного процессора.

✎ Пример

Начертим графики зависимости времени выполнения от числа каскадов s при двух значениях времени $\tau = 2$ и 4 для процессора со следующими характеристиками: $L = 128$, $k = 4$. Значение s будем изменять от 1 до 128.

✎ Решение

Решение легко найти, подставив указанные параметры в полученное выше уравнение. На рисунке 6.7 по оси абсцисс отложен двоичный логарифм числа каскадов.

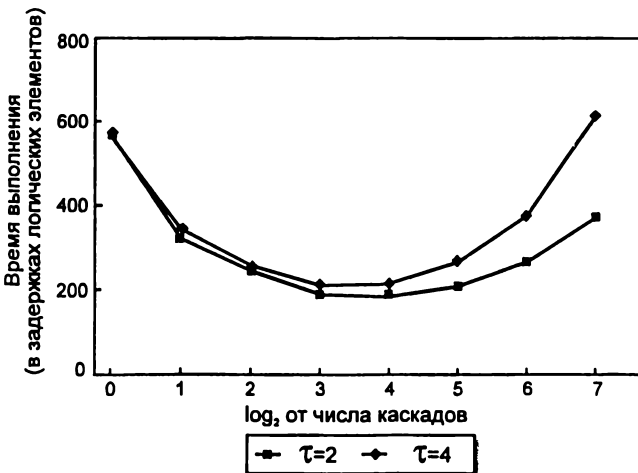


Рис. 6.7. Оптимальное число каскадов конвейера.

✓ **Замечание**

Из рисунка 6.7 видно, что оптимальное число каскадов для $\tau = 2$ приблизительно равно 16. Кривая имеет пологий характер, поэтому точное значение минимума лежит за пределами точности этой модели. Ранние RISC-процессоры использовали конвейеры с 4–5 каскадами; их называют процессорами с недостаточной конвейеризацией, поскольку число каскадов меньше оптимального, обеспечивающего максимум производительности. В большинстве современных конвейерных процессоров конвейеры имеют 10–16 каскадов. На рис. 6.7 проиллюстрировано также снижение времени выполнения при уменьшении значения τ от 4 до 2. Уменьшение перекося сигналов является на сегодня первостепенной задачей проектировщика очень больших интегральных схем.

Другое соображение, которое приходится принимать в расчет при делении конвейера на каскады, заключается в нахождении оптимального логического уровня каскадов. Каскад 3 базового конвейера выполняет две функции: декодирования и чтения регистров. Другие конвейеры, например конвейер целочисленных команд Intel Pentium, декодируют в одном каскаде, а выбирают из регистров операнды в другом — в каскаде выполнения. Проектировщик имеет большую широту выбора в организации каскадов. Его важнейшей задачей является максимально возможное уравнивание логических путей всех каскадов.

Поскольку все регистры синхронизируются от единого источника тактовых сигналов, тактовый период определяется как сумма задержек логики отдельных каскадов и времени τ :

$$\text{тактовый период} = \text{MAX}[(L1 + \tau), (L2 + \tau), \dots, (L3 + \tau)]$$

✎ **Примеры**

1. Процессор содержит конвейер, состоящий из 5 каскадов. Число логических уровней каскадов составляет 8, 13, 10, 11 и 12; τ равно двум задержкам логических элементов. Каков должен быть тактовый период этого процессора (в задержках логических элементов)?

✎ **Решение**

Тактовый период равен

$\text{MAX} [(8+2), (13+2), (10+2), (11+2), (12+2)] = 15$ задержек логических элементов.

2. Взяв конвейер из предыдущего примера, проектировщик может разделить логический путь второго каскада на два каскада, включающих 6 и 7 задержек логических элементов. Если использовать модель базового конвейера с задержкой, равной $s - 1$, и длиной после-

довательности $k = 4$ команд, будет ли предложенное разделение конвейера разумным?

Решение

Новый тактовый период будет равен $\text{MAX}(10, 8, 9, 12, 13, 14) = 14$ задержек логических элементов. Время выполнения (в задержках логических элементов) для каждого варианта составит:

время выполнения = $(s + k - 1)$ тактовых периодов

время выполнения (первый вариант) = $(5 + 4 - 1) \times 15 = 120$ задержек логических элементов

время выполнения (второй вариант) = $(6 + 4 - 1) \times 14 = 126$ задержек логических элементов

✓ Замечание

Приведенный пример иллюстрирует изучение альтернативных вариантов с помощью аналитических моделей. Первый вариант несколько быстрее и поэтому предпочтительнее. Однако не следует забывать ограниченность этих моделей, поэтому при наличии существенных доводов в пользу того или иного варианта их надо принять во внимание.

6.3. Задержки в конвейере

Вернемся теперь к вопросу о задержках и о способах снижения их влияния. Используемые нами модели учитывали задержку $s - 1$ в конце каждой последовательности из k команд. Эта задержка соответствует худшему случаю и будет служить для нас отправной точкой для последующих обсуждений. Задержки в конвейерном процессоре возникают по следующим причинам.

1. Задержки памяти: время запаздывания при обращении к памяти превышает один такт. В гл. 3 обсуждался вопрос об управлении задержками памяти.

2. Задержки переходов: изменение хода программы с переходом на отдаленную команду.

3. Задержки структурных сбоев: процессор вынужден приостановиться, так как две или более команд конкурируют за один и тот же ресурс.

4. Задержки сбоев данных: команда приостанавливается ввиду наличия взаимозависимости данных в параллельно выполняемых командах.

5. Задержки прерываний: программа прерывается, и вызывается на выполнение обработчик прерывания; когда обработчик завершается, управление программным счетчиком возвращается в прерванную программу.

Как уже отмечалось выше, в стационарном режиме конвейер обрабатывает одну команду за такт. Однако к времени стационарного состояния следует добавить разнообразные задержки. Задержки имеют две характеристики: вероятность P_i того, что команда встретится с задержкой или породит ее, и значение D_i этой встретившейся или порожденной задержки. Ранее мы получили модель CPI для конвейера

$$CPI = 1 + \frac{1}{k}(s - 1)$$

В этой формуле $1/k$ есть вероятность того, что следствием выполнения команды будет задержка $s - 1$. В общем случае

$$CPI = 1 + \sum_{i=1}^n P_i D_i$$

Такая простая линейная зависимость преувеличивает влияние задержек на работу конвейера. Например, задержка кэш-памяти может скрыть задержку сбоя данных. Тем не менее эта модель в какой-то мере описывает влияние задержек, как поодиночке, так и в комбинации, на производительность конвейера. В общем эта модель дает наиболее неблагоприятную оценку CPI.

6.3.1. Задержки переходов

Команда условного перехода может привести к одному из двух действий: 1) переход не выполняется, и продолжается последовательное выполнение расположенных далее команд потока и 2) переход выполняется, и в результате изменения значения программного счетчика начинает выполняться другая последовательность команд. Этот результат условного перехода не может быть известен заранее. Если ход выполнения программы изменяется, то некоторое количество команд, которые к этому моменту частично выполнились, должны быть отброшены, что приведет к значительной задержке.

Для оценки стратегий условных переходов используются два параметра. Первый — это вероятность P_b того, что выполняемая команда является командой перехода. Второй представляет собой взвешенная средняя задержка при переходе (WABD от weighted average branch delay) для выполненных и невыполненных переходов. CPI процессора с задержками переходов определяется следующим образом:

$$CPI = 1 + P_b \text{ WABD}$$

1	В								
s''		В							
s'			В						
s				В					

Рис. 6.8. Таблица резервирования для выбора стратегии переходов.

Целью выбора стратегии переходов является уменьшение или устранение WABD, что позволит приблизить значение CPI к 1. Параметр P_b определяется выполняемой программой и не зависит от выбранной стратегии переходов.

На рисунке 6.8 показано использование таблиц резервирования для анализа стратегий переходов. При выполнении анализа мы вводим следующие обозначения каскадов конвейера, имеющих значение для выбора стратегии переходов:

- каскад 1 — каскад выборки команды,
- каскад s'' — каскад декодирования,
- каскад s' — каскад, на котором становится известен эффективный адрес мишени перехода,
- каскад s — каскад, на котором становится известно действие условного перехода⁶¹⁾.

Если нумеровать эти каскады от 1, начав с каскада выборки команды, то $s'' \leq s' \leq s$. Символ В обозначает движение команды перехода по конвейеру. Например, для базового конвейера, изображенного на рис. 6.1, $s'' = 2$, $s' = 3$ и $s = 3$. Команда перехода, обозначенная символом В, начинается в каскаде 1 и перемещается к каскаду s .

Стратегия замораживания конвейера

Изучение статистических данных по командам переходов показывает, что приблизительно 60% всех команд переходов действительно имеют своим результатом переход, а 40% переход не возбуждают, а имеют результатом продолжение выполнения последовательности команд потока. Обозначим как P_{bt} вероятность того, что переход выполняется. Если основываться на упомянутой статистике, $P_{bt} = 0,6$, а $1 - P_{bt} = 0,4$. Параметры P_b и P_{bt} мы будем в дальнейшем называть параметрами рабочей нагрузки.

Простейшая стратегия переходов, какую только можно придумать, заключается в так называемом замораживании конвейера в тот момент, когда команда перехода декодируется в каскаде s'' . Выборка команд приостанавливается до тех пор, пока в каскаде s не проявится действие команды перехода. Эта стратегия изображена на рис. 6.9.

⁶¹⁾ Совсем не обязательно, чтобы s' , s'' и s были различными каскадами. Для процессора Berkeley RISC I $s'' = s' = s = 2$.

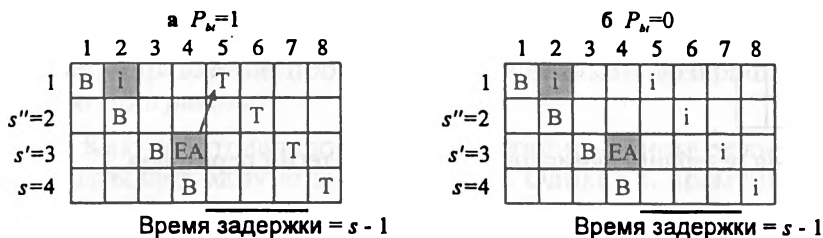


Рис. 6.9. Стратегия замораживания конвейера.

Нам хотелось бы определить для этой стратегии значение WABD, чтобы затем вычислить CPI. При этом следует иметь в виду два случая: случай а, когда переход выполняется ($P_{bt} = 1$), и случай б, когда переход не выполняется ($P_{bt} = 0$).

Случай а с $P_{bt} = 1$ показывает, как действует стратегия перехода, если переход выполняется. Команда перехода декодируется в каскаде s'' в момент t_2 . В то же время выполняется выборка очередной команды i , следующей за командой перехода В. Поскольку декодированная команда является командой перехода, команда i отбрасывается (что обозначено в таблице затенением клетки) и все время, пока команда перехода проходит по конвейеру, другие команды не выбираются. В момент t_3 в каскаде s'' вычисляется эффективный адрес, который сохраняется в буфере (буфер не нужен, если $s'' = s$). В момент t_4 становится известно, что действием команды перехода является переход, и выполняется выборка команды Т, эффективный адрес которой был сохранен в буфере. После этого буфер очищается. Задержка в этом случае составляет $s - 1$ тактов.

Схожим образом мы можем рассмотреть случай б с $P_{bt} = 0$, когда переход не выполняется. Когда в момент t_4 станет известно действие команды перехода, осуществляется выборка команды i , и нормальный поток команд начинает перемещаться по конвейеру. Задержка для этого случая равна $s - 1$.

Для стратегии замораживания WABD равно

$$\text{WABD (замораживание)} = P_{bt}(s - 1) + (1 - P_{bt})(s - 1) = s - 1$$

Вероятность того, что команда является командой перехода, составляет P_b . Таким образом, значение CPI для конвейера, использующего стратегию замораживания, составит

$$\text{CPI (замораживание)} = 1 + P_b \text{WABD} = 1 + P_b(s - 1)$$

При этой стратегии и выполняемый, и не выполняемый переходы дают одинаковую задержку $s - 1$. В результате такая модель совпадает с базовой моделью, разработанной ранее на основании таблицы резервирования рис. 6.4, где $1/k = P_b$.

✎ Примеры

1. Рассмотрим конвейер с $s'' = 1$, $s' = 3$ и $s = 4$. Параметры рабочей нагрузки равны $P_b = 0,3$ и $P_{bt} = 0,65$. Каково значение CPI для такого конвейера?

✎ Решение

$$CPI = 1 + P_b (s - 1)$$

$$CPI = 1 + 0,3 \times (4 - 1) = 1 + 0,9 = 1,9$$

✓ Замечание

Из-за переходов производительность стационарного режима этого конвейера ухудшилась: значение CPI вместо 1 стало равно 1,9, что соответствует потери производительности на 90%.

2. Если каскад s объединяется с каскадом s'' , и при этом тактовый период должен быть увеличен на 20%, повышается ли производительность?

✎ Решение

$$CPI = 1 + 0,3 \times (3 - 1) = 1,6$$

$$\text{время выполнения} = CPI \times \text{тактовый период} = 1,6 \times 1,2 = 1,92$$

Мы имеем уменьшение производительности; $S = 1,9/1,92 = 0,989$.

✓ Замечание

Изменение производительности очень мало, и выбор варианта следует производить исходя из фактического увеличения тактового периода.

Стратегия с предсказанием выполнения перехода

С точки зрения производительности стратегия замораживания дает очень плохие результаты. Можно ли предложить что-то лучшее? Возможно, мы можем извлечь пользу из того известного факта, что приблизительно 60% всех команд переходов приводят к выполнению перехода, и включить как-то эти сведения в аппаратную реализацию. Таблица резервирования для такой стратегии показана на рис. 6.10. Случай а выполнения перехода рассмотрен в левой части рисунка. Если в каскаде s'' декодируется команда, выборка очередных команд потока прекращается, а уже выбранные команды отбрасываются. Когда в каскаде s'' в момент t_3 станет известен эффективный адрес перехода, в следующем такте t_4 выбирается команда по адресу перехода, еще до

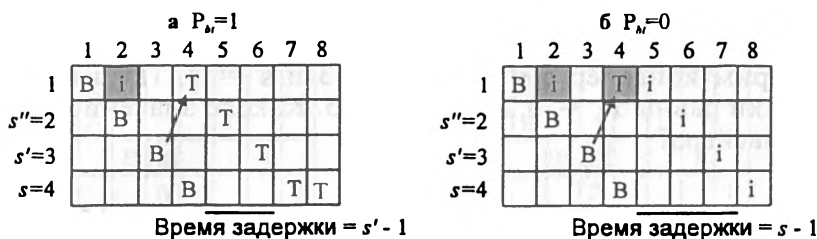


Рис. 6.10. Стратегия с предсказанием выполнения перехода.

того, как станет известно, произойдет ли переход⁶²⁾. Мы поступаем так потому, что предсказываем выполнение перехода. Если это предсказание оправдывается, задержка равна $s'' - 1$.

Если переход не выполняется (случай б в правой части рисунка), то отбрасываются не только последующие команды потока, но и заранее выбранная команда-мишень, поскольку наше предсказание оказалось неправильным. В этом случае должна быть выбрана следующая команда из потока. Задержка для этого случая будет $s - 1$.

Теперь можно вычислить значения WABD и CPI для стратегии с предсказанием выполнения перехода (PT-стратегия):

$$WABT(PT) = P_{bt}(s' - 1) + (1 - P_{bt})(s - 1) = s - 1 - P_{bt}(s - s')$$

и

$$CPI(PT) = 1 + P_b WABD = 1 + P_b[s - 1 - P_{bt}(s - s')]$$

Стратегия с предсказанием невыполнения перехода

Почему бы не исследовать стратегию переходов, которая предсказывает невыполнение перехода, т. е. противоположность рассмотренной выше стратегии? Таблица резервирования для этой стратегии показана на рис. 6.11.

Поскольку аппаратúra разработана в предположении, что переход не выполняется, в этой стратегии по предположению осуществляется

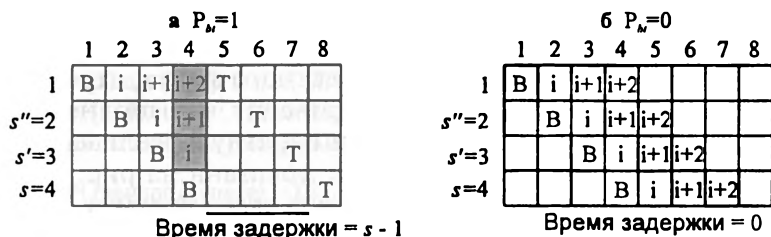


Рис. 6.11. Стратегия с предсказанием невыполнения перехода.

⁶²⁾ Так называемое выполнение по предположению.

выборка возможно большего количества команд потока; конкретно $s - 1$, или 3 команды для нашего примера. Аппаратура ожидает выяснения действия команды перехода в каскаде s перед тем, как будет выбираться команда мишень. В этом случае адрес мишени должен быть сохранен в буфере. Если предсказание оказалось неверным и переход выполняется, как показано в случае **а** на рис. 6.11, то задержка равна $s - 1$. Если, однако, переход не выполняется, как и было предсказано, задержка равна нулю (случай **б** на рис. 6.11).

Значения WABD и CPI для стратегии с предсказанием невыполнения перехода (PNT-стратегия) вычисляются следующим образом:

$$WABT(PNT) = P_{bt}(s - 1) + (1 - P_{bt})0 = P_{bt}(s - 1)$$

и

$$CPI(PNT) = 1 + P_b P_{bt}(s - 1)$$

✎ Пример

Для конвейера и данных рабочей нагрузки из предыдущего примера вычислите значения CPI для стратегий с предсказанием выполнения и невыполнения перехода и сравните их со значением CPI для стратегии замораживания (F-стратегия).

✍ Решение

$$CPI(PT) = 1 + 0,3 \times [4 - 1 - 0,6 \times (4 - 3)] = 1,72$$

$$CPI(PNT) = 1 + 0,3 \times 0,6 \times (4 - 1) = 1,54$$

$$CPI(F) = 1 + 0,3 \times 3 = 1,9$$

✓ Замечание

Этот пример показывает, что, используя стратегии с предсказанием, можно добиться существенного уменьшения CPI. Любопытно, что стратегия с предсказанием невыполнения перехода оказывается лучшей из трех рассмотренных, несмотря на то, что переходы чаще выполняются, чем не выполняются. Причина этого заключается в том, что для случаев невыполнения перехода, т. е. в 40% всех переходов, достигается огромный выигрыш — нулевая задержка. Этот результат, казалось бы, противоречит закону Амдала, который утверждает, что следует повышать быстродействие наиболее вероятного варианта. Если, однако, вероятности событий приблизительно равны, но имеется возможность существенно сократить время выполнения события с меньшей вероятностью, то может оказаться, что именно такое решение будет наилучшим.

Обобщенное предсказание для буферов мишени перехода и кэшей

В рассмотренных выше моделях предполагалось, что предсказание о выполнении или о невыполнении перехода, аппаратно реализуется в процессоре его разработчиком. Нельзя ли пойти еще дальше и настраивать предсказание динамически по ходу выполнения программы? Оказывается, можно: динамическое предсказание вполне осуществимо и может еще сильнее снизить CPI процессора.

Внимательно рассмотрев две описанные выше стратегии, можно заметить, что в системе динамического предсказания возникают четыре возможных события. Эти события и соответствующие им задержки (полученные из рис. 6.10 и 6.11) перечислены в табл. 6.1.

Таблица 6.1. События обобщенного предсказания и их задержки

Предсказание перехода	Фактически	Задержка
Будет выполнен	Выполнен	$s' - 1$
Будет выполнен	Не выполнен	$s - 1$
Не будет выполнен	Выполнен	$s - 1$
Не будет выполнен	Не выполнен	0

Идеальный предсказатель (предсказатель, который правильно предсказывает и выполняемые, и невыполняемые переходы) должен обеспечить осуществление только первого и четвертого событий; второе и третье события никогда возникать не будут. Точность предсказания обозначается A_p , и для идеального предсказателя $A_p = 1$.

Вероятности событий при использовании идеального предсказателя приведены в табл. 6.2. Поскольку предсказатель идеален, вероятности первого и четвертого событий равны соответственно P_{bt} и $1 - P_{bt}$. Оба оставшихся события имеют нулевую вероятность возникновения. Это приводит нас к модели WABD:

$$WABT(A_p = 1) = P_{bt}(s' - 1) + (1 - P_{bt})0 = P_{bt}(s' - 1),$$

$$CPI(A_p = 1) = 1 + P_b P_{bt}(s' - 1)$$

Таблица 6.2. Идеальное предсказание

Предсказание перехода	Фактически	Вероятность	Задержка
Будет выполнен	Выполнен	P_{bt}	$s' - 1$
Будет выполнен	Не выполнен	0	$s - 1$
Не будет выполнен	Выполнен	0	$s - 1$
Не будет выполнен	Не выполнен	$1 - P_{bt}$	0

Этот результат замечателен в том отношении, что при использовании идеального предсказателя местоположение каскада s' оказывается решающим, в то время как положение s не имеет значения. В результате проектировщик, желающий с целью уменьшения CPI выбрать эффективную стратегию переходов, должен решить две задачи:

1. Разработать хороший предсказатель.
2. Поместить каскад s' в начало конвейера (если $s' = 1$ и $A_p = 1$, то $WABD = 0$).

Не забудьте, что здесь придется идти на компромиссы. Комбинирование каскадов с целью переноса s' вперед снизит CPI, но увеличит тактовый период. В результате производительность может возрасти, а может и не возрасти; необходимы дальнейшие оценки. В последующих параграфах мы рассмотрим другие возможности помещения s' в начало конвейера.

Динамическое предсказание перехода

Первое требование для высокопроизводительной стратегии переходов — наличие хорошего предсказателя переходов. В этом разделе мы рассмотрим предсказатели переходов, которые динамически модифицируют предсказания по ходу выполнения программы. Такие динамические приемы подстраивают предсказание для каждой конкретной команды перехода в процессе ее выполнения. Динамическое предсказание перехода выступает в трех основных формах: 1) предсказание, основанное на характеристиках команды перехода, 2) предсказание, основанное на характеристиках адреса перехода, 3) предсказание, основанное на истории выполнения. В работе [1] можно найти более полное обсуждение предсказаний переходов.

Характеристики команд перехода

Прямолинейный способ придания командам перехода определенной характеристики заключается во включении в код команды бита, устанавливаемого программистом или компилятором. Этот бит указывает процессору на наиболее вероятное направление перехода; во время выполнения бит не модифицируется. Например, при программировании цикла бит направления будет установлен в состояние «выполнить переход». Если цикл выполняется 100 раз, предсказание будет оправдываться в 99% случаев. Бит предсказания перехода используется, в частности, в команде перехода процессора PowerPC.

Коды операции команды перехода предоставляют возможность предсказания действия команды в каскаде s'' . Например, безусловный переход всегда выполняется и, следовательно, всегда должно предсказываться его выполнение. Статистический анализ кодов операций команд переходов показывает, что команды условных перехо-

дов также имеют особенности, указывающие на более вероятное их действие, которые могут использоваться для предсказания [5].

🔗 **Пример**

Рассмотрим стратегию переходов, основанную на анализе кода операции команды перехода. Все безусловные переходы предсказываются, как выполняемые, а все условные переходы предсказываются, как невыполняемые. Доля безусловных переходов среди всех переходов составляет P_{uncb} .

- А. Каково значение WABD для этой стратегии?
- Б. Используя данные предыдущего примера и полагая, что $P_{uncb} = 0,3$, найти CPI процессора.

✍ **Решение**

Эту задачу можно решить, построив таблицу вероятностей событий и их задержек. Определим P_{cbit} , как вероятность выполнения условного перехода. Напомним, что P_{bt} определяет вероятность того, что переход выполнится при усреднении по всем переходам. В таблице 6.3 перечислены четыре возможных • состояния стратегии перехода вместе с вероятностями событий и их задержками.

Таблица 6.3. События переходов и их задержки

Тип перехода	Стратегия	Фактически	Вероятность	Задержка
Условный	PNT	Выполнен	$(1 - P_{uncb})P_{cbit}$	$s - 1$
Условный	PNT	Не выполнен	$(1 - P_{uncb})(1 - P_{cbit})$	0
Безусловный	PT	Выполнен	P_{uncb}	$s' - 1$
Безусловный	PT	Не выполнен	0	0
			Сумма = 1	

Из таблицы 6.3 находим

$$WABT = (1 - P_{uncb})P_{cbit}(s - 1) + P_{uncb}(s' - 1)$$

Каково значение P_{cbit} ? Для нахождения P_{cbit} отметим еще раз, что P_{bt} определяет среднюю вероятность выполнения перехода. Найдем взвешенное среднее вероятностей условных и безусловных переходов:

$$P_{bt} = (1 - P_{uncb})P_{cbit} + P_{uncb} \times 1$$
$$P_{cbit} = (P_{bt} - P_{uncb}) / (1 - P_{uncb})$$

Подставив данные, получим $P_{\text{cbl}} = (0,6 - 0,3)/(1 - 0,3) = 0,428$. Теперь можно вычислить для этой стратегии WABD и CPI:

$$\text{WABD} = (1 - 0,3) \times 0,428 \times (4 - 1) + 0,3 \times (3 - 1) = 1,49$$

$$\text{CPI} = 1 + P_{\text{b}} \text{WABD} = 1 + 0,3 \times 1,49 = 1,44$$

✓ Замечание

При выполнении предсказания по коду операции CPI для PNT-стратегии уменьшился с 1,54 до 1,44. Результаты, получаемые с помощью предложенной модели, можно проверить, положив P_{unclb} равным 0 и 1 и удостоверившись, что мы приходим к моделям с предсказанием выполнения и с предсказанием невыполнения перехода. Кроме того, положив $P_{\text{unclb}} = 0$ (т. е. предположив, что все переходы являются условными), получим $P_{\text{cbl}} = P_{\text{b}}$, чего и следовало ожидать.

Характеристики адреса перехода

Адрес мишени перехода имеет две характеристики, которые можно использовать для предсказания действия перехода. К этим характеристикам относятся направление и расстояние перехода. Короткие переходы, а также переходы назад характерны для программных циклов и обычно выполняются. Длинные переходы, а также переходы вперед используются для серьезных изменений в ходе программы и чаще не выполняются.

Как делаются эти предсказания? Рассмотрим относительный адрес мишени перехода (относительно текущего содержимого PC). Определить направление перехода не составляет труда, так как смещение в команде перехода является целым числом со знаком, поэтому для предсказания может быть использован знаковый бит.

Определение расстояния перехода представляет собой более сложную задачу. Прежде всего следует задать пороговое значение, скажем, 108 команд. Как теперь выполнить сравнение смещения и порогового значения? Вычитание добавит значительное время к задержке каскада декодирования. Более перспективно установить для порога значение, равное целой степени 2 (не 108, как было предложено выше, а 128) и, анализируя биты смещения, фиксировать превышение заданного порога.

История переходов

Статистические исследования показывают, что если сведения о направлении перехода для данной команды перехода собираются и накапливаются в аппаратуре («история перехода»), то при следующем выполнении этой команды перехода можно сделать обоснованное предсказание. Если, например, в коде команды перехода содержится

бит, который устанавливается или сбрасывается в зависимости от действия текущей команды, этот бит является хорошим предсказателем для следующего выполнения. Уже выполненный переход с большей вероятностью будет выполнен, чем не выполнен, и в следующий раз, когда программа подойдет к той же команде. Некоторое осложнение заключается в том, что, поскольку бит входит в команду, мы имеем дело с самомодифицирующимся кодом.

Более практично иметь поле битов истории в кэше команд, где такое поле может быть связано с каждой командой. Эти биты не являются частью команды; они присоединяются к команде в кэше. Исследования показали, что чем больше битов включено в поле истории, тем точнее будет предсказание. Для иллюстрации этого утверждения в табл. 6.4 и 6.5 приведены данные по 1-битовой и 2-битовой организациям поля истории. Эти данные взяты из работы [5], они были получены с помощью контрольной программы. В первом столбце показаны история и предсказание. Второй столбец показывает (курсивом) действие команды. Символы *N* и *T* используются для указания на невыполняемый (not taken) и выполняемый (taken) переходы соответственно. Для 1-битовой истории (табл. 6.4) пары *NN* возникают с вероятностью 0,309, в то время как пары *NT* лишь с вероятностью 0,071.

Таблица 6.4. Вероятности сочетаний переходов для 1-битового поля истории

История(предсказание)	Следующий переход	Вероятность
<i>N</i> (<i>N</i>)	<i>N</i>	0,309
<i>N</i> (<i>N</i>)	<i>T</i>	0,071
<i>T</i> (<i>T</i>)	<i>N</i>	0,046
<i>T</i> (<i>T</i>)	<i>T</i>	0,565

Таблица 6.5. Вероятности сочетаний переходов для 2-битового поля истории

История(предсказание)	Следующий переход	Вероятность
<i>NN</i> (<i>N</i>)	<i>N</i>	0,294
<i>NN</i> (<i>N</i>)	<i>T</i>	0,015
<i>NT</i> (<i>N</i>)	<i>N</i>	0,051
<i>NT</i> (<i>N</i>)	<i>T</i>	0,019
<i>TN</i> (<i>T</i>)	<i>N</i>	0,017
<i>TN</i> (<i>T</i>)	<i>T</i>	0,039
<i>TT</i> (<i>T</i>)	<i>N</i>	0,013
<i>TT</i> (<i>T</i>)	<i>T</i>	0,552

Аппаратура, исходя из значения бита истории, даст предсказание N , если в предыдущий раз переход не выполнялся, и T , если в предыдущий раз переход выполнялся. Если поле истории составляет 1 бит, точность предсказания A , равна $0,309 + 0,565 = 0,874$.

Данные для поля истории размером 2 бита, приведенные в табл. 6.5, взяты из той же работы [5]. Читать эти данные следует справа налево; так, NT обозначает, что при последнем выполнении команды перехода переход выполнялся (T), а при предпоследнем выполнении перехода не выполнялся (N). Вероятности правильных предсказаний выделены жирным шрифтом.

Два выполненных перехода подряд дадут предсказание выполнения перехода с точностью $0,552/(0,013 + 0,552) = 0,97$. Два невыполненных перехода подряд дадут предсказание невыполнения с точностью $0,294/(0,294 + 0,015) = 0,95$. Если два бита истории имеют различающиеся значения, предсказание основывается на данных приведенной таблицы. Так, NT даст предсказание N , потому что $0,051 > 0,019$. Усредненная точность предсказаний по данным конкретной контрольной программы составляет $0,294 + 0,051 + 0,039 + 0,552 = 0,936$; мы получили улучшение на 10% по сравнению с 1-битовой схемой.

✎ Пример

Определим точность предсказания по 2-битовому полю истории при использовании следующей стратегии. Если оба бита истории одинаковы (NN или TT), будем предсказывать N или T , соответственно. Если биты различаются (NT или TN), предскажем по последнему событию: N укажет на N , T – на T .

✎ Решение

Таблица 6.6 получена из табл. 6.5 путем замены предсказаний для случаев, когда биты истории различаются. Вероятности правильных предсказаний выделены жирным шрифтом.

Точность предсказания является суммой вероятностей, для которых предсказание оказалось правильным:

$$0,294 + 0,019 + 0,017 + 0,552 = 0,882.$$

✓ Замечание

Точность лишь незначительно выше, чем для 1-битовой (0,874) схемы. Аппаратная реализация такой схемы почти так же сложна, как в случае первого варианта анализа 2-битового поля истории, а точность предсказания существенно ниже.

Таблица 6.6. Модифицированная таблица вероятностей для 2-битового поля истории

История (предсказание)	Следующий переход	Вероятность
NN(N)	<i>N</i>	0,294
NN(N)	<i>T</i>	0,015
NT(T)	<i>N</i>	0,051
NT(T)	<i>T</i>	0,019
TN(N)	<i>N</i>	0,017
TN(N)	<i>T</i>	0,039
TT(T)	<i>N</i>	0,013
TT(T)	<i>T</i>	0,552

Любые схемы предсказаний в конвейерном процессоре имеют определенные недостатки. Один из недостатков связан с тем, что состояние истории команды условного перехода обновляется лишь в каскаде *s*, после того, как станет известно действие команды. Если конкретная команда перехода выполняется с интервалом меньшим, чем *s*, предсказание делается на основе устаревшей информации. Другими словами, последнее выполнение команды не учитывается. Из-за этой задержки 3-битовая история может оказаться не лучше двухбитовой.

Другой недостаток, проявляющийся в некоторых схемах предсказаний, например, в 1-битовой схеме табл. 6.4, заключается в том, что для неестественных последовательностей переходов, например *T*, *N*, *T*, *N*, *T*, , предсказание действия перехода будет всегда неверным. Далее, при контекстных переключениях восстановление истории потребует некоторого времени, и в течение этого времени точность предсказаний может оказаться сниженной. Разумеется, информацию об истории можно сохранять при контекстном переключении в качестве части состояния процессора.

Уменьшение *s'*

Теперь, когда мы поняли, как получить точность предсказания, приближающуюся к 1, посмотрим, как можно уменьшить *s'*. Вспомним, что *s'* — это каскад конвейера, на котором определяется эффективный адрес. Для решения нашей задачи учтем то обстоятельство, что эффективный адрес мишени команды перехода обычно не изменяется от одного выполнения команды к другому. Иными словами, для эффективных адресов мишеней характерна временная локализация, что дает возможность использовать кэш-память для сохранения предварительно вычисленного и уже использованного эффективного адреса. После следующего выполнения команды перехода эффективный

адрес может быть доступен немедленно, в результате чего $s' = 1$, и $WABD = 0$. Кэш-память эффективных адресов может быть реализована в двух формах: буфер мишени перехода (branch target buffer, BTB) и кэш мишени перехода (branch target cache, BTC).

Буфер мишени перехода

Блок-схема ВТВ показана на рис. 6.12. Элемент ВТВ состоит из трех полей: эффективный адрес команды перехода, эффективный адрес мишени перехода и данные истории, необходимые для предсказания. Адреса команд от каскада выборки команды направляются в кэш команд и в ВТВ. Организация ВТВ и доступ к нему могут соответствовать любой схеме организации кэш-памяти: прямой, частично-ассоциативной или полностью ассоциативной. Если в ВТВ происходит попадание (команда перехода использовалась ранее и все еще находится в кэше) и данные истории предсказывают, что переход будет выполняться, адрес мишени перехода пересылается в кэш команд, и команда-мишень выбирается в каскад декодирования.

Поскольку обращение к ВТВ происходит последовательно с обращением к кэшу команд, может потребоваться увеличение тактового периода процессора или использование второго такта; таким образом, $s' = 2$. При реализации ВТВ возникает ряд вопросов. Что происходит, если в ВТВ произошел промах? Что происходит, если было попадание в ВТВ, но предсказывается невыполнение перехода? Решение всех этих вопросов требует составления таблицы событий и определения действий процессора в ответ на каждое и неизбежных задержек.

Мы должны определить некоторые параметры. P_{btb} представляет собой вероятность ВТВ-попадания, P_{pbt} — вероятность предсказания выполнения перехода, а A_p — точность предсказания. В таблице 6.7 показаны события, задержки и стратегии по умолчанию для поведения ВТВ. Значения задержки по умолчанию для тех случаев, когда предсказание не оправдывается, взяты из табл. 6.1. Для случаев ВТВ-промаха принята стратегия замораживания конвейера.

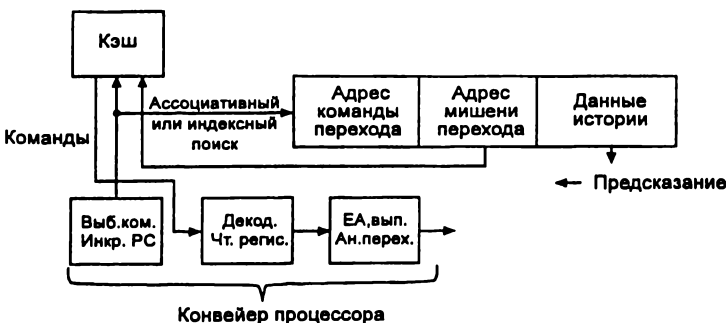


Рис. 6.12. Буфер мишени перехода ВТВ.

Таблица 6.7. События ВТВ и задержки

ВТВ	Предсказание	Фактически	Вероятность	Задержка
Попад.	Выполнится	Выполнен	$P_{bth} P_{pbt} A_p$	0
Попад.	Выполнится	Не выполнен	$P_{bth} P_{pbt} (1 - A_p)$	$s - 1$
Попад.	Не выполнится	Не выполнен	$P_{bth} (1 - P_{pbt}) A_p$	0
Попад.	Не выполнится	Выполнен	$P_{bth} (1 - P_{pbt}) (1 - A_p)$	$s - 1$
Промех	Заморозить	Безразлично	$1 - P_{bth}$	$s - 1$
			Сумма = 1	

Воспользовавшись табл. 6.7, мы можем написать уравнение для WABD. Обратите внимание на то, что для двух событий задержка равна 0, а для всех остальных $s - 1$:

$$WABT = [P_{bth} P_{pbt} (1 - A_p) + P_{bth} (1 - P_{pbt}) (1 - A_p) + (1 - P_{bth}) (s - 1)]$$

$$WABT = [1 - (P_{bth} A_p)] (s - 1)$$

Обратите внимание на то, что в модель WABD входит местоположение каскада s . Однако s имеет значение, лишь когда A_p заметно меньше 1. Кроме того, размер ВТВ может быть увеличен, что повысит интенсивность попаданий и снизит влияние неправильных предсказаний переходов или большого значения s .

Обращение к ВТВ и выбор пути выполнения или невыполнения перехода основаны на предсказании. Что произойдет, если предсказание не оправдалось или имел место ВТВ-промах? Вспомним, что действие перехода неизвестно до каскада s . Таким образом, должен быть параллельный конвейер, выполняющий в таких случаях стратегию по умолчанию. Другими словами, этот параллельный конвейер действует так, как если бы ВТВ не существовал. В таблице 6.7 в качестве стратегии по умолчанию указано предсказание невыполнения перехода с задержкой $s - 1$.

Пример

Воспользуемся рассмотренным выше примером конвейера с $s \approx 4$, $P_b = 0,3$ и $P_{bt} = 0,6$. Примем также, что $P_{bth} = 0,9$ и $A_p = 0,9$. Каковы для этой стратегии значения WABD и CPI?

Решение

$$WABD = (1 - 0,9 \times 0,9)(4 - 1) = 0,57$$

$$CPI = 1 + 0,3 \times 0,57 = 1,17$$

✓ **Замечание**

Обратите внимание на то, что величина $P_{\text{рв}}$ выпала из нашей модели. Если ВТВ имеет указанные выше параметры, CPI процессора приблизительно на 10% меньше, чем теоретическое значение для стационарного состояния, равное 1. Этот пример демонстрирует положительную роль ВТВ в снижении CPI.

Кэш мишени перехода

Другим способом уменьшения значения s' является использование кэша мишени перехода (ВТС). Зачем сохранять адрес мишени, а затем через него обращаться к кэшу команд, как это делается в случае использования ВТС? Почему бы не сохранять сами команды-мишени? Именно такой подход используется в ВТС. Блок-схема ВТС приведена на рис. 6.13, где показан также конвейер по умолчанию. ВТВ весьма похож на ВТС за исключением того, что в тех случаях, когда происходит попадание, а предсказание является выполнением перехода, поток команд-мишеней направляется в каскад декодирования. Такая схема устраняет необходимость в удлинении тактового периода или задержке цикла такта из-за дополнительной задержки обращения к кэшу команд. Для рассмотренной стратегии $s' = 1$.

При проектировании ВТС необходимо обеспечить источник команд, следующих за командами переходов. Для этого ВТС должен хранить команду перехода вместе со следующими i командами-мишенями; обычно i равно 4 или 8. Например, ВТС процессора AM29000 хранит последовательность из четырех команд-мишеней.

При проектировании как ВТС, так и ВТВ необходимо найти время для обновления этих буферов. Обновление может быть двух ви-

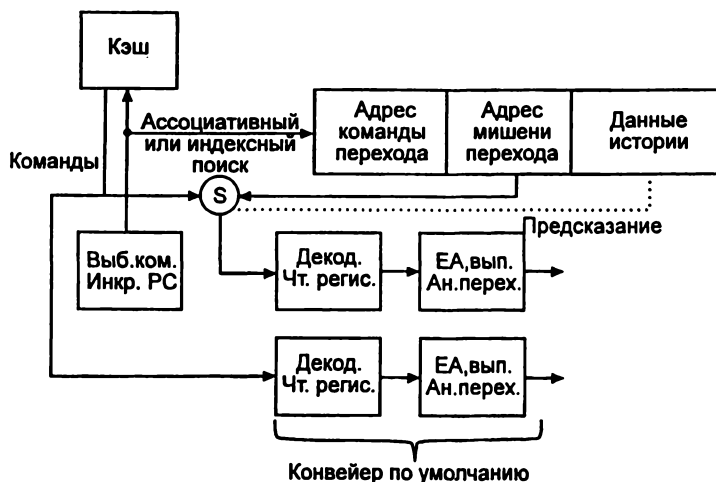


Рис. 6.13. Кэш мишени перехода ВТС.

дов. Обновление первого вида требуется, если произошел промах (данная команда перехода ранее не использовалась) и новый элемент должен быть сформирован целиком. Второй вид обновления требуется в тех случаях, когда модифицировать надо поле истории. В обоих случаях для обновления ВТВ или ВТС надо изыскать время, что позволит снизить производительность системы. Заметьте, что не все обращения к ВТВ или ВТС приводят к необходимости обновления. Если для хранения истории используется 1 бит, то при правильности предсказания бит истории обновлять не нужно. Экономия на времени обновления может скомпенсировать меньшую точность предсказания.

Поскольку и ВТВ, и ВТС по сути представляют собой кэш-память, необходимо предусмотреть средства для удаления элемента в тех случаях, когда кэш заполнен, но требуется создание нового элемента. Здесь можно использовать те же стратегии, что для кэшей команд или данных. При этом если ВТВ или ВТС спроектированы как частично-ассоциативные, то следует иметь в виду возможность кэш-промаха по конфликту наборов.

✎ Пример

Пусть обновление ВТВ занимает один такт, причем оно требуется для каждого выполнения команды перехода; частота обновлений равна P_b . Какова величина CPI процессора с учетом тактов обновления? Найдём CPI, используя параметры предыдущего примера.

✎ Решение

Формула WABD для ВТВ, полученная из табл. 6.7, модифицирована с учетом такта обновления. Для каждой выполняемой команды перехода требуется обновление, занимающее один такт:

$$WABT = (1 - P_{bth} A_p)(s - 1) + 1$$

$$CPI = 1 + P_b [(1 - P_{bth} A_p)(s - 1) + 1]$$

$$CPI = 1 + 0,3(0,57 + 1) = 1,47$$

✓ Замечание

Такты обновления увеличивают CPI на 0,3 такта ($1,47 - 1,17 = 0,3$), чем заметно снижают производительность. Избежать потери производительности поможет вариант с двухпортовыми ВТВ или ВТС. Другое решение вопроса заключается в сохранении обновляемой

информации в буфере и выполнении обновления в тех тактах ВТВ, которые не являются попаданиями.

Задержанные переходы

В RISC-процессорах с короткими конвейерами удачно использовалась стратегия, получившая название задержанных переходов. Эта стратегия требует взаимодействия между программным обеспечением и аппаратными средствами.

Компилятор после каждой команды перехода вставляет $s - 1$ команд пор⁶³⁾. В результате конвейер не выполняет полезной работы в ожидании разрешения перехода в каскад s . Затем компилятор пытается заменить команды пор полезными командами, которые допустимо выполнять в местах расположения команд пор без нарушения логики программы. В ряде исследований было показано, что в среднем одна команда пор может быть заменена полезной командой.

Описанная стратегия эффективна для конвейерных процессоров с $s \leq 5$. Для более длинных конвейеров команд пор становится слишком много, что существенно увеличивает размер программы.

6.3.2. Структурные конфликты

Структурные конфликты возникают, когда в некотором такте появляются два или более требований к одному и тому же ресурсу. Сама природа конвейерного процессора требует параллелизма; процессор должен быть спроектирован так, чтобы в одно и то же время выполнялось как можно больше различных действий. Требование обеспечения параллелизма может привести к одновременным обращениям к некоторому ресурсу.

Рассмотрим функции ALU в процессоре. В процессоре с последовательным выполнением (см. гл. 3) одно ALU может выполнять три функции: инкрементировать программный счетчик, вычислять эффективный адрес и выполнять операцию, соответствующую текущей команде. В конвейерном процессоре, который обрабатывает одновременно несколько команд, одно ALU не может выполнять три операции одновременно. Имеются три способа обеспечения логически правильного выполнения программы:

- 1) использовать приоритетную схему, которая будет задерживать все запросы, кроме одного, что приведет к задержкам для других запросов;

⁶³⁾ Команда пор не делает абсолютно ничего. Она не приводит ни к каким побочным результатам, за исключением инкрементирования программного счетчика.

- 2) дублировать ресурс, например, включить в процессор три экземпляра ALU;
- 3) использовать быстродействующий ресурс, который позволит обслужить все запросы в течение одного такта.

Выбор приоритета

В целом ряде случаев приоритетная схема является предпочтительным методом борьбы со структурными сбоями. Один из таких случаев — использование общего кэша в качестве кэша второго уровня. Вспомним пример в гл. 4: если на первом уровне используется разделенный кэш, лишь небольшая доля обращений к памяти (командам или данным) приводит к промаху и обслуживается кэшем второго уровня. Промахи первого уровня чаще всего некоррелированы, т. е. не возникают одновременно. Таким образом, имеется лишь незначительная вероятность того, что два промаха, возникнув одновременно, приведут к задержке в силу двух обращений к кэшу второго уровня. Несмотря на то, что вероятность одновременного обращения к общему кэшу второго уровня невелика, проектировщик обязан обеспечить приоритетную логику для разрешения этого структурного конфликта, если он все-таки возникнет.

Схожая ситуация возникает на шине, которая соединяет общий кэш и реальную память и совместно используется вводом и выводом. Здесь вступает в дело стратегия записи в кэш. При использовании кэша данных со сквозной записью шина занята приблизительно на 50%, потому что все промахи чтения и записи требуют передачи данных по шине. Обычно в этом случае операциям ввода-вывода дается более высокий приоритет, так как они характеризуются меньшей частотой, чем кэш-промахи. Кроме того, как это будет показано в гл. 8, задержка в обращении к шине некоторых устройств ввода-вывода приводит к значительным потерям в производительности.

Внутренние шины процессора также могут использовать приоритетный выбор. Выходы узлов выполнения операций умножения-деления и сложения с плавающей запятой процессора IBM S360-91 сообщают используют шину, называемую общей шиной данных. Эта шина пересылает результаты в регистровый файл и обратно на входы узлов выполнения. Если оба узла выполнения выдают результаты в том же такте, приоритетная логика задерживает один из результатов на один такт. Проектировщики этого процессора учитывали тот факт, что вероятность появления такого события невелика и результирующая задержка не будет иметь особого значения.

✎ Пример

Из рисунка 6.1 видно, что два каскада конвейера (каскады 2 и 5) могут обращаться к регистровому файлу. В зависимости от конкретных команд, которые проходят через конвейер в данный момент, регистровый файл может получить в одном такте два обращения чтения и одно обращение записи. Пусть регистровый файл является двухпортовым, т. е. может выполнять две операции чтения в одном такте. Если в том же такте возникает операция записи, она должна быть задержана приоритетной логикой. Будем далее считать, что в $1/4$ тактов отсутствуют обращения к регистровому файлу, в $1/4$ тактов возникает два чтения, в $1/4$ тактов — два чтения и одна запись и в $1/4$ тактов — запись в регистровый файл. Если игнорировать остальные задержки, каким будет CPI процессора?

✎ Решение

Только в $1/4$ тактов, когда возникают два чтения и одна запись, будет наблюдаться останов в регистровом файле. Структурный сбой разрешается приоритетной логикой. В таких тактах конвейер приостанавливается, или задерживается, на один такт:

$$CPI = 1 + \text{вероятность задержки} \times \text{задержка}$$

$$CPI = 1 + 0,25 \times 1 = 1,25$$

Дублирование ресурсов

Если вероятность конфликта столь велика, что приоритетный выбор некоторого ресурса приводит к существенной потере в производительности, этот ресурс может быть дублирован. Можно привести ряд примеров такого подхода к решению проблемы структурных сбоев. Например, в некоторых конвейерных процессорах используются три ALU: один для инкрементирования PC, другой для вычисления эффективного адреса и третий в качестве узла выполнения операции. Обратите внимание на то, что в этом случае все ALU должны быть высокоспециализированы.

✎ Пример

Базовый конвейер осуществляет выборку команды в каждом такте и в командах загрузки-запоминания выполняет операции чтения или записи. Пусть команды загрузки-запоминания составляют 0,3 всех команд. При использовании общего кэша команды загрузки-запоминания должны задерживаться на один такт приоритетной логикой. При использовании разделенного кэша задержка не нужна. Откройте рис. 4.10, где были изображены варианты топологии кэш-памяти. Если

использовать разделенный кэш и игнорировать кэш-промахи, насколько повысится быстродействие процессора в единицах CPI?

Решение

$CPI = 1 + \text{вероятность задержки} \times \text{задержка}$

$CPI (\text{общий кэш}) = 1 + 0,3 \times 1 = 1,3$

$CPI (\text{разделенный кэш}) = 1 + 0,3 \times 0 = 1,0$

повышение быстродействия = $1,3/1,0 = 1,3$

✓ Замечание

Добавление второго кэша приводит к существенному повышению производительности процессора. Ввиду относительно низкой стоимости электронных схем дублирование ресурсов широко используется в современных процессорах для решения проблем структурных конфликтов.

В некоторых процессорах дублируются буферы ассоциативной трансляции TLB, в результате чего трансляция имен для команд и данных не вызывает структурных конфликтов.

При дублировании ячеек памяти необходимо следить за тем, чтобы содержимое этих ячеек было согласованным (см. гл. 4). В противном случае выполнение программы может оказаться логически неверным. Проектировщик должен решить, что хуже: потерять время на поддержание согласованности или иметь задержки, вносимые приоритетной логикой обращения к общему ресурсу.

6.3.3. Взаимозависимость данных

Взаимозависимость данных возникает в выполняемой программе в тех случаях, когда программа не может выполняться правильно в силу запаздывания конвейера. В последовательном процессоре взаимозависимость данных возникнуть не может. Имеются три класса взаимозависимостей:

1. Чтение после записи (RAW); этот класс также носит название реальной зависимости.
2. Запись после чтения (WAR); этот класс также носит название антизависимости.
3. Запись после записи (WAW); этот класс также носит название выходной зависимости.

Как будет показано ниже, эти три класса взаимозависимостей обнаруживаются аппаратно, причем аппаратура обеспечивает правиль-



Рис. 6.14. Взаимозависимость типа RAW.

ное выполнение как обычными, так и специализированными средствами.

Взаимозависимость «чтение после записи»

Такого рода взаимозависимость возникает, когда команде требуется операнд, являющийся результатом предыдущей команды, но в силу запаздывания в конвейере этот результат еще не получен. Рассмотрим программный фрагмент, состоящий из двух команд; команда *j* следует за командой *i*. При этом между командами *i* и *j* могут находиться ни одной, одна или несколько команд:

i $R1 = R2 + R3$

j $R4 = R1 + R6$

Здесь возникает взаимозависимость типа RAW по регистру *R1*, так как команда *j* не может быть правильно выполнена, пока команда *i* не поместит свой результат в регистр-приемник *R1*. Имейте в виду, что взаимозависимости типа RAW являются следствием неудачной последовательности команд в программе; хороший компилятор может переставить программные строки и уменьшить частоту появления взаимозависимостей типа RAW, хотя все такие взаимозависимости устранены быть не могут.

Единственная цель проектировщика заключается в уменьшении задержек при возникновении взаимозависимости типа RAW. Эта цель может быть выражена следующим образом:

$$CPI = 1 + \text{вероятность взаимозависимости типа RAW} \times \text{задержка RAW} = 1 + P_{\text{raw}} D_{\text{raw}}$$

Определим теперь значение задержки RAW. На рисунке 6.14 приведена таблица резервирования для фрагмента из двух команд, выполняемого на 5-каскадном конвейере рис. 6.1⁶⁴). Команда *i* проходит по конвейеру, причем ее результат записывается в регистровый файл. Поскольку команда *j* нуждается в этом результате, она задерживается на три такта, пока результат не сможет быть прочитан из регистрового файла. Для многих программ зависимость типа RAW между соседни-

⁶⁴) Обратите внимание на то, что каскад памяти обойден, несмотря на то, что квант времени показан занятым. Такая процедура упрощает управление конвейером, не влияя на его производительность.

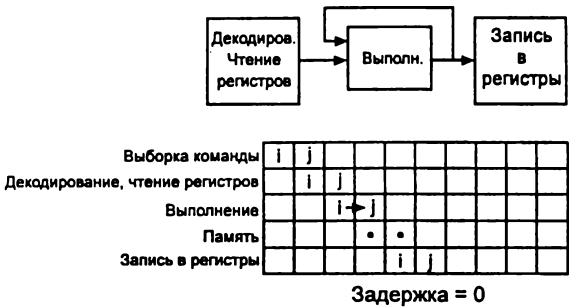


Рис. 6.15. Продвижение данных.

ми командами, обозначенная P_{raw} , характеризуется вероятностью, близкой к 0,2. Итак, CPI такого процессора равно:

$$CPI = 1 + P_{raw}D_{raw}$$
$$CPI = 1 + 0,2 \times 3 = 1,6$$

Поскольку обнаружилось серьезное ухудшение производительности, что-то следует предпринять, чтобы скомпенсировать потери. Прием, используемый для уменьшения или устранения потерь в производительности, называется продвижением данных (forwarding). Три последних каскада базового конвейера и таблица резервирования изображены на рис. 6.15.

Логический путь построен так, что выход узла выполнения поступает на вход того же узла в обход регистрового файла. В результате выполнение команды j может начаться в следующем же такте. Результат команды i тем не менее записывается в регистровый файл, поскольку он может понадобиться последующей команде. Реализация продвижения данных при $P_{raw} = 0,2$ дает следующее значение CPI:

$$CPI = 1 + P_{raw} D_{raw}$$
$$CPI = 1 + 0,2 \times 0 = 1,0$$

Продвижение данных приводит к существенному повышению производительности. Если узел выполнения включает не один, а два каскада, то задержка соседней команды составит один такт, если же в узле выполнения три каскада, то два такта. Для узла выполнения с E каскадами задержка RAW соседней команды будет равна:

$$\text{задержка RAW при наличии продвижения данных} = E - 1$$



Рис. 6.16. RAW-управление.

Мы обсудили вопросы производительности при наличии конфликтов чтения после записи (RAW). В некоторых случаях для уменьшения задержек продвижение данных использовать нельзя. Как в таких случаях спроектировать конвейер, чтобы обеспечить логическую правильность программы? Другими словами, как ввести задержки, когда это необходимо? Решение этой задачи заключается в использовании управляющих битов в регистровом файле, как это показано на рис. 6.16, где рассмотрен тот же программный фрагмент, что и ранее:

i R1 = R2 + R3
 j R4 = R1 + R6

Каждое слово регистрового файла содержит управляющий бит, как это показано для регистров R1, R2, R3 и R6. После декодирования команды управляющий бит регистра-приемника (R1 для команды i) в следующем такте устанавливается в 1. Следующая команда j анализирует управляющие биты ее регистров-источников, в нашем примере R1 и R6. Если любой из этих битов установлен в 1, команда приостанавливается, биты анализируются в каждом последующем такте, и когда оба становятся равными 0, команда j перемещается на следующий каскад конвейера. Когда команда i записывает свой результат в регистровый файл, управляющий бит сбрасывается в 0, указывая тем самым, что взаимозависимость типа RAW была разрешена. Такая форма управления используется в целочисленном процессоре IBM S360 и многих современных микропроцессорах.

Взаимозависимость «запись после чтения»

Такого рода взаимозависимость возникает, когда нарушается последовательность выполнения команд и последующая команда запи-

сывает результат в операнд-источник еще до того, как предыдущая команда прочитала этот операнд. В базовом конвейере (см. рис. 6.1) чтение регистра должно происходить перед тем, как последующая команда может записать что-то в этот регистр. Каким же образом может нарушиться последовательность выполнения команд? В некоторых процессорах дешифратор может передать команду узлу (узлам) выполнения вне очереди. Например, если команда i ожидает разрешения RAW-взаимозависимости, дешифратор может выдать команду j , для которой взаимозависимость отсутствует. Рассмотрим программный фрагмент из трех команд, выполняемый на процессоре, который может выдавать команды вне очереди:

$$i \quad R2 = R1 + R8$$

$$j \quad R9 = R4 \times R2$$

$$k \quad R4 = R7 + R8$$

Здесь имеет место взаимозависимость типа RAW по регистру $R2$ между командами i и j , в результате чего команда j будет приостановлена. Команда k не имеет взаимозависимостей своих операндов, поэтому она может быть передана в узел выполнения. Команда k может завершиться и записать результат в регистр $R4$ перед тем, как $R4$ будет прочитан командой j , что приведет к помещению в $R9$ неправильного результата.

Взаимозависимость предотвращается путем буферизации операндов-источников. Когда команда i поступает в дешифратор, значение $R4$ помещается в буфер $B1$, а значение $R2$ — в буфер $B2$, когда это значение будет получено командой i :

$$j \quad R9 = R4 \times R2$$

$$(R4) \rightarrow B1$$

$$(R2) \rightarrow B2, \text{ когда это значение будет получено командой } i.$$

Команда j преобразуется следующим образом:

$$j \quad R9 = B1 \times B2$$

Буферизация входных данных для команды j устраняет WAR-взаимозависимость, так как команда k после своего завершения может свободно записывать в $R4$. Часто организация конвейера предусматривает буферы на входах ALU, и тогда эти буферы могут использоваться для разрешения конфликтов WAR. Такое использование буферов в общем случае носит название переименования.

Взаимозависимость «запись после записи»

Эта взаимозависимость, известная также под именем выходной, возникает, когда команда выполняется вне очереди, а следующая команда осуществляет запись в приемник еще до того, как в этот же

приемник запишет предыдущая команда. Рассмотрим такой программный фрагмент:

```
i R4 = R3 + R1
j R4 = R7 + R8
k R9 = R4 + R5
```

Предположим, что имеется взаимозависимость, приостанавливающая команду *i*, которая теперь завершается после команды *j*. В этом случае команды будут фактически выполняться в таком порядке:

```
j R4 = R7 + R8
i R4 = R3 + R1
k R9 = R4 + R5
```

Команда *k* получит в *R4* неправильный операнд-источник, и ее результат будет неверен. Для разрешения WAW-конфликта можно использовать буферизацию. При наличии буферизации приведенный программный фрагмент следует переписать таким образом⁶⁵⁾:

```
i R4 = buff1 = R3 + R1
j R4 = buff2 = R7 + R8
k R9 = buff2 + R5
```

Даже если команда *i* завершится после команды *j*, команда *k* будет иметь правильные операнды-источники. Однако здесь возникает RAW-взаимозависимость по буферу *buff2*, которая должна быть разрешена, как было описано выше. При наличии буферов взаимозависимость разрешается не полностью, так как *R4* все еще записывается в регистровый файл вне очереди. По этой причине буферы нельзя считать универсальным решением обсуждаемой проблемы.

Взаимозависимости в других местах

В предыдущих параграфах обсуждались взаимозависимости в процессорном регистровом файле общего назначения. Конфликты и взаимозависимости всех трех видов могут появляться и в других местах, включая память, кэши, регистры битов признаков результата и регистры адресов. Проектировщик должен выявить и идентифицировать все потенциальные взаимозависимости и обеспечить аппаратные средства для их разрешения, чтобы предотвратить нарушение логики выполняемой программы.

⁶⁵⁾ В процессорах с переименованием регистров переписывание программы выполняется автоматически аппаратурой процессора.

Если в компьютере предусматривается ввод-вывод, отображаемый на память (подробнее этот вопрос будет описан в гл. 7), для устранения конфликтов нельзя использовать буферы, так как для правильной работы устройств ввода-вывода операции записи в память должны следовать в заданном порядке. Взаимозависимости типа WAW могут быть разрешены путем насильственной установки последовательности операций записи. В составе команд процессора PowerPC имеется команда `eieio` (*enforce in-order execution of I/O* – принудительный порядок выполнения команд ввода-вывода), которая заставляет команды загрузки и запоминания выполняться в том порядке, в котором они следуют в программе. Эта команда включается в программу в качестве преамбулы к командам, которые будут выполнять загрузку или запоминание в устройствах ввода-вывода, отображаемых на память. Процессор Intel Pentium Pro, который использует ввод-вывод, отображаемый на память, требует, чтобы область памяти, отведенная под устройства ввода-вывода, адресовалась непосредственно и не кэшировалась. В памяти предусматриваются аппаратные средства, обеспечивающие запись в заданном порядке. Этот вопрос будет подробнее обсуждаться в разделе 7.1.

6.4. Прерывания

Общие вопросы использования прерываний в последовательных процессорах были рассмотрены в гл. 3. Разработка конвейерного процессора, в котором одновременно выполняется ряд команд, ставит перед проектировщиком новые вопросы. Нам хотелось бы, чтобы прерывания обрабатывались правильно. Это значит, что при наличии нескольких одновременно выполняемых команд процессор должен правильно вернуться в прерванную программу и продолжить ее выполнение. Если такой возврат не может быть гарантирован, про прерывания говорят, что они неправильны (*imprecise*).

Когда возникает прерывание, источником прерывания может быть каскад конвейера; например, на этапе декодирования обнаружен неправильный код операции. В этом случае каскад декодирования будет называться прерывающим каскадом. С другой стороны, при поступлении внешнего прерывания проектировщик должен идентифицировать каскад, который является прерываемым. Для того чтобы прерванная программа могла продолжить свое выполнение, прерванная или прерывающая команда должна сохранить свое значение программного счетчика.

Любой каскад конвейера, в зависимости от типа прерывания, может быть прерван или стать прерывающим. Отсюда следует, что дол-

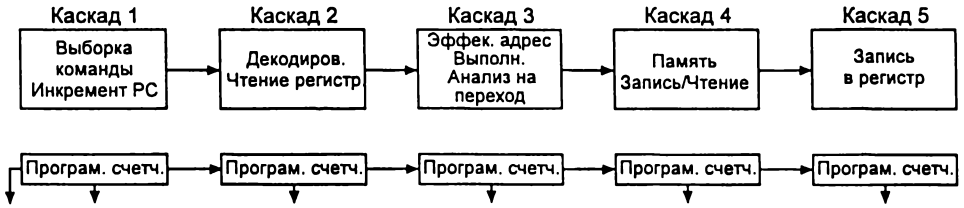


Рис. 6.17. Сохранение программного счетчика.

жен быть способ сохранения содержимого программного счетчика для каждой команды, проходящей через конвейер. Это означает, что необходим конвейер программного счетчика, параллельный основному конвейеру (рис. 6.17)⁶⁶.

Важной деталью режима прерываний является идентификация сохраненного программного счетчика. Для исследования этого вопроса нам надо рассмотреть три типа прерываний.

Асинхронные или внешние	Эти прерывания могут быть инициированы в любой момент такими событиями, как запрос устройства ввода-вывода. Проектировщик выбирает каскад процессора, который должен сохранить свое значение программного счетчика.
Синхронные, или внутренние	Эти прерывания инициируются событиями, возникающими в конкретном каскаде конвейера. Примерами являются отказ страницы, неправильный код операции или арифметическая ошибка. Каскад, в котором происходит данное событие, или предыдущий, сохраняет значение своего программного счетчика.
Программные	Эти события представляют собой явно включенные в программу команды, требующие от процессора выполнения некоторой функции. Примерами являются программное переключение контекста или вызов функции ввода-вывода. В этом случае сохранение программного счетчика обычно выполняется в каскаде декодирования, если только для завершения операции не требуется вычисление эффективного адреса.

⁶⁶) В дополнение к этому должен быть предусмотрен параллельный конвейер для переноса управляющей информации, с помощью которой каждому каскаду сообщается, что делать.

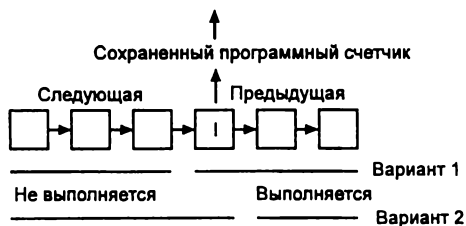


Рис. 6.18. Варианты правильной обработки прерываний (с любезного согласия Jones and Bartlett Publishers).

Чтобы получить правильную обработку прерываний в конвейерном процессоре, следует обеспечить выполнение трех условий [6]:

1. Команды, которые предшествуют сохраняемому значению программного счетчика, должны быть полностью завершены, а состояние процессора должным образом модифицировано.
2. Команды, которые следуют за сохраненным значением программного счетчика, не должны быть выполнены; они не должны также модифицировать состояние процессора.
3. Команда, на которую указывает сохраненный программный счетчик, должна удовлетворять одному из условий 1 или 2.

Как удовлетворить этим трем условиям правильной обработки прерываний? На рисунке 6.18 показаны два варианта проектирования: вариант 1 и вариант 2. Имейте в виду, что обработка разных типов прерываний может быть реализована по-разному; единство способов обработки не требуется и не достигается в большинстве процессоров. Например, обнаружение на этапе декодирования неправильного кода операции приводит к сохранению относящегося к нему значения программного счетчика, но команда выполнена не будет (вариант 2). Если же на этапе декодирования будет обнаружена команда контрольного останова, то содержимое программного счетчика будет сохранено, а команда выполнена (вариант 1).

Невыполненные команды, следующие за прерванной командой, могут обслуживаться по-разному. Если команды приводят к ранним побочным результатам, как это имеет место при использовании автоинкрементного режима адресации в архитектуре VAX, эти побочные результаты должны быть 1) отменены, а команды выполнены заново после обработки прерывания, или 2) сохранены, а команды запущены заново после обработки прерывания. Для процессоров, в которых ранние побочные результаты не наблюдаются, последующие команды могут быть отброшены и выполнены заново после завершения работы обработчика прерывания. Теория RISC-проектирования требует, чтобы ранних побочных результатов не было, что позволяет использовать для обслуживания последующих команд этот простой подход.

В большинстве процессоров предыдущим командам разрешается выполниться до конца. Другими словами, предыдущие команды «прогоняются» сквозь конвейер. Эта стратегия дает хорошие результаты до тех пор, пока одна из команд, прогоняемых сквозь конвейер, сама не генерирует прерывание. Что случится, если прогоняемая команда создаст прерывание? Проектировщик должен учитывать возможность такого события; в некоторых процессорах в этом случае возбуждается сигнал ошибки, указывающий на ситуацию, с которой процессор не может справиться.

К другим событиям, требующим особого рассмотрения, относится одновременное возникновение двух или более прерываний, если, например, неправильный код операции зафиксирован в том же такте, что и внешнее прерывание. И для таких случаев проектировщик должен рассмотреть все возможные варианты и предусмотреть аппаратные средства их разрешения или возбуждение сигнала ошибки.

6.5. Суперскалярные конвейеры

В предыдущих разделах рассматривались конвейеры, в которых команды декодировались и направлялись в узел выполнения по очереди. В этом случае CPI для стационарного состояния равен 1. Разумно задать вопрос: «Почему бы не декодировать и не отправлять в узел выполнения более одной команды за раз, получив тем самым CPI меньше 1?» Именно такой подход и используется в реализациях, называемых суперскалярами. Идея суперскаляра проиллюстрирована на рис. 6.19. В верхней части рисунка показан базовый конвейер, в который между

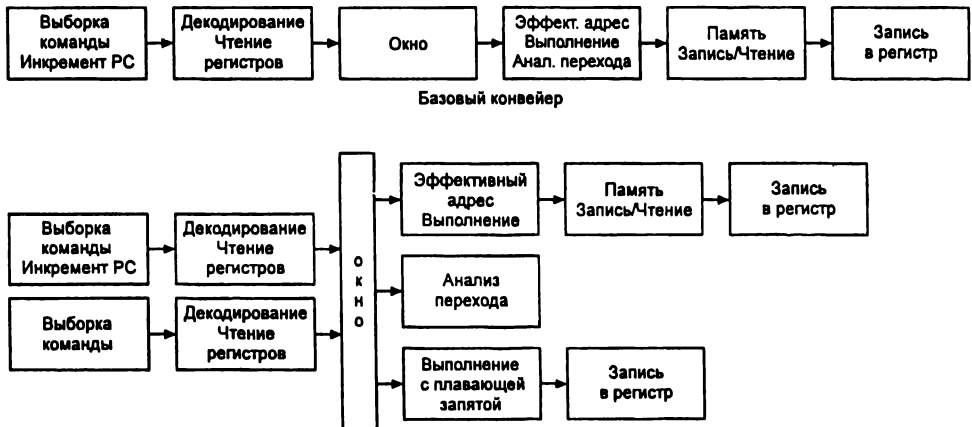


Рис. 6.19. Суперскалярная топология.

каскадами декодирования и выполнения вставлено окно. С точки зрения базового конвейера это окно представляет собой просто регистр на выходе каскада декодирования.

Суперскалярная архитектура второй степени показана в нижней части рис. 6.19. В ней имеются идентичные конвейерные каскады для выборки и декодирования команды; однако инкремент РС выполняется только в одном из них. Степень дублирования каскадов, расположенных перед окном (называемым также узлом диспетчеризации), обозначает степень суперскаляра s). На рисунке показаны узел целочисленного выполнения, узел выполнения переходов и узел выполнения плавающей запятой. Окно может передавать ноль, одну, две или три команды за один такт. Однако средняя скорость передачи не должна превышать двух команд за такт. Поскольку конвейерные каскады, установленные за окном, не являются просто копиями базового конвейера, эта схема носит название неоднородного суперскаляра. Примеры современных процессоров, содержащих неоднородные суперскаляры, перечислены в табл. 6.8.

Разработаем теперь модель производительности суперскалярных процессоров. Таблицы резервирования для этой модели показаны на рис. 6.20. Они относятся к конвейеру, состоящему из четырех каскадов: выборки команд, декодирования и чтения регистров, выполнения и записи в регистры — причем каждый из этих каскадов дублирован⁶⁷⁾. Для данного примера $s = 4$, $\sigma = 2$ и $k = 4$.

Таблица 6.8. Примеры суперскалярных топологий

Процессор	σ	Узлы выполнения
RS/6000	4	Целые, плавающая запятая (умножение, сложение), переход, ввод-вывод
PowerPC 601	3	Целые, плавающая запятая (умножение, сложение, деление), переход
Intel Pentium	2	Двойные-целые, плавающая запятая (сложение), плавающая запятая (умножение), переход
Intel i860	2	Целые-переходы, плавающая запятая (сложение), плавающая запятая (умножение)
AMD-K5	4	Целые, целые/сдвиг, плавающая запятая, переход, загрузка-запоминание
MIPS-T5	4	Двойные-целые, плавающая запятая (сложение), плавающая запятая (умножение/деление/квадратный корень), загрузка-запоминание
MC88110	2	Двойные-целые-переход, целые (умножение), целые (деление), плавающая запятая (сложение), загрузка-запоминание, битовое поле, двумерная графика

⁶⁷⁾ Однородный суперскаляр такого вида не используется на практике. Однако он удобен в качестве основы для разработки модели производительности.



Рис. 6.20. Модель производительности суперскаляра.

Поскольку команды в кэше команд не обязательно выровнены, следует рассмотреть два случая выравнивания. В случае а команды выровнены и число тактов, требуемое для выполнения программного фрагмента, равно $s + (k/\sigma) - 1$. Если две первые команды не выровнены (случай б), число тактов составит $s + k/\sigma$. Если вероятность выравнивания равна $1/\sigma$, взвешенное среднее число тактов составит

$$\begin{aligned}
 &= \frac{1}{\sigma} \left(s + \frac{k}{\sigma} - 1 \right) + \left(1 - \frac{1}{\sigma} \right) \left(s + \frac{k}{\sigma} \right) = \\
 &= s + \frac{k - 1}{\sigma}
 \end{aligned}$$

CPI для суперскалярного процессора степени $s = 2$ равно

$$\text{CPI} = \frac{s + \frac{k - 1}{\sigma}}{k} = \frac{1}{\sigma} + \frac{1}{k} \left(s - \frac{1}{\sigma} \right)$$

Положив $\sigma = 1$, мы получим модель, разработанную ранее для обычного, не суперскалярного ($\sigma = 1$) процессора. Повышение быстродействия суперскалярного процессора по отношению к несуперскалярному составит

$$\begin{aligned} \text{повышение быстродействия} &= \frac{\text{такты, несуперскалярный}}{\text{такты, суперскалярный}} = \frac{s+k-1}{s+\frac{k-1}{\sigma}} = \\ &= \frac{\sigma(s+k-1)}{\sigma s+k-1} \end{aligned}$$

Если $k \rightarrow \infty$, повышение быстродействия $\rightarrow \sigma$.

Если $\sigma \rightarrow \infty$, повышение быстродействия $\rightarrow 1 + \frac{k-1}{s}$

Обратите внимание на то, что для больших значений повышение быстродействия ограничено задержками, задаваемыми параметром k , и длиной конвейера s . Другими словами, повышение быстродействия суперскалярного процессора в конечном счете ограничивается локальным параллелизмом, присущим программе. Процессор не может обрабатывать параллельно больше команд, чем получает их из программы. Эта зависимость показана на рис. 6.21. При увеличении σ повышение быстродействия возрастает линейно до точки, где параллелизм уровня команд (instruction level parallelism, ILP) ограничивает его дальнейший рост.

Во многих случаях ILP находится в диапазоне 2–4. Именно поэтому для многих современных суперскалярных процессоров характерно значение σ в диапазоне 2–4, как это показано в табл. 6.8. Действительно, если предполагаемые программы имеют относительно небольшое значение ILP, нет смысла увеличивать σ , что потребует дополнительной площади кристалла.

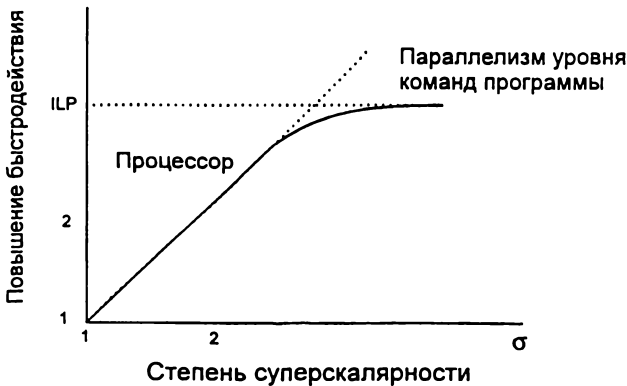


Рис. 6.21. Повышение быстродействия, ограниченное σ и ILP.

✎ Пример

Предположим, что программный фрагмент, использованный в гл. 3 для иллюстрации архитектурного регистрового файла загрузки-запоминания (табл. 3.3), выполняется на суперскалярном процессоре, изображенном на рис. 6.19. Каково значение ILP этой программы и каким будет повышение быстродействия этого процессора по отношению к несуперскалярному? Считаем, что кэш-промахи отсутствуют.

✍ Решение

Архитектура процессора трехадресная, включает команды регистра-регистр и загрузки-запоминания. Программный фрагмент, требуемый для выполнения функции, имеет такой вид:

Загрузка: $R(1) \leftarrow M(B)$

Загрузка: $R(2) \leftarrow M(C)$

Сложение: $R(3) \leftarrow R(1) + R(2)$

Запоминание: $M(A) \leftarrow R(3)$

Суперскалярный процессор использует целочисленный конвейер для операций загрузки-запоминания, а также операций ALU, что является потенциальным источником структурных конфликтов. При двух целочисленных конвейерах две команды загрузки могут быть выданы одновременно; однако команда запоминания должна следовать за командой сложения из-за RAW-взаимозависимости. Можно заключить, что при отсутствии приостановок и задержек этот фрагмент из четырех команд потребует трех тактов. Минимальное значение CPI составит $3/4 = 0,75$; однако CPI может оказаться хуже из-за потерь времени на разрешение RAW-взаимозависимостей и структурных конфликтов в системе памяти.

✓ Замечание

RAW-взаимозависимости по R1 и R2 носят название истинных взаимозависимостей загрузки. Эти взаимозависимости обычно приводят к задержкам приостановки даже при реализации продвижения данных.

Обратите внимание на то, что, как это видно из табл. 6.8, в процессорах AMD-K5, MIPS-T5 и MC88110 в дополнение к целочисленным конвейерам предусмотрены конвейеры загрузки-запоминания. В результате в некоторых программах операции загрузки и запоминания в другой команде могут выдаваться из окна в одном такте.

6.6. Конвейерные процессоры и требования, предъявляемые к памяти

Конвейерный процессор предъявляет к системе памяти серьезные требования, связанные с операциями выборки команд и данных. В силу параллелизма в работе конвейера в одном такте могут быть затребованы одна выборка команды и до одного чтения или записи данных (см. рис. 6.22). Для суперскалярных процессоров эти требования умножаются на величину s . Чтобы удовлетворить таким требованиям, в современных процессорах предусматриваются разделенные кэши, расширенные кэши команд для их выборки и буферы записи, позволяющие изолировать операции записи и чтения данных.

Кроме того, в архитектурах RISC-типа регистровый файл должен быть многопортовым, чтобы поддерживать одновременно выполняемые операции записи и чтения, как это показано на рис. 6.23. В двухместных операциях на каждую команду приходится два чтения и одна запись. Для суперскалярных процессоров порты для регистров должны в худшем случае поддерживать $\sigma \times (2 \text{ чтения} + 1 \text{ запись})$ регистровых циклов на один такт.

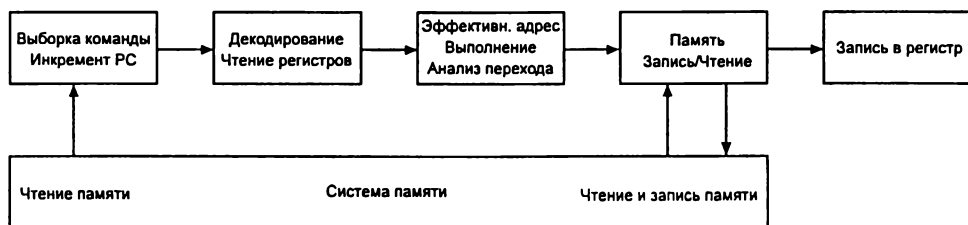


Рис. 6.22. Запросы к памяти.

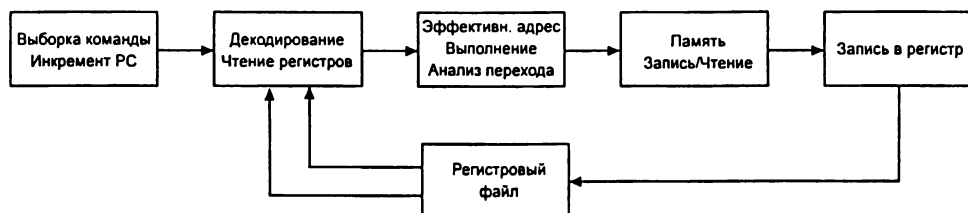


Рис. 6.23. Обращения к регистровому файлу.

 Пример

Рассмотрим суперскалярный процессор с $\sigma = 2$. Рабочая нагрузка характеризуется следующими цифрами: 30% всех команд принадлежат к ALU-типу, 30% — загрузка и запоминание, остальные 40% команд не используют регистровый файл. Каково взвешенное среднее число обращений к регистровому файлу за 1 такт?

 Решение

События и их вероятности собраны в табл. 6.9. Взвешенное среднее число обращений к регистрам составит $1,2 \sigma$ и для $\sigma = 2$ взвешенное среднее число обращений к регистровому файлу за 1 такт равно 2,4.

Таблица 6.9. Обращения к регистровому файлу за 1 такт

Событие	Обращения	Вероятность	Произведение
Команда ALU	3σ	0,3	$0,9 \sigma$
Загрузка или запоминание	σ	0,3	$0,3 \sigma$
Другие	0	0,4	0
Сумма = 1			$1,2 \sigma$

 Замечание

Значение 2,4 обращений к регистровому файлу за 1 такт является оценкой худшего случая. В силу параллелизма обработки обращения не будут выполняться одновременно, как это предполагается в приведенном решении. Модельные испытания подтверждают это соображение: для процессора $\sigma = 4$ приблизительно 90% всех тактов имеют три или менее обращений к регистровому файлу [3].

Проектировщик, устраняя структурные конфликты, может попытаться вместо организации многопортового регистрового файла предусмотреть его дублирование. В таком случае следует обеспечить согласованность регистровых файлов, что может потребовать использования большей поверхности кристалла и привести к снижению быстродействия по сравнению с многопортовым регистровым файлом.

Литература

1. Cragon, H. G. (1991). *Branch Strategy Taxonomy and Performance Models*, IEEE Computer Society Press, Los Alamitos, CA.
2. Davidson, E. S. (1971). «The Design and Control of Pipelined Function Generators», presented at the IEEE Conference on Systems, Networks and Computers, Oaxtepec, Mexico, January 1971.

3. Johnson, M. (1991). *Superscalar Microprocessor Design*, Prentice-Hall, Englewood Cliffs, NJ.
4. Kogge, P. M. (1981). *The Architecture of Pipelined Computers*, Hemisphere, New York.
5. Lee, R. L. (1984). «Branch Prediction Strategies and Branch Target Buffer Design», *Computer*, 21:1.
6. Smith, J. E. and Pleszlumn, A. R. (1985). «Implementation of Precise Interrupts in Pipelined Processors», *IEEE Transactions on Computers*, C-34:3, pp. 36–44.

Упражнения

- 6.1. Сколько случаев выравнивания следует рассмотреть при моделировании суперскалярного процессора с $\sigma = 3, 4$ или 5 ?
- 6.2. Покажите, что модель CPI, разработанная для $\sigma = 2$ в соответствии с табл. 6.20, дает общее решение для любого σ .
- 6.3. Покажите с помощью таблиц резервирования, что RAW-задержки при использовании продвижения данных составляют $E - 1$.
- 6.4. Каковы значения RAW-задержки для различных значений E и расстояния между командами с взаимозависимостями? Дайте ответ в табличной форме.
- 6.5. Разработайте модель WABD для стратегии задержанных переходов.
- 6.6. Покажите все шаги в вычислении оптимального числа каскадов конвейера.
- 6.7. Рассмотрите общую кэш-память уровня L2, которая получает промахи от кэш-памяти уровня L1 для команд и данных. Считайте, что число коррелированных обращений равно произведению вероятностей двух промахов: $P_{\text{miss}}(\text{команды}) \times P_{\text{miss}}(\text{данные})$. Предположите, что ISA имеет одну команду и одно обращение к данным на команду. Если интенсивность промахов для обеих L1-кэшей равна 0,1, каково среднее время обращения к общему кэшу?
- 6.8. С помощью таблиц резервирования покажите, что задержка продвижения данных для соседних команд составляет $E - 1$ тактов.
- 6.9. Опишите ситуацию, в которой каскад конвейера записи в регистр может вызывать прерывание или оказаться прерываемым каскадом.
- 6.10. Выведите формулу для определения числа тактов $s + (k - 1)/\sigma$ для суперскалярного процессора степени 2 с вероятностью выравнивания $1/\sigma$.
- 6.11. Какие условия или ситуации могут возникнуть, если из окна на рис. 6.19 не выдано ни одной команды?
- 6.12. Для суперскалярного процессора, изображенного на рис. 6.19, рассмотрите последовательность из двух команд: загрузка плавающей запятой, сложение плавающей запятой

$$i \quad R_x = M(R_y)$$

$$j \quad R_z = R_x + R_w$$

Покажите путь продвижения данных, который повысит производительность этого программного фрагмента. Объясните, почему повышается производительность.

6.13. Установите все пути продвижения данных в суперскалярном процессоре, изображенном на рис. 6.19, которые позволят снизить потерю производительности, возникающую в силу RAW-зависимостей.

6.14. Для базового конвейера $s'' = 2$, $s' = s = 3$ вычислите CPI для трех стратегий переходов: замораживания, предсказания выполнения, предсказания невыполнения. Считайте, что параметры рабочей нагрузки составляют $P_b = 0,4$ и $P_{bt} = 0,6$.

6.15. Модифицируйте пример раздела 6.3.2, включив в него кэш-промахи. Одноуровневый общий кэш с прямым отображением имеет объем 8 Кбайт. Разделенные кэши команд и данных с прямым отображением имеют объем по 4 Кбайт. Используйте значения для P_{miss} из табл. 4.8 – 4.10. Транспортное время передачи из реальной памяти в общий кэш составляет 10 тактов. Воспользуйтесь рис. 4.10, где были приведены топологии одноуровневого общего кэша и одноуровневого разделенного кэша.

6.16. Вы модернизируете проект ВТВ процессора и можете увеличить число битов на 50%. Как вы используете эти дополнительные биты? Увеличите число входов, тем самым уменьшив интенсивность промахов? Увеличите размер таблицы истории, чтобы повысить точность предсказаний? Сделаете что-нибудь другое? Обоснуйте свой ответ, вычислив уменьшение значения WABD.

6.17. Разработайте вариант модели WABD для ВТВ (см. табл. 6.7). Измените случай по умолчанию, когда имеется ВТВ-промах при предсказании выполнения и при предсказании невыполнения.

6.18. Измените пример, приведенный после табл. 6.7, используя по умолчанию стратегии РТ и PNT. Какую стратегию по умолчанию вы предпочтете? Обоснуйте свой ответ.

6.19. Проанализируйте взвешенное среднее число обращений к регистровому файлу на такт при продвижении по базовому конвейеру ($\sigma = 1$) приведенного ниже потока команд. Поток команд: загрузка, ALU, загрузка, другая, ALU, другая, другая, другая, ALU, запоминание. Вычислите долю команд, имеющих 0, 1, 2 и 3 обращений к регистровому файлу. Как эти результаты согласуются с данными табл. 6.3?

6.20. Разделенные кэши команд и данных устраняют структурные конфликты при обращениях к памяти. Обдумайте требования поддержания согласованности этих двух кэшей.

ГЛАВА 7

ВВОД-ВЫВОД

7.0. Введение

Процессоры нуждаются в источнике входных данных и приемнике выходных. Кроме этого, процессорам требуется запоминающая среда для хранения больших объемов промежуточных данных. Система ввода-вывода (I/O от input-output) удовлетворяет требования процессора, подключая память процессора к устройствам ввода-вывода. Эти устройства широко различаются по запаздыванию и пропускной способности; поэтому важнейшей функцией системы ввода-вывода является согласование характеристик устройств ввода-вывода с системой памяти. В перечень устройств ввода-вывода входят такие разные объекты, как клавиатуры, модемы и жесткие диски. Для каждого из этих устройств характерны свои требования по запаздыванию и пропускной способности, которые должны быть удовлетворены для правильного функционирования системы.

В работе [1] дана классификация устройств ввода-вывода, воспроизведенная в модифицированной форме на рис. 7.1. Запоминающие устройства предоставляют пространства памяти с быстрым доступом. Системно-резидентные устройства и базы данных находятся обычно в виртуальном адресном пространстве и предоставляют память с малым запаздыванием. Архивные запоминающие среды, напротив, обычно требуют для своей работы участия человека, который активизирует их при возникновении необходимости в получении или запоминании архивной информации.



Рис. 7.1. Классификация устройств ввода-вывода.

Устройства получения-отправления обычно используются как средства коммуникации компьютера с другими устройствами и, как правило, не могут хранить информацию. Дискеты и zip-диски можно отнести как к архивным, так и к локальным средам. Система Windows98 может использовать программу Internet Explorer в качестве средства доступа и к запоминающим устройствам, и к устройствам получения-отправления, размывая различия в классификации устройств.

7.1. Архитектура системы ввода-вывода

Система ввода-вывода должна предоставлять средства для передачи информации между памятью процессора и различными устройствами. Природа этого интерфейса ведет к серьезным сложностям проектирования. Соотношение пропускной способности памяти и устройств может колебаться в широких пределах — от, скажем, 5000000:1 до 1:1. Например, память может иметь пропускную способность до 50 Мбайт/с, в то время как клавиатура выдает символы со скоростью приблизительно 10 байт/с. С другой стороны, графические дисплеи могут принимать данные со скоростью около 30 Мбайт/с. Очень велик также и диапазон различий времени запаздывания — от 500000:1 до 1:1. Примером могут служить устройства памяти (70 нс) и диски (20 мс). Различия в запаздывании и пропускной способности приводят к необходимости организации буферов. Эти буферы могут быть реализованы аппаратно или помещены в реальную память соответствующим программным обеспечением.

Устройства ввода-вывода обычно асинхронны по отношению к тактам процессора и памяти, а также и к программам, выполняемым процессором. Асинхронность требует наличия синхронизаторов аппаратного уровня, которые будут описаны в разделе 7.5, а также синхронизирующих примитивов в ISA процессора, которые обсуждаются в гл. 8.

Основные задачи системы ввода-вывода заключаются в следующем:

1. Установить связь между памятью и устройством ввода-вывода.
2. Синхронизировать передачу данных и организовать управление ею.
3. Предоставить буферизацию для тех случаев, когда пропускные способности и запаздывание устройств получения и отправления различаются.
4. Выполнить при необходимости преобразование кодов.
5. Прервать при необходимости работу процессора.
6. Завершить выполненную успешно операцию.

Эти задачи решаются как аппаратно, так и программно в различных пропорциях. При увеличении аппаратной составляющей про-

пуская способность и стоимость возрастают, а при увеличении программной составляющей — снижаются. Управление внешними устройствами можно разбить на две большие категории: программный ввод-вывод и сопроцессорный ввод-вывод^{*)}.

Программный ввод-вывод

Программный ввод-вывод, который также называют непосредственным, организуется выполняемой процессором программой, которая берет на себя все управление операциями ввода-вывода и передачи данных. Система программного ввода-вывода, показанная на рис. 7.2, включает несколько архитектурных регистров (по меньшей мере, входной и выходной регистры), которые адресуются с помощью специальных команд пересылки данных. Эти команды передают информацию от аккумулятора (или другого регистра общего назначения) в (из) регистры ввода-вывода. Периферийное устройство подключается непосредственно к битам этих регистров.

В регистрах ввода-вывода выделяются биты для сигналов управления, идущих как в устройство, так и от него, и для передачи данных. Некоторые реализации включают специальный регистр состояния, который хранит состояние устройства и биты управления им, так что биты управления не смешиваются с битами данных. В реализацию может также входить регистр-счетчик, отсчитывающий число пересылок, чтобы не надо было использовать процессор для организации счета под управлением программы.

Когда главной программе требуется операция ввода-вывода, она приостанавливается и вызывается подпрограмма ввода-вывода. Эта подпрограмма выполняет побитовое управление и пересылку данных между памятью и устройством. После завершения пересылки управ-



Рис. 7.2. Программный ввод-вывод.

^{*)} Под этим термином автор понимает специальные аппаратные узлы (возможно, включающие в себя специализированные процессоры), служащие для управления внешними устройствами. — *Прим. перев.*

ление возвращается из подпрограммы ввода-вывода в приостановленную главную программу. В некоторых ситуациях программа ввода-вывода включается в текст основной программы в виде встроенного кода.

Наиболее серьезной проблемой при организации программного ввода-вывода является невозможность параллельного выполнения ввода-вывода и программы. На время выполнения подпрограммы ввода-вывода главная программа приостанавливается, что приводит к потере производительности.

Пример

Рассмотрим процессор с программным вводом-выводом, выполняющим команды со скоростью 0,1 MIPS. Подпрограмма ввода-вывода для записи на дискету требует 10000 команд, а дискета характеризуется полным запаздыванием 100 мс. Дисковые запросы инициируются по ходу программы каждую секунду. Какова эффективность всей системы?

Решение

Главная программа приостанавливается на 100 мс плюс время выполнения подпрограммы ввода-вывода:

$$\begin{aligned}\text{время приостановки} &= 100 \times 10^{-3} \text{ с} + \frac{10000}{100000} \text{ с} = \\ &= 100 \times 10^{-3} \text{ с} + 100 \times 10^{-3} \text{ с} = \\ &= 200 \times 10^{-3} \text{ с}\end{aligned}$$

$$\begin{aligned}\text{эффективность} &= 100 \times \frac{\text{время задачи}}{\text{полное время}} = \\ &= 100 \times \frac{1 \text{ с}}{1 \text{ с} + 200 \text{ мс}} = \\ &= 83\%\end{aligned}$$

Замечание

Многие современные однокристальные микрокомпьютеры и контроллеры используют программный ввод-вывод. Потеря эффективности часто не имеет значения, поскольку основным фактором проектирования является возможно более низкая стоимость. При помещении подпрограммы управления вводом-выводом в весьма плотное ПЗУ полная площадь кристалла уменьшается по сравнению с реализацией других способов управления вводом-выводом.

Программный ввод-вывод использовался на первом компьютере, построенном на базе фон-неймановской ISA — машине EDSAC — в Кембриджском университете под руководством Мориса Уилкса [8]. Вспомним из гл. 3, что EDSAC имел средства вызова подпрограмм и возврата из них. В качестве среды ввода данных использовалась перфолента, а в качестве выходного устройства телетайп. Команды ввода и вывода были включены в ISA:

- I, n *Считать следующую строку отверстий на входной ленте и поместить результирующее целое число, умноженное на 2^{-16} , в ячейку памяти n.*
- O, n *Напечатать символ, посланный ранее на телетайп, и послать на телетайп следующий символ, представленный пятью старшими разрядами в ячейке памяти n.*⁶⁸⁾

IBM 701, один из ранних коммерческих компьютеров, появившийся на рынке в 1953 г., использовал программный ввод-вывод. В этом компьютере в качестве запоминающей среды были установлены четыре магнитных барабана; устройствами ввода-вывода служили карточный считыватель, карточный перфоратор, принтер, магнитная лента и графический дисплей с фотоаппаратом для получения документальной копии.

Команды ввода-вывода компьютера 701 имели более высокий лексический уровень, чем в случае EDSAC. Две команды — выбор чтения (read select, RDS) и выбор записи (write select, WRS) — устанавливали путь к адресуемому устройству и активизировали само устройство (например, запускали считыватель с магнитной ленты). Другие команды выполняли операции чтения или записи. Все передачи шли через регистр MQ множителя-частного:

- CPY (copy and skip) — копирование и пропуск,
- WEF (write end of file) — запись конца файла,
- REW (rewind tape) — перемотка ленты,
- RDB (read backwards) — чтение назад,
- SNS (sense) — опознание,
- LDA (locate drum address) — найти адрес на барабане.

В процессоре Intel 8080 использовались простые команды ввода IN и вывода OUT. Эти команды обращались к линиям шины памяти и перемещали байт между аккумулятором и линиями шины данных, выставляя одновременно байтовый адрес на адресных линиях шины памяти. В микросхемах специального назначения были реализованы более высокие уровни ввода-вывода, такие как буферы и каналы. Среди линий шины были предусмотрены три бита состояния, с помощью

⁶⁸⁾ Обратите внимание на функцию команды вывода «CISC-стиля»: напечатать один символ, установить второй символ.

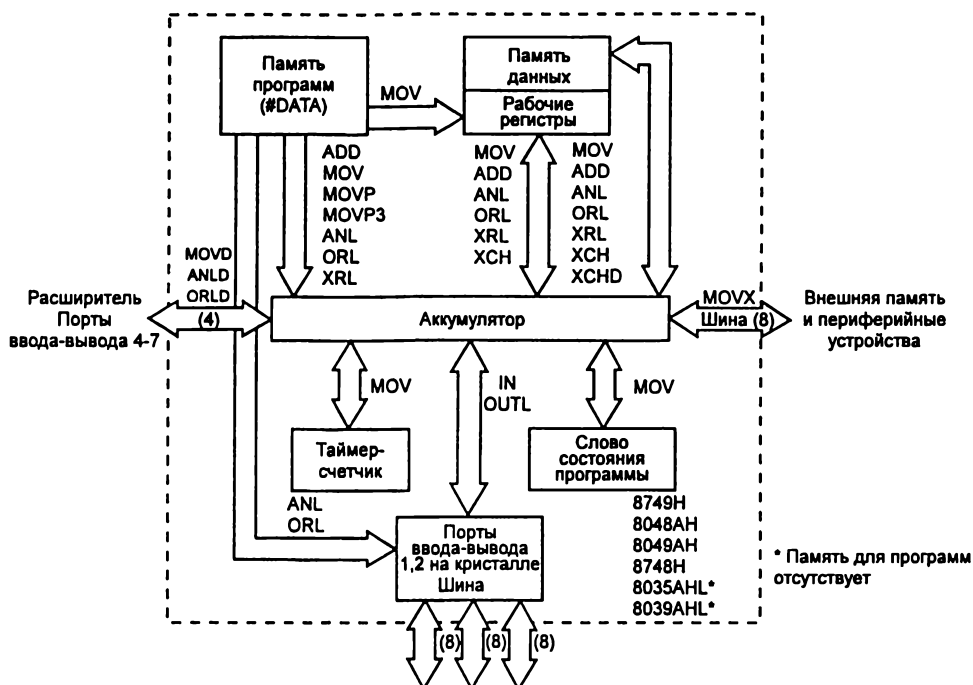


Рис. 7.3. Блок-схема микроконтроллера Intel 8048 (с любезного разрешения Intel Corporation).

декодирования которых можно было определить, что выполняется не операция с памятью, а ввод-вывод: 101 — IO-запись, 110 — IO-чтение.

Для управления простыми устройствами вроде микроволновых печей, контроллеров клавиатуры или игр с ручным управлением широко используются однокристалльные микроконтроллеры, например Intel i8048, изображенный на рис. 7.3. Эти компьютеры используют программный ввод-вывод с пересылкой либо битов, либо байтов.

В процессоре Intel 8048 предусмотрены три 8-битовых порта ввода-вывода и 8-битовая шина. ISA включает некоторое число команд, выполняющих перемещение данных между портами и аккумулятором. Выходы могут быть замаскированы путем установки или сброса битов непосредственными операндами.

Ради совместимости снизу вверх эти команды **IN** и **OUT** включаются и в современные процессоры типа Pentium Pro. Эти процессоры допускают использование в качестве приемника или источника любых из регистров **AL, AX** или **EAX**, в зависимости от размера адресуемого порта (8, 16 или 32 бит). Windows 95 использует программный ввод-вывод для реализации некоторых из своих функций ввода-вывода.

Ввод-вывод, отображенный на память

Ввод-вывод, отображенный на память, представляет собой другую форму программного ввода-вывода, при котором адреса устройств отображаются на адресное пространство памяти, как это показано на рис. 7.4. Командой вывода в этом случае служит обычная команда записи, которая заносит требуемое сочетание битов по выделенному адресу памяти. Команда ввода представляет собой обычную команду загрузки, которая считывает содержимое по адресу памяти.

Главным преимуществом ввода-вывода, отображенного на память, по сравнению с программным вводом-выводом^{*)} является то, что он не требует ни специальных команд пересылки, ни специальных регистров ввода-вывода^{**)}. Главный недостаток — выделение некоторой области адресного пространства, которая должна трактоваться специальным образом как аппаратурой, так и разработчиками программного обеспечения. Для разработчиков аппаратуры биты, отображаемые на память, должны быть реальными триггерами, а их адреса в адресном пространстве памяти заблокированы для дешифратора адресов реальной памяти. С точки зрения программирования любой вариант проекта, противоречащий однородному распределению адресов памяти, проблематичен.

Ввод-вывод, отображаемый на память, требует полного участия процессора, как и в случае программного ввода-вывода. Таким образом, здесь также имеет место потеря эффективности, связанная с выполнением программы управления вводом-выводом.

Процессор Intel Pentium Pro поддерживает как программный ввод-вывод, так и ввод-вывод, отображаемый на память. Любые команды процессора, служащие для обращения к памяти, могут использоваться для адресации отображенных на память адресов ввода-вывода.

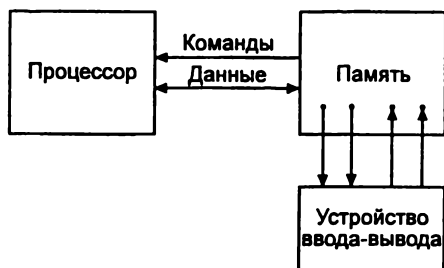


Рис. 7.4. Ввод-вывод, отображенный на память.

^{*)} Через порты. — Прим. перев.

^{**) Разумеется, специальные регистры или ячейки внешней памяти, входящие в состав внешнего устройства, требуются и в том, и в другом случаях. — Прим. перев.}

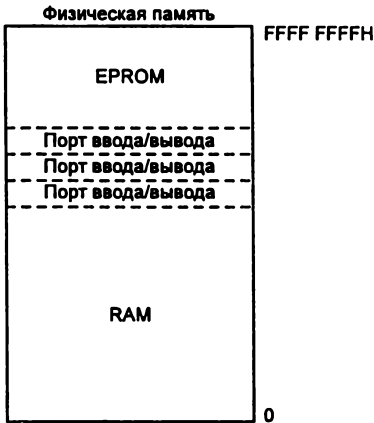


Рис. 7.5. Ввод-вывод, отображенный на память, процессора Pentium Pro (с любезного разрешения Intel Corporation).

Пример распределения адресного пространства приведен на рис. 7.5. Команда пересылки может передавать данные между любыми регистрами процессора и областями памяти, выделенными под ввод-вывод. Другие команды, например ADD, AND, OR или TEST, могут также иметь один из аргументов-источников в пространстве ввода-вывода. Логические операции полезны для управления отдельными битами в регистрах управления и состояния устройства ввода-вывода, которые отображены на один или несколько адресов памяти.

Отображение на память ввода-вывода усложняет процедуру кэширования. Как можно кэшировать устройства, подключенные к пространству ввода-вывода? В общем случае ввод-вывод отображается на некэшируемую область памяти. Это пространство не кэшируется по двум причинам: во-первых, промахи кэш-страниц приведут к некоторой неопределенности моментов ввода-вывода, и во-вторых, быстрые запись и чтение не столь важны для устройств ввода-вывода, как для команд и данных процессора.

Отсутствие кэширования адресов, отображенных на память, в процессоре Pentium Pro обеспечивает выполнение операций записи по порядку (этот вопрос обсуждался в разделе 6.3.3). Если две команды, управляющие периферийным устройством через пространство памяти, завершатся не в том порядке, в каком они расположены в программе, последовательность данных, посылаемая в устройство, окажется неверной, что приведет к неправильной работе устройства. Например, биты команды могут попасть в устройство после битов данных.

Некэшируемые адреса устанавливаются с помощью набора регистров типа памяти (MTRR), которые задают границы областей ввода-вывода. Тем самым регистры MTRR отображают пространство ввода-вывода на некэшируемые адреса.

Поскольку IBM S360 не использует ввод-вывод, отображенный на память, для этого процессора проблемы записи в память в неправильном порядке не затрагивают работу внешних устройств. Однако в других случаях правильный порядок записи может понадобиться, и для его обеспечения в процессоре предусмотрены соответствующие средства. Процессор PowerPC, как уже отмечалось в разделе 6.3.3, для обеспечения правильного порядка записи в память использует команду `eieio`.

Прерывания ввода-вывода

В приведенном выше примере, посвященном эффективности программного ввода-вывода, 50% времени простоя программы пришлось на ожидание реакции дискеты. Ввиду столь низкой эффективности вы почти не встретите в высокопроизводительных системах примеры чисто программного или отображенного на память ввода-вывода. Если, однако, работать в режиме прерываний, то большая часть времени простоя может быть использована для вычислений. Этот режим использовался еще такими ранними компьютерами, как Univac 1103 и IBM 701.

На рисунке 7.6 показано действие прерываний на примере устройства ввода-вывода, отображаемого на память. Процессор посылает устройству команду начать операцию (обычно с помощью программно доступного регистра ввода-вывода). Поставив запрос на ввод-вывод, процессор возвращается в главную программу. Когда истечет время запаздывания и устройство будет готово к передаче, оно посылает в процессор сигнал прерывания. Процессор обслуживает прерывание, выполняя передачу данных. При этом устройство посылает в процессор различную информацию о состоянии с помощью каналов программного ввода-вывода.

Прерывания могут быть использованы и с программным вводом-выводом, и с вводом-выводом, отображенным на память; в процессоре Intel Pentium Pro реализованы оба способа. Детали обслуживания прерываний были описаны в гл. 5 и 6.

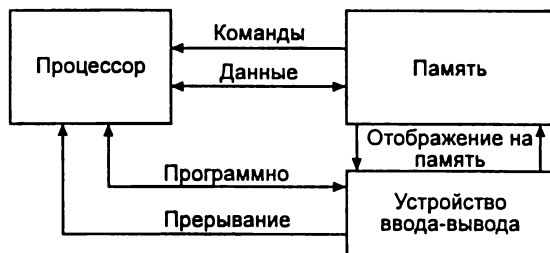


Рис. 7.6. Ввод-вывод с прерываниями.

Обратите внимание на то, что система прерываний предоставит возможность выполнить дополнительные вычисления, только если эти дополнительные вычисления требуются, или если запрос на ввод-вывод может быть поставлен задолго до того момента, когда он будет реализован. Одно или оба эти условия могут выполняться в мультипрограммных системах, которые обсуждались в гл. 4, но не на однопользовательских рабочих станциях.

✎ Пример

Дополним пример программного ввода-вывода прерыванием. Инициация дисковой передачи требует 100 команд, а отклик на прерывание в конце интервала запаздывания диска требует еще 100 команд. Какова эффективность системы?

✍ Решение

Запаздывание длительностью 100 мс может быть использовано для выполнения задач полезной работы. Однако полное время увеличится, так как необходимо выполнить 200 команд, связанных с вызовом подпрограммы и выходом из нее:

$$\text{полезное время} = 1c + 0,1c = 1,1c$$

$$\text{полное время} = 1,2c + \frac{200}{100000}c = 1,202c$$

$$\text{эффективность} = 100 \times 1,1/1,202 = 91,5\%$$

✓ Замечание

Мы видим, что использование прерываний повысило эффективность с 83% до 91,5%. Однако в некоторых случаях может оказаться, что время выполнения команд, добавленных для обслуживания прерываний, будет больше, чем время, сэкономленное при наложении вычислений на интервал запаздывания устройства.

Сопроцессорный ввод-вывод

Если для управления вводом-выводом и пересылки данных используется процессор, то это сопровождается потерей эффективности. В результате естественным образом возникает идея использовать для управления вводом-выводом второй процессор, или сопроцессор. Автономный сопроцессор впервые появился в машине IBM 709 в 1959 г.; он выполнял все задачи по вводу-выводу, освободив основной процессор для выполнения присущей ему работы. На рисунке 7.7 показан сопроцессор, соединяющий устройство ввода-вывода с системной

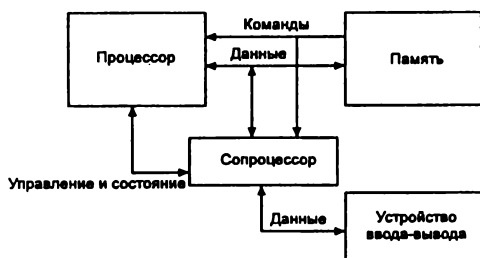


Рис. 7.7. Сопроцессорный ввод-вывод.

шиной. Данные управления и состояния передаются между процессором и сопроцессором по каналам, аналогичным тем, что используются в программном вводе-выводе. Процессор посылает команду в сопроцессор, который затем автономно выполняет операции ввода-вывода.

Включение в систему сопроцессора делает ее мультипроцессорной. Все проблемы, возникающие при проектировании мультипроцессора, характерны и для системы сопроцессорного ввода-вывода. К этим проблемам относятся синхронизация программ, синхронизация аппаратуры, защита памяти и согласованность памяти. Студентам, желающим познакомиться с этими проблемами и их решением, можно рекомендовать прослушать курс по мультипроцессорам.

Существует два основных типа сопроцессоров: с каналом программного ввода-вывода и с прямым доступом к памяти (direct memory access, DMA).

Канал программного ввода-вывода. Канал программного ввода-вывода представляет собой дополнительный процессор, обеспечивающий функции ввода-вывода с помощью процедуры программного ввода-вывода. Этот процессор вместе с ISA специального назначения полностью автономен и может работать параллельно с основным процессором.

Источником команд для канала программного ввода-вывода может быть либо основная память, как это показано на рис. 7.7, либо отдельная память. Пример использования для этой цели основной памяти можно найти в каналах машин IBM 360/370/390. Программы управления каналом помещаются в память операционной системой. Когда процессор приступает к обслуживанию ввода-вывода, в программный счетчик канала загружается указатель, после чего начинается выполнение. Функционирование канала IBM описано ниже в разделе 8.4.

Канал с отдельной памятью использовался в периферийном процессоре CDC 6600. Каждый из периферийных процессоров имеет собственную локальную память программ, хранящую его программу управления. Достоинством отдельной памяти является то, что пропускная способность команд основной памяти не уменьшается в процессе выполнения операции ввода-вывода.

Сравнивая архитектуры команд канала IBM S360/370/390 с периферийным процессором CDC 6600, можно обнаружить, что ISA канала IBM относится к числу архитектур высокого уровня и весьма похожа на CISC-архитектуру. Для периферийного процессора, напротив, характерна ISA низкого уровня типа RISC.

Прямой доступ к памяти (DMA). DMA является вариантом программного ввода-вывода. DMA-сопроцессор получает от операционной системы список параметров, характеризующих передачу данных. В качестве примеров таких параметров можно привести тип передачи, имя устройства, начальный адрес памяти, длину передаваемых данных и различные управляющие параметры. Список параметров может быть передан DMA по системной шине или по выделенным каналам, аналогичным каналам программного ввода-вывода.

DMA обычно реализуется аппаратно в виде конечного автомата. Список параметров используется в качестве входов этого автомата; автомат выполняет управление вводом-выводом и передачу данных согласно списку параметров.

Резюме по методам управления вводом-выводом

В таблице 7.1 собраны основные характеристики обсуждавшихся выше методов управления вводом-выводом с указанием их достоинств и недостатков.

Таблица 7.1. Методы управления

Тип ввода-вывода	Преимущества	Недостатки
Программный	Простота, небольшой объем дополнительной аппаратуры.	Потеря эффективности процессора на выполнении подпрограммы ввода-вывода. Необходимость в специальных регистрах и командах передачи.
С отображением на память	Простота, небольшой объем дополнительной аппаратуры. Отсутствие специальных команд передачи.	Потеря эффективности процессора на выполнении подпрограммы ввода-вывода. Необходимость в специальной адресной логике для адресации памяти.
По прерываниям; программный или с отображением на память	Простота, небольшой объем дополнительной аппаратуры, повышенная эффективность ввиду использования процессором интервала запаздывания устройства.	Программный: необходимы специальные регистры и команды передачи. С отображением на память: необходима специальная адресная логика для адресации памяти.

Таблица 7.1. *Продолжение*

Тип ввода-вывода	Преимущества	Недостатки
Сопроцессор; программный канал или DMA	Полный параллелизм, отсутствие потерь эффективности процессора. Передачи непосредственно в/из памяти, минуя регистры процессора.	Необходим значительный объем дополнительной аппаратуры. Может потребоваться увеличение пропускной способности основной памяти для поддержки дополнительных команд и передач данных.

7.2. Требования к устройствам ввода-вывода

Конкретное применение процессора диктует типы и количество устройств ввода-вывода. Требования к устройствам различаются от почти никаких для специализированных контроллеров до множества для больших универсальных вычислительных машин. Мы рассмотрим два применения процессоров: персональные компьютеры (ПК) и большие вычислительные машины (иногда называемые серверами).

Устройства ввода-вывода для персонального компьютера

Персональный компьютер используется в качестве однопользовательской среды с ограниченной мультипроцессорной активностью. В общем случае при перемещении небольших объемов данных предпочтительны устройства с малым запаздыванием, в то время как при перемещении больших объемов лучше использовать устройства с высокой пропускной способностью. Операционные системы типа Windows 95 или Windows 98 поддерживают и используют виртуальную память. Для того чтобы работать одновременно с большим количеством значительных по объему резидентных программ, требуется относительно большое дисковое пространство на жестком диске. Учитывая широкое использование Интернета, необходима связь с различными коммуникационными системами. Архивация данных и резервное копирование всей системы требуют архивных запоминающих устройств.

Для поддержки графических языков программирования и для сложных графических приложений (как САПР, изготовление чертежей, игр), необходимы высококачественные видеосистемы с большими экранами. Развлекательные системы требуют наличия высококаче-

ственной аудиоаппаратуры и устройств для компакт-дисков (CDROM), поддерживающих локальные базы данных.

Для того чтобы удовлетворить всем этим требованиям, производители ПК предлагают покупателям широкий спектр устройств ввода-вывода. Краткий перечень этих устройств с их характеристиками дан в табл. 7.2, учитывающей приведенную на рис. 7.1 иерархию устройств.

Таблица 7.2. Устройства ввода-вывода персональных компьютеров

Запоминание	Системно-резидентные	Жесткий диск	4–17 Гбайт, 5400 об/мин, среднее время поиска 8,5–9,5 мс
	Базы данных	CDROM	650 Мбайт на диск, 1,8–16 Мбайт/с, среднее запаздывание 75–118 мс
	Архивы	zip-дисковод	100 Мбайт на диск, среднее время поиска 29 мс, 20 Мбайт/с, среднее время старт/стоп 3 с
Получение/отправление	Удаленные	Модем	56 Кбод, стандарт V.90
	Локальные среды	Сканер	Планшет, однопроходный, штриховый, полутонный и цветной, 75, 150, 300, 600 дюйм на пиксел
	Интерактивные	Клавиатура	104 клавиши, скорость данных равна скорости ввода
		Монитор	17 дюйм SVGA, 0,27 мм на шаг точек, разрешение в режиме построчной развертки 1280×1024, горизонтальная синхронизация 30–70 кГц, вертикальная синхронизация 50–100 Гц

Устройства ввода-вывода для серверов

Большие вычислительные машины и серверы предоставляют вычислительные ресурсы большому числу пользователей с помощью терминалов и периодического выполнения больших пакетных заданий. Многие машины такого рода предоставляют не столько возможности для вычислений, сколько базы данных. Типичными применениями серверов являются системы заказа авиационных билетов, проверка кредитных карточек или прием заказов. Такие приложения обычно называют системами обработки транзакций; запрос на транзакцию поступает в базу данных, проводит транзакцию и обновляет базу данных в соответствии с совершенной транзакцией. Системы обработки транзакций обычно представляют собой системы

реального времени в том смысле, что клиент ждет завершения транзакции. Поэтому производительность таких систем обычно измеряется числом транзакций в секунду. На Рождество, например, сервер компании кредитных карточек может принимать до 50000 запросов на транзакцию в секунду.

Для работы серверов требуются в основном внешние устройства двух типов: жесткие диски в качестве системно-резидентной памяти и сетевой доступ для удаленных терминалов. Исследования, проведенные компанией IBM, показали, что требования к жестким дискам составляют приблизительно 4,5 Гбайт на 1 MIPS вычислительных возможностей⁶⁹⁾. Для компьютера на 500 MIPS требуется диск емкостью 2250 Гбайт.

✎ Пример

Рассмотрим систему обработки транзакций на базе ПК. Скорость работы ПК — 100 MIPS; время обращения к диску составляет 20 мс. Для каждой транзакции требуется 4 обращения к диску и 100000 команд. Какова пропускная способность системы в транзакциях в секунду?

✎ Решение

Полное время транзакции равно сумме дискового времени и времени процессора:

$$\begin{aligned}\text{время транзакции} &= 4 \times 20 \times 10^{-3} + \frac{100000}{100 \times 10^6} = \\ &= 80 \times 10^{-3} + 1 \times 10^{-3} = 81 \text{ мс},\end{aligned}$$

$$\text{число транзакций в секунду} = \frac{1}{81 \times 10^{-3}} = 12,35$$

✓ Замечание

Даже если свести до нуля время процессора, скорость обработки транзакций ограничено 4 обращениями к диску. Для того, чтобы ПК стал эффективным обработчиком транзакций, необходимо усовершенствовать его систему ввода-вывода, обеспечив поддержку многих параллельных обращений к диску. Следует заметить, что именно это и делается сегодня множеством изготовителей ПК.

Единственный жесткий диск, которым обычно комплектуется ПК, не удовлетворяет требованиям, предъявляемым системами обработки

⁶⁹⁾ Для типичного ПК характерна величина 0,005 Гбайт/MIPS.

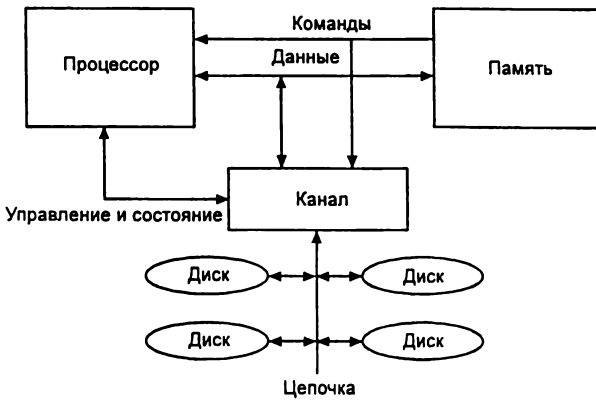


Рис. 7.8. Цепочка дисков большой вычислительной машины.

транзакций, ни по емкости, ни по пропускной способности, ни по запаздыванию. К каналу ввода-вывода должно быть подсоединено несколько дисковых устройств, как это показано на рис. 7.8. Канал подключается к жесткому диску с помощью тракта, называемого цепочкой. Цепочки обычно имеют избыточность, так что в случае отказа цепочки операция может быть продолжена. В дополнение к этому в состав большой машины или сервера может входить много контроллеров и цепочек, что позволяет подключать несколько сотен дисков.

Требования удаленного доступа обеспечиваются в больших машинах с помощью различных специализированных и абонентских сетевых систем. В общем объем данных, передаваемых в транзакции, невелик; поэтому требования к пропускной способности относительно скромны. Обмен информацией между терминалами и главной машиной осуществляется по наземным и радиорелейным линиям и через спутники. Запаздывание выделенных коммуникационных трактов, даже при использовании геосинхронных спутников, вносит незначительный вклад в полное время запаздывания, определяемое обращением к дискам. Ключевыми компонентами систем связи являются коммутационные компьютеры, которые маршрутизируют информацию в сети. В большой машине линии связи подключаются к каналному контроллеру, который устанавливает очередь запросов на обслуживание.

Всемирная Паутина, в отличие от процессоров транзакций, требует и малого запаздывания, и большой пропускной способности. Малое запаздывание необходимо для обеспечения небольшого времени отклика, а высокая пропускная способность — для передачи значительного объема графики, видео и звука. WWW будет обсуждаться в разделе 7.6.1.

7.3. Шины и контроллеры

Две первые задачи системы ввода-вывода, сформулированные в разделе 7.1, заключаются в установлении соединений между памятью и устройствами ввода-вывода, а также управлении передачей данных через эти соединения. В большинстве современных компьютеров для объединения основных компонентов системы используется одна или несколько шин. Шина служит информационным трактом между компонентами; по ней передаются адреса, команды и данные. Упрощенное изображение шины приведено на рис. 7.9. Это та же шина, что и на рис. 7.7, служащая для связи компонентов в сопроцессорной системе ввода-вывода. Шина соединяет процессор, основную память и устройства ввода-вывода с помощью их контроллеров. Вспомним также обсуждавшийся в гл. 4 вопрос о роли шины в передаче команд и данных между процессором и памятью.

Шины можно классифицировать по-разному: по назначению, управлению или принципу передачи данных [9].

Назначение: специализированное или общее.

Управление: централизованное или распределенное.

Передача данных: синхронная или асинхронная.

Таким образом, всего существует восемь комбинаций каждого из указанных типов. В последующих разделах мы рассмотрим характеристики шести типов шины и их использование в современной вычислительной технике.

Специализированная шина и шина общего назначения

Основное различие между этими шинами заключается в том, что специализированная шина соединяет два физических устройства, в то время как шина общего назначения связывает несколько устройств. Эти два типа шин показаны на рис. 7.10. Специализированные шины используются в тех случаях, когда требуются столь высокие характеристики времени реакции и пропускной способности, что совмест-



Рис. 7.9. Структура простой шины.

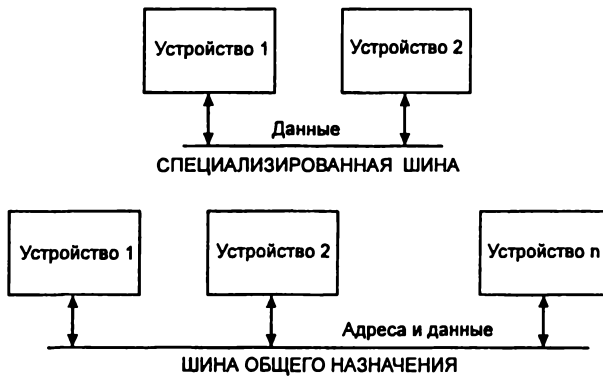


Рис. 7.10. Специализированная шина и шина общего назначения.

ное использование шины несколькими устройствами может существенно снизить производительность системы. В таких случаях проектировщик предусматривает связь по специализированной шине. Обратите внимание на то, что специализированная шина не содержит адресных линий, поскольку и источник, и приемник адресуются неявным образом. Устройство 1 всегда посылает данные устройству 2, а устройство 2 — устройству 1. Иногда встречаются однонаправленные специализированные шины, в которых данные передаются только в одну сторону. Например, однонаправленной может быть шина, соединяющая память и графический контроллер.

Ввиду того что специализированные шины используются либо внутри процессора, либо в специальных высокопроизводительных установках, не являющихся устройствами общего назначения, они в дальнейшем рассматриваться не будут. Мы остановимся на средствах управления и передачи данных, применяемых только с шинами общего назначения.

Шины общего назначения отличаются от внутренних шин процессора, описанных в гл. 5. Специализированная внутренняя шина управляется внешним конечным автоматом или микропрограммным контроллером, в то время как шина общего назначения получает адрес приемника от передатчика данных.

Для шины общего назначения типично совместное ее использование несколькими пользователями; одновременно поступающие запросы шины должны разрешаться с помощью тех или иных приемов. Каждое устройство, подключаемое к шине общего назначения, может представлять собой только приемник или передатчик, но может выполнять функции и того, и другого, образуя приемопередатчик. Например, основные функции контроллера принтера заключаются в приеме данных и послыке информации о состоянии. А дисковый кон-

троллер и посылает, и принимает данные, и посылает информацию о состоянии.

Централизованное и распределенное управление

Управление шиной общего назначения может быть либо централизованным, либо распределенным. Основная задача управления состоит в решении вопроса, предоставлять или не предоставлять устройству доступ к шине. И при централизованном, и при распределенном управлении все устройства рассматриваются единым образом, однако они могут обладать различными приоритетами доступа. Так, если одним из устройств является процессор, ему может быть предоставлен наивысший приоритет доступа к шине. Однако в некоторых системах наивысший приоритет предоставляется устройству ввода-вывода, чтобы не снижать производительность системы; например, если диск не сможет выполнить чтение и пропустит целый оборот, будет потеряно много миллисекунд. Более детальное обсуждение приоритетов лежит за рамками этой книги.

Централизованное управление. Единый аппаратный управляющий узел фиксирует запрос и предоставляет доступ к шине запрашивающему устройству. В обязанности контроллера входит разрешение ситуации с одновременными запросами и назначение запросам приоритетов. При наличии централизованного контроллера используются по меньшей мере три варианта (с модификациями) подключения устройств ввода-вывода: гирляндное подключение, опрос с глобальным счетчиком, опрос с локальным счетчиком. Все эти варианты представлены на рис. 7.11.

Обычно узел, содержащий контроллер шины, называют ведущим, а узел управляющей логики в устройстве — ведомым. Заметим, что можно встретить многочисленные варианты обсуждаемых ниже конфигураций.

Гирлянда. В этом случае управляющая линия от контроллера проходит через все устройства последовательно. Если устройство нуждается в доступе к шине, оно посылает контроллеру шины запрос. Контроллер посылает ответный сигнал-запрос о доступности шины в первое устройство с целью определения его текущего состояния активности. Если это устройство не нуждается в доступе к шине, оно пересылает сигнал-запрос следующему устройству и т. д. Запрашивающее устройство отзывается на этот запрос сигналом занятости шины и получает доступ к шине.

К преимуществам гирляндного управления устройствами относятся простота, низкая стоимость и расширяемость. Недостатком является ненадежность: если управляющая линия разрывается, все устройства, подключенные за точкой разрыва, выпадают из системы. Кроме того, такое управление оказывает предпочтение первым устройствам гирлянды, что может привести к «оседланию» ими шины.

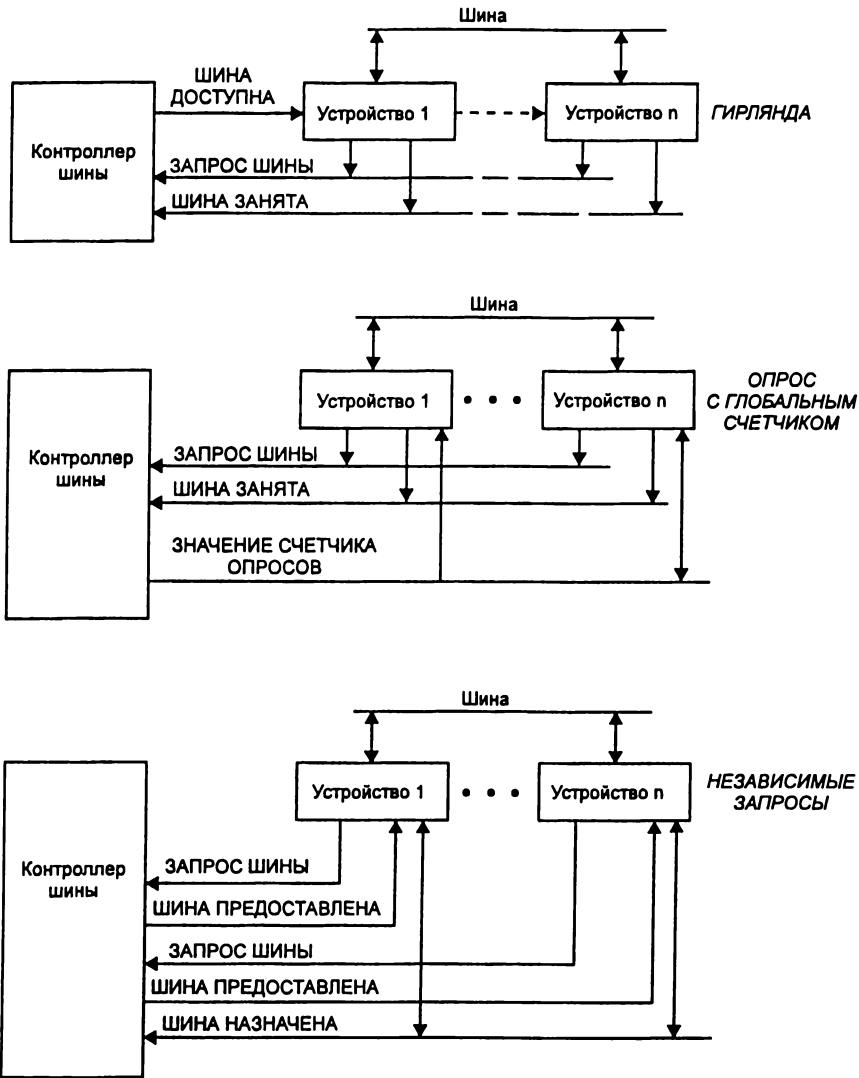


Рис. 7.11. Варианты централизованного управления.

Опрос с глобальным счетчиком. Каждое устройство может послать контроллеру свой запрос шины. Контроллер, получив этот запрос, опрашивает линии запроса с целью идентификации запросчика, а также определения того, какое устройство использует шину, если такое устройство имеется. Последовательность опроса определяется счетчиком в контроллере, который начинает счет с 1, а затем при каждом опросе инкрементируется. Устройство считается идентифицирован-

ным, когда значение счетчика оказывается равным номеру устройства. Если шина свободна или когда она освобождается, запрос устройства удовлетворяется.

Такая система более надежна, чем гирляндное соединение, однако та же проблема неравнозначности устройств остается, так как счетчик в каждом цикле опроса начинает счет с 1. Однако этот недостаток можно устранить, если назначить устройствам случайные адреса или использовать псевдослучайный глобальный счетчик. Расширение в пределах возможностей контроллера реализуется достаточно просто.

Независимые запросы. Централизованное управление с независимыми запросами получает от различных устройств сигналы запроса шины, которые могут рассматриваться процессором как сигналы прерываний. Информация о состоянии шины хранится в контроллере; все разрешения на доступ к шине выдаются с учетом этой информации. Когда контроллер определяет, что устройство может использовать шину, этому устройству посылается сигнал разрешения доступа к шине.

Такая система обладает гибкостью, поскольку приоритеты устройств могут задаваться путем программной записи в контроллер шины таблицы приоритетов. Время запаздывания меньше, чем для двух других систем, в силу того, что сокращается обмен информацией между контроллером и устройствами.

Распределенное управление. Управляющие функции распределены между всеми устройствами на шине. Главное достоинство такого управления заключается в том, что система легко поддается расширению добавлением в нее модулей. Как и в случае централизованного управления, можно встретить три основных варианта построения системы: гирлянда, опрос и независимые запросы. На рисунке 7.12 эти варианты представлены в упрощенной форме.

Гирляндное соединение. Устройство посылает свой запрос по линии запроса шины, которая подсоединена к линии доступности шины. Когда устройство получает сигнал доступности шины, оно передает его следующему устройству, если только оно само не является источником запроса. Этот процесс продолжается до тех пор, пока сигнал доступности шины не дойдет до запрашивающего устройства; в этот момент запрос удовлетворяется. Если шина используется, ни одно из устройств не может выдать запрос шины.

Опрос. Возможность опроса предоставляется всем устройствам. Если устройство, занимающее шину, более в ней не нуждается, оно помещает некоторый, зависящий от устройства, код на линию кода опроса и оповещает остальные устройства по линии доступности шины. Если код опроса соответствует коду устройства, нуждающемуся в доступе к шине, это устройство отзывается сигналом захвата шины. Первое устройство, которое перед этим занимало шину, снимает сигналы кода опроса и доступности шины, после чего второе устройство, сняв сигнал захвата шины, начинает ее использовать.

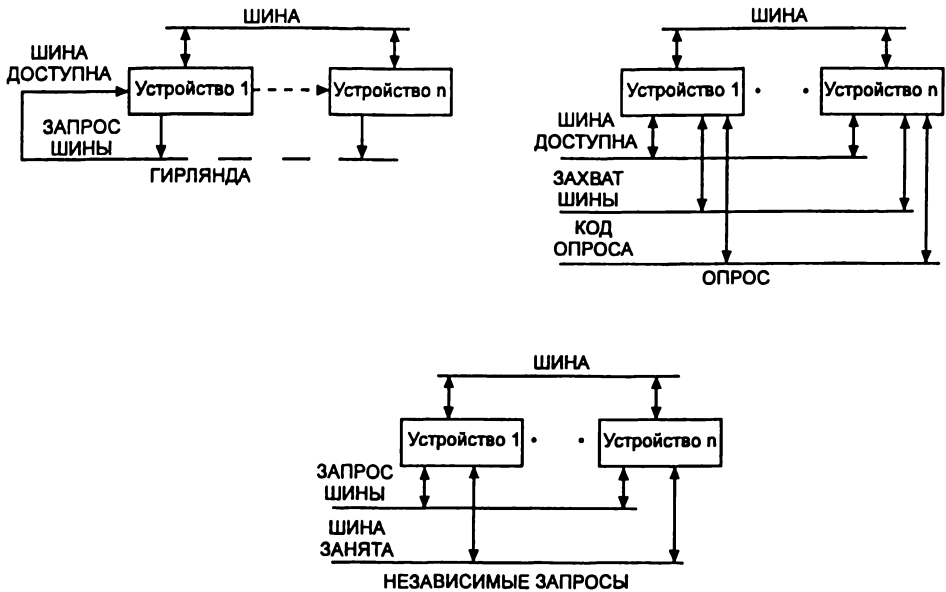


Рис. 7.12. Варианты распределенного управления.

Если опрашивающее устройство не получает сигнал захвата шины, код опроса изменяется с помощью какого-либо алгоритма и опрос продолжается. Алгоритм изменения кода может заключаться попросту в инкременте счетчика, что приведет к кольцевому опросу всех устройств, но может быть и более сложным, учитывающим приоритеты устройств. Например, устройство 3 после завершения работы с шиной может всегда спрашивать устройство 7, не желает ли теперь оно использовать шину. В то же время устройство 4 может выбрать для себя алгоритм опроса со случайным кодом.

Независимые запросы. В этом случае любое устройство, использующее шину, сигнализирует об этом по линии занятости шины. Другое устройство, нуждающееся в доступе к шине, помещает на линии запроса шины свой запрос, включающий код приоритета. Когда текущий пользователь завершает работу с шиной и освобождает ее, он также снимает свой сигнал с линии занятости шины. Все устройства анализируют все запросы шины, и устройство, которое распознает себя, как имеющее наивысший приоритет, берет на себя управление шиной и сигнализирует об этом по линии занятости шины.

Синхронная и асинхронная передача данных

Пересылка адресов, управляющей информации и данных между двумя устройствами может синхронизироваться тактовыми сигнала-

ми или выполняться асинхронно, с саморегулированием, без каких-либо тактовых сигналов [4, 6]. Каждый из этих методов имеет свои достоинства и недостатки, которые будут рассмотрены ниже.

Синхронная передача. Упрощенное изображение синхронной шины (части шины, связанной с данными и синхронизацией), соединяющей между собой два устройства, приведено в левой части рис. 7.13. Обратите внимание на схожесть этого рисунка со схемой конвейера, приведенной на рис. 6.5. Данные передаются между платой в правом гнезде (слоте) и платой в левом гнезде. Передатчик и приемник синхронизируются от общего источника, входящего в состав левой платы. Минимальный тактовый период составляет

$$T_{\text{такт}}(\text{мин}) = \text{задержка шины (макс)} + [\text{перекос тактового сигнала (макс)} + \\ + \text{время установления}] = \text{задержка шины (макс)} + [\text{задержка шины (макс)} + \\ + \text{время установления}] \approx 2 \times \text{задержка шины (макс)} + \text{время установления}$$

Тактовый период составляет приблизительно 2 задержки шины и определяет предел пропускной способности. Этот предел зависит от физической длины шины. Однако перекос тактового сигнала представляет одну половину тактового периода. Если этого компонента (одной задержки шины) не будет, пропускная способность удвоится; тактовый период станет приблизительно равным одной задержке шины на операцию передачи.

Для удаления из тактового периода этого компонента используют относительно сложные цепи, которые позволяют уменьшить перекос на каждом гнезде шины до приемлемой величины 1–2 задержки логического элемента. Как это показано в правой части рис. 7.13, в

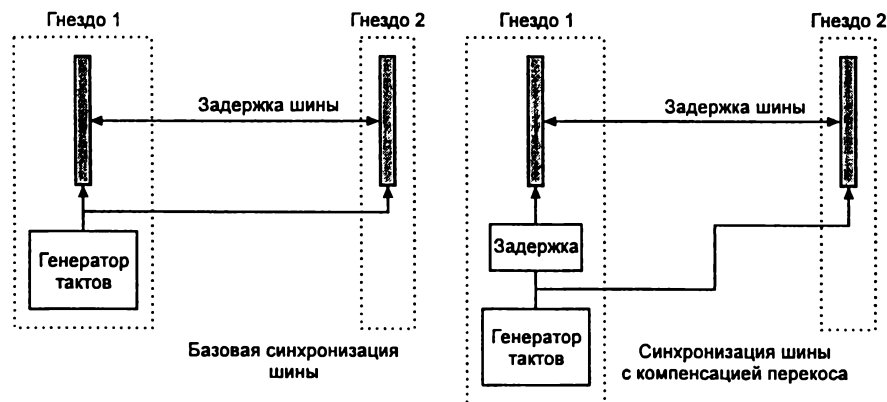


Рис. 7.13. Синхронизация синхронной шины.

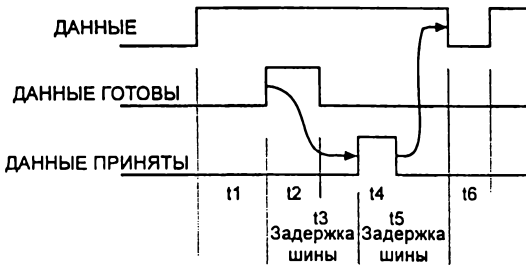


Рис. 7.14. Асинхронная передача.

линии, подводящей тактовые сигналы к платам, включаются задержки, величина которых обратна расстоянию от тактового генератора. В результате тактовые сигналы поступают на все платы одновременно. Компенсация перекоса увеличивает пропускную способность шины приблизительно в два раза. Даже если в настоящий момент используется только одно гнездо расширения, причем ближайшее к передатчику, тактовый период должен выбираться, исходя из предположения, что в дальнейшем может быть занято и самое дальнее гнездо.

Асинхронная передача. Как было показано выше, при синхронной передаче данных по шине тактовый период должен выбираться настолько большим, чтобы шина работала правильно в самом худшем случае, когда платы устройств разнесены максимально. Для преодоления этого ограничения был предложен асинхронный метод передачи данных по шине. В асинхронной шине передачи осуществляются с максимально возможной скоростью и с минимальными задержками, отвечающими физическому состоянию шины. По мере добавления новых плат синхронизация каждой платы подстраивается автоматически. Разработано и используется большое количество асинхронных протоколов; мы здесь рассмотрим только простейший.

Временная диаграмма асинхронного обмена приведена на рис. 7.14, где рассмотрен случай, когда источник посылает данные приемнику.

Источник помещает данные на шину и через время t_1 передает сигнал готовности данных, имеющий ширину t_2 . Когда приемник получит этот сигнал через время t_3 , он фиксирует данные в своем внутреннем регистре и может при необходимости выполнить проверку четности. После этого приемник информирует передатчик сигналом приема данных о том, что передача данных прошла успешно. Если обнаружен какой-либо сбой, источнику посылается сигнал ошибки данных (не показанный на рисунке). После истечения времени t_6 источник может инициировать новый обмен.

Этот процесс оказывается довольно медленным, так как следующая передача данных источником может начаться после по меньшей

мере двух задержек шины ($t_3 + t_5$). Имеются другие протоколы, которые уменьшают число задержек и увеличивают тем самым пропускную способность системы, однако их рассмотрение выходит за рамки этой книги.

7.3.1. Примеры шин

Как уже отмечалось выше, всего можно предложить восемь вариантов шин, имеющих различные комбинации описанных параметров. Специализированные шины проектируются под требования конкретных трактов данных, таких как на рис. 7.11. С другой стороны, в качестве шин общего назначения обычно используются стандартизованные шины; именно эти шины представляют для нас особый интерес. В таблице 7.3 приведены примеры четырех шин общего назначения с указанием их свойств согласно нашей классификации: централизованное и распределенное управление, а также синхронная и асинхронная передача данных.

Таблица 7.3. Примеры шин общего назначения

Управление	Передача данных	Примеры	Спецификация
Централизованное	Синхронная	PCI	PCI SIG v2.1
Централизованное	Асинхронная	IPI	ANSI X3.129
Распределенное	Синхронная	Multibus II	ANSI/IEEE 1296
Распределенное	Асинхронная	VME	IEEE 1014
		FutureBus	IEEE 896.1

7.4. Последовательная передача данных

Описанные выше шины обычно проектируются в расчете на параллельную передачу некоторого числа битов, например 32 бит. В некоторых случаях в силу особенностей передающей среды или данных используется последовательная передача. В качестве примеров такой передачи можно назвать связь мыши или клавиатуры с процессором, а также обмен данными по телефонным линиям. При последовательной передаче передатчик посылает приемнику синхронизованные данные, а приемник должен каким-то образом по потоку принимаемых данных установить синхронизацию тактов.

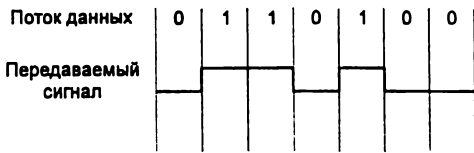


Рис. 7.15. NRZ-кодирование.

Установка синхронизации тактов

В большинстве последовательных систем связи данные передаются на различные расстояния, и передача тактовых сигналов с нулевым перекосом оказывается невозможной. Поток данных должен кодироваться таким образом, чтобы приемник мог извлечь из него информацию о синхронизации тактов. Такая передача данных носит название передачи с неявной синхронизацией.

Ниже будут описаны три метода установки синхронизации тактов. В разделе 7.5 мы остановимся на случаях, когда установка синхронизации потоком принимаемых данных оказывается невозможной.

Без возврата к нулю (Nonreturn to zero, NRZ). При NRZ-кодировании 1 передается одним уровнем сигнала, а 0 другим, как это показано на рис. 7.15. Тактовый сигнал приемника, частота которого устанавливается равной частоте передатчика, синхронизируется каждым перепадом входящего сигнала данных. В этом случае для длинных последовательностей нулей или единиц не происходит перепадов входящего сигнала, и генератор приемника остается без средств синхронизации. По этой причине стабильность тактового генератора приемника и способ синхронизации приобретают особую важность.

В ПК для связи устройств с относительно низкой пропускной способностью используется новый стандарт последовательной шины USB (universal serial bus). В этом стандарте используется NRZ-передача на расстояния до 5 м при пропускной способности 15 Мбит/с. Для связи с такими устройствами, как мышь, используется более дешевый низкоскоростной подканал с частотой 1 Мбит/с.

Частотная модуляция (FM, frequency modulation). В этом случае 1 и 0 передаются разными частотами, как это показано на рис. 7.16. В силу

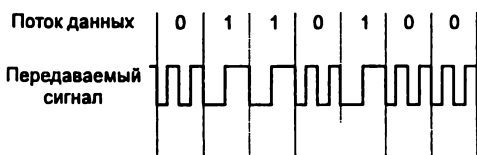


Рис. 7.16. Частотная модуляция.

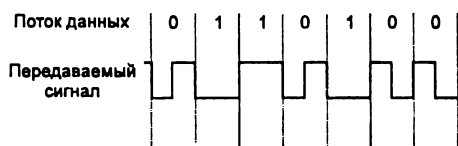


Рис. 7.17. Манчестерское двухфазное кодирование.

того что внутри каждого бита имеются перепады сигнала, синхронизация осуществляется более просто.

В некоторых модемах используется вариант частотной модуляции, позволяющий повысить пропускную способность последовательного канала. Например, при наличии четырех частот можно предложить следующую схему кодирования:

Частота 1 00

Частота 2 01

Частота 3 10

Частота 4 11

Передающий модем извлекает из потока передаваемых данных пары битов и выбирает частоту передачи на заданный интервал времени. Принимающий модем анализирует частоту и восстанавливает поток данных⁷⁰⁾.

Манчестерский код. Другой способ установки синхронизации тактов, имеющий несколько вариантов, получил название манчестерского кода. На рисунке 7.17 показан вариант двухфазного кодирования.

При таком способе передачи каждый следующий бит сопровождается перепадом сигнала, что существенно облегчает синхронизацию тактов. Однако этот код является контекстно-зависимым, что можно увидеть на рис. 7.17, сравнив коды двух последовательных единиц. В общем случае дешифратор для восстановления фактического потока данных должен анализировать пары битов.

Протоколы последовательной передачи

Последовательные коммуникационные протоколы можно классифицировать по методу синхронизации и формату передаваемых данных.

Синхронизация

Синхронная передача Синхронизация является распределенной; для обеспечения синхронизации биты передаются непрерывно, без остановки.

⁷⁰⁾ Телефон с тональным набором использует одну из форм FM-кодирования: кодируются 10 цифр плюс 2 управляющих символа.

Асинхронная передача Линия находится в состоянии покоя (либо 1, либо 0), пока не начнет передаваться сообщение. В этот момент приемник синхронизирует свой тактовый генератор.

Данные

Символьная передача Кодировются символы, например, кодом ASCII.

Блочная передача Сообщение является двоичным.

Из этих двух характеристик можно образовать четыре комбинации. Ниже обсуждаются три из них: передачи асинхронная символьная, синхронная символьная и синхронная блочная. Асинхронно-блочный протокол, видимо, не используется.

Поскольку во всех случаях передается поток битов, приемник прежде всего должен, используя принимаемый поток данных, установить синхронизацию тактов, как это описывалось выше. После установки синхронизации потока единиц и нулей возникает вопрос, где начинается и где оканчивается символ или сообщение? Таким образом, необходимо предусмотреть кодирование, допускающее установку синхронизации символов и сообщений. Простым примером является азбука Морзе: после передачи каждого символа переменной длины передатчик приостанавливается, прежде чем начать передачу следующего символа. Разделение слов осуществляется по контексту.

Асинхронно-символьный протокол. При использовании этого протокола линия остается в состоянии покоя, пока передатчик не пошлет стартовый бит, за которым следуют биты символов, как это показано на рис. 7.18. В зависимости от используемого набора символов, число битов в символе может достигать до 8, при этом к ним может быть добавлен бит контроля четности. И передатчик, и приемник должны пользоваться единой спецификацией символов. После передачи битов символа линия возвращается в состояние покоя. Перед тем как начнет передаваться следующий символ, должно пройти какое-то время.

Другим видом асинхронно-символьного протокола является стандарт RS-232. В этом случае вид символа и тактовая частота должны быть независимо установлены и в передатчике, и в приемнике. В со-

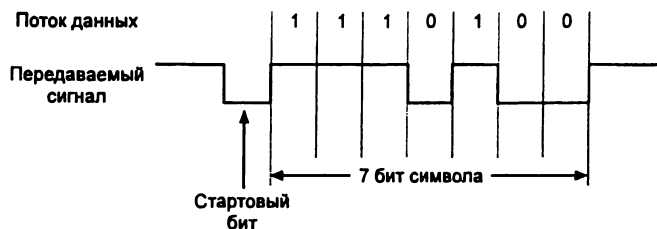


Рис. 7.18. Асинхронно-символьный протокол.

стоянии покоя на линии установлен ноль, который называют также маркером, а признаком начала передачи символа служит единица, которую называют пробелом. Пока линия находится в состоянии покоя, приемник с целью обнаружения стартового бита проверяет уровень сигнала в линии с удвоенной тактовой частотой.

В таблице 7.4 показаны варианты кодов символов RS232. Имеются три разновидности кодов: 5-битовый, 7-битовый и 8-битовый. Для 7- и 8-битовых кодов имеются программируемые модификации.

Таблица 7.4. Коды символов RS232

Тип кода	Поток битов
5-бит.	Стартовый бит + 5 бит данных + (1÷2 бит маркера)
7-бит.	Стартовый бит + 7 бит данных + (1÷2 бит маркера)
7-бит.	Стартовый бит + 7 бит данных + 0 + (1÷2 бит маркера)
7-бит.	Стартовый бит + 7 бит данных + 1 + (1÷2 бит маркера)
7-бит.	Стартовый бит + 7 бит данных + контроль четности + (1÷2 бит маркера)
7-бит.	Стартовый бит + 7 бит данных + контроль нечетности + (1÷2 бит маркера)
8-бит.	Стартовый бит + 8 бит данных + (1÷2 бит маркера)
8-бит.	Стартовый бит + 8 бит данных + контроль четности + (1÷2 бит маркера)
8-бит.	Стартовый бит + 8 бит данных + + контроль нечетности + (1÷2 бит маркера)

Синхронно-символьный протокол.^{*)} В этом протоколе текстовое сообщение, которое может иметь фиксированную или переменную длину, окружается различной синхронизирующей и управляющей информацией, как это показано на рис. 7.19. Передатчик непрерывно посылает поток символов SYN (0010110), в результате чего приемник постоянно поддерживает синхронизацию битов и символов. Приемник ожидает символа SOH (0000001), который указывает на начало сообщения и обеспечивает синхронизацию сообщения. Заголовок

SYN	SYN	SOH	Заголовок	STX	Текстовое сообщение	ETX	BCC
	SYN	Устанавливает синхронизацию		0010110			
	SOH	Начинает заголовок сообщения		0000001			
	STX	Начало текста		0000010			
	ETX	Конец текста		0000011			
	BCC	Контрольная сумма блока символов					
	Заголовок	Адрес и управляющая информация					

Рис. 7.19. Синхронно-символьный протокол.

^{*)} Такой протокол иногда называют кодограммным. – Прим. ред.

Флаг	Адрес	Управл.	Двоичное сообщение	Контрольная сумма кадра	Флаг
		Флаг	01111110		
		Адрес	8 бит		
		Управление	8 бит		
		Контрольная сумма кадра	16 бит		

Рис. 7.20. Синхронно-блочный протокол HDLC.

предоставляет информацию о длине текстового сообщения. Вслед за заголовком посылается символ STX, за которым начинается тело сообщения. Заканчивается сообщение символом ETX и контрольной суммой блока.

Рассматриваемый протокол использует 7-битовые символы ASCII с контролем нечетности. В наборе символов ASCII предусмотрены управляющие коды, которые гарантированно не могут встретиться в сообщении.

Синхронно-блочный протокол. Одним из многих блочно-ориентированных протоколов синхронизации сообщений является протокол HDLC (high-level-data link control, высокоуровневый протокол управления каналом). Этот протокол, как показано на рис. 7.20, предваряет каждое сообщение преамбулой, называемой флагом. При отсутствии передачи по линии непрерывно передаются флаги, что позволяет поддерживать синхронизацию битов и символов. Начало сообщения характеризуется адресным полем, в котором не допускается символ 01111110. Сообщение заканчивается контрольной суммой кадра для проверки правильности принятой информации (в пределах возможности контрольной суммы).

Двоичное сообщение может включать любые комбинации единиц и нулей; в сообщении может даже присутствовать код флага 01111110. Поэтому передатчик должен выявлять такие коды в сообщении и вставлять в них дополнительный ноль (что называется вставкой бита, bit stuffing), чтобы код в сообщении не был спутан с кодом флага. Таким образом,

код в сообщении 01111110
преобразуется в 011111010

Приемник, установив сначала синхронизацию по полям адреса и управления, удаляет вставленный ноль.

✎ Пример

Каким образом приемник, использующий протокол HDLC, узнает, что код сообщения, состоящий из 5 единиц и одного нуля (111110), не имеет вставленного нуля и является истинным кодом сообщения?

✎ Решение

Приемник контролирует появление в сообщении 9-битовой последовательности 011111010. Эта последовательность имеет вставленный бит нуля.

7.5. Тактовая синхронизация

Синхронизация двух или большего числа программ будет рассмотрена в гл. 8. В настоящем разделе мы обсудим проблемы синхронизации двух аппаратных систем, которые обмениваются данными, но имеют отдельные тактовые генераторы [7], и в которых не предусмотрено восстановление тактовой синхронизации. Например, канал ввода-вывода соединяет дисковую память (со своим тактовым генератором) с реальной памятью процессора (тоже со своим тактовым генератором).

Рисунок 7.21 иллюстрирует результат использования несинхронизированных тактовых генераторов. Логический сигнал от системы с другим тактовым генератором поступает на вход триггера с собственными тактами. Две тактовые последовательности не синхронизированы. Если перепад асинхронного входного логического сигнала возникает в течение тактового сигнала или его перепада, триггер может перейти на некоторое время в метастабильное состояние, после которого результирующее состояние будет неопределенным. Если выход триггера используется для управления двумя схемами с различ-

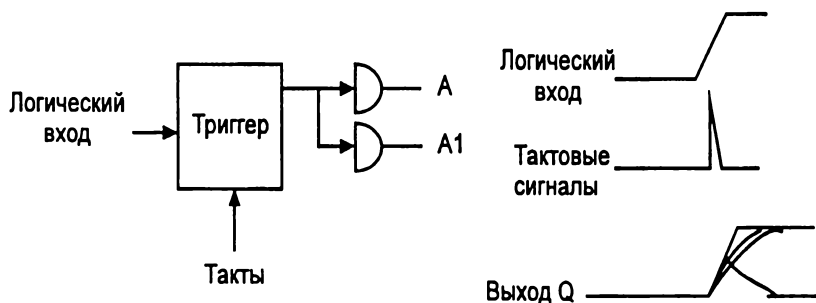


Рис. 7.21. Метастабильность.

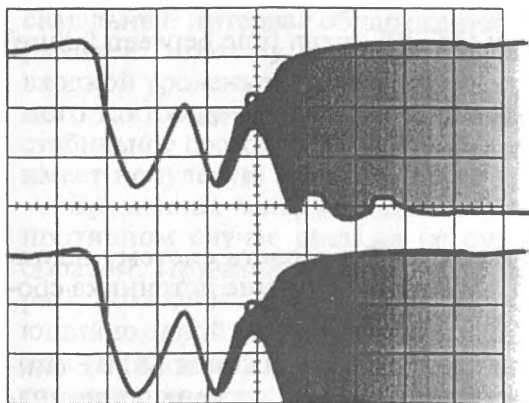


Рис. 7.22. Метастабильность синхронизатора. 1 нс на деление, 1 В на деление. (© 1973 IEEE).

ными порогами, выходные сигналы этих схем могут на некоторое время оказаться логически неравными. Другими словами, A не будет равно $A1$. Поскольку эти сигналы используются в качестве входных для других логических элементов, конечный результат предсказать невозможно, т. е. мы получаем систему с ненадежным поведением.

На рисунке 7.22 показаны оциллографические снимки выходных сигналов Q и Q' двух ТТЛ элементов НЕ-И с перекрестными связями [2]. R и S входы этих элементов переключаются одновременно с целью моделирования одновременного с тактовыми сигналами воздействия перепадов входных логических сигналов. Видно, что длительность состояния метастабильности не имеет четких границ. Небольшая доля случаев переключения приводит к весьма длительному периоду метастабильности, более 9 нс в примере на рис. 7.22.

Неправильное функционирование аппаратуры из-за метастабильности случается не часто, но диагностировать его очень трудно, так как сбои возникают случайным образом.

✎ Пример

Чтобы проиллюстрировать частоту синхронизационных сбоев, рассмотрим следующий пример. Система пытается выполнять синхронизацию каждые 16 тактов (для обеспечения квити́рования шины процессорной памяти), тактовый период составляет 100 нс, а синхронизатор отказывает один раз на 10^{11} попыток синхронизации.

Решение

Среднее время между отказами (MTBF, mean time between failure) синхронизации равно

$$(\text{число тактов} \times \text{частота тактов} \times \text{интенсивность отказов синхронизатора}) = \\ = 16 \times (100 \times 10^{-9}) \times (1 \times 10^{11}) = 1,6 \times 10^5 \text{ с} = 1,85 \text{ день}$$

✓ Замечание

Значение MTBF достаточно мало, чтобы сделать систему нестабильной, и достаточно велико, чтобы обнаружение источника сбоев стало весьма сложной задачей.

Предыдущий пример предлагает два метода снижения MTBF синхронизации: уменьшение частоты синхронизации и уменьшение значения MTBF синхронизатора. Имейте в виду, что до сих пор не удалось спроектировать безотказный синхронизатор.

Частоту синхронизации можно уменьшить, если обеспечить передачу на каждую синхронизацию большего количества битов. Если, например, выполнять синхронизацию не каждого передаваемого бита, а лишь каждого блока размером 1 Кбайт, частота синхронизации снизится в 8192 раза.

После того как частота синхронизации снижена, можно заняться снижением величины MTBF синхронизатора. Здесь используются различные приемы. Первый шаг — использование в схеме синхронизатора сверхвысокочастотных транзисторов. В этом случае уменьшается интервал времени, в течение которого синхронизатор проявляет метастабильность.

Другой способ уменьшения величины MTBF синхронизатора показан на рис. 7.23. Такой синхронизатор работает следующим образом. Тактовый сигнал задерживается на такое время, после которого вероятность для триггера 1 находиться в метастабильном состоянии снижается до приемлемого уровня. Из рисунка 7.22 видно, что чем больше время задержки, тем меньше вероятность метастабильности. По истечении этой задержки тактовый сигнал подается на триггер 2. Из-за введения задержки вероятность изменения сигнала на входе



Рис. 7.23. Синхронизатор с уменьшенной интенсивностью отказов по причине метастабильности.

триггера 2 невелика, и метастабильность не возникает. Однако максимальный интервал обнаружения метастабильности триггера 1 не ограничен, поэтому остается незначительная вероятность того, что входной уровень триггера 2 претерпит изменение в течение задержанного тактового сигнала, в результате чего триггер 2 перейдет в метастабильное состояние. Таким образом, рассмотренный синхронизатор имеет ненулевую вероятность проявления метастабильности.

Временная задержка должна быть меньше тактового периода; в противном случае система не будет функционировать правильным образом. Причина этого заключается в том, что если на вход триггера 1, пока он находится в метастабильном состоянии, поступит следующий входной сигнал, триггер, скорее всего, останется в метастабильном состоянии. Таковы сложные и противоречивые требования, с которыми приходится сталкиваться при проектировании систем с очень высокими тактовыми частотами.

✎ Пример

Рассмотрим синхронизатор, изображенный на рис. 7.23. Пусть задержка тактового сигнала, достаточная для устранения большей части метастабильных состояний первого триггера, составляет 15 нс. Каково минимальное значение тактового периода и максимальная тактовая частота для этой системы?

✍ Решение

минимальный тактовый период (мин) = 15 нс

максимальная тактовая частота (макс) = $1/15 \text{ нс} = 66,6 \text{ МГц}$

7.6. Теория очередей

Если в системе к одному ресурсу может возникнуть несколько запросов (в гл. 6 такая ситуация была названа структурным конфликтом), то для хранения запросов до того момента, когда они смогут быть обслужены, используются очереди. Теория очередей разрабатывалась как закрытая модель для оценки поведения таких систем без обращения к компьютерному моделированию. Две обсуждаемые ниже модели весьма просты и игнорируют многие сложности реальных систем. Тем не менее эти модели могут дать быстрые и достаточно точные оценки поведения таких систем. Наиболее важный результат применения этих моделей — среднее время, требуемое для обслуживания запроса. Мы обсудим модели открытых и закрытых систем. Проектировщик обычно может определить тип своей системы и выбрать правильную модель для оценки ее поведения.

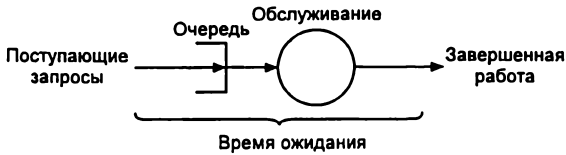


Рис. 7.24. Модель открытой системы с очередью.

7.6.1. Модель открытой системы с очередью

Открытая система показана на рис. 7.24. В ней используется очередь типа «первый вошел, первый обслуживается» (FIFO). Запросы на обслуживание поступают, помещаются в очередь и обслуживаются, когда данный запрос достигает конца очереди. Будем называть временем нахождения в очереди полное время ожидания в системе (время очереди плюс время обслуживания).

Открытая модель с очередью приложима не только к дисковой системе ввода-вывода, но и к взлетно-посадочным полосам аэропортов, кассам магазинов и интернет-серверам. Вывод формулы для времени нахождения в очереди основывается на двух предположениях:

1. Интенсивности поступления запросов и их обслуживания распределены по закону Пуассона.
2. Запросы поступают, обслуживаются и никогда не требуют обслуживания повторно в пределах временных рамок модели.

Распределение Пуассона

$$P_n = \frac{\lambda^n}{n!} e^{-\lambda},$$

для $\lambda = 3$ приведено на рис. 7.25.

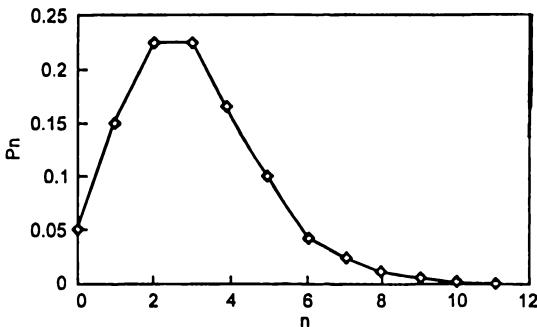


Рис. 7.25. Распределение Пуассона для $\lambda = 3$.

Введем некоторые обозначения. Будем считать, что средняя интенсивность поступления запросов равна λ , средняя интенсивность обслуживания μ , а среднее время ожидания (время очереди + время обслуживания) равно t_q . Эти величины связаны следующей зависимостью:

$$t_q = \frac{1}{\mu - \lambda}$$

Рассмотрим пределы изменения λ :

При $\lambda \rightarrow 0$, $t_q \rightarrow 1/\mu$,

и

при $\lambda \rightarrow \mu$, $t_q \rightarrow \infty$

Из опыта мы знаем, что если мы, например, подходим к кассиру в банке и перед нами нет никого в очереди (другими словами $\lambda \rightarrow 0$), мы будем обслужены немедленно и среднее время ожидания составит $1/\mu$. С другой стороны, если клиенты приходят с той же средней интенсивностью, что и интенсивность обслуживания ($\lambda \rightarrow \mu$), среднее время ожидания для некоторых клиентов может быть бесконечным.

Пример

Интернет-сервер может выполнять 100 транзакций в секунду, а запросы поступают со скоростью 80 запросов в секунду. Каково будет среднее время ожидания?

Решение

$$t_q = 1/(\lambda - \mu) = 1/(100 - 80) = 1/20 = 0,05 \text{ с}$$

Замечание

Среднее время ожидания составит 0,05 с, т. е. в 5 раз больше, чем для незагруженной системы (0,01 с).

Нет необходимости знать значения λ и μ . Модель можно нормализовать с помощью понятия нагрузки сервера (называемого также коэффициентом использования) — параметра, который в некоторых случаях проще оценить или измерить. Нагрузка определяется как $\rho = \lambda / \mu$ и

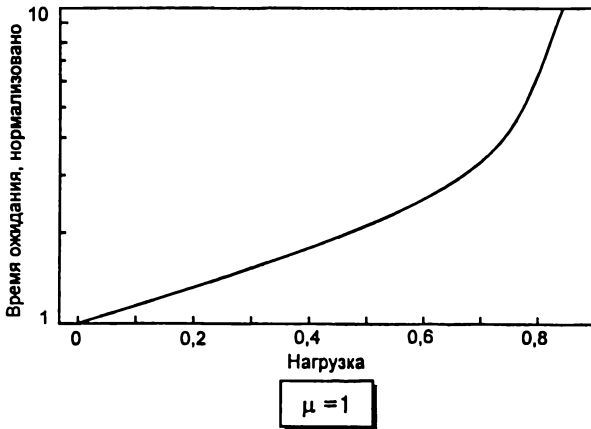


Рис. 7.26. Время ожидания в зависимости от нагрузки.

$$t_q = \frac{1}{\mu - \lambda} = \frac{\frac{1}{\mu}}{\frac{\mu}{\mu} - \frac{\lambda}{\mu}} = \frac{1}{\mu(1 - \rho)}$$

График нормализованного времени ожидания в зависимости от нагрузки для $\mu = 1$ приведен на рис. 7.26. Для нагрузки 0,5 время ожидания вдвое превышает время обслуживания. Для нагрузки 0,8

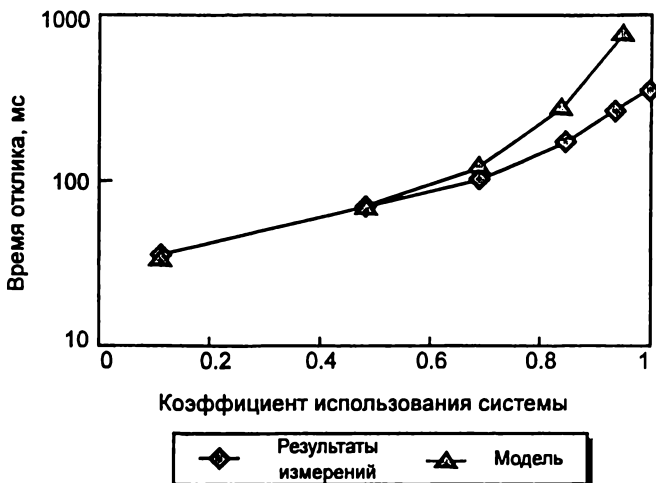


Рис. 7.27. Время отклика дисковой системы.

время ожидания уже в 5 раз превышает время обслуживания — результат, идентичный полученному в предыдущем примере. График доведен только до значения нагрузки 0,9, так как асимптотическое поведение t_q приводит к бесконечному времени ожидания для нагрузки, равной 1.

На рисунке 7.27 показана зависимость времени отклика системы от коэффициента ее использования для матрицы магнитных дисков [3]. Значение t_q для нулевой нагрузки равно приблизительно 35 мс. По мере увеличения нагрузки время отклика увеличивается (при $\rho = 0,5$ приблизительно до 70 мс), как это и предсказывается моделью. Модель дает достаточно точные результаты вплоть до нагрузки 0,7. Расхождение возникает из-за того, что операционная система отклоняет запросы, чтобы время отклика и длина очереди не возрастали бесконечно.

✎ Пример

Средняя интенсивность взлетов со взлетной полосы аэропорта составляет 0,5 взлетов в минуту, или 1 взлет за каждые 2 минуты, не считая ожидания в очереди на взлет^{*)}. В настоящее время нагрузка взлетной полосы составляет 50%. Согласно прогнозу трафик аэропорта увеличится в следующем году на 50%. Насколько возрастут задержки взлета?

✎ Решение

В настоящее время среднее значение t_q , включая само время взлета, равно

$$\frac{1}{\mu(1-\rho)} = \frac{1}{0,5(1-\rho)} = \frac{2}{1-0,5} = 4 \text{ мин.}$$

При увеличении трафика интенсивность взлетов не возрастает^{**)}, и среднее время ожидания во взлетной очереди будет равно

$$\frac{2}{1-(0,5 \times 1,5)} = 8 \text{ мин.}$$

Таким образом, процентный рост времени ожидания составит

$$\frac{100(8-4)}{4} = 100\%$$

^{*)} Другими словами, время обслуживания составляет 2 мин. — *Прим. перев.*

^{**)} Правильнее сказать, не возрастает время взлета (2 мин), хотя увеличивается время ожидания в очереди на взлет. — *Прим. перев.*

✓ **Замечание**

Мы видим, что время ожидания в открытой очереди подчиняется экспоненциальной зависимости. Увеличение интенсивности отправления самолетов на 50% приводит к 100% росту среднего полного времени ожидания. Среднее время ожидания в очереди на взлет возросло с 2 мин до 6 мин, что дает увеличение на 200%.

Среднее число запросов в открытой системе (очередь + обслуживание) определяется законом Литтла. Этот закон утверждает, что число клиентов в системе равно скорости поступления, умноженной на время ожидания:

$$\text{среднее число запросов в системе} = \lambda \cdot t_q$$

Поскольку ρ запросов обслуживаются, число запросов в очереди равно числу запросов в системе минус ρ , т. е. $\lambda \cdot t_q - \rho$. Таким образом, по мере приближения t_q к бесконечности размер очереди также стремится к бесконечности. В реальных системах очередь должна быть конечной, и когда очередь фиксированного размера заполняется, запросы отвергаются. Именно это сделала операционная система с матрицей дисков (см. рис. 7.27).

Вспомним первый пример раздела 7.6.1. Что случится, если число запросов к интернет-серверу подскочит до 200 в минуту как следствие «раскаленной» домашней страницы⁷²⁾? Глубина входной очереди и время ожидания в очереди должны стать равными ∞ , что невозможно. В такой ситуации запросы отсылаются назад пользователю с сигналом «занято»; в случае аэропорта самолеты не выпускаются из ворот на

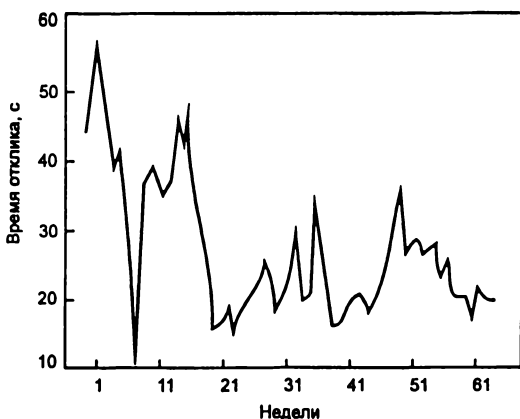


Рис. 7.28. Время отклика WWW (данные предоставлены Keynote Inc.).

⁷²⁾ Во время Олимпийских игр в Нагано в 1998 г. интернет-сайт получил 634716480 запросов за 16 дней, в среднем 459 запросов в секунду. Пиковая интенсивность доходила до 1840 обращений в секунду.

летное поле. Очередь не разрушается, запрос просто возвращается пользователю.

✎ Примеры

1. На рисунке 7.28 приведены данные о времени отклика WWW за период 64 недели. Мы хотим выяснить, что послужило причиной существенного уменьшения времени отклика за этот период приблизительно от 55 с до 20 с. За это время наблюдались периоды малого времени отклика, но уровень, достигнутый в конце, представляется достаточно стабильным.

✎ Решение

Чтобы получить разумный ответ на поставленный вопрос, мы должны сделать несколько предположений. Предположим, что данные о времени отклика получены для пользователей, подключенных к своим провайдерам, и не характеризуют недостаток модемов. Далее, разумно предположить, что интенсивность запросов возросла за рассматриваемый период на 50%. Мы также полагаем, что емкость серверов существенно возросла; в результате уменьшились задержки очередей к серверам. Будем считать, что задержки относятся к серверам с их очередями; другими словами, задержки передачи данных отсутствуют. Будем исходить из модели открытых систем с очередями. Примем, что среднее время обслуживания сервером составляет один запрос в секунду ($\mu = 1$), а полное время отклика (t_q) на первой неделе было 55 с. Каково повышение быстродействия серверов?

Прежде всего определим среднюю интенсивность запросов в начале периода

$$t_q = \frac{1}{\mu - \lambda}$$

$$\lambda = \mu - \frac{1}{t_q} = 1 - \frac{1}{55} = 0,982 \text{ запросов в секунду}$$

Новые серверы привели к повышению скорости обслуживания (среднее время отклика 20 с) при росте интенсивности запросов. Какова новая скорость обслуживания в предположении, что интенсивность запросов увеличилась на 50%?

$$\mu = \frac{1}{t_q} + \lambda = \frac{1}{20} + 0,982 \times 1,5 = 1,52 \text{ обслуживаний в секунду}$$

Повышение быстродействия серверов составит

$$S = \mu \text{ (новое)} / \mu \text{ (старое)} = 1,52 / 1 = 1,52$$

✓ Замечание

Очевидно, что на первой неделе серверы были сильно перегружены, поскольку значение λ было близко к μ , а нагрузка составила $0,982/1 = 0,982$. Увеличение скорости обслуживания на 50% привело к повышению быстродействия и уменьшению времени отклика в $55 / 20 = 2,75$ раз при незначительном снижении нагрузки серверов до величины $\lambda/\mu = 0,982 \times 1,5 / 1,52 = 0,969$.

При такой нагрузке систему все еще следует считать сильно перегруженной.

2. Сколько запросов ожидает в очереди к интернет-серверам на 1-й и 64-й неделях?

✍ Решение

Согласно закону Литтла среднее число запросов в очереди равно полному числу запросов в системе минус вероятность обслуживания одного запроса. Для первой недели среднее число запросов в очереди равно

$$\lambda t_q - \rho = 0,982 \times 55 - 0,982 = 54 - 0,982 \approx 53$$

По тому же закону Литтла среднее число запросов в очереди на 64-й неделе равно

$$0,982 \times 1,5 \times 20 - 0,969 = 29,46 - 0,98 \approx 29$$

✓ Замечание

Среднее число запросов в очереди уменьшилось с 53 до 29, т. е. на 45%.

3. Переформулируйте закон Литтла, введя в него зависимость от нагрузки ρ .

✍ Решение

число запросов в системе $= \lambda t_q =$

$$= \lambda / [\mu(1 - \rho)] = \rho / (1 - \rho)$$

✓ **Замечание**

Выводы из закона Литтла можно сравнить с предыдущими выкладками. Начальная нагрузка интернет-серверов в нашем примере составила 0,982. Число запросов в системе равно

$$0,982/(1 - 0,982) = 0,982/0,018 \approx 54,$$

что совпадает с полученным выше результатом.

7.6.2. Модель закрытой системы с очередью

Рассмотрим теперь вторую полезную для нас модель, именно, модель закрытой системы. В этой модели определяется среднее время ожидания для закрытой системы, например для сервера с некоторым числом терминалов, как это показано на рис. 7.29. В этой модели мы имеем один сервер, одну входную очередь и заданное число терминалов.

Пользователи системы составляют запрос к серверу и помещают его в очередь. Через некоторое время сервер возвращает результат запроса, и пользователь составляет следующий запрос. В отличие от открытой системы, здесь запросы и ответы циркулируют внутри системы.

Введем некоторые обозначения. Будем считать, что время обслуживания сервером равно t_s , нагрузка сервера ρ , число терминалов n , время обдумывания (время, требуемое пользователю на формирование следующего запроса после получения ответа) t_i и время ожидания ответа W . Эти величины связаны следующей зависимостью:

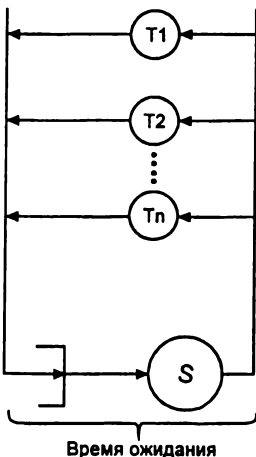


Рис. 7.29. Модель закрытой системы с очередью.

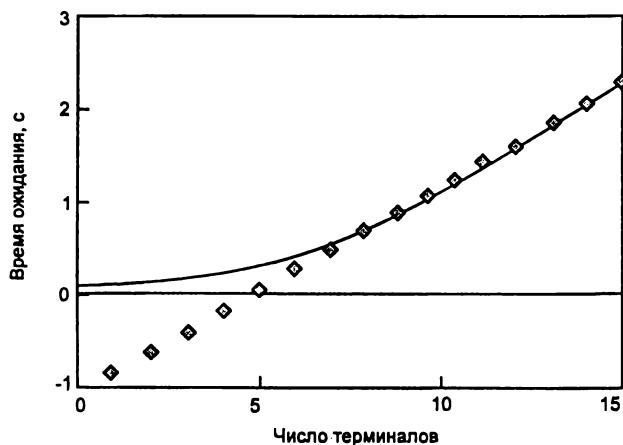


Рис. 7.30. Время ожидания для закрытой системы.

$$W = \frac{nt_s}{\rho} - t_s$$

Нагрузка сервера может изменяться от 0 (если активных терминалов нет) до 1 (если запросы насыщают сервер, не оставляя свободного серверного времени). Если сервер насыщен, то $\rho = 1$, а время ожидания равно

$$W = nt_s - t_s$$

Обратите внимание на то, что мы получили линейную зависимость, в отличие от экспоненциальной, характерной для открытых систем. Кроме того, при отсутствии нагрузки на сервер $\rho = 0$ и W становится очень большим — абсурдный результат! Для рассматриваемой модели было получено итеративное решение [5], устраняющее эту нелепость; мы покажем приближительное решение для конкретных значений параметров.

Пусть $t_s = 0,1$ с, $t_k = 1$ с, а $\rho = 1$ (случай насыщения). На рис. 7.30 показана зависимость времени ожидания от числа терминалов. Видно, что при небольшом числе терминалов время становится отрицательным — еще один абсурдный результат, устраняемый итеративным решением.

Каков же правильный график времени ожидания? При одном терминале других запросчиков нет, и время ожидания попросту равно времени обслуживания сервером. Для этой точки на графике выбрано произвольное значение 0,1 с. По мере роста числа терминалов ρ увеличивается до 1, и система насыщается приблизительно при 10 терминалах. При дальнейшем увеличении числа терминалов время ожидания возрастает, причем наклон кривой равен t_s . Закрытые сис-

темы часто работают в режиме насыщения, поэтому область насыщения представляет значительный интерес.

✎ Пример

Сервер с 100 терминалами работает в режиме насыщения; измеренное среднее время ожидания равно 1,7 с, а измеренное среднее время обдумывания равно 2,0 с. Пользователи жалуются на долгое ожидание, и администратор желает уменьшить среднее время ожидания до 0,5 с. Как это сделать?

✎ Решение

Эту проблему можно решить двумя способами: уменьшить число терминалов или поставить более производительный сервер. Для нахождения числа терминалов, при котором будет обеспечено требуемое время ожидания, мы сначала должны найти t_s базовой системы:

$$W = nt_s - t_i$$

$$1,7 = 100t_s - 2$$

$$t_s = \frac{1,7+2}{100} = 0,037 \text{ с}$$

Найдем число терминалов, при котором время ожидания уменьшится до 0,5 с:

$$W = nt_s - t_i$$

$$0,5 = 0,037n - 2$$

$$n = \frac{0,5+2}{0,037} = 67,5 \approx 67 \text{ терминалов}$$

Альтернативное решение проблемы длительного ожидания заключается в повышении производительности сервера. Сначала найдем требуемое значение t_s :

$$W = nt_s - t_i$$

$$0,5 = 100t_s - 2$$

$$t_s = \frac{0,5+2}{100} = 0,025 \text{ с}$$

Требуемое повышение быстродействия сервера составит:

$$\text{повышение быстродействия} = t_s(\text{старое}) / t_s(\text{новое}) = 0,037/0,025 = 1,48$$

✓ **Замечание**

Для получения требуемого времени ожидания администратор будет вынужден убрать 1/3 наличных терминалов, на что он вряд ли пойдет. Более разумное решение заключается в модернизации сервера с тем, чтобы увеличить его производительность приблизительно на 50%.

Следует сделать предупреждение относительно времени обдумывания. В нашей модели оно присутствует как фиксированное значение. Однако имеются многочисленные экспериментальные доказательства (да и ваш опыт подскажет вам то же самое), того, что по мере снижения времени ожидания время обдумывания также снижается. Более быстрой системе мы ставим больше запросов, чем более медленной. Поэтому в вышеприведенном примере администратор должен определить требуемую производительность при значении $t_i = 1$, а не $t_i = 2$.

Литература

1. Blaauw, G. A. and Brooks, F. P. Jr. (1997). *Computer Architecture, Concepts and Evolution*, Addison-Wesley Longman, Reading, MA.
2. Chaney, T. J. and Molnar, C. E. (1973). «Anomalous Behavior of Synchronizer and Arbiter Circuits», *IEEE Transactions on Computers*, C-22:4.
3. Chen, P. (1989) «An Evaluation of Redundant Arrays of Inexpensive Disks Using an Amhdal 5890», M. S. Thesis, Technical Report UCB/CSD 89/506, p. 341, Computer Science Division, University of California, Berkeley.
4. Del Corso, D., Kirmann, H., and Nicoud, J. D. (1986). *Microcomputer Buses and Links*, Academic, New York.
5. Denning, P. J. and Buzen J. P. (1978). «The Operational Analysis of Queuing Network Models», *ACM Computing Surveys*, 10:3.
6. Di Giacomo, J., ed. (1990). *Digital Bus Handbook*, McGraw-Hill, New York.
7. Kleeman, L. and Cantoni, A. (1987). «Metastable Behavior in Digital Systems», *IEEE Design and Test*, 4:6.
8. Randell, B., ed. (1973). *The Origins of Digital Computer: Selected Papers*, Springer-Verlag, New York, 1973.
9. Thurber, K. J., Jensen, E. D., Jack, L. A., Kenney, L. L., Patton, P. G., and Anderson, L. C., (1972). «A Systematic Approach to the Design of Digital Bussing Structures», *Fall Joint Computer Conference*, AFIPS Press, Montvale, NJ, pp. 719–736.

Упражнения

7.1. Рассмотрите пример для времени отклика WWW. Насколько изменится результат, если время обслуживания уменьшить от 1 с до 0,1 с? Получите конкретный ответ.

7.2. Составьте программу с использованием фон-неймановской архитектуры (табл. 1.2), в которой счетчик, управляющий передачами программного ввода-вывода, находится в ячейке памяти.

7.3. Получите выражение для определения процентного роста времени ожидания в очереди при увеличении ρ от ρ_1 до ρ_2 . Считайте, что μ не изменяется. Проверьте вашу формулу на примере аэропорта с взлетными полосами.

7.4. Шина компьютера имеет длину 0,5 м, а плата с тактовым генератором установлена в середине шины. Какая потребуется задержка компенсации перекоса (в наносекундах) для тактов, поступающих на логические схемы этой серединной платы? Вам придется сделать предположение о времени распространения сигнала по шине до ее конца.

7.5. В примере с терминалами (раздел 7.6.2) вычислите необходимое повышение производительности сервера, если $k=1$ вместо $k=2$.

7.6. Воспользовавшись документацией к процессору Intel 8080, составьте полный список кодов для трех выводов, упоминаемых в разделе 7.1.

7.7. Вернитесь к рис. 7.13, где в линии тактовых сигналов включены задержки, компенсирующие перекося. Какой должна быть задержка для каждого слота в предположении, что всего на шине установлено 5 плат?

7.8. Почему распределенная синхронизация тактовыми сигналами без перекося при использовании последовательной связи невозможна? Этот вопрос обсуждался в разделе 7.4.

7.9. Вы видели график на рис. 7.25. На вид это распределение Пуассона. Найдите для него значение λ .

7.10. Найдите повышение быстродействия сервера в последнем примере раздела 7.6.2, если при снижении среднего времени отклика до 0,5 с время обдумывания уменьшится до 1 с.

7.11. В примере программного ввода-вывода предполагалось, что скорость процессора составляет 0,1 MIPS. Пересчитайте этот пример для значений 0,1, 1, 10 и 100 MIPS. Начертите график зависимости эффективности от скорости процессора.

7.12. В примере раздела 7.2 система обеспечивает 12,35 транзакций в секунду. Пусть требуется получить скорость 500 транзакций в секунду. Процессор останется тем же, но будут добавлены диски. Сколько потребуется дисков?

ГЛАВА 8

РАСШИРЕНИЯ НАБОРА КОМАНД

8.0. Введение

Прикладные системы команд (ISA) предоставляют операции, адреса и типы данных для разнообразных научных и деловых приложений (см. гл. 3). Однако этих команд не хватает для обеспечения работы процессоров, используемых в современных системах с виртуальной памятью, параллельным вводом-выводом и операционными системами, а также в мультипроцессорных конфигурациях. Для того чтобы обеспечивать поддержку этих дополнительных задач, предусматривают расширение набора команд. Каждое из таких расширений ISA требует специальных типов данных, адресов и операций.

Программы, обеспечивающие реализацию расширенных функций, обычно пишутся не прикладными программистами, а разработчиками программного обеспечения. Чтобы предохранить эти программы от вмешательства пользователей в их работу, в процессорах предусматриваются два режима работы: режим пользователя и режим супервизора.

Расширения ISA в целом называют по-разному: управляющими командами в документации к IBM S360 [4], командами защищенного режима в описаниях Intel Pentium [7] и привилегированными командами в документации к Pentium Pro [8]. В этой книге мы будем использовать термин «привилегированные команды».

Привилегированные команды не могут выполняться вместе с прикладными, когда процессор находится в режиме пользователя. Если, однако, процессор работает в режиме супервизора, могут выполняться и прикладные, и привилегированные команды. Если сделать попытку выполнить привилегированную команду в режиме пользователя, каскадом декодирования конвейера возбуждается прерывание. Разделение привилегированных и прикладных команд во многих компьютерах осуществляется с помощью переключателя режима. Команды, которые переключают процессор из режима супервизора в режим пользователя, являются привилегированными, однако прикладные команды могут привести к возбуждению привилегированного режима.

🔗 Пример

Привести пример перехода из режима пользователя в режим супервизора и назад в режим пользователя.

✍ Решение

В процессорах IBM S360, S370 и S390 команда вызова супервизора (SVC) является прикладной. Эта команда вызывает прерывание, передающее управление в супервизор, который, в свою очередь, загружает новое слово состояния программы (PSW) из ячейки памяти 96, в котором управляющий бит (бит 15) установлен в состояние режима супервизора⁷³. PSW пользователя сохраняется в ячейке памяти 32. После того как прерывание вызова супервизора было обслужено, привилегированная команда загрузки PSW (LPSW) восстанавливает старое PSW и возвращается в прерванную программу.

✓ Замечание

Если при нахождении процессора в режиме пользователя начнет выполняться привилегированная команда, на стадии декодирования команды будет возбуждено прерывание неправильного кода операции.

В IBM S390 были введены полупривилегированные команды, которые можно выполнять в режиме пользователя, если программа имеет соответствующие полномочия. Аналогично этому в процессоре Intel Pentium Pro предусмотрены четыре уровня привилегий. Уровень 0 является уровнем ядра операционной системы, уровни 1 и 2 отводятся под обслуживающие системные программы (драйверы устройств и проч.), а на уровне 3 располагаются приложения пользователя.

В настоящей главе мы обсудим привилегированные команды, их действие, типы данных и адреса. В разделе 8.7 будут описаны два класса расширенных команд пользователя: мультипроцессорная поддержка и MMX-усовершенствования процессора Intel Pentium.

8.1. Поддержка ядра операционной системы

В настоящем разделе рассматривается поднабор привилегированных команд, служащих для поддержки ядра операционной системы. Среди команд операционной системы имеется ряд команд, схожих с при-

⁷³ IBM называет этот бит P (problem state bit, бит проблемного состояния).

кладными, но использующих другие регистры. Процессор может включать набор регистров, адресуемых только привилегированными командами. В сущности, можно считать, что в процессор пользователя вложен другой процессор. Однако есть и общие ресурсы, такие, как кэш-память, ALU и регистры, которые используются всеми командами.

Состояние процессора включает в себя системные типы данных, представляющие собой коллекции битов, битовые поля и слова, назначенные регистрам, которые используются как индикаторы некоторых состояний процесса или процессора. Эти регистры содержат биты индикации, определяющие в каждый момент времени операционный режим и условия работы процессора. Регистры состояния процессора обычно содержат биты индикации, о которых шла речь в гл. 3 в связи с действием команд переходов. Помимо данных типа регистров состояния определены адреса памяти, связанные с этими типами данных, а также операции, которые могут над ними выполняться. В процессорах IBM S370 и Intel Pentium Pro имеются следующие регистры состояния:

IBM S370 [5]

PSW

Регистры управления

Intel Pentium Pro [8]

Регистр флагов

Регистры управления

Регистр-указатель команд

PSW, регистр флагов и регистр-указатель команд содержат биты, которые могут динамически изменяться в процессе выполнения программы. Управляющие регистры содержат биты, которые при выполнении программы обычно не модифицируются. Эти биты устанавливают операционный режим и участвуют в обмене с памятью в случае контекстного переключения или, в некоторых процессорах, при вызове подпрограммы.

Типы данных, описываемые в настоящей главе, содержат биты и поля в компьютерном слове. Студентам нет необходимости запоминать эти назначения, поскольку детальная информация всегда может быть получена из технических описаний. Однако нужно понимать, зачем созданы эти типы данных, и иметь общее представление об их функциях и назначении.

Типы данных ядра операционной системы нужны для управления системой, а не для получения какого-либо результата. Поэтому они никогда не покидают систему в качестве выходных данных. Эти типы данных передаются в специализированные регистры процессора, где они и выполняют предназначенные им функции. Такие данные могут

модифицироваться привилегированными командами в процессе выполнения программы или как побочный результат при выполнении прикладных команд.

IBM S360, S370 и S390

Семейство компьютеров IBM S360 с самого начала разработки было ориентировано на управление с помощью операционной системы. Для этих машин был предложен ряд аппаратных и программных концепций, облегчающих разработку и использование операционной системы. Рассмотрим типы данных, адреса и операции, поддерживающие ядро операционной системы.

Типы данных. На рисунке 8.1 показаны слова состояния программы PSW для режима базового управления (BC, basic control) IBM S360 и расширенного управления (EC, extended control) IBM S370. Машина IBM S370 представляет собой расширенный вариант S360 с включением виртуальной памяти, что требует дополнений к PSW S360. Обратите внимание на то, что программный счетчик, называемый здесь адресом команды, включен в состав PSW вместе с 2 битами признака результата (CC). Биты 0–7 служат для управления прерываниями.

4-битовый ключ обеспечивает защиту 2048-байтовых блоков памяти⁷⁴). Каждому блоку в памяти соответствует 7-битовый ключ, назы-

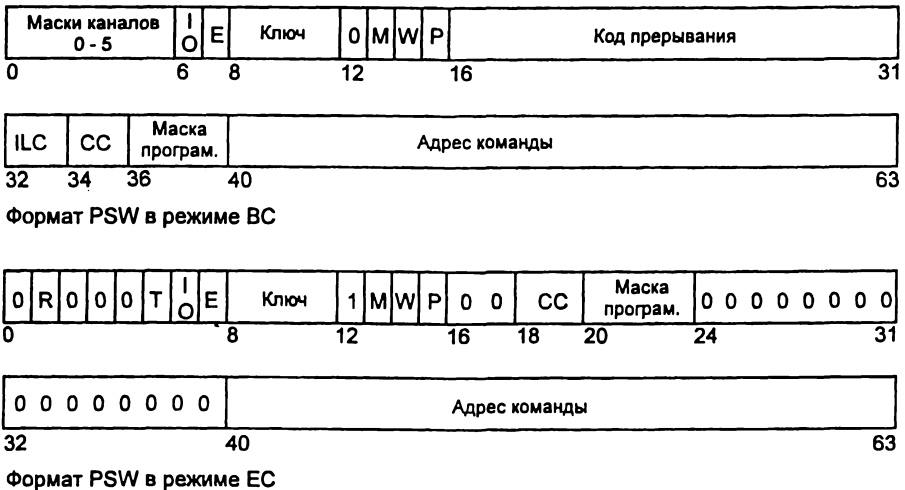


Рис. 8.1. PSW IBM S360, S370 (с любезного разрешения International Business Machines Corporation).

⁷⁴) Блок может соответствовать, но может и не соответствовать странице виртуальной памяти, поскольку S370 может работать с 4096-байтовыми страницами.

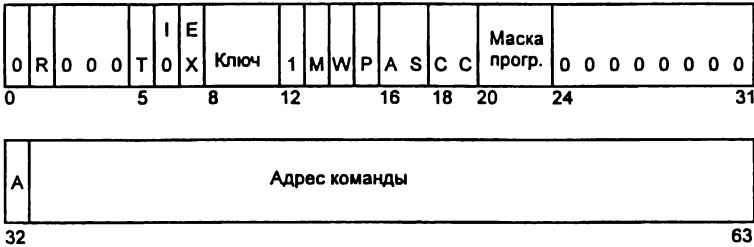


Рис. 8.2. PSW IBM S390 (с любезного разрешения International Business Machines Corporation).

аемый ключом защиты памяти. Для операций записи из CPU 4-битовый ключ в PSW должен соответствовать 4-битовому ключу защиты памяти. Чтение в CPU разрешается только, если ключи совпадают или если бит 4 ключа защиты памяти равен нулю. Этот механизм обеспечивает защиту от несанкционированных операций чтения или записи. Ключ в PSW может быть модифицирован привилегированными командами.

4-битовая маска программы расширяет код признака результата CC признаками переполнения фиксированной запятой, десятичного переполнения, исчезновения порядка плавающей запятой, а также потери значимости, когда результирующая мантисса операции с плавающей запятой равна нулю. Если какой-либо из битов маски устанавливается в 1, возбуждается прерывание. В этом случае ядро операционной системы определяет источник прерывания путем анализа кода признака результата, ключа, а также кода прерывания.

PSW IBM S390 для режима ЕС имеет тот же формат полей, за исключением адреса команды, ключа и некоторых других битов, как это показано на рис. 8.2. Основное отличие от формата S370 заключается в увеличении адреса команды с 24 до 31 бит. Полное описание PSW можно найти в справочной литературе.

В дополнение к типу данных PSW, в IBM S370 предусмотрен 16-словный 32-битовый регистровый файл управления (CR)⁷⁵⁾. Один или несколько битов регистров этого файла назначаются функциям, нуждающимся в управлении или спецификациях. Некоторые биты предназначены для трансляции адресов виртуальной памяти, другие — для параллельной обработки (например, маска времени дня). Часть битов контролирует запись событий, возникающих в программе, например, успешных переходов. Наконец, биты определяют, как осуществляется машинный контроль: с выходом из сеанса или без выхода. В таблице 8.1 перечислены основные функции каждого из слов CR. Более детальную информацию можно найти в описаниях архитектуры IBM S370.

⁷⁵⁾ В процессоре S360 не было регистрового файла управления; этот файл был добавлен в S370 и расширен в S390.

Таблица 8.1. Назначение слов регистрового файла управления
CR IBM S370

Слово	Основное назначение
0	Параллельная обработка, динамическая трансляция адреса, таймер
1	Динамическая трансляция адреса
2	Маски каналов
3	Свободно
4	Свободно
5	Свободно
6	Свободно
7	Свободно
8	Монитор
9	Запись событий программы
10	Запись событий программы
11	Запись событий программы
12	Свободно
13	Свободно
14	Машинный контроль
15	Машинный контроль

Адреса. Содержимое PSW и CR может переноситься в память или из нее с помощью специальных команд, которые требуют указания как источника, так и приемника. PSW и CR адресуются по умолчанию кодом операции команды, а память адресуется с помощью вычисленного эффективного адреса.

Система S370 предоставляет область реальной памяти с непосредственной адресацией; другими словами, эти адреса не отображаются системой виртуальной памяти. Эта область памяти с адресами 0–1FC используется для хранения слов состояния программы PSW, регистрового файла управления CR и информации о машинном контроле. Предусмотрено также место для хранения содержимого регистров плавающей запятой и общего назначения. Непосредственное отображение этой области предотвращает ее выгрузку на диск; ее содержимое всегда доступно процессору.

Операции. Перед тем как описывать операции, полезно познакомиться с шестью форматами команд IBM S370, приведенными на рис. 8.3. Команды могут иметь длину 2, 4 или 6 байт. Предусмотрено большое разнообразие вариантов адресации. Для отождествления упоминаемых далее команд с их форматами будут указываться их двухсимвольные коды форматов, например RR для формата регистр-регистр.

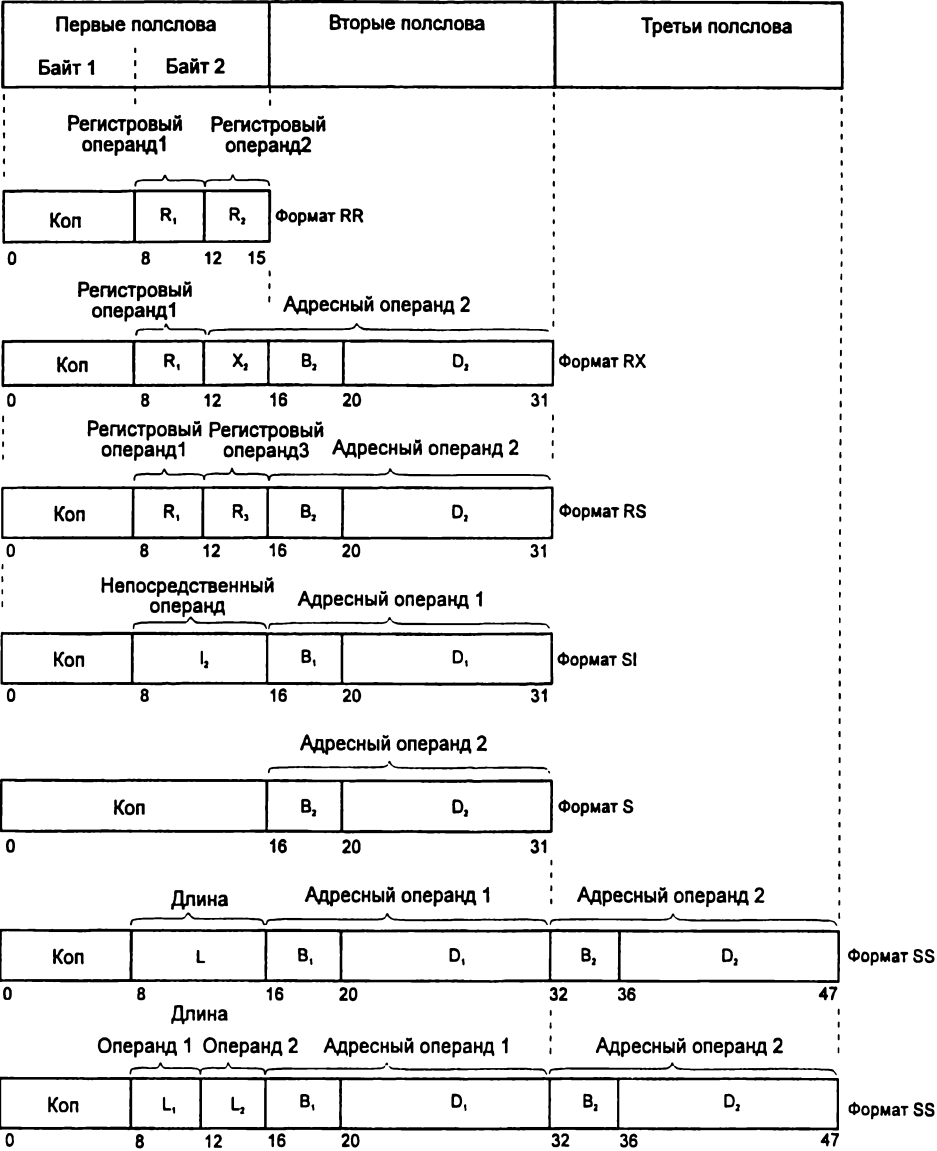


Рис. 8.3. Форматы команд IBM S370 (с любезного разрешения International Business Machines Corporation). Коп – код операции.

В IBM S360 предусмотрено 11 привилегированных команд: 5 для ядра (плюс 1 непривилегированная команда), 4 для ввода-вывода и 2 для параллельной обработки. Набор команд S360 невелик; в IBM S370 таких команд уже 26, а в IBM S390 – 55 (включая полупривилегиро-

ванные). В этой книге для простоты рассматривается набор команд S360.

В таблице 8.2 перечислены команды, используемые для модификации состояния процессора путем загрузки и запоминания битов PSW. Заметьте, что ключ защиты памяти может и загружаться в регистр, и сохраняться в памяти. Однако маска системы и маска программы могут быть только загружены в PSW. Диагностические команды вызывают аппаратные прерывания, характерные для каждой модели процессора.

Intel Pentium Pro

Процессор Intel Pentium Pro является потомком процессора Intel 8008, который при разработке не был ориентирован на управление с помощью операционной системы. Поэтому в Pentium Pro исходная простая архитектура была расширена добавлением в нее специальных средств. Операционная система использует четыре уровня привилегий, типы данных состояния процессора, а также адреса и операции, служащие для поддержки ядра операционной системы.

Типы данных. Типы данных состояния процессора в Intel Pentium Pro включают биты, назначенные регистру указателя команд (EIP), регистру флагов (EFLAG) и четырем управляющим регистрам CR⁷⁶. Все эти регистры являются аппаратными; они определяют порядок выполнения команд и операционные режимы процессора, а также оказывают помощь в трансляции виртуальных адресов.

Таблица 8.2. Команды, модифицирующие состояние процессора IBM S360

Название	Мнемоника (формат)	Функция
Включить ключ защиты памяти ⁷⁾	ISK (RR)	Ключ защиты памяти, соответствующий блоку в памяти, адресуемому через R ₂ , помещается в биты 24–28 регистра, адресуемого через R ₁
Установить ключ защиты памяти ⁷⁾	SSK (RR)	Биты 24–28 регистра, адресуемого через R ₁ , помещаются в ключ защиты памяти, соответствующий блоку памяти, адресуемому через R ₂
Загрузить PSW ⁷⁾	LPSW (SI)	Текущее содержимое PSW замещается двойным словом $m(D_2 + (B_2))$. Текущее PSW не сохраняется
Установить маску системы ⁷⁾	SSM (SI)	Адресуемый байт заменяет маску системы текущего PSW

⁷⁶⁾ Буква E была добавлена к обозначениям IP и FLAGS в процессоре Intel 386 вместе с расширением этих регистров до 32 бит (в Intel 286 эти регистры имели 16 бит).

Таблица 8.2. *Продолжение*

Название	Мнемоника (формат)	Функция
Установить маску программы	SPM (RR)	Биты 2–7 регистра, адресуемого через R ₁ , заменяют код признака результата и маску программы текущего PSW
Диагностика ^{*)}	Нет	CPU выполняет встроенные тесты аппаратных средств, специфические для конкретной модели

^{*)}Привилегированная

Регистр EIP принадлежит к 32-битовым типам данных; он указывает на первый байт команды. EIP инкрементируется после декодирования каждой команды; величина инкремента равна числу байтов в текущей команде.

На рисунке 8.4 показано распределение битов регистра флагов (битов индикации). Биты 0–7 уже обсуждались в гл. 3 (рис. 3.12) при рассмотрении битов индикации применительно к командам переходов. Детальную информацию об использовании битов регистра флагов можно найти в справочнике пользователя Intel Pentium Pro.

Четыре управляющих регистра Intel Pentium Pro показаны на рис. 8.5. Отметим некоторые интересные моменты, касающиеся этих регистров. База каталога страниц, хранящаяся в CR3, уже обсуждалась в гл. 4 (рис. 4.6) в связи с вопросом о расширении линейного адреса до 52 бит. Заметьте также, что в CR2 находится линейный адрес отказа страницы, который используется обработчиком отказа страниц для определения точки отказа.

Адреса. Указатель команд, регистр флагов и управляющие регистры CR реализованы аппаратно и загружаются из памяти или сохраняются в памяти с помощью регистров общего назначения, которые были приведены на рис. 3.8. Требуется указание двух адресов: источника и приемника. Для регистра флагов и регистров CR оба адреса неявно включены в код операции. Указатель команд адресуется неявно с помощью кода операции, когда этот регистр сохраняется при переключении контекста.

Операции. Для работы с регистром флагов и управляющими регистрами в составе ISA имеются только специальные команды пересылки, приведенные в табл. 8.3. Биты состояния можно перемещать в память или из нее через регистр АН, а содержимое управляющих регистров — с помощью регистров общего назначения, изображенных на рис. 3.8. Регистр флагов можно также протолкнуть в стек программы или вытолкнуть из него. Операции с указателем команд были описаны в разделе об обработке прерываний.

Таблица 8.3. Команды для работы с регистром флагов и управляющими регистрами

Название	Мнемоника	Функция
Загрузка флагов состояния в регистр АН	LAHF	Младший байт регистра EFLAG пересылается в регистр АН
Сохранение АН в битах флагов	SAHF	Флаги SF, ZF, AF, PF и CF пересылаются в регистр флагов. Биты 1, 3 и 5 не изменяются
Проталкивание EFLAGS в стек программы ^{*)}	PUSHF	Содержимое регистра флагов проталкивается в стек программы после декремента указателя стека
Выталкивание из стека программы в регистр флагов ^{*)}	POPF	Вершина стека программы выталкивается в регистр флагов, после чего указатель стека инкрементируется
Пересылка в/из управляющих регистров ^{*)}	MOV CRn	32-битовое слово пересылается между регистром общего назначения и одним из управляющих регистров. Всего имеется восемь таких команд

*) Привилегированная

Заметьте, что некоторые биты регистра флагов устанавливаются или сбрасываются как побочный результат выполнения прикладных команд. Состояние регистра флагов сохраняется при выполнении контекстного переключения.

8.2. Поддержка виртуальной памяти

Основными задачами команд поддержки виртуальной памяти ISA являются размещение в памяти, защита и управление таблицами сегментов и страниц, которые используются для трансляции виртуальных адресов в адреса реальной памяти. Принципы использования страниц и сегментов в системах виртуальной памяти были описаны в гл. 4.

IBM S370

Основное отличие IBM S370 от IBM S360 заключается в добавлении виртуальной памяти. В модели IBM S360 виртуальная память не была реализована из-за жестких временных рамок, в которых велись разработки аппаратного и программного обеспечения машины⁷⁷⁾.

⁷⁷⁾ Предвестник модели S370, машина S360 Model 67, уже имела виртуальную память.

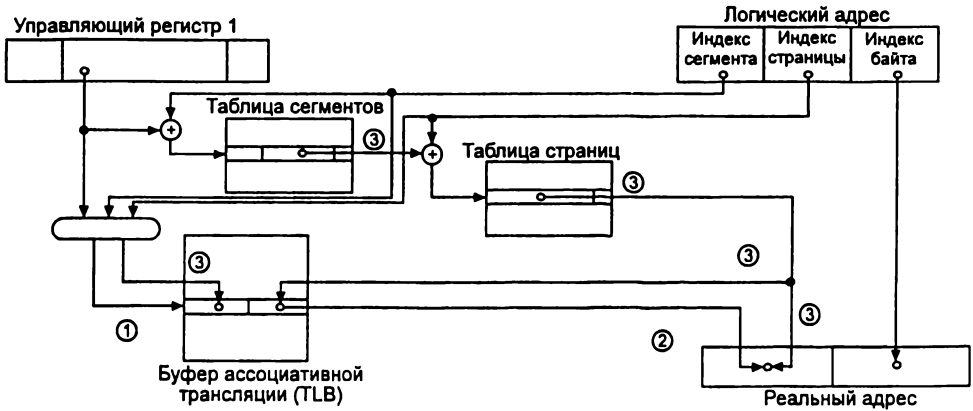


Рис. 8.6. Трансляция виртуальных адресов (с любезного разрешения International Business Machines Corporation).

В документации IBM виртуальная память носит название динамической трансляции адресов. В PSW для режима ЕС отведен бит 5 (Т), показанный на рис. 8.1, который, будучи установлен, включает механизм динамической трансляции адресов. Если же этот бит сброшен, все эффективные адреса являются непосредственными адресами реальной памяти. На рисунке 8.6, схожем с рис. 4.35, показан процесс трансляции адресов в IBM S370 с помощью сегментов и страниц.

Типы данных. Трансляция адреса требует четырех типов данных: адреса таблицы сегментов в CR1, адреса таблицы страниц в таблице сегментов, адреса страницы в таблице страниц и элементов TLB; все эти типы данных приведены на рис. 8.7. Управляющие регистры и TLB реализованы аппаратно, в то время как таблицы сегментов и страниц находятся в реальной памяти. Вспомним (см. гл. 4), что поля логического адреса могут иметь различную длину в зависимости от размеров сегментов и страниц.

В регистре CR 1 имеются два поля, 18-битовый адрес таблицы сегментов и 8-битовая длина сегмента; сегмент может занимать до 2^8 страниц. 6 бит остаются свободными. Индекс в таблице сегментов формируется путем сложения адреса таблицы сегментов, взятого из CR1, с индексом сегмента, извлеченным из логического адреса.

Элемент таблицы сегментов содержит 21-битовый адрес таблицы страниц, который, после сложения с индексом страницы из логического адреса образует индекс в таблице страниц. 4-битовое поле содержит длину таблицы страниц в единицах, равных $1/16$ от максимального размера таблицы. Это значение сравнивается с индексом страницы в логическом адресе, чтобы удостовериться, что адресуемый элемент таблицы страниц не находится за пределами таблицы страниц.

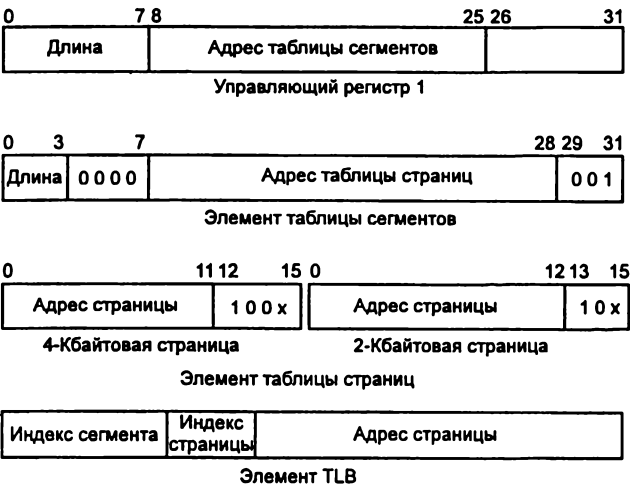


Рис. 8.7. Типы данных виртуальной памяти IBM S370.

IBM S370 поддерживает два размера страниц: 2 Кбайт и 4 Кбайт; выбор размера осуществляется с помощью битов 8 и 9 регистра CR0. Формат элемента таблицы страниц также приведен на рис. 8.7. Каждый элемент содержит 16 бит. Бит 12 или 13, в зависимости от размера страницы, является битом присутствия страницы. Если бит присутствия равен 0, трансляция продолжается. Если он равен 1, возбуждается прерывание трансляции страницы. Для формирования адреса реальной памяти адрес страницы сцепляется с индексом байта из логического адреса.

Элемент буфера ассоциативной трансляции TLB включает теги, входящие в состав полей индексов сегмента и страницы. Теги устанавливают наличие элемента и адреса страницы реальной памяти. Размер и организация TLB зависит от конкретной модели IBM S370.

Адреса. Типы данных поддержки виртуальной памяти назначают полям адресов, как это показано в табл. 8.4. Заметьте, что для элементов таблиц сегментов и страниц используются нетранслированные адреса реальной памяти. В результате эти таблицы располагаются в реальной памяти, так что при обращении к ним в процессе обработки отказа страницы исключается вторичный отказ страницы.

Операции. Для явной загрузки и запоминания управляющих регистров предусмотрены две привилегированные команды (см. табл. 8.5). Непривилегированная команда пересылки (из набора прикладных команд) помещает элементы таблицы сегментов и таблицы страниц на выделенные для них места в памяти. Команда PTLB очищает TLB, а команда SPX выполняет то же самое как побочный результат.

Таблица 8.4. Типы данных поддержки виртуальной памяти IBM S370

Тип данного	Назначение	Адрес
CR1	Аппаратный регистр	Подразумевается в коде операции
Элемент таблицы сегментов	Реальная память	Нетранслированный адрес реальной памяти
Элемент таблицы страниц	Реальная память	Нетранслированный адрес реальной памяти
TLB	Аппаратные регистры	Подразумеваются в командах, обрабатывающих отказ страницы

Таблица 8.5. Команды поддержки виртуальной памяти IBM S370

Название	Мнемоника (формат)	Функция
Загрузка управляющих регистров ^{*)}	LCTI (RS)	Набор управляющих регистров от (R ₁) до (R ₃) загружается из $m(D_2 + (B_2))$
Запоминание управляющих регистров ^{*)}	STCTL (RS)	Набор управляющих регистров от (R ₁) до (R ₃) запоминается в $m(D_2 + (B_2))$
Пересылка	MVC (SS)	Байты в количестве L пересылаются из $m(D_1 + (B_1))$ в $m(D_2 + (B_2))$
Очистка TLB ^{*)}	PTLB (S)	Все данные в TLB помечаются как недостоверные
Установка префикса ^{*)}	SPX (S)	SPX используется в многопроцессорных конфигурациях. Пометка TLB как недостоверного является побочным результатом.

^{*)} Привилегированная

Intel Pentium Pro

Intel Pentium Pro использует систему виртуальной памяти со страницами и сегментами, как это было описано в разделе 4.6. На рисунке 8.8 показана система образования виртуального адреса вместе со страничной трансляцией для 4-Кбайтовых страниц. Этот рисунок повторяет рис. 4.36. Другая конфигурация трансляции виртуальных адресов предусмотрена для 4-Мбайтовых страниц; выбор способа трансляции определяется состоянием бита 4 (PSE) управляющего регистра CR4, показанного на рис. 8.5.

Типы данных. Для поддержки виртуальной памяти с сегментами и страницами в процессоре Pentium Pro предусмотрены шесть типов данных. К ним относятся селекторы сегментов в сегментных регистрах, управляющий регистр CR3, элементы таблиц дескрипторов сегмен-

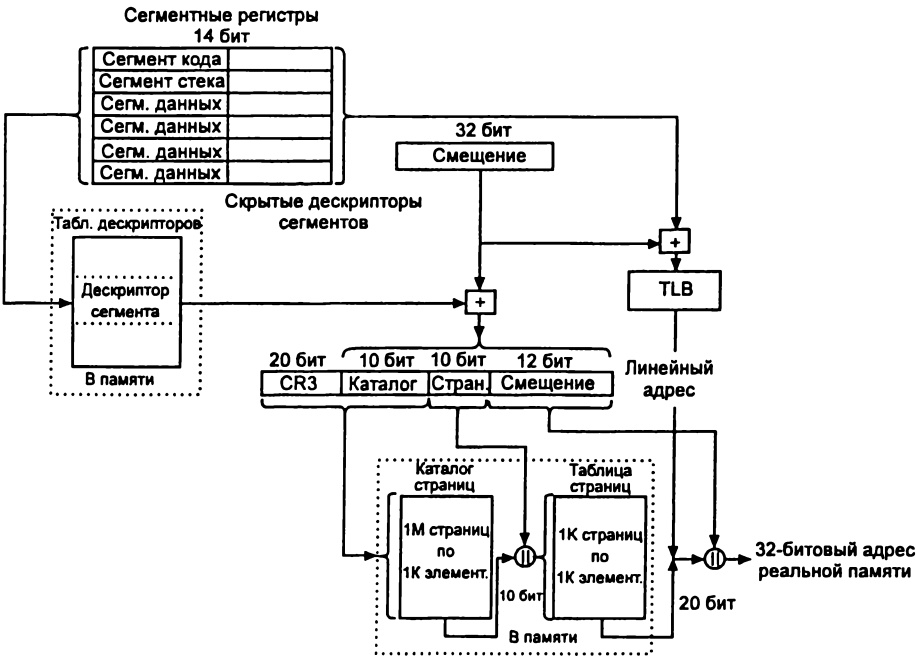


Рис. 8.8. Система виртуальных адресов с сегментацией в Pentium Pro.

тов, элементы таблиц каталога страниц, элементы таблиц страниц и элементы TLB.

На рисунке 8.9 показан формат данных типа селектор сегмента. Этот тип данных хранится в аппаратных регистрах, как это было описано в разделе 4.6, и содержит 14-битовый индекс в таблице дескрипторов сегментов. Всего имеются два типа таблиц дескрипторов сегментов, глобальный и локальный; их выбор осуществляется с помощью бита 2 селектора. Два бита RPL определяют один из четырех уровней привилегий доступа, кратко затронутых в разделе 8.0.

Назначение битов управляющих регистров было приведено на рис. 8.5. 20 старших битов CR3 расширяют линейный адрес до 52 бит.

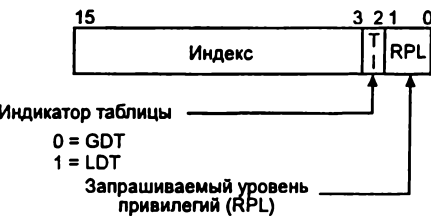
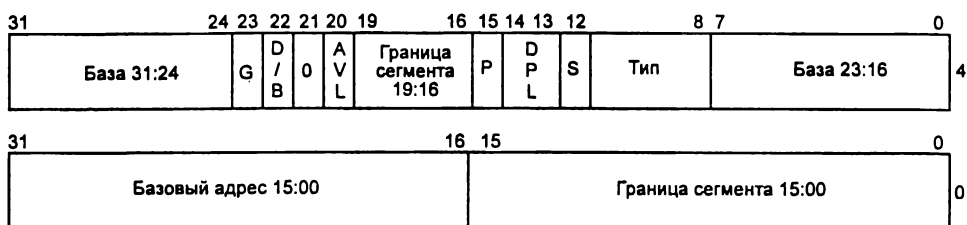


Рис. 8.9. Селектор сегмента (с любезного разрешения Intel Corporation).



AVL - Свободен для использования системным программным обеспечением
D/B - Размер операции по умолчанию (0=16-битовые сегменты, 1=32-битовые сегменты)
DPL - Уровень привилегий дескриптора
G - Дробность
P - Присутствие сегмента
S - Тип дескриптора (0=системный, 1=прикладной)

Рис. 8.10. Данное типа дескриптора сегмента Pentium Pro (с любезного разрешения Intel Corporation).

Биты PCD и PWT управляют кэшированием каталога страниц и элементов таблиц страниц в кэше данных процессора.

На рисунке 8.10 показан тип данных дескрипторов сегмента. Дескриптор занимает 8 байт, изображенных на рисунке в виде двух 32-битовых слов с номерами 0 и 4. Двумя важнейшими компонентами дескриптора сегмента являются базовый адрес сегмента и его предел. 32-битовый базовый адрес формируется из битов 16–31 слова 0, а также битов 0–7 и 24–31 слова 4. Аналогично граница сегмента формируется из битов 0–15 слова 0 и битов 16–19 слова 4, составляющих в сумме 20 бит.

Обратите внимание на флаг дробности G, входящий в дескриптор сегмента. Если этот флаг равен 0, размер сегмента может устанавливаться от 1 байт до 1 Мбайт с дробностью 1 байт. Если G равен 1, размер сегмента может принимать значения от 4 Кбайт до 4 Гбайт с дробностью 4 Кбайт (что как раз соответствует размеру страницы).

Если значение 20 старших битов 32-битового смещения, или эффективного адреса процессора больше 20-битового поля границы сегмента, возбуждается прерывание ошибки адресации; это прерывание должно обрабатываться операционной системой.

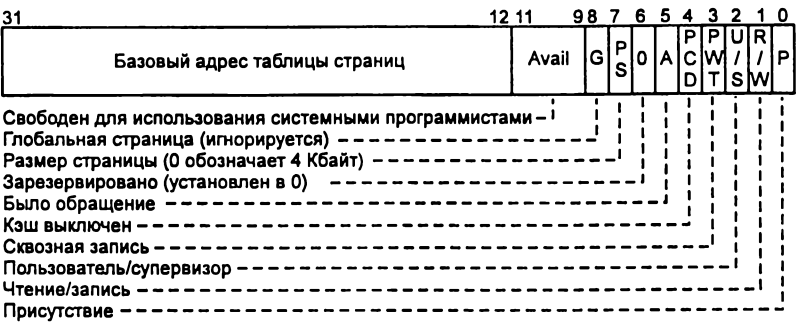
✎ Пример

Привести выкладки, показывающие максимальный размер сегмента для двух значений дробности.

✎ Решение

Граница сегмента равна 2^{20} единиц дробности. Поэтому для $S = 0$ размер (1-байт приращение) $= 1 \times 2^{20} = 2^{20} = 1$ Мбайт

Элемент каталога страниц (4-Кбайтовые таблицы страниц)



Элемент таблицы страниц (4-Кбайтовые страницы)

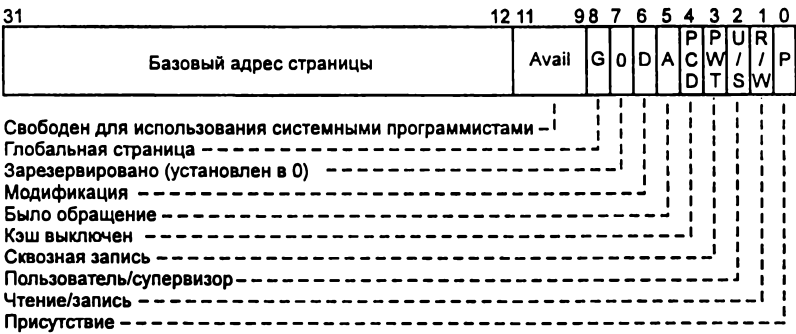


Рис. 8.11. Элементы каталога страниц и таблицы страниц Pentium Pro (с любезного разрешения Intel Corporation).

для $S = 1$ размер (4-Кбайт приращение) = $2^{12} \times 2^{20} = 2^{32} =$
= 4 Гбайт

Вспомним (см. рис. 8.8), что линейный адрес формируется путем сложения 32-битового базового адреса из дескриптора сегмента с 32-битовым смещением. 10 старших битов линейного адреса сцепляются с 20-ю битами из CR3 с образованием индекса в каталоге страниц, который также показан на рис. 4.6.

Элементы каталога страниц и таблицы страниц показаны на рис. 8.11. Каталог страниц индексируется с целью нахождения базового адреса таблицы страниц. Этот базовый адрес таблицы страниц сцепляется с 10 битами, извлеченными из линейного адреса, с образованием индекса в таблице страниц.

Элементы каталога страниц и таблицы страниц практически одинаковы; в них содержатся 20-битовый базовый адрес таблицы страниц и 20-битовый базовый адрес страницы. Обратите внимание на бит

присутствия Р. В элементе каталога страниц этот бит указывает на наличие или отсутствие в памяти элемента таблицы страниц, в то время как бит Р в элементе таблицы страниц указывает на наличие или отсутствие самой этой страницы в реальной памяти.

Организация TLB Pentium Pro была показана на рис. 4.21. Тип данного представляет собой предварительно транслированный базовый адрес страницы реальной памяти; такой же тип можно найти в элементе таблицы страниц. Бит достоверности позволяет объявить любой элемент недостоверным с помощью привилегированной команды.

Адреса. В таблице 8.6 показаны шесть типов данных, служащих для поддержки виртуальной памяти, с указанием мест их расположения и способов адресации.

Таблица 8.6. Типы данных поддержки виртуальной памяти Pentium Pro

Тип данного	Расположение	Адрес
Селектор сегмента	Аппаратный регистр	Неявно задается способом обращения к памяти: CS, SS, DS, ES, FS или GS.
CR3	Аппаратный регистр	Автоматически сцепляется с линейным адресом. Подразумевается в коде операции или как приемник при пересылках.
Дескриптор сегмента	Реальная память	Нетранслированная реальная память
Таблица каталога страниц	Реальная память	Нетранслированная реальная память.
Элемент таблицы страниц	Реальная память	Нетранслированная реальная память.
TLB	Аппаратный регистр	Подразумевается в командах обработки отказов страниц.

▣ Пример

При наличии в реальной памяти таблицы дескрипторов, каталога страниц и таблицы страниц какую долю пространства реальной памяти (максимум) могут занимать эти таблицы? Общий объем памяти составляет 2^{32} байт.

✍ Решение

Таблица дескрипторов индексируется 14-битовым указателем, а дескрипторы сегментов имеют размер 4 байт^{*)}.

^{*)} При длине дескриптора 8 байт результат по формуле (1) составит $100 \times 2^{14} \times 2^{3/2^{32}} = 0,003\%$. — Прим. пер.

Максимальная доля памяти, занимаемая этим объектом, равна $100 \times 2^{14} \times 2^3/2^{32} = 0,0015\%$ (1)

Каталог таблиц индексируется 30-битовым указателем, а элементы каталога имеют размер 4 байт. Максимальная доля памяти, занимаемая этим объектом, равна

$$100 \times 2^{30} \times 2^2/2^{32} = 100\%$$

Таблица страниц индексируется 20-битовым указателем, а элементы таблицы страниц имеют размер 4 байт. Максимальная доля памяти, занимаемая этим объектом, равна

$$100 \times 2^{20} \times 2^2/2^{32} = (100 \times 2^{22})/2^{32} = 0,1\%$$

Операции. В таблице 8.7 собраны команды обслуживания типов данных, поддерживающих виртуальную память. Поскольку данные типа дескриптора сегмента, каталога страниц и таблицы страниц сохраняются в реальной памяти, для загрузки и запоминания этих типов данных достаточно непривилегированных пользовательских команд пересылки. Если, однако, команды пересылки используются с управляющими регистрами, а процессор находится в защищенном или реальном режимах, будут возбуждаться прерывания. Поэтому для выполнения пересылок с управляющими регистрами в качестве операндов автоматически используется привилегированная форма команд пересылки.

Таблица 8.7. Команды поддержки виртуальной памяти Intel Pentium Pro

Название	Мнемоника	Функция
Пересылка в/из сегментного регистра	MOV	Пересылает 16-битовое слово между сегментным регистром и памятью.
Объявление недоверным элементом TLB ^{*)}	INVLPG	Объявляет недоверным элемент TLB, указанный операндом-источником, адресующим память.
Пересылка в/из управляющего регистра CR ^{*)}	MOV CR3	Объявляет недоверными все элементы TLB, если выполняется запись в CR3.
Возврат из процедуры	RET	Как побочный результат команды RET загружается сегментный регистр CS.

^{*)} Привилегированная

Пять селекторов сегментов (за исключением CS) загружаются командами MOV. Элементы TLB загружаются лишь как побочный эффект при обработке отказа страницы. Однако для объявления недоверными элементов TLB предусмотрена специальная команда INVLPG. TLB объявляется недоверным также как побочный эффект некоторых других команд. Содержимое TLB никогда не сохраняется в памяти, даже при контекстных переключениях.

8.3. Поддержка прерываний

Общие принципы прерываний были описаны в гл. 3. В настоящем разделе мы рассмотрим особенности расширений наборов команд для обслуживания прерываний в процессорах IBM S360 и Intel Pentium Pro.

IBM S360

Мы уже обсуждали вопросы об источниках прерываний и о том, как прерывания обслуживаются. Процессор IBM S360 — с аппаратным и микропрограммным управлением и конвейером — обслуживает прерывания таким же образом. Не отличаются в этом отношении и процессоры IBM S370 и S390. В этом разделе мы остановимся на использовании слова состояния программы PSW в процессе обслуживания прерывания и при возвращении управления в прерванную программу.

Вспомним (см. рис. 8.1), что адрес команды хранится в PSW. На рисунке 8.12 текущее активное PSW обозначено, как «Текущее PSW». Показаны также два набора из пяти ячеек памяти для PSW, названные «Старые PSW» и «Новые PSW». Каждая пара PSW соответствует одному из пяти классов прерываний. На рисунке показана также область памяти, названная «Область сохранения».

Когда возникает прерывание, определяется его класс, и аппаратура пересылает текущее PSW в ячейку для старого PSW, соответствующую классу прерывания (шаг 1). Кроме этого, аппаратура перемещает соответствующее новое PSW в регистр текущего PSW (шаг 2). Это PSW было заранее подготовлено операционной системой; оно запустит обработчик прерывания соответствующего класса. На шаге 3, также вы-

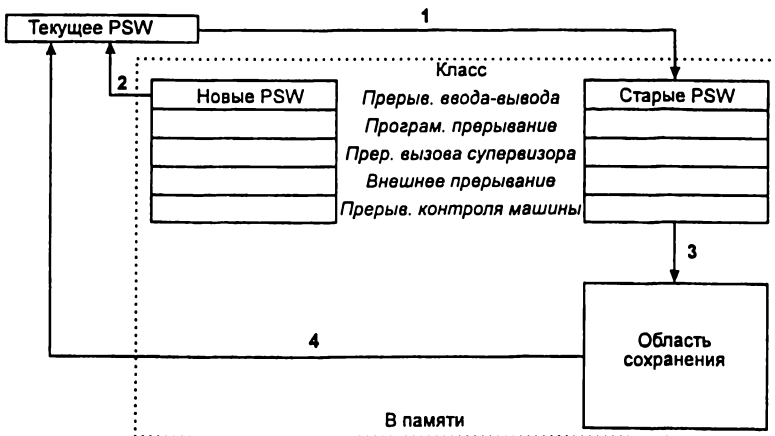


Рис. 8.12. Обслуживание прерываний в IBM S360.

полняемом аппаратно, старое PSW перемещается в область сохранения, освобождая ячейку старого PSW для другого прерывания того же класса. Когда обработчик прерывания завершит свою работу, он выполнит команду LPSW, которая вернет текущее PSW в свое исходное состояние; шаг 2 или 4.

Типы данных. В IBM S360 к типам данных, поддерживающим прерывания, относится только PSW (см. рис. 8.1). Как уже упоминалось, PSW процессоров IBM S360, S370 и S390 различаются.

Адреса. На рисунке 8.12 были показаны три области реальной памяти, участвующие в обслуживании прерываний: новые PSW, старые PSW и область сохранения. Для работы с ними требуются два адреса: адрес неотображенной памяти и адрес, подразумеваемый в коде операции. Команды, в качестве побочного результата которых происходит перемещение PSW, также используют адреса неотображенной памяти.

Операции. Единственная явная команда для поддержки прерываний — это команда LPSW, которая восстанавливает состояние регистра текущего PSW (табл. 8.8). В зависимости от адреса, загрузка осуществляется либо из нового PSW, либо из области сохранения. Операционная система для подготовки в памяти различных PSW использует и другие непривилегированные команды.

Таблица 8.8. Команда поддержки прерываний IBM S360

Название	Мнемоника (формат)	Функция
Загрузка PSW	LPSW (SI)	Текущее PSW заменяется двойным словом из ячейки $m(D_2 + (B_1))$. Текущее PSW не сохраняется.

Intel Pentium Pro

Pentium использует векторную систему прерываний. Если возбуждается прерывание или исключение, состояние процессора проталкивается в стек. Возврат из прерывания выталкивает сохраненные в стеке данные и восстанавливает состояние прерванной программы.

Типы данных. Перечисленные в табл. 8.9 типы данных, связанные с состоянием процессора, сохраняются в ответ на возбуждение прерывания или исключения и восстанавливаются при возврате.

Адреса. Регистры, содержащие эти данные, адресуются неявным образом при возбуждении прерывания или исключения, при возврате из прерывания или исключения, а также как побочный результат инкрементирования или декрементирования стека.

Таблица 8.9. Типы данных поддержки прерываний процессора Intel Pentium Pro

Название	Описание
EFLAGS	Регистр флагов, см. рис. 3.12.
CS	Селектор сегмента команд ^{*)} .
Регистр EIP	Указатель команд, содержащий текущее значение программного счетчика.
Регистр ESP	Указатель стека, см. рис. 3.7.

^{*)} Сегментный регистр команд содержит селектор сегмента, который служит индексом в таблице дескрипторов.

Операции. Некоторые из операций, которые загружают, запоминают или изменяют типы данных поддержки прерываний, перечислены в табл. 8.10. Приведенный список не полон; он служит лишь для иллюстрации предоставляемых процессором типов команд.

Таблица. 8.10. Команды поддержки прерываний процессора Intel Pentium Pro

Название	Мнемоника	Функция
Вызов процедуры обработки прерывания	INT	Вызывает обработчик прерывания, находящийся по адресу приемника. В качестве побочного результата проталкивает в стек EFLAGS, CS и EIP.
Возврат из процедуры	RET	В качестве побочного результата выталкивает из стека EFLAGS, CS и EIP.
Пересылка ESP	MOV ESP	Загружает указатель стека из ячейки с эффективным адресом в реальной памяти.
Выталкивание из стека	POP	Выталкивает из стека содержимое регистров DS, ES, SS, FS или GS и инкрементирует указатель стека.

8.4. Поддержка ввода-вывода

Общие принципы ввода-вывода были описаны в гл. 7. В настоящем разделе рассматриваются элементы, служащие для поддержки ввода-вывода, применительно к двум процессорам: IBM S360 и Intel Pentium Pro.

Операции. Всего в IBM S360 имеются четыре команды ввода-вывода, перечисленные в табл. 8.11. Операция ввода-вывода инициируется CPU с помощью привилегированной команды SIO. Эта команда перемещает значение программного счетчика из CAW, находящегося в ячейке реальной памяти с адресом 72, в программный счетчик адресуемого канала, как это показано на рис. 8.14. Канал начинает выполнение своих команд, хранящихся в основной памяти. В любой момент CPU с помощью привилегированной команды TIO может загрузить в PSW информацию о состоянии канала и устройства. Команда TCH помещает в текущее PSW информацию о состоянии канала. Команда HIO останавливает выполнение текущей канальной программы и передачу данных.

Таблица 8.11. Команды поддержки ввода-вывода IBM S360

Название	Мнемоника (формат)	Функция
Начать ввод-вывод*) SIO (S)		Пересылает CAW (адрес 72) в канал, адресуемый через $(D_1 + (B_1))$ и заставляет канал начать выполнение канальной программы.
Тестировать ввод-вывод*) TIO (S)		Устанавливает признаки результата для состояния канала и устройства в PSW процессора по адресу $(D_1 + (B_1))$.
Тестировать канал*) TCH (S)		Устанавливает признаки результата для состояния канала в PSW процессора по адресу $(D_1 + (B_1))$.
Остановить ввод-вывод*) HIO (S)		Завершает операции канала и устройства.

*) Эти команды могут предвшаться префиксом REP (повторение) для пересылки строк.

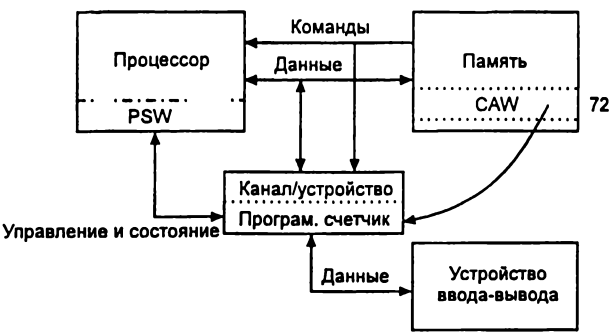


Рис. 8.14. Операции ввода-вывода IBM S360.

Intel Pentium Pro

Процессор Pentium обеспечивает два способа подключения внешних устройств: через порты ввода-вывода и с помощью отображения ввода-вывода на память. Ввод-вывод Pentium Pro является результатом эволюции многих поколений процессоров, поддерживая при этом совместимость с ранними процессорами и программами.

Типы данных. С вводом-выводом, в сущности, не связаны никакие типы данных. Операции ввода-вывода могут пересылать биты между регистрами, ячейками памяти и портами. Эти биты назначаются конкретным операциям ввода-вывода, выполняемым системой.

Адреса. Пространство ввода-вывода представляет собой отдельное адресное пространство из 64К 8-битовых портов, пронумерованных от 0 до FFFFh. Два последовательных 8-битовых порта могут использоваться, как 16-битовый порт, и два последовательных 16-битовых порта могут использоваться, как 32-битовый порт. Адресное пространство ввода-вывода изображено на рис. 7.5.

Операции. Предусмотрен набор команд, выполняющих операции ввода-вывода через порты. Однако порты назначены адресным линиям памяти процессора. Команды ввода-вывода указывают, что на адресных линиях установлен адрес ввода-вывода, а не адрес памяти⁷⁸⁾. Данные передаются от устройств ввода-вывода или к ним с помощью линий данных памяти. Команды ввода-вывода с частичным описанием перечислены в табл. 8.12. Эти команды не являются привилегированными. Команды ввода-вывода с отображением на память обсуждались ранее в гл. 7. Заметьте, что для передачи данных по адресам ввода-вывода, отображенного на память, можно использовать любые прикладные команды пересылки.

Таблица 8.12. Команды ввода-вывода процессора Intel Pentium Pro

Название	Мнемоника	Функция
Ввод из порта	IN	Вводит байт, слово или двойное слово в регистр AL, AX или EAX.
Вывод в порт	OUT	Выводит байт, слово или двойное слово из регистра AL, AX или EAX.
Ввод из порта в строку ^{*)}	INS	Вводит байт, слово или двойное слово в память по адресу ES:[EDI].
Вывод строки в порт ^{*)}	OUTS	Выводит байт, слово или двойное слово из памяти с адресом DS:[ESI].

^{*)} Эти команды могут предваряться префиксом REP (повторение) для пересылки строк.

⁷⁸⁾ Процессоры, более ранние, чем Pentium Pro, имели специальный вывод для сигнала различения обмена с памятью и передач ввода-вывода.

8.5. Поддержка кэш-памяти

Первоначально концепция кэш-памяти была неразрывно связана с прозрачностью этой памяти с точки зрения пользователя — кэш просто уменьшал запаздывание памяти. В некоторых современных процессорах такой подход уже не оправдывается. Часто прямое вмешательство программы позволяет обеспечить дополнительные возможности и рост производительности по сравнению с прозрачным интерфейсом с памятью. Однако концепция прозрачного кэша по-прежнему используется в процессоре IBM S390. Документ «Принципы действия», входящий в техническую документацию к IBM S390, на с. 2-2 утверждает следующее: «Программа не замечает влияния физической организации или использования выделенной запоминающей среды, кроме как на повышение производительности» [6]. Поэтому семейство IBM S360 не удастся использовать для иллюстрации поддержки кэш-памяти с помощью расширенного набора команд. Вместо этого мы рассмотрим процессоры PowerPC и Pentium Pro.

События, следующие за кэш-промахом, рассматриваются как побочные результаты команд загрузки и запоминания. Проектировщики некоторых современных процессоров считают, что эта точка зрения может быть развита с предоставлением программе возможности прямого вмешательства в работу кэш-памяти. Такое программное вмешательство находит свое применение в двух областях:

- 1) в прямом инициировании побочных результатов, как основной цели команды;
- 2) задании стратегии кэш-памяти для обеспечения большей гибкости, чем предусматривается аппаратно реализованной стратегией по умолчанию.

PowerPC

Команды поддержки кэш-памяти команд и данных обсуждаются здесь только в качестве примера. В силу функционального различия кэшей команд и данных для их поддержки предусмотрены отдельные наборы команд. Детальное описание этих команд можно найти в работе [10].

Управление влиянием побочных результатов на кэш-память. В процессоре PowerPC предусмотрены шесть команд, которые позволяют осуществлять непосредственное управление кэшами команд и данных. Две из этих поддерживающих команд (см. табл. 8.13) обеспечивают повышение производительности системы памяти. Четыре команды поддерживают мультипроцессорные приложения, и они будут описаны здесь, а не в разделе 8.6, посвященном мультипроцессорным системам. Эти команды используются для повышения уровня согласованности кэш-памяти в мультипроцессорных системах.

Таблица 8.13. Команды поддержки кэш-памяти PowerPC

Название	Мнемоника	Функция	Назначение
Обращение к блоку кэша данных	dcbt	Выбирает кэш-блок, соответствующий виртуальному адресу $((RA) + (RB))$, и помещает его в кэш, как было бы сделано в качестве побочного результата при промахе.	Заранее выбирает кэш-блок перед тем, как он фактически понадобится. Последующая команда загрузки, использующая этот адрес, приведет к попаданию в кэш.
Обращение к блоку кэша с целью запоминания данных	dcbst	Выбирает кэш-блок, соответствующий виртуальному адресу $((RA) + (RB))$, и помещает его в кэш, как было бы сделано в качестве побочного результата при промахе в модифицированном блоке.	Заранее выбирает кэш-блок перед тем, как он фактически понадобится. Последующая команда запоминания, использующая этот адрес, приведет к кэш-попаданию. Блок находится в кэш-памяти; поэтому операция выделения не требуется для команды запоминания.
Сброс в ноль кэша данных	dcbz	Устанавливает в ноль все байты в блоке с адресом $((RA) + (RB))$.	Используется для обеспечения согласованности кэш-памяти.
Запоминание блока кэша данных	dcbst	Записывает в память блок с адресом $((RA) + (RB))$.	Используется для обеспечения согласованности кэш-памяти.
Очистка блока кэша данных	dcbf	Для блока с адресом $((RA) + (RB))$: если не модифицирован, объявить недостоверным; если модифицирован, скопировать в память; если отсутствует, объявить недостоверными другие кэши процессора.	Используется для обеспечения согласованности кэш-памяти.

Таблица 8.13. *Продолжение*

Название	Мнемоника	Функция	Назначение
Объявление недостоверным блоку кэша обеспечения команд	icbi	Объявить недостоверным блок с адресом ((RA) + (RB)).	Используется для обеспечения согласованности кэш-памяти.
Синхронизация команд	isync	Ожидает завершения всех предыдущих команд, затем отбрасывает все предварительно выбранные команды.	Используется для обеспечения контекстной синхронизации.

Типами данных являются блоки кэш-памяти, а адресами являются транслированные адреса реальной памяти с соответствующими адресами кэш-памяти.

Команды поддержки кэш-памяти не являются привилегированными, поскольку они предназначены для использования в прикладных программах для повышения производительности и обеспечения согласованности кэш-памяти в мультипроцессорных системах. Первые две команды в таблице имеют своей основной целью загрузку кэш-памяти еще до того момента, когда эти данные потребуются, что позволяет повысить производительность системы.

Управление кэш-стратегией. Вспомним (гл. 4), что организация и стратегии кэш-памяти выбираются таким образом, чтобы удовлетворить ожидаемому характеру последовательности адресации. Однако операционная система может динамически настраивать некоторые параметры кэш-стратегии, чтобы она соответствовала ожидаемому характеру коротких последовательностей адресации.

Тип данных представляет собой набор битов режима, которые хранятся в элементах таблиц страниц. Этот тип данных включает в себя три бита, управляющих стратегией, или режимом: W, I и M. Использование этих битов описано в табл. 8.14. Стратегии определяются по отношению к страницам, поэтому разные страницы могут использовать разные стратегии. Кэш-адреса — это транслированные адреса реальной памяти.

Операции. Поскольку биты управления режимом назначены 128-битовым элементам таблиц страниц, только команды записи и запоминания таблиц страниц в реальной памяти имеют отношение к управлению кэш-режимом. Поскольку PowerPC присуща архитектура загрузки-запоминания, таблицы страниц пересылаются в памяти последовательностью команд загрузки и запоминания.

Таблица 8.14. Управление кэш-стратегиями в PowerPC

Бит	Назначение
W	Сквозная запись. Запись должна обновить более высокие уровни кэша, реальную память или ввод-вывод. В противном случае используется обратная запись.
I	Кэширование запрещено. В противном случае все обращения кэшируются.
M ^{*)}	Согласованность памяти. Вся память должна быть согласована. В противном случае — несогласована.

^{*)} Полезна для мультипроцессорных приложений, кратко затронутых в разделе 4.3.

Intel Pentium Pro

Процессор Intel Pentium Pro может быть использован в мультипроцессорных конфигурациях, и для него желательно иметь средства оптимизации программ. По этой причине в процессоре предусмотрены команды для прямого управления его кэш-памятью.

Управление влиянием побочных результатов на кэш-память. Процессор Intel Pentium Pro предоставляет две команды, непосредственно воздействующие на кэши уровней L1 и L2 и косвенно — на кэш уровня L3. Эти команды (они являются привилегированными) приведены в табл. 8.15.

Термин «недостоверный» (invalid) используется для обозначения того, что биты достоверности (V) установлены в состояние недостоверности. Некоторые авторы используют для описания того же действия термины «очистка» (purge) или «сброс» (flush). Последний термин может ввести в заблуждение, поскольку в конвейерных процессорах он используется для обозначения того, что выполняемая работа прошла сквозь конвейер и завершена.

Управление кэш-стратегией. На рисунке 8.4. были изображены управляющие регистры CR Pentium Pro. В регистре CR0 биты 30 (CD — выключение кэша) и 29 (NW — без сквозной записи) устанавливают стратегию кэш-памяти, как это показано в табл. 8.16. Из-за того что биты стратегии расположены в CR0, стратегия является глобальным понятием. Вспомним, что, в отличие от этого, в PowerPC стратегия кэш-памяти устанавливается постранично.

Два бита управляющего регистра определяют стратегию кэширования для таблиц страниц. Стратегия устанавливается в регистре CR3 битами 4 (PCD — запрет кэширования на уровне страниц) и 3 (PWT — прозрачная запись на уровне страниц). Стратегии, устанавливаемые с помощью этих битов, показаны в табл. 8.17.

Таблица 8.15. Команды поддержки кэш-памяти процессора Intel Pentium Pro

Название	Мнемоника	Функция	Назначение
Объявить недостоверными внутренние кэши	INVD	Объявляет недостоверными кэши уровней L1 и L2 и вызывает специальный цикл шины для инициирования объявления недостоверным внешнего кэша L3. Данные в кэшах L1, L2 и L3 не записываются обратно в реальную память ^{*)} .	Позволяет операционной системе объявить кэш недостоверным при перезагрузке системы или контекстном переключении.
Обратная запись с объявлением кэша недостоверным	WBINVD	Выполняется обратная запись всех модифицированных блоков кэшей L1 и L2 в реальную память, после чего блоки объявляются недостоверными. Вызывает специальный цикл шины для выполнения обратной записи кэша L3 и объявления его недостоверным.	Позволяет операционной системе объявить кэш недостоверным при перезагрузке системы или контекстном переключении. Выполняется обратная запись всех кэшей, приводящая к обновлению всех изменений.

^{*)} Кэши данных и команд уровня L1 находятся на кристалле процессора. Общий кэш уровня L2 находится в том же корпусе, а кэш уровня L3 находится вне корпуса процессора (см. рис. 4.10).

Таблица 8.16. Стратегия кэш-памяти процессора Intel Pentim Pro

CD	NW	Кэш-стратегия
0	0	Обратная запись
0	1	Сквозная запись
1	x	Кэш-память отключена

Таблица 8.17. Стратегия страниц кэш-памяти процессора Intel Pentim Pro

PCD	PWT	Кэш-стратегия
0	x	Таблица страниц не кэшируется
1	0	Таблица страниц кэшируется, сквозная запись
1	1	Таблица страниц кэшируется, обратная запись

8.6. Мультипроцессорная поддержка

Современные компьютеры представляют собой набор автономных процессоров, работающих совместно, но асинхронно. Примером такого мультипроцессора является система одиночного процессора с автономным сопроцессором канала ввода-вывода. Две или более программ, выполняемых на одном процессоре, также требуют синхронизации; однако последующее обсуждение посвящено совместной работе двух или более процессоров. Из-за того что эти процессоры функционируют асинхронно, для правильного выполнения программы требуется синхронизация их работы. Синхронизация может быть достигнута на двух уровнях: 1) на уровне электронных цепей, где процессоры снабжаются отдельными таймерами, и 2) на программном уровне, где синхронизация организуется явным образом самой программой.

Синхронизация на уровне цепей обсуждалась в гл. 7; в настоящем разделе рассматривается только программный уровень синхронизации. Две или более совместно работающих программ должны быть синхронизованы таким образом, чтобы никакая переменная не использовалась процессором, если ее модифицирует другой процессор. Наиболее примитивным средством синхронизации является использование флагов (семафоров), которые предоставляют программам возможности блокировки друг друга [1].

Семафоры используются следующим образом. Рассмотрим два процессора, обращающихся к общей памяти посредством шины, как это показано на рис. 8.15. В памяти имеется данное, к которому время от времени обращаются оба процессора и которое они могут модифицировать. Необходимо обеспечить следующее условие: если процессор А загрузил данное и модифицирует его, процессор В не должен иметь к нему доступ, пока процессор А не запишет модифицированное данное обратно в память. В противном случае процессор В получит неверное значение этого данного.

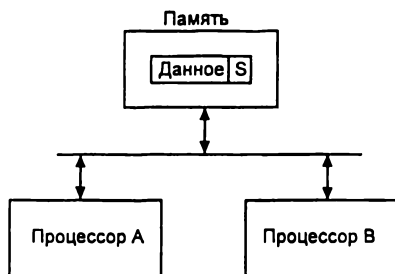


Рис. 8.15. Мультипроцессорная синхронизация.

Примерами приложений, требующих синхронизации, являются предотвращение продажи одного и того же места в самолете нескольким клиентам, одновременно подавшим заявку на место, и недопущение подачи сигнала «занято» двум абонентам, набравшим одновременно один и тот же номер. Обе ситуации маловероятны, но следует обеспечить невозможность их появления.

Как показано на рис. 8.15, каждому данному или каждому блоку данных или каждой защищаемой программе назначается семафор S. Если процессор А может с помощью семафора сообщить процессору В, что данное недоступно, описанные выше ошибочные операции будут предотвращены.

Состояние двоичного однобитового семафора можно описать следующим образом:

$S=1$ Данное используется

$S=0$ Данное не используется⁷⁹⁾

Процессор обращается к семафору и определяет, может ли он использовать данное или оно занято другим процессором. Чтение семафора должно выполняться таким образом, чтобы никакой другой процессор не мог обратиться к семафору, пока длится это действие. Другими словами, следует организовать взаимоисключающий доступ к семафору в реальной памяти. Принципиальный подход к решению этой проблемы заключается в проектировании команды, которая будет выполнять операцию чтение-модификация-запись над ячейкой памяти, содержащей семафор; на это требуется два цикла памяти.

Одним из способов достижения взаимоисключающего доступа к семафору является использование атомарной команды. Атомарной называется команда, которая в процессе ее выполнения не может быть прервана или завершена другим процессором, а семафор, соответственно, не может быть доступен другому процессору, пока выполняется эта команда.

Для достижения атомарной синхронизации в процессорах IBM S360 и Intel x86 (еще до появления Pentium Pro) использовались два совершенно различных способа. Эти способы будут описаны в последующих разделах.

IBM S360

Возможным типом синхронизирующей команды является команда анализа и установки, впервые использованная в IBM S360 и включен-

⁷⁹⁾ Очевидно, что с равным успехом могут использоваться и противоположные значения семафора.

ная в систему команд (ISA) процессоров S370 и S390 [2]. Эта команда является непривileгированной, атомарной и выполняет следующие шаги⁸⁰⁾:

1. В качестве побочного результата команда блокирует шину памяти для других пользователей.
2. Она копирует состояние семафора S в биты CC PSW.
3. Она выполняет запись, устанавливающую S в состояние 1. Если S уже 1, его значение не изменяется. (Шаги 1, 2 и 3 вместе представляют собой команду анализа и установки. После выполнения этой команды блокировка шины памяти снимается.)
4. Процессор командой условного перехода анализирует признак результата, чтобы определить, был ли предоставлен доступ к данному. Если CC=0, доступ предоставлен, осуществляется переход к шагу 5. Если CC=1, доступ не предоставлен; процессор пытается получить доступ повторно, возвращаясь к шагу 1.
5. Процессор использует защищенное данное, что может включать его модификацию, и переходит к шагу 6.
6. Он выполняет атомарную команду запоминания, которая сбрасывает S в 0, тем самым разрешая доступ к данному другим пользователям, и продолжает свою работу.

Типы данных. С процедурой синхронизации связаны два типа данных. Первое является семафором, причем это не одиночный бит, а байт. Такой подход приводит к потере 7 бит, но упрощает реализацию, учитывая, что данная архитектура адресует байты; все биты этого байта равны либо 1, либо 0.

Второй тип данного — это 2-битовое поле CC в PSW, изображенном на рис. 8.1 и 8.2. Эти биты устанавливаются в 00, если значение семафора равно 0, и в 01, если значение семафора равно 1.

Адреса. Эффективный адрес семафорного байта определяется, как базовый адрес плюс смещение. Другой адрес команды анализа и установки, признак результата CC, подразумевается в коде операции. Атомарная команда запоминания использует в качестве источника непосредственный операнд, заполненный нулями.

Операции. Синхронизация обеспечивается двумя атомарными командами, приведенными в табл. 8.18. При реализации команды анализа и установки возникает ряд сложностей. Так, в течение выполнения команды анализа и установки шина и память заблокированы для других пользователей. Поскольку эта команда может занять много тактов, эффективность использования шины падает. Можно представить себе такой крайний случай, когда при первом обращении к семафору он находится на диске в виртуальном адресном пространстве, обраба-

⁸⁰⁾ IBM не использует термин «атомарный». Вместо этого используется фраза «с точки зрения другого CPU представляется ячейкой с взаимоисключающей модификацией». Канал ввода-вывода не рассматривается, как процессор, и взаимоблокировки не предусмотрены.

тывается отказ страницы, и все это время система памяти будет заблокирована для других пользователей.

Таблица 8.18. Команды синхронизации процессора IBM S360

Название	Мнемоника (формат)	Функция
Анализ и установка	TS (SI)	Байт по адресу ($D_1 + (B_1)$) копируется в биты признаков результата и во все биты адресуемого байта записывается 1.
Пересылка	MVI (SI)	Непосредственный байт с одними нулями (в битах 8–15) пересылается по адресу ($D_1 + (B_1)$)

Другие мультипроцессорные команды. В IBM S360 предусмотрена возможность непосредственной связи процессоров без использования системной шины. Каждый процессор имеет восемь линий прямого ввода, восемь линий прямого вывода и восемь сигнальных линий прямой передачи (биты 8–15 команд формата SI). Две команды, приведенные в табл. 8.19, используют эти линии для организации связи двух процессоров системы.

Таблица 8.19. Команды межпроцессорных связей процессора IBM S360

Название	Мнемоника (формат)	Функция
Прямое чтение	RDD (SI)	Биты 8–15 команды помещаются на линии выходного сигнала. Байт с линий прямого ввода записывается в байт памяти по адресу ($D_1 + (B_1)$)
Прямая запись	WRD (SI)	Биты 8–15 команды помещаются на линии выходного сигнала. Байт памяти по адресу ($D_1 + (B_1)$) помещается на линии прямого вывода

В процессорах IBM предусмотрен еще сигнал строба с временными параметрами, характерными для модели, но это не имеет отношения к вопросу об ISA.

Intel Pentium

Процессор Intel Pentium выполняет команду анализа и установки приблизительно так же, как и IBM S360, за тем исключением, что в

дополнение к этой команде предусмотрен префикс LOCK для блокировки шины и памяти явным образом, а не в результате побочного эффекта⁸¹⁾. Блокировка шины осуществляется на время выполнения одной команды, следующей за префиксом LOCK, после чего снимается. Команда синхронизации BTS, осуществляющая анализ и установку бита, передает значение семафора в флаг CF процессора и устанавливает семафор в 1. После этого осуществляется анализ флага CF с целью определения, предоставлен ли доступ к данному.

Типы данных. Имеются два типа данных: 1-битовый двоичный семафор и флаг CF регистра флагов (см. рис. 8.4).

Адреса. Вспомним, что архитектура x86 может адресовать биты. Эта операция осуществляется наложением маски на адресуемый байт. В результате, в отличие от IBM S360, семафором может служить один бит. Флаг CF, сохраняющий значение семафора, не нуждается в явном адресе, поскольку его адрес подразумевается в коде операции.

Операции. Первый байт команды анализа и установки бита содержит префикс LOCK (см. рис. 3.15). Имеется целый ряд вариантов команды анализа и установки бита; однако в табл. 8.20 показаны только команды BTS и BTR.

Таблица 8.20. Команды синхронизации процессора Intel Pentium

Название	Мнемоника	Функция
Блокировка		Преобразует следующую за префиксом команду в атомарную.
Анализ и установка бита	BTS	Запоминает адресуемый бит во флаге CF и устанавливает адресуемый бит в 1.
Анализ и сброс бита	BTR	Запоминает адресуемый бит во флаге CF и сбрасывает адресуемый бит в 0.

Процессор Pentium предоставляет несколько вариантов команд синхронизации, которые позволяют создавать семафоры со значениями, отличными от обсуждаемых. Так, $S = 1$ может означать «свободен», а $S = 0$ — «занят». Кроме того, для установки и отмены синхронизации можно использовать две команды с префиксом LOCK: BTS и BTR.

⁸¹⁾ Префикс LOCK могут использовать 18 команд ISA. При этом Pentium Pro обеспечивает автоматическую блокировку шины, и необходимость в префиксе LOCK отпадает.

8.7. MMX – другие усовершенствования прикладной системы команд (ISA)

В течение последних лет был предложен целый ряд усовершенствований ISA с целью обеспечения поддержки конкретных прикладных задач. Примерами таких усовершенствований являются векторные суперкомпьютеры, микропроцессоры для цифровой обработки сигналов (DSP) и LISP-машины для символьной обработки.

Стремительный рост мультимедийной обработки привел к появлению усовершенствованной ISA для такого рода приложений. Примером является MMX-расширение архитектуры Pentium Pro [9]. Мы обсудим это расширение в том же ключе, что и другие варианты ISA: типы данных, адреса и операции.

Типы данных. Проектировщики корпорации Intel заметили, что основными типами данных для мультимедийных алгоритмов являются 8-битовые пиксели и 16-битовые образы звуков [11]. Кроме этого, иногда используются 32-битовые и 64-битовые целые. Поскольку ширина шины памяти процессора Pentium составляет 64 бит, адресуемый элемент памяти (AU) составляет 64 бит, а остальные типы данных размещаются или упаковываются в этот элемент. Типы данных MMX показаны на рис. 8.16. Они представляют собой упакованный байт, упакованное слово (16-битовое слово), упакованное двойное слово и четверное слово. Некоторые из этих типов данных считаются целыми, другие – ординальными (порядковыми).

Адреса. Проектировщики поставили перед собой задачу добавления в ISA MMX-расширений при сохранении совместимости снизу вверх и минимальном увеличении площади кристалла. Таким обра-

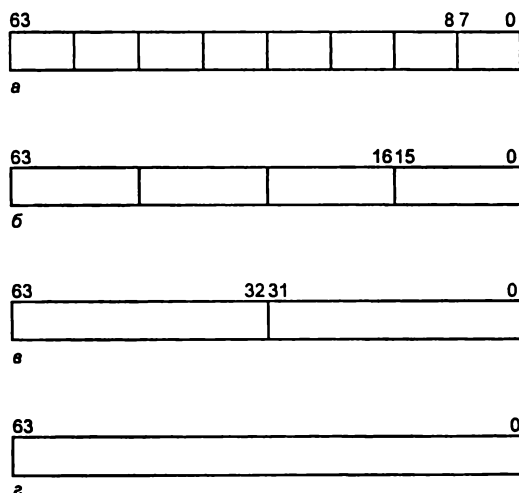


Рис. 8.16. Типы данных MMX (© [11]).

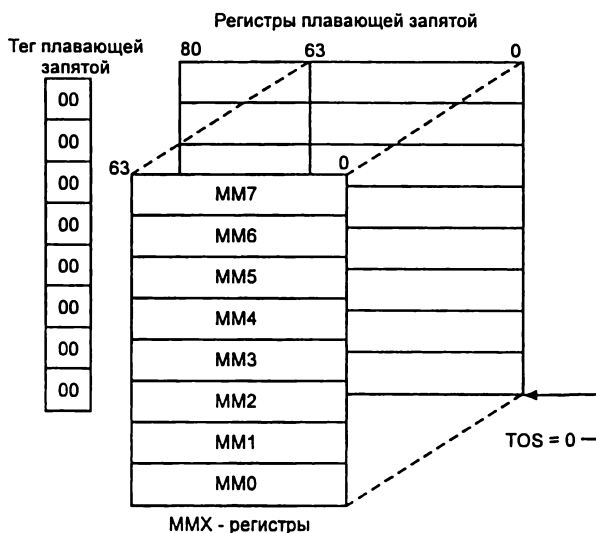


Рис. 8.17. Регистровый файл MMX (© [11]). TOS – вершина стека.

зом, необходимо было сохранить базовую архитектуру памяти. Требования MMX привели к архитектуре регистрового файла с загрузкой-запоминанием. Далее, для работы с 64-битовыми типами данных не годятся целочисленные регистры Pentium Pro. Было принято решение использовать для MMX-данных восемь 80-битовых регистров плавающей запятой. При этом предполагалось, что операции с плавающей запятой и MMX не будут использоваться одновременно.

Для обработки, которая виделась в MMX-приложениях, регистры плавающей запятой имели тот недостаток, что были организованы в виде стека, в то время как MMX-задачи требуют произвольного доступа к операндам. В итоге цепи регистров плавающей запятой были преобразованы так, что давали возможность обращаться к ним в двух режимах (произвольного доступа и стека). Как показано на рис. 8.17, 64-битовые типы данных занимают в регистровом файле IEEE-формата место мантииссы.

Биты порядка и знака (биты 64–79) устанавливаются в 1, так что если по ошибке к регистровому файлу, заполненному MMX-данными, обратится операция плавающей запятой, будет возбуждено прерывание.

В дополнение к адресам регистров необходимо еще адресовать память с целью выполнения операция загрузки и запоминания. Эти адреса памяти не отличаются от остальных адресов памяти ISA Pentium Pro. Так, эффективный адрес отображается на виртуальный адрес с сегментацией и страничной трансляцией. MMX-данные кэшируются в кэше данных.

Операции. MMX-операции, добавленные к ISA Pentium Pro, состоят из базовых арифметических и логических операций применительно к упакованным типам данных. Все управление программой, например, переходы, обслуживаются обычными командами традиционной ISA. Новых битов индикации (флагов) не понадобилось. Операторные команды Pentium Pro требуют два адреса со следующими вариантами:

$R_i \leftarrow R_i \text{ операция } R_j$ (двухадресная регистровая)

$R_i \leftarrow R_i \text{ операция } M(EA)$ (множественный аккумулятор)

MMX-модификация потребовала 64-бит ALU. Это ALU может быть разделено под управлением дешифратора команд на два 32-битовых ALU, четыре 16-битовых ALU или восемь 8-битовых ALU. Поскольку многие мультимедийные данные имеют размер 8 и 16 бит, одна команда в одном цикле может выполнить 8 или 4 целочисленных операций. Такое подразделение ALU было впервые использовано в ILLIAC-IV в 1960-х годах и было названо Флинном [3] единой командой со многими данными (single instruction multiple data, SIMD).

Обратите внимание на то, что для любого из этих типов данных обрабатывается полное содержимое 64-битового AU: 8 байт, 4 слова, 2 двойных слова или 1 четверное слово. Поскольку процессор имеет конвейер, то при выполнении, например, сложения байтов, производительность составляет 8 операций за такт, однако CPI = 1. Единицы CPI практически неприменимы, если измеряется MMX-производительность.

✎ Пример

Если CPI не годится для измерения MMX-производительности, то что годится?

✍ Решение

В главе 2 для оценки производительности суперкомпьютеров при выполнении операций с плавающей запятой использовалась мера MFLOPS. Предложим такую меру производительности: миллионы целочисленных операций в секунду (millions of integer operations per second, MIOPS).

✓ **Замечание**

Для 300-МГц ММХ-процессора, выполняющего 8-битовые целочисленные операции регистр-регистр, максимальная производительность в МІOPS составит:

$$\begin{aligned} \text{MІOPS} &= \frac{\text{такты́ая частота}}{10^6} \times \frac{64}{\text{число битов в операндах}} = \\ &= \frac{300 \times 10^6}{10^6} \times \frac{64}{8} = 300 \times 8 = 2400 \end{aligned}$$

При обработке сигналов и изображений используются дополнительные режимы сложения и вычитания, называемые арифметикой с насыщением. Эта арифметика требует, чтобы в тех случаях, когда результат сложения или вычитания оказывается больше или меньше пределов представимых значений, вызывались функции ограничения, которые устанавливают в качестве результата максимальное или минимальное представимое значение. В результате переполнение или потеря значимости не отражаются в состоянии флагов. Вместо циклического оборачивания действует усечение результата до его представимого значения. Приводимые ниже примеры показывают сложение и вычитание 8-битовых целых без знака 109 и 202.

	Сложение	Вычитание
109	01101101	01101101
202	+11001010	−11001010
	<u>100110111</u>	<u>110100011</u>
Результат с насыщением	11111111	00000000
Результат с оборачиванием	00110111	10100011

В таблице 8.21 приведены 47 архитектурных команд ММХ. Система команд ММХ предусматривает варианты для операций регистр-регистр и операций с множественным аккумулятором. Заметьте, что, за двумя исключениями, мнемонические обозначения команд начинаются с буквы Р (от слова *packed*, упакованный). Некоторые из типов данных знаковые, другие без знака (s, o); некоторые арифметические действия выполняются с оборачиванием, другие с насыщением (w, s).

Таблица 8.21. ММХ-команды

Операция	Варианты	Длина типа данного	Знак/ без знака	Оборачивание/ насыщение	Полное количе- ство
PADD; PSUB		b, w, dw	s,o	w	12
PADDs, PSUBs		b, w	s	s	2
PADDUS, PSUBUS		b, w	o	s	2

Таблица 8.21. Продолжение

Операция	Варианты	Длина типа данного	Знак/ без знака	Оборачивание/ насыщение	Полное количе- ство
PCMPE, PCMPGT	=, >	b, w, dw	o	w	6
PMULH, PMULL	Результат либо старш., либо младш.	w	s	w	2
PMADDWD		w	s	w	1
PUNPCKH, PUNPCKL	Старший или младший	b, w, dw	s	s	6
PACKSS, PACKUS		w	s, o	s	3
PAND, PANDN	AND, NOT	qw	o	w	2
PSLL, PSRL	Логический сдвиг влево, вправо	w, dw, qw	o	w	6
PSRA	Арифмет. сдвиг вправо	w, dw	s	w	2
MOV, MOVQ	Пам. → Рег. d, q Рег. → Рег. Рег. → Пам.			2	
EMMS	Очистка стека	Теги регистров			1
Итого					47

Для того чтобы дать представление о выполнении MMX-операций, опишем три интересные команды. Первая, приведенная на рис. 8.18, является командой упакованного сравнения, в которой анализируется условие «больше чем» для упакованных 16-битовых ординальных операндов. При наличии в регистровом файле двух 64-битовых AU каждое слово в одном регистре сравнивается с соответствующим словом в другом регистре. Слово одного регистра затирается единицами, если слово в первом регистре больше слова во

51	3	5	23
>	>	>	>
73	2	5	6
000 ... 0	111 ... 1	000 ... 0	111 ... 1

Рис. 8.18. Команда упакованного сравнения (© [11])

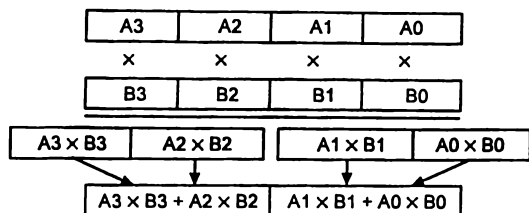


Рис. 8.19. Команда упакованного умножения и сложения (© [11]).

тивном случае слово затирается нулями. Второй операнд для операции с множественным аккумулятором может быть в памяти. Варианты этой команды могут работать с байтами или двойными словами.

Команды упакованного умножения и сложения (PMADDWD) выполняют операции умножения и сложения, весьма важные для приложений обработки сигналов (см. рис. 3.18, изображающий программу матричного умножения). Такая MMX-команда, показанная на рис. 8.19, имеет в качестве операнда-источника четыре упакованных 16-битовых слова со знаком. Команда умножает слова, образуя четыре 32-битовых произведения, которые записываются во временные регистры реализации. Далее эти произведения попарно складываются и упаковываются в два 32-битовых упакованных двойных слова, которые замещают один из операндов-источников. Второй упакованный операнд может быть в памяти, если обработка выполняется с использованием множественного аккумулятора.

✎ Пример

Каково максимальное значение MIOPS для команды PMADDWD? Запозывание этой команды составляет 3 такта, но она конвейеризована, и каждый такт выдается одна новая пара результатов. Будем считать, что частота тактов равна 300 МГц.

✍ Решение

Команда PMADDWD выполняет четыре умножения и два сложения за такт, или 6 операций. Таким образом,

$$\text{MIOPS} = \frac{300 \times 10^6}{10^6} \times 6 = 1800$$

Третья команда, которую мы хотим рассмотреть — распаковка данных, упакованных в младшие биты (PUNPCKLBW). Эта команда распаковывает байты, слова и двойные слова. На рисунке 8.20 показана

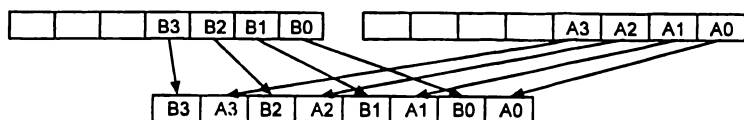


Рис. 8.20. Распаковка байтовых данных (© 11)).

распаковка байтов. Команда берет четыре младших байта из слова, записанного в регистровом файле, и чередует с младшими байтами другого источника (который может быть в регистре или в памяти). Результат записывается в регистр первого операнда-источника.

Команды упаковки и распаковки требуются для подготовки данных в SIMD-стиле MMX-архитектуры. К этим командам относятся `PACKSS`, `PACKUS`, `PUNPCKH` и `PUNPCKL`.

8.7.1. Другие виды специальной поддержки ISA Intel

ISA Pentium Pro предоставляет специальную поддержку для разработки программного обеспечения. Каждая модель семейства Pentium включает характерные для модели регистры (*model-specific registers*, MSR), в которых хранятся типы данных поддержки. Это средство полезно для таких задач, как отладка программ, настройка системы и контроль кэш-промахов.

В процессоре имеются несколько специализированных архитектурных регистров, конкретно, регистры отладки, регистр контроля производительности и регистр отметки времени. В ISA включены специфические команды для задания функций этих регистров, а также для их загрузки и запоминания. Более детальное описание всех этих средств можно найти в справочнике пользователя, входящем в комплект документации к Intel Pentium Pro [8]. Однако здесь мы все же рассмотрим регистр контроля производительности.

В процессоре Intel Pentium Pro имеются два 40-битовых счетчика производительности, которые могут одновременно контролировать различные внутренние для процессора события. Счетчики могут подсчитывать как число событий, так и продолжительность временного интервала, измеряемого числом тактов. Два 32-битовых регистра устанавливают конкретную функцию, которая будет контролироваться счетчиками производительности, и для установки битов этих регистров служат привилегированные команды.

Счетчики производительности могут контролировать события из обширного списка, включающего 68 событий. Некоторые из них перечислены в табл. 8.22; полный список можно найти в технической документации. Из приведенного краткого перечня возможностей

кументации. Из приведенного краткого перечня возможностей Pentium Pro в части контроля производительности видно, что многие параметры, необходимые для построения модели производительности и описанные нами в гл.2, можно получить с помощью аппаратных средств, встроенных в процессор. Эти параметры, в свою очередь, могут быть использованы для настройки программ и выявления возможных усовершенствований аппаратных реализаций, которые можно с успехом применить в будущих моделях процессора.

Таблица 8.22. События контроля производительности Intel Pentium Pro

Контролируемый узел	Описание события
Кэш данных	• Все обращения к памяти • Взвешенное число тактов, приходящихся на необслуженные промахи кэша данных
Узел выборки команд	• Число выборок команд • Число промахов TLB команд • Число промахов выборки команд
Кэш L2	• Число выборок команд из L2 • Число загрузок данных в L2 • Число записей данных из L2
Узел плавающей запятой	• Число незавершенных вычислительных операций с плавающей запятой • Число выполненных вычислительных операций с плавающей запятой

Заметьте, что описываемые возможности контроля производительности схожи с соответствующими средствами IBM S370. В таблице 8.1 можно найти слова 9, 10 и 11, выделенные для записи программных событий.

Литература

1. Dijkstra, E. W. (1968). «Co-Operating Sequential Processes», in *Programming Languages*, F. DeGenuys, ed., Academic, New York.
2. Falkoff, J. W., Iverson, K. E., and Sussenguth, E. H. (1964). «A Formal Description of System/360», *IBM Systems Journal*, 3:3.
3. Flynn, M. J. (1966). «Very High-Speed Computing Systems», *Proceedings of the IEEE*, 54, pp. 1901–1990.
4. IBM (1970). «IBM S/360 Principles of Operation», order GA22-6821-8.
5. IBM (1976). «IBM System/370 Principles of Operation», order GA22-7000-5.
6. IBM (1977). «Enterprise System Architecture/390 Principles of Operation», order SA-7201-04.

8. Intel Corporation (1996). «Pentium Pro Family Developer's Manual», Vol. 1–3, order 000901-001.
9. Intel Corporation (1997). «Intel Architecture MMX™ Technology, Programmer's Reference Manual», order 243007-003.
10. May, C., Silha, E., Simpson, R., and Warren, H., eds. (1994). *The PowerPC Architecture*, Morgan Kaufmann, San Francisco, CA.
11. Peleg, A. and Weiser, U. (1996). «MMX Technology Extension to the Intel Architecture», *IEEE Micro*, 16:4, pp. 42–52.
12. McGhan and O'Connor, (1998). «PicoJava: A Direct Execution Engine for Java Bytecode», *Computer*, October.

Упражнения

- 8.1. Опишите канальную систему команд процессора IBM S360/370 с помощью типов данных, адресов и операций.
- 8.2. Какую команду IBM S360 можно использовать для определения по признаку результата возможности доступа к данному, защищенному семафором?
- 8.3. Изучите техническую документацию процессора Intel Pentium Pro по вопросу контроля производительности и предложите способ определения интенсивности промахов кэша команд и кэша данных.
- 8.4. Статья Мак-Гана и О'Коннора [12] описывает процессор для выполнения Java-программ. Напишите реферат объемом не более 500 слов, посвященный наиболее ярким чертам архитектуры и реализации этого процессора.
- 8.5. Опишите значение номеров 1, 2 и 3 в кружках на рис. 8.6.
- 8.6. В разделе 8.4 описаны четыре сегментно-страничные конфигурации, используемые в процессоре IBM S370. Для каждой из этих конфигураций укажите число битов, складываемых в двух сумматорах, и число битов, сцепляемых в регистре реального адреса (см. рис. 8.6). Вспомните, что S370 использует 24-битовый эффективный, или логический, адрес.
- 8.7. Какова длина виртуального адреса Pentium Pro?
- 8.8. На рисунке 8.8 изображена система виртуальной адресации с сегментами процессора Pentium Pro для 4-Кбайтовых страниц. Система может также поддерживать 1-Мбайтовые страницы. Покажите, изменятся ли для 1М-байтовых страниц линейные адреса и таблицы страниц.
- 8.9. Объясните, почему для трансляции виртуальных адресов в процессоре S370 используются два сумматора (рис. 8.6), а в процессоре Pentium Pro — лишь один сумматор (рис. 8.8)?
- 8.10. Вспомните команды синхронизации, описанные в разделе 8.6. Что происходит, когда два или более запросов на доступ к семафору поступают в одном такте?
- 8.11. Выведите формулу для определения скорости процессора в единицах MIOPS для примера в разделе 8.7.
- 8.12. Какие действия выполняются над двумя 32-битовыми результатами, получаемыми в команде умножения и сложения на рис. 8.19? Другими словами, как эти результаты можно использовать в качестве входных данных для последующих команд? Заметьте, что операнды-источники имеют длину 16 бит.

Предметный указатель

AC 24
ALU 17,20
AU 87,179

BTB 295
BTC 295

CAS 228
CAW 387
CCW 387
CISC 77, 138-142
CPI 46
CPU 15
CR 370
CSW 388

DMA 18
DRAM 20

EA 89
EBP 99
EIP 125
ELB 167
ELD 169
ESP 99

IBR 22
IPC 46
IR 22, 23
ISA 19, 76, 78

LSB 88
LSD 169

MAR 22, 23
MDR 22, 23
MFLOPS 72
MIPS 70
MQ 22
MSB 88

NAND 92

Pb 282
PC 22
Pop 82, 85
PSW 125
Push 82, 85

RAM 20
RAS 227
RISC 20, 77, 142, 143

S 88

TLB 196
TOS 84

WABD 282
WAM 51, 58
WGM 58
WHM 53, 58

XOR 92

Автомат конечный 16
Адрес виртуальный 182
- логический 89
- памяти 137
- регистра 137
- трансляция динамическая 377
- эффективный (EA) 89,90
Аккумулятор 82
- множественный 85
Архитектура набора команд
(ISA) 76, 78
- фон-неймановская 19-29

Биты, бюджет 85-88
- залипания 171
- избыточные 167
- индикации 127
- сторожевые 167-169
Буфер ассоциативной трансляции
(TLB) 196, 197
- мишени (BTB) 295, 296
Быстродействие 46

Ввод-вывод, поддержка 387-390
- программный 322-325
- с отображением на память
326-328
- сопроцессорный 329
Взаимозависимость «запись после
записи» (WAW) 306, 307

- «запись после чтения» (WAR) 305, 306
- «чтение после записи» (RAW) 302-305

Время отклика 72, 180

Выбор поздний 210

- ранний 185

Выталкивание (pop) 82

Вычитание 154-158, 163, 164

Данные, продвижение 304

- типы 107
- - кодирование 173, 174
- тракт 249

Деление 159-162, 164

Диаграммы состояния 251

Загрузка-запоминание 82

Задержки в конвейере 281, 282

- переходов 282

Закон Амдала 60-62

- Гроша 69-72

- Литгла 358

- Мура 63-66

Замораживание 283

Исключения 101

Кадр страничный 182

Каскад 273

Код самомодифицирующийся 21

- с избытком 112

Кодирование типов данных 173, 174

Коды операций 132-137

Команда атомарная 397

Команды арифметические 92, 93

- интерпретация 25
- логические 92, 93
- пересылки 93, 94
- привилегированные 366
- системы 19, 78
- статистика 145
- управления 94
- частота 145

Конвейер, замораживание 283

- разбиение 276

- суперскалярный 311

Контроллеры 336

Конфликты структурные 299

Кривая обучаемости 66-69

Кэш мишени (BTC) 295

Кэш-память 197, 232

- ассоциативная 204, 205

- данных 211-213

- дисковая 241, 242

- многоуровневая 215-219

- общая 213, 214

- поддержка 391

- специальная 219

- с прямым отображением 206-208

- частично-ассоциативная 205, 206

Кэш-попадание 205

Кэш-промах 205

Ловушки 96

Локализация временная 184

- пространственная 184

Макро-ISA 260

Мантисса 115, 121

Метастабильность 350-352

Микро-ISA 260

Микрокоманда переходов 264

- формат 264

Микропрограмма 265

- размер 268

Мишень 95

Модель фон-неймановская 14

МОП 63

Обучаемости кривая 66-69

Округление 169-173

Операнд 81

Операции 91-107

- коды 132-137

Основание 115

Отношение процентное 47

Очереды теории 353

Память виртуальная, производительность 194

- - страничная 181

- дисковая 238-242

- иерархическая 181

- реальная (основная, оперативная) 181

- режимы адресации 89

- сегментированная виртуальная 234-238

- с перекрытием 222

- страничная виртуальная 181

Передача асинхронная 343

- последовательная 344

- синхронная 342

Переходы, задержка 282

- история 291

- предсказание выполнения 285

- - невыполнения 286

- - обобщенное 288
- таблица истинности 256
- ПЗУ 14
- Порядок 115, 122
- смещенный 116, 165
- Прерывания 101, 308
- ввода-вывода 328
- маскируемые 106
- немаскируемые 106
- поддержка 385
- Приоритет 300
- Программа, слово состояния (PSW) 125
- Пропускная способность 180
- Просмотр назад 184
- предварительный 184
- Проталкивание (push) 82
- Протоколы последовательной передачи 346
- Процесс, состояние 96, 179
- Процессор базовый с конвейеризацией 272
- состояние 17, 96, 98
- Распределение битовое 85-88
- Регистр базовый 82
- Регистры архитектурные 22
- реализации 22
- Результат побочный 79
- Ресурсы, дублирование 301
- Сегмент 234
- Сегментация виртуальная 238
- страничная 235
- Сектор диска 239
- Семафоры 396
- Синхронизация мультипроцессорная 396
- тактов 345, 350
- Система закрытая с очередью 354
- открытая с очередью 361
- Система команд 19, 78
- Слово состояния программы (PSW) 125
- Сложение 154-158, 163, 164
- Смещение 82, 89
- Состояние процесса 96, 179
- процессора 17, 96, 98
- Средние по времени 49, 51
- по отношению 54-58
- по скорости выполнения 51-53
- Стек, вершина 84
- выборка (pop) 84
- запись (push) 85
- Страница 182
- интенсивность отказа 183
- Суперскаляр неоднородный 311
- степень 311
- Счетчик программный 80, 125-127
- Считывание-запись 82
- Темп роста сложный 59
- Тракт данных 249
- Точность 166
- Трансляция адресов динамическая 377
- ассоциативная 196
- инверсная 190
- прямая 186-190
- Умножение 158, 159
- Управление аппаратное 250-259
- микропрограммное 260-269
- распределенное 340
- централизованное 338
- Флаги 129
- Функции архитектурные (аппаратные) 76
- программные 76
- Функция ограничения 92
- Характеристика переходная 72, 73
- Хеширование 191
- ЦМД 63
- Шина общего назначения 336
- специализированная 336

Учебное издание

Харви Крейгон

АРХИТЕКТУРА КОМПЬЮТЕРОВ И ЕЕ РЕАЛИЗАЦИЯ

Зав. редакцией *Т. Г. Хохлова*

Ведущий редактор *Н. М. Савина*

Художественный редактор *В. П. Григорьев*

Технический редактор *Е. В. Денюкова*

Корректор *А. В. Игумнов*

Оригинал-макет подготовлен *Ю. В. Сныткиным*

Подписано к печати 20.02.04. Формат 70×100/16.

Бумага офсетная. Печать офсетная. Объем 13,0 бум. л.

Усл. печ. л. 33,8. Уч.-изд. л. 25,68. Изд. № 6/9745.

Тираж 2000 экз. Зак. № 0413390.

Издательство «МИР»

Министерства культуры и средств массовых коммуникаций РФ.

107996, ГСП-6, Москва, 1-й Рижский пер., 2.

Диапозитивы изготовлены в издательстве «Мир»

Отпечатано в полном соответствии

с качеством предоставленных диапозитивов

в ОАО «Ярославский полиграфкомбинат»

150049, Ярославль, ул. Свободы, 97

