П.Брамм, Д.Брамм

Микропроцессор 80386 и его программирование

Издательство «Мир»

Микропроцессор 80386 и его программирование

Микропроцессор 80386 и его программирование

80386
A Programming and Design Handbook

Penn Brumm

Consultant/Development
Project Leader
Control Data Corporation

Don Brumm
Staff Systems Programmer
Operating Systems Development
Amdahl Corporation

TAB Professional and Reference Books
Division of TAB BOOKS Inc.

П.Брамм, Д.Брамм

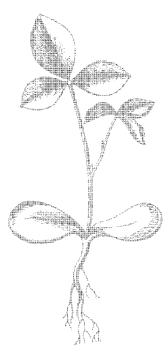
Микропроцессор 80386 и его программирование

Перевод с английского

канд. техн. наук А. И. Шереметьева
и канд. пед. наук К. Г. Финогенова

под редакцией
д-ра физ.-мат. наук Н. Г. Волкова

Москва «Мир» 1990



Scan AAW

~~ББК 32.973-01~~
Б87
~~УДК 681.3~~

Брамм П., Брамм Д.

Б87 Микропроцессор 80386 и его программирование: Пер. с англ.– М.: Мир, 1990,– 448 с., ил.
ISBN 5-03-001441-1

В книге американских специалистов подробно описаны аппаратные и программные средства 32-разрядного микропроцессора (МП) 80386 фирмы Intel. Рассмотрены архитектура, способы адресации, работа памяти, локальная шина и ввод-вывод. Большое внимание уделено описанию организации многозадачного и многопроцессорного режимов. Приведены подробные сведения по основным компонентам микропроцессорного комплекта. Даны указания по программированию и примеры специальных программ для МП 80386. Приведена система команд МП 80386.

Для разработчиков и изготовителей микропроцессорной техники и профессиональных программистов.

Б $\frac{2404020000\text{-}165}{041(01)\text{-}90}$ 145-90 **ББК 32.973-01**

Редакция литературы по информатике и робототехнике

ISBN 5-03-001441-1 (русск.)
ISBN 0-8306-2937-8 (англ.)

© TAB Books, 1987
© перевод на русский язык, Финогенов К. Г., Шереметьев А. И., 1990

Предисловие редактора

Вряд ли сегодня следует много говорить о том радикальном изменении наших представлений о возможностях вычислительной техники, которое произошло после появления микропроцессоров и созданных на их базе персональных компьютеров. Персональные ЭВМ, созданные на базе 16-разрядных микропроцессоров, стали распространенным инструментом для инженеров, ученых, экономистов, широко используются для обучения. Несмотря на большую роль, которую сыграли эти ЭВМ в первые годы своего развития, в настоящее время ощущается потребность в более совершенных машинах, которые совмещали бы достоинства персональных компьютеров с большими возможностями вычислений и уровнем сервиса.

Реакцией на эту потребность явилась разработка 32-разрядных микропроцессоров и создание на их основе рабочих станций, обладающих большим быстродействием, имеющих многозадачный режим, лучший сервис, в частности графические средства, а также способных включаться в вычислительные сети. Ожидается, что объем мирового производства рабочих станций в ближайшие пять лет вырастет не менее чем в 5 раз.

Хотя в нашей стране не налажен выпуск 32-разрядных микропроцессоров и микроЭВМ на их основе, тем не менее во многие организации поступает довольно много таких ЭВМ по импорту. В связи с этим возникает настоятельная необходимость в литературе с описанием характеристик 32-разрядных микропроцессоров с тем, чтобы дать возможность советским специалистам проектировать программное обеспечение для таких ЭВМ. Есть надежды и на выпуск отечественных 32-разрядных микропроцессоров.

Предлагаемая книга посвящена одному из массовых 32-разрядных микропроцессоров, разработанному фирмой Intel. Книга охватывает все аппаратные и программные характеристики микропроцессора 80386. Описывается его архитектура, особенности организации взаимодействия с внешними устройствами, система команд, методы программирования. Особое внимание уделяется проблемам совместимости с более ранними моделями. Заключительная часть книги посвящена общим вопросам вычислительной техники. Она носит вспомогательный характер.

По своему характеру книга является справочником и предназначена для системных программистов, проектировщиков вычислительных систем, а также студентов и аспирантов соответствующих специальностей, всех лиц, интересующихся микропроцессорами. Книга написана на высоком профессиональном уровне, содержит достаточные для практических целей сведения.

Предисловие и гл. 1–7 перевел А. И. Шереметьев, остальное – К. Г. Финогенов.

Н. Г. Волков

Предисловие

Наиболее существенным достоинством 32-разрядного процессора 80386 фирмы Intel (США) является то, что он привносит микропроцессоры в мир виртуальных вычислительных систем, открывая дверь перед коммерческими и научными приложениями, ранее невозможными для настольных ЭВМ. Второе не менее важное достоинство заключается в том, что он снимает барьеры, которые разработчики системного и прикладного программного обеспечения обнаруживали в микропроцессорах (МП) прежних поколений. Дополнительные усовершенствования включают расширение формата команды до 32 бит, программную модель для большого пространства линейных адресов и функции разбиения памяти на страницы. Наиболее важные для приложений эти преимущества реализованы «способом супермножества», что обеспечивает полную совместимость с программными продуктами, разработанными для МП 8086/8088 и 80286 той же фирмы.

Вообще говоря, эта книга рассчитана на четыре категории читателей. К первой категории относятся читатели, желающие понять историю и архитектуру обработки данных или физическую сущность таких изделий, как МП 80386. Ко второй категории относятся программисты, которым необходимо освоить новую систему фирмы-производителя комплексного оборудования и программировать ее работу в мире пользователя. Во вторую категорию входят также те программисты, которые работают в фирмах-производителях программной продукции и должны быть в курсе новейших достижений в области аппаратных средств. К третьей категории читателей относятся конструкторы фирм, производящих электронное оборудование, которые хотят построить на основе этого МП систему. Такой конструктор прежде всего нуждается в хорошем обзоре архитектуры нового МП. Наконец, существуют руководители, которые должны быть в курсе дел своих подчиненных, относящихся к указанным выше трем категориям читателей.

Трудно структурировать материал, когда пишешь о таком взаимосвязанном и взаимообусловленном предмете, как МП 80386: что должно идти первым, вторым и т. д.? В конце концов мы решили начать с основных стандартных блоков, таких как регистры и типы данных. При планировании этой книги был принят во внимание еще один фактор. Часто люди являются специалистами в одной области знаний и новичками в другой. Там, где они являются новичками, кратко изложенный вводный материал будет способствовать быстрому вхождению в курс дела и устранит недопонимания.

Книга разбита на части так, чтобы помочь каждому из ее потенциальных читателей сразу же определить, с какого места лучше начать читать. Гл. с 1 по 5, составляющие I часть книги, дают обзор архитектуры МП 80386 для того, чтобы провести читателя от его внутренней структуры к связанным с ним подсистемам, которые должны работать совместно с МП. Кроме того, в этой части рассматриваются некоторые основные понятия.

Гл. с 6 по 10, составляющие II часть этой книги, предназначены для системного программиста и/или руководителя программного проекта и посвящены системе команд МП 80386. Описана каждая команда из полного набора, и действие большинства из них проиллюстрировано так, чтобы читателю были понятны все детали взаимодействия команды с различными регистрами и системными флагами. Примеры написаны как упражнения по установке системных флагов и изменению содержимого регистров. Приведенные программы не являются наиболее эффективными или совершенными способами реализации какой-либо функции и не предлагаются в качестве руководства по программированию.

Гл. с 11 по 14 входят в III часть книги и содержат общий обзор системы. Это именно то место, куда следует обратиться, чтобы почерпнуть новый материал по таким темам, как

конвейеризация, концепции интерфейса и организация памяти. В приложениях приведена дополнительная информация, которая ближе знакомит читателя с этим многогранным и оригинальным МП. В приложении 1 дается общая информация о фирме Intel и ее стратегиях проектирования, приведших к разработке МП 80386. Приложение 2 содержит справочную информацию о различных регистрах, битах и флагах, используемых в МП 80386. Приложение 3 поможет читателю, когда он столкнется с некоторыми, прежде незнакомыми жаргонными сокращениями. В приложении 4 описано взаимодействие МП 80386 с магистралью MULTIBUS I, II. В приложении 5 приведен текст программы на языке ассемблера, которая демонстрирует несколько команд, допустимых только в защищенном режиме (подробнее этот режим описан в I части книги). Программа была написана с использованием платы-адаптера для МП 80386, вставленной в гнездо процессора системы на основе МП 80286. В качестве ассемблера был использован макроассемблер фирмы Microsoft Corp. (США), который воспринимает коды операций для моделей МП от 8086 до 80286; специфические для МП 80386 команды были введены в виде шестнадцатеричных кодов.

Микропроцессор 80386 обеспечивает беспрецедентную производительность на микрокомпьютерном уровне с сохранением при этом совместимости с предыдущими моделями по принципу «снизу-вверх». Благодаря этому пользователи могут с большей эффективностью применять существующее программное обеспечение без дополнительных затрат по его адаптации к новой системе, а будущие программные разработки раскроют всю потенциальную мощь МП 80386.

Было бы трудно написать подобную книгу, содержащую подробные сведения о новом микропроцессоре, без поддержки заинтересованной фирмы Intel, которая предоставила нам необходимую документацию и плату-адаптер с МП 80386. Особую благодарность хотим выразить сотруднику этой фирмы Роду Скиннеру за его щедро потраченное время и техническую поддержку. Помощь сотрудницы фирмы Дженни Маркез была также исключительно полезной. Квалифицированных технических рецензентов трудно найти, особенно таких, кто добровольно потратил бы свое плотно расписанное по часам время. Мы благодарим Дейва Бердсола и Гэри Льюиса за их усилия по составлению рецензий и технических комментариев. Нам помогали также Ханк Та и Тим Луонг – инженеры фирмы Cassidy Computers, которые собрали и испытали нашу систему на основе МП 80286, установили и протестировали плату-адаптер с МП 80386. Наконец, выражаем нашу искреннюю благодарность Бесс Клейтон за ее продолжительную поддержку и помощь, без которой многое не могло бы произойти.

Глава 1

Введение в МП 80386 фирмы Intel

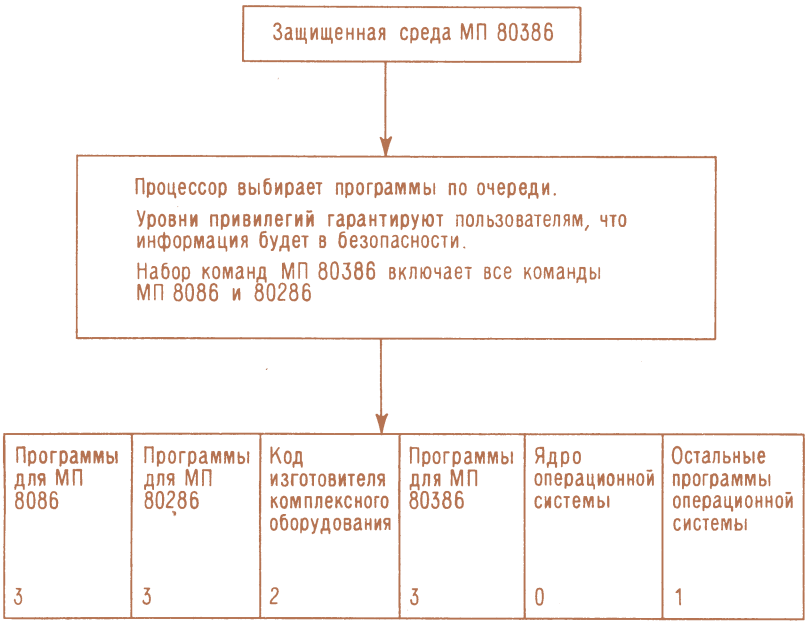
МП 80386 вышел на рынок с уникальным преимуществом. Он является единственным 32-разрядным МП, для которого пригодно существующее прикладное программное обеспечение стоимостью 6,5 млрд. долл., написанное для МП предыдущих моделей от 8086/88 до 80286 (клон IBM PC).

Говорят, что системы совместимы, если программы, написанные на одной системе, успешно выполняются на другой. Если совместимость распространяется только в одном направлении, от старой системы к новой, то говорят о совместимости снизу вверх. Совместимость снизу вверх на объектном уровне поддерживает капиталовложения конечного пользователя в программное обеспечение, поскольку новая система просто заменяет более медленную старую. Микропроцессор 80386 совместим снизу вверх с предыдущими поколениями МП фирмы Intel. Это означает, что программы, написанные специально для МП 80386 и использующие его специфические особенности, обычно не работают на более старых моделях. Однако, так как набор команд МП 80386 и его модули обработки являются расширениями набора команд предшествующих моделей, программное обеспечение последних совместимо снизу вверх с МП 80386.

Специфическими особенностями МП 80386 являются многозадачность, встроенное управление памятью, виртуальная память с разделением на страницы, защита программ и большое адресное пространство. Аппаратная совместимость с предыдущими моделями сохранена посредством динамического изменения разрядности магистрали.

МП 80386 выполнен на основе технологии CHMOS III фирмы Intel, которая вобрала в себя быстродействие технологии HMOS (МДП высокой плотности) и малое потребление мощности технологии CMOS (КМДП). МП 80386 предусматривает переключение программ, выполняемых под управлением различных операционных систем, таких как MS-DOS и UNIX. Это свойство позволяет разработчикам программ включать стандартное прикладное программное обеспечение для 16-разрядных МП непосредственно в 32-разрядную систему. Процессор определяет адресное пространство как один или несколько сегментов памяти любого размера в диапазоне от 1 байт до 4 Гбайт ($4 \cdot 2^{30}$ байт). Эти сегменты могут быть индивидуально защищены уровнями привилегий и таким образом избирательно разделяться различными задачами. Механизм защиты основан на понятии иерархии привилегий или ранжированного ряда. Это означает, что разным задачам или программам могут быть присвоены определенные уровни, которые используются исключительно для данной задачи (рис. 1.1).

Заметим, что в иллюстрациях некоторые биты регистров являются неопределенными или отмечены как зарезервированные фирмой Intel для использования в будущем. Не вводите зависимость от состояния любого из этих битов при тестировании значений определенных битов регистра. Маскируйте ненужные биты.



Сегменты памяти с различными уровнями привилегий

Не вводите также зависимость от способности сохранять информацию, записанную в любом из неопределенных или зарезервированных битов. При загрузке регистров всегда сбрасывайте эти биты в нуль или оставляйте их значения без изменения.

1.1. Основные определения

Прежде чем двигаться дальше, дадим определения нескольким основным терминам. Эти термины часто используются как в этой книге, так и в статьях по МП 80386. Многие из них имеют значения, специфические для фирмы Intel и/или для систем на базе МП 80386.

Сегмент. Начиная с МП 8086, фирма Intel ввела понятие сегмента, который определяется как блок адресного пространства. В МП 8086 сегмент имеет максимальный размер 64 Кбайт или 65536 байт. К МП 80386 это ограничение больше неприменимо. Программисты теперь могут рассматривать сегменты как одномерные подпространства длиной до 4 Гбайт.

Вентиль. Логический элемент, пропускающий только определенные процессы. МП 80386 с помощью дескрипторов вентилей обеспечивает защиту потоков управления между исполнительными сегментами с различными уровнями привилегий. Существуют четыре типа вентилей – «вызов», «ловушка», «прерывание» и «задача», описанные в гл. 6.

Дескриптор. Структура данных, используемая для определения характеристик программного элемента. Например, дескриптор описывает запись данных, сегмент или задачу.

Таблица. Собрание данных, расположенных по строкам и столбцам для ссылок на них или хранимых как массив. Доступ к элементам таблицы возможен путем

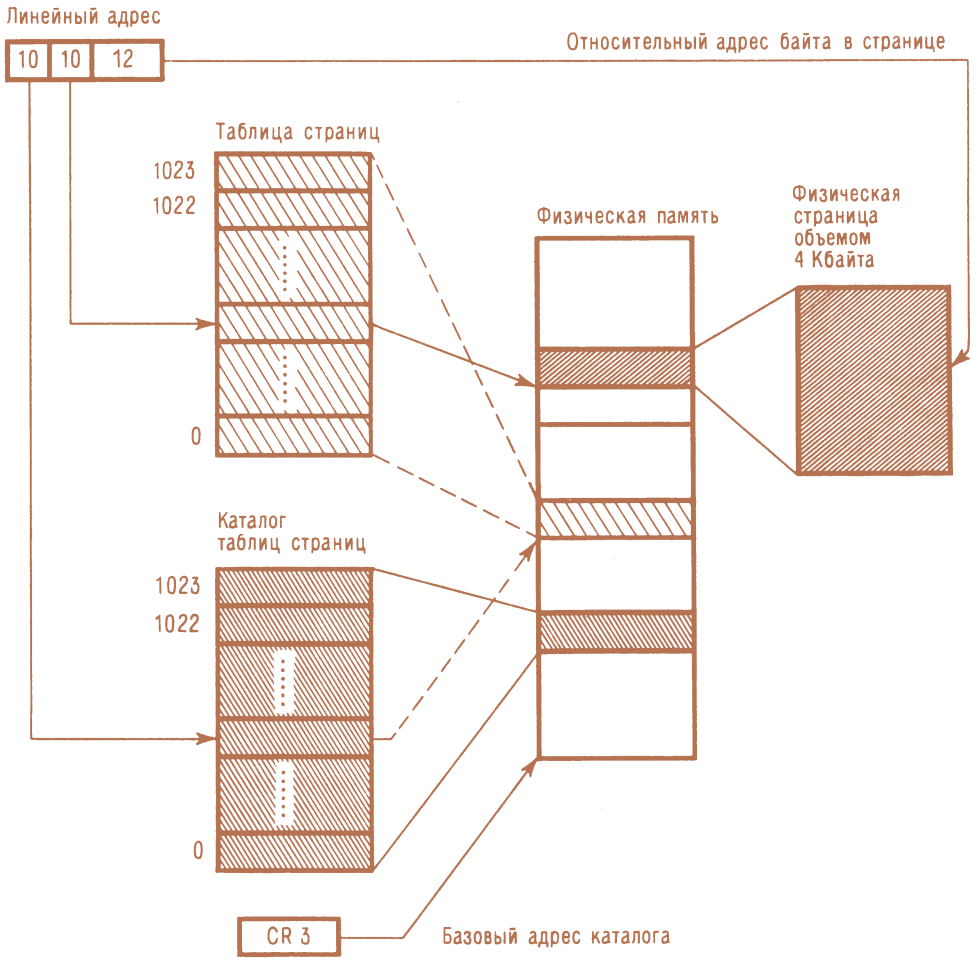


Рис. 1.2. Отображение линейного адреса в физический.

непосредственного вычисления их адреса, если известны селектор и базовый адрес.

Пространство линейных адресов. Адрес указывает размещение регистра, конкретное место в среде хранения или некоторые другие источники или приемники данных. В МП 80386 пространство линейных адресов простирается от 0 до 4 Гбайт. Линейный адрес указывает конкретный байт в этом пространстве (рис. 1.2).

Логический адрес. Отметим, что концептуальной параллели между пространством линейных адресов и пространством, используемым для логической адресации, не существует. Логический адрес состоит из селектора и относительного адреса. Селектор указывает на некоторый дескриптор сегмента, частью которого является линейный базовый адрес этого сегмента. Относительный адрес говорит о том, как далеко в сегменте расположен требуемый байт. Различные термины, использованные в этом предварительном описании, подробно рассматриваются на протяжении всей I части книги. Логический адрес упомянут здесь для того, чтобы ввести его понятие и указать на отличие от других типов адресов.

Физический адрес. Адрес, который действительно выбирает тот элемент памяти, где расположен требуемый байт. В МП 80386 линейный и физический адреса различаются, только когда выполняется разбиение памяти на страницы.

Задача. Основная, уникальная функция программы или системы. Это может быть один эпизод выполнения программы. Задачи также называют процессами.

Сегмент состояния задачи (TSS). Структура данных, описанная и указываемая дескриптором, в которой хранится состояние (прерванной) задачи. Системное программное обеспечение создает TSS и помещает в них начальную информацию, такую как правильные указатели стеков для программ обслуживания прерываний.

Микрокод. Список малых программных шагов, а также множества функций управления, отрабатываемых исполнительной логикой компьютерной системы, декодирующей команды. Этот код лежит ниже уровня языка ассемблера.

Разбиение на страницы. Процедура, которая передает последовательность байтов, называемую страницей, между основной памятью и массовой памятью. Разбиение памяти на страницы упрощает алгоритмы операционной системы, предназначенные для обмена информацией между двумя типами памяти, так как оно обеспечивает единый механизм для управления физической структурой памяти.

Флаг. Индикатор, состояние установки/сброса которого используется для информирования более поздней секции программы о выполнении или невыполнении некоторого условия.

1.2. Обзор 32-разрядного микропроцессора

Интеграция функций связывает элементы системы в одну структуру, которая не может быть разделена без потери установленных для нее функций. Микропроцессор 80386 имеет интегрированное управление памятью и архитектуру защиты, которые включают регистры трансляции адреса, аппаратные средства многозадачности и механизм защиты для поддержки ряда операционных систем. Управление памятью и механизм защиты транслируют логические адреса в физические и инициируют защиту, необходимую для поддержки целостности задач в многозадачной среде.

1.2.1. Основные блоки

В своей основе МП 80386 состоит из шести блоков, работающих параллельно: блок интерфейса с магистралью (BIU), блок предварительной выборки команд, блок декодирования команд, исполнительный блок (EU), блок управления сегментами и блок разбиения на страницы (рис. 1.3).

Интерфейс представляет собой физическое или логическое средство, которое соединяет смежные компоненты, схемы, оборудование или системные элементы. Блок BIU обеспечивает интерфейс между МП 80386 и его окружением. Он принимает внутренние запросы для выборки команд от блока предварительной выборки команд и для обмена данными от блока EU и устанавливает приоритет этих запросов. Одновременно он генерирует или обрабатывает сигналы для исполнения текущего цикла магистрали. К ним относятся сигналы адреса, данных и управления для обращения к внешней памяти и устройствам ввода-вывода. Блок BIU управляет также интерфейсом с внешними задатчиками магистрали и сопроцессорами.

Для того чтобы заранее получать команды или данные перед их фактическим использованием, существует функция опережающего просмотра программы, кото-

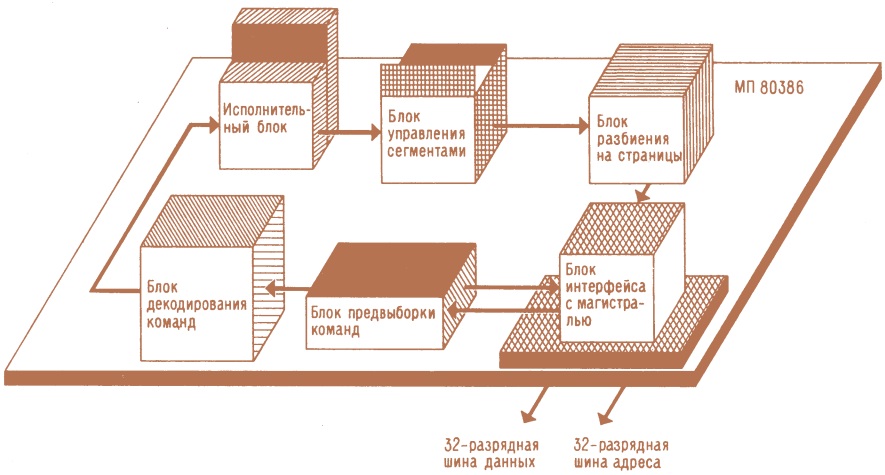


Рис. 1.3. Шесть основных блоков МП 80386.

рую в МП 80386 выполняет блок предвыборки команд. Когда блок BIU не занимает цикла магистрали для исполнения команды, блок предвыборки команд использует его для последовательной выборки из памяти байтов команд. Эти команды хранятся в 16-байтовой очереди команд в ожидании обработки блоком декодирования команд.

Блок декодирования команд преобразует байты команды из этой очереди в микрокод. Декодированные команды в ожидании обработки блоком EU хранятся в очереди команд глубиной в три команды, работающей по принципу FIFO («первым вошел – первым вышел»). Непосредственные данные и относительные адреса в коде операции также берутся из очереди команд.

Блок EU выполняет команды из очереди команд и взаимодействует со всеми другими блоками, требуемыми для завершения выполнения команды. Для ускорения выполнения команд с обращением к памяти блок EU приступает к их выполнению до завершения выполнения предыдущей команды. Так как команды с обращением к памяти встречаются часто, то благодаря такому перекрытию по времени производительность повышается примерно на девять процентов.

Регистры общего назначения (РОН) встроенного типа используются для таких операций, как двоичное сложение или вычисление и модификация адресов. Они все чаще используются вместо специализированных регистров-аккумуляторов. Блок EU содержит восемь 32-разрядных РОНов, применяемых как для вычисления адресов, так и для операций с данными. Этот блок содержит также 64-разрядный регистр, применяемый для ускорения операций сдвига, циклического сдвига, умножения и деления.

Блок управления сегментами преобразует логические адреса в линейные по запросу блока EU. Для ускорения этого преобразования текущие дескрипторы сегментов помещаются во встроенную кэш-память. Во время трансляции адресов блок управления сегментами проверяет, нет ли нарушения сегментации. Эти проверки выполняются отдельно от проверок нарушений статической сегментации, осуществляемых блоком проверки защиты. Оттранслированный линейный адрес направляется в блок разбиения на страницы.

Если механизм разбиения на страницы включен, соответствующий блок транслирует линейные адреса в физические. Если же этот механизм выключен, то это

означает, что физический адрес совпадает с линейным и трансляция не нужна. Для ускорения трансляции адресов в кэш-память дескрипторов страниц помещаются каталог недавно использованных страниц, а также информация о входах в таблицу страниц в буфере трансляции адресов. Затем блок разбиения на страницы пересылает физические адреса в блок BIU для выполнения цикла обращения к памяти или устройствам ввода-вывода.

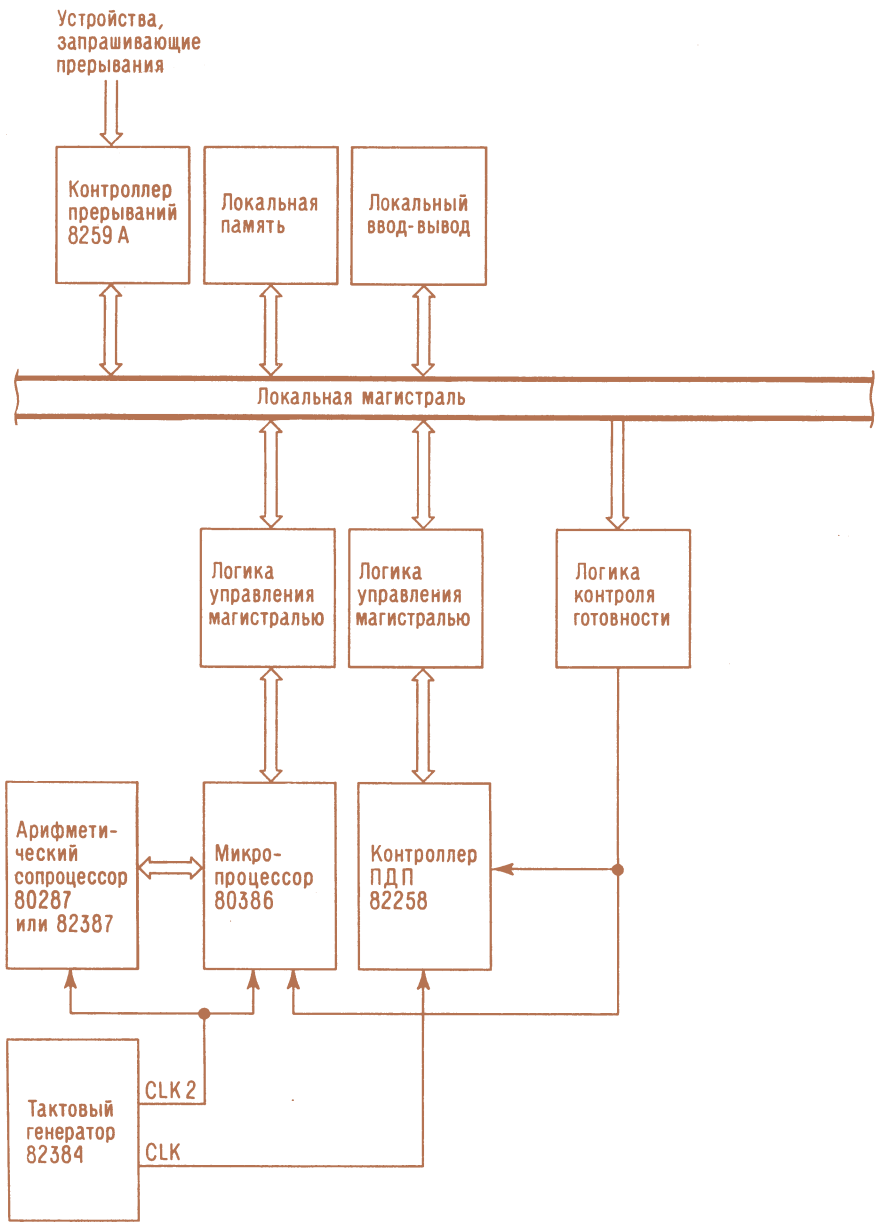


Рис. 1.4. Структурная схема системы на основе МП 80386.

Микропроцессор 80386 (рис. 1.4) использует 32-разрядные регистры и шины данных для поддержки адресов и типов данных такой же разрядности. Благодаря этому доступ к 32-разрядной памяти может быть завершен всего лишь за два такта генератора, т.е. возможен обмен информацией по магистрали со скоростью до 32 Мбайт/с при частоте тактового генератора 16 МГц. Процессор адресует до 4 Гбайт физической памяти и до 64 Тбайт виртуальной памяти[1)].

1.2.2. Устройство управления памятью

Предназначенное для манипуляции сегментами устройство управления памятью (MMU)[2)] состоит из блока сегментации и блока разбиения на страницы. Блок сегментации обеспечивает четыре (от 0 до 3) уровня защиты с целью изоляции и защиты друг от друга прикладных программ и операционной системы. Этот компонент также позволяет легко создавать перемещаемые программы и данные и обеспечивает их совместное использование.

Блок разбиения на страницы, входящий в состав блока MMU, работает на более низком и прозрачном для процесса сегментации уровне. Эта прозрачность позволяет управлять пространством физических адресов отдельно и независимо от управления сегментами. Каждый сегмент отображается в пространство линейных адресов, которое в свою очередь отображается в одну или несколько страниц объемом 4 Кбайт. Для реализации эффективной системы виртуальной памяти МП 80386 полностью поддерживает способность рестарта (повторного запуска) в случае отказа во всех страницах и сегментах.

1.2.3. Архитектура режима реальных адресов и защищенного режима

Микропроцессор 80386 имеет два режима работы: режим реальных адресов, называемый реальным режимом, и защищенный режим. *Реальный режим* используется преимущественно с целью установки процессора для работы в защищенном режиме, а также с целью выполнения программ МП предыдущих поколений. *Защищенный режим* обеспечивает возможности разбиения на страницы, управления памятью и механизма привилегий МП 80386. Эти конструктивные свойства дают совместимость на уровне объектного кода с более ранними МП фирмы Intel. Реальный режим, защищенный режим, среда виртуального МП 8086 кратко рассматриваются в этом разделе, а подробнее в гл. 8.

Реальный режим. При подаче сигнала сброса или при включении питания устанавливается реальный режим, причем МП 80386 работает как очень быстрый МП 8086, но, по желанию программиста, с 32-разрядным расширением. В реальном режиме МП 80386 имеет такую же базовую архитектуру, что и МП 8086, но обеспечивает доступ к 32-разрядным регистрам. Механизм адресации, размеры памяти и обработка прерываний (с их последовательными ограничениями) МП 8086 полностью совпадают с аналогичными функциями МП 80386 в реальном режиме.

Для формирования логического адреса в реальном режиме используются два компонента. Для определения линейного базового адреса сегмента используется 16-разрядный селектор. Полученный базовый адрес используется как 32-разрядный физический адрес. Различие двух режимов состоит в способе вычисления базового адреса.

[1)] 1 Терабайт = 1 Тбайт = 2^{40} байт.– *Прим. ред.*

[2)] Другое название – диспетчер памяти.– *Прим. ред.*

Перемещаемость программ или данных заключается в том, что они могут быть размещены в разные моменты времени в различных областях памяти, не требуя от системы или прикладных программ своей модификации. Перемещение сегмента производится в МП 80386 точно так же, как и в МП 8086. В селекторе сегмента 16-разрядное значение сдвигается влево на четыре бита (т. е. умножается на 16) для формирования 20-разрядного базового адреса сегмента. Исполнительный адрес расширяется четырьмя старшими нулевыми битами для получения 20-разрядного значения и добавляется к базовому адресу. Таким образом формируется линейный адрес, который совпадает с физическим адресом, поскольку в реальном режиме разбиение на страницы не используется.

Прерывания и исключения нарушают нормальный порядок работы системы или программы. В реальном режиме они обрабатываются точно так же, как и в МП 8086. Список прерываний, воспринимаемых МП 80386, приведен в гл. 3. В реальном режиме таблица дескрипторов прерываний (IDT) МП 80386 представляет собой таблицу векторов реальных прерываний МП 8086 и занимает пространство от нулевого реального адреса до реального адреса 1024 (4 байт на вектор прерывания с 256 возможными входами).

Единственным способом выхода из реального режима является явное переключение в защищенный режим. В защищенный режим МП 80386 входит при установке бита включения защиты (PE) в нулевом регистре управления (CR0) с помощью команды пересылки (MOV to CR0). Для совместимости с МП 80286 с целью установки бита PE может быть также использована команда загрузки слова состояния машины LMSW.

Процессор повторно входит в реальный режим в том случае, если программа командой пересылки сбрасывает бит PE регистра CR0.

Защищенный режим. Полные возможности МП 80386 раскрываются в защищенном режиме. Программы могут исполнять переключение между процессами с целью входа в задачи, предназначенные для режима виртуального МП 8086. Каждая такая задача проявляет себя в семантике МП 8086 (т. е. в отношениях между символами и приписываемыми им значениями независимо от интерпретирующего их оборудования). Это позволяет выполнять на МП 80386 программное обеспечение для МП 8086 – прикладную программу или целую операционную систему. В то же время задачи для виртуального МП 8086 изолированы и защищены как друг от друга, так и от главной операционной системы МП 80386.

Подобно реальному режиму, в защищенном режиме используется 16-разрядный селектор для указания в определенной операционной системной таблице индекса, который содержит 32-разрядный базовый адрес сегмента. Физический адрес формируется путем добавления базового адреса, полученного из этой таблицы, к относительному адресу. Программы, написанные для МП 80286, выполняются на МП 80386 без модификации. Этот МП поддерживает также дескрипторы, используемые в МП 80286, поскольку зарезервированное фирмой Intel младшее слово дескриптора равно нулю.

1.3. Типы данных

Микропроцессор 80386 подразделяет память на 8-разрядные байты, 16-разрядные слова и 32-разрядные двойные слова. Байт образуют восемь соседних битов, начинающихся с любого логического адреса, и эти биты нумеруются с 0 до 7, причем бит 0 является самым младшим значащим битом. На рис. 1.5 показано сравнение основных типов данных.

Слова хранятся в порядке «старший байт, младший байт». Таким образом,

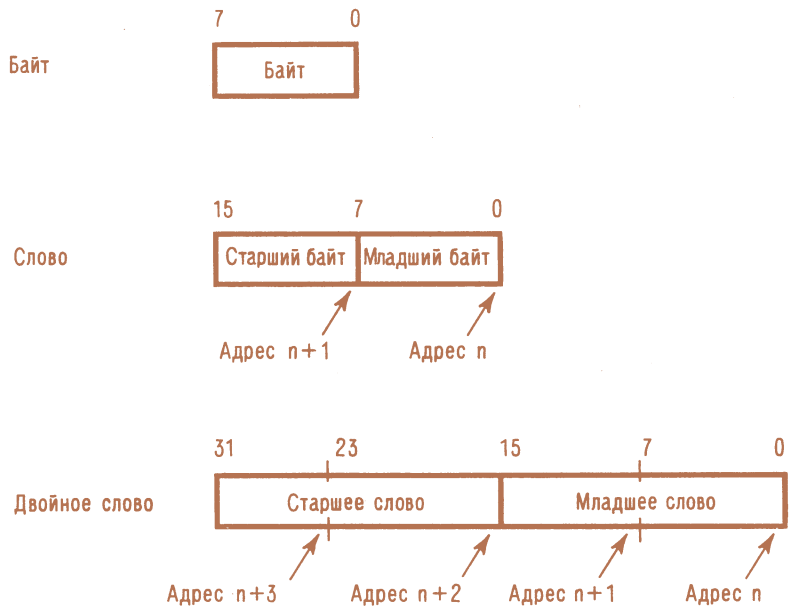


Рис. 1.5. Основные типы данных.

значение, которое в регистре AX было бы 0201, в памяти было бы представлено как 0102 – числа в шестнадцатеричной системе счисления. (Кажется, что байт-ориентированные передачи и обращение байтов при обмене с регистрами останутся с нами навсегда.) То есть слова содержат 16 бит и младший байт хранится по меньшему адресу, а старший байт – по большему адресу. Каждый байт внутри слова имеет свой собственный адрес. Адресом слова или двойного слова является адрес соответствующего младшего байта.

Для достижения максимальной гибкости в структурах данных и эффективного использования памяти слова необязательно выравнивать по четным адресам. Двойные слова также необязательно выравнивать по адресам, кратным четырем. Однако при использовании в системной конфигурации с 32-разрядной магистралью обмен данными между процессором и памятью производится двойными словами, начинающимися с адресов, кратных четырем. Процессор преобразует запросы невыровненных слов или двойных слов в последовательность запросов, приемлемых для интерфейса с памятью. Таким образом, отсутствие выравнивания данных снижает производительность из-за увеличения количества обращений к памяти. Для повышения производительности структуры данных и стеки следует выравнивать по четным адресам и по адресам, кратным четырем.

Поскольку предварительная выборка с образованием очереди команд осуществляется внутри МП, то команды не обязаны быть выровнены по границам слов или двойных слов. Однако при таком выравнивании скорость обработки несколько увеличивается.

В дополнение к байтам, словам и двойным словам МП 80386 поддерживает также следующие дополнительные типы данных.

Неупакованный двоично-десятичный тип – распакованное байтовое представление десятичной цифры от 0 до 9. Распакованные десятичные числа хранятся как беззнаковые байтовые значения по одной цифре в каждом байте. Значение цифры определяется младшим полубайтом. Старший полубайт должен быть равным нулю

при делении и умножении, но может иметь любое значение при вычитании или сложении.

Упакованный двоично-десятичный тип – упакованное байтовое представление двух десятичных цифр от 0 до 9. Каждая цифра хранится в своем полубайте. Цифра в старшем полубайте является более значимой. Диапазон упакованного десятичного байта составляет от 0 до 99.

Битовое поле – непрерывная последовательность битов, в которой каждый бит рассматривается как независимая переменная. Битовое поле может начинаться с любого бита любого байта и может быть длиной до 32 бит.

Битовая строка – подобно битовому полю, битовая строка является непрерывной последовательностью битов. Битовая строка может начинаться с любого бита любого байта и иметь длину до $(2^{32}-1)$ бит.

Ближний указатель – 32-разрядный логический адрес, который представляет собой относительный адрес внутри сегмента. Ближние указатели используются как в сплошной, так и в сегментированной модели памяти.

Дальний указатель – 48-разрядный логический адрес из двух компонентов: 16-разрядного сегмента и 32-разрядного относительного адреса. Дальние указатели используются программистами только в том случае, когда конструкторы системы выбирают сегментированную организацию памяти.

Целый тип – знаковое двоичное значение, содержащееся в 32-разрядном двойном слове, 16-разрядном слове или 8-разрядном байте. Все операции предполагают представление чисел в дополнительном коде. Знаковый бит расположен в бите 7 в байте, в бите 15 в слове и в бите 31 в двойном слове. Он равен нулю для положительных чисел и единице для отрицательных. Поскольку этот старший бит используется как знаковый, то 8-разрядное (байт) целое число может изменяться в диапазоне от -128 до $+127$, 16-разрядное (слово) целое число в диапазоне от -32768 до $+32767$, а 32-разрядное (двойное слово) целое число в диапазоне от -2^{31} до $+2^{31}-1$. Нулевое значение имеет положительный знак.

Обычный тип – беззнаковое двоичное значение, содержащееся в 32-разрядном двойном слове, 16-разрядном слове или 8-разрядном байте. Все биты определяют величину числа. Значение числа для байта составляет от 0 до 255, для слова от 0 до 65535 и для двойного слова от 0 до $2^{32}-1$.

Строка – непрерывная последовательность байтов, слов или двойных слов. Строка может содержать от 0 до $2^{32}-1$ байтов или 4 Гбайт.

1.4. Регистры

Регистр является устройством временного хранения данных и используется с целью облегчения арифметических, логических и пересылочных операций. Регистры МП 80386 являются расширением регистров прежних МП 8086, 80186, 80286. Все 16-разрядные регистры МП предыдущих поколений содержатся внутри 32-разрядной архитектуры.

Микропроцессор 80386 включает шесть непосредственно доступных программисту регистров селекторов сегментов, которые содержат указатели сегментов. Значения этих селекторов могут быть загружены при исполнении программы и являются специфичными для задачи. Это значит, что регистры сегментов перезагружаются автоматически при переключении МП 80386 на другую задачу. За регистрами селекторов сегментов стоят реальные регистры кэш-памяти сегментов, которые содержат описания сегментов, указываемых селектором. Это сделано на аппаратном уровне для того, чтобы избежать дополнительной выборки из памяти в случае, когда требуется описание сегмента.

Регистры	Использование в реальном режиме		Использование в защищенном режиме		Использование в режиме виртуального МП 8086	
	Загрузка	Сохранение	Загрузка	Сохранение	Загрузка	Сохранение
Регистры общего назначения	Да	Да	Да	Да	Да	Да
Регистры сегментов	Да	Да	Да	Да	Да	Да
Регистр флагов	Да	Да	Да	Да	IOPL	IOPL
Регистры управления	Да	Да	PL = 0	PL = 0	Нет	Да
Регистр таблицы GDT	Да	Да	PL = 0	Да	Нет	Да
Регистр таблицы LDT	Нет	Нет	PL = 0	Да	Нет	Да
Регистр таблицы IDT	Да	Да	PL = 0	Да	Нет	Нет
Регистр транзакции	Нет	Нет	PL = 0	Да	Нет	Нет
Регистры управления отладкой	Да	Да	PL = 0	PL = 0	Нет	Нет
Регистры проверки	Да	Да	PL = 0	PL = 0	Нет	Нет

PL = 0 — регистр доступен только при нулевом текущем уровне привилегий.
IOPL — в режиме виртуального МП 8086 команды PUSHF и POPF сделаны чувствительными к уровню привилегий ввода-вывода.

Рис. 1.6. Доступность регистров.

Существуют некоторые отличия в возможностях доступа к регистрам в реальном и защищенном режимах, показанные на рис. 1.6.

1.4.1. Регистры общего назначения (РОН)

Восемь регистров общего назначения имеют длину в 32 бит и содержат адреса или данные. Они поддерживают операнды-данные длиной 1, 8, 16, 32 и (при использовании двух регистров) 64 бит; битовые поля от 1 до 32 бит; операнды-адреса длиной 16 и 32 бит. Эти регистры называются EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP.

Доступ к младшим 16 бит этих регистров выполняется независимо. Это делается в большинстве ассемблеров при использовании 16-разрядных имен регистров: AX, BX, CX, DX, SI, DI, BP и SP. Формат регистров общего назначения показан на рис. 1.7.

1.4.2. Регистр системных флагов

Регистр EFLAGS управляет вводом-выводом, маскируемыми прерываниями, отладкой, переключением задач и включением исполнения в режиме виртуального МП 8086 в защищенной многозадачной среде – все это в дополнение к флагам состояния,

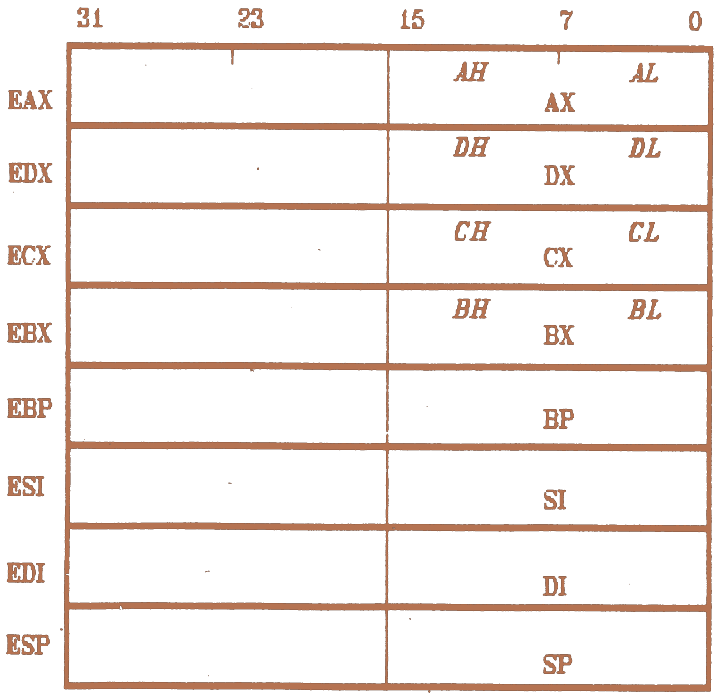


Рис. 1.7. Регистры общего назначения.

которые отражают результат исполнения команды. На рис. 1.8 показано содержимое регистра EFLAGS. Младшие 16 бит его представляют собой 16-разрядный регистр флагов и состояния МП 80286, называемый FLAGS, который наиболее полезен при исполнении программ для МП 8086 и 80286.

В нижеследующих описаниях термин «установлен» означает «установлен в 1», а термин «сброшен» означает «установлен в 0».

VM (режим виртуального МП 8086), бит 17. Бит обеспечивает режим виртуального МП 8086 в защищенном режиме. Если он установлен, когда процессор находится в защищенном режиме, то МП 80386 переключается на работу в режиме виртуального МП 8086. Бит VM может быть установлен только двумя способами: 1) в защищенном режиме командой IRET, только если текущий уровень привилегий равен нулю; 2) переключением задачи на любом уровне привилегий.

RF (флаг возобновления), бит 16. Временно приостанавливает обработку исключений для отладки (т. е. возвращает к нормальному исполнению программы) так, что исполнение команды может быть повторено после обработки исключения для отладки, не вызывая немедленно обработку другого исключения для отладки.

NT (вложенная задача), бит 14. Процессор использует этот флаг для управления последовательностью прерываемых и вызываемых задач. Команда CALL временно передает управление от программы к подпрограмме. При завершении этой подпрограммы управление возобновляется с команды, следующей за командой CALL. Бит NT влияет на выполнение команды IRET.

IOPL (уровень привилегий ввода-вывода), биты 12–13. Это двухбитовое поле используется в защищенном режиме. Биты IOPL показывают наивысшее значение текущего уровня привилегий (CPL), позволяющее выполнять команды ввода-вывода, не приводя к обработке исключения 13 или обращения к битовой карте

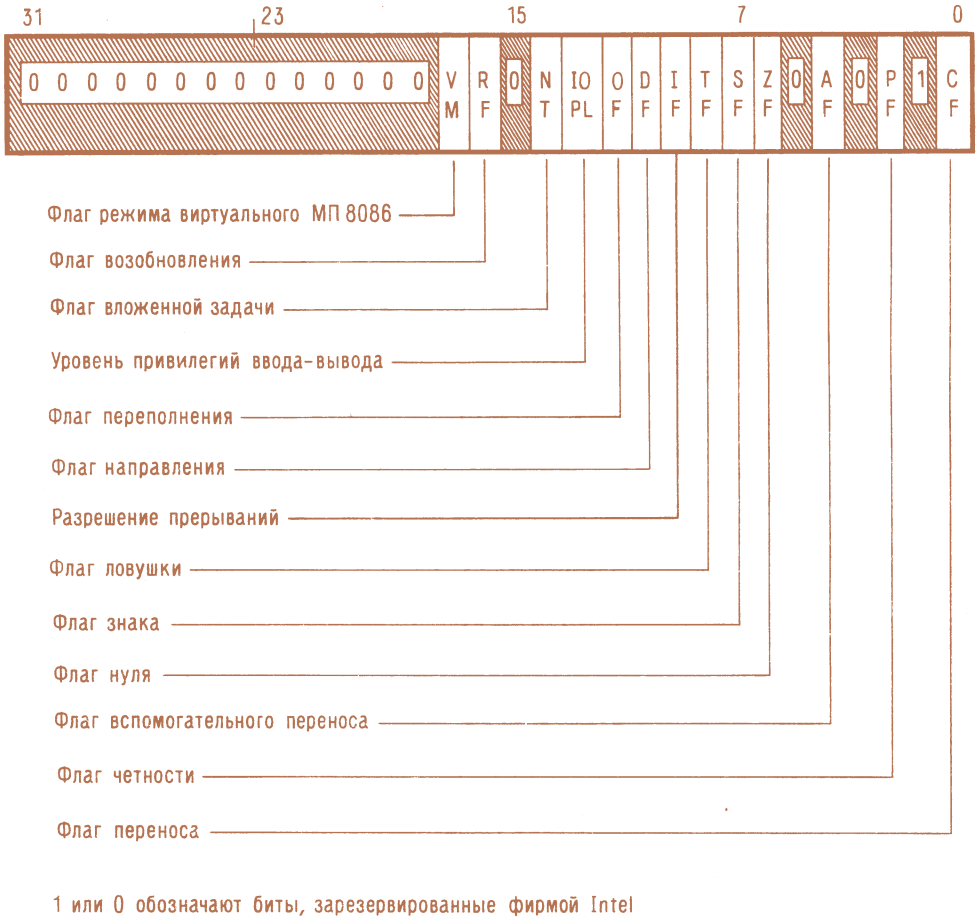


Рис. 1.8. Регистр EFLAGS.

разрешения ввода-вывода. Это поле показывает также наивысшее значение CPL, которое позволяет изменять бит IF при выборке нового значения из стека в регистр EFLAGS (команды выборки POP и загрузки PUSH в стек рассматриваются в гл. 10).

OF (флаг переполнения), бит 11. Флаг установлен, если операция привела к переносу (заему) в знаковый (самый старший) бит результата, но не привела к переносу (заему) из самого старшего бита, или наоборот.

DF (флаг направления), бит 10. Определяет, должны ли регистры ESI и/или EDI инкрементироваться или декрементироваться во время операций со строками. Если флаг сброшен, то они инкрементируются, иначе декрементируются.

IF (разрешение прерывания), бит 9. Установка флага IF позволяет МП воспринимать запросы внешних (маскируемых) прерываний. Очистка этого бита запрещает такие прерывания. Флаг не влияет на обработку как маскируемых внешних прерываний, так и исключений.

TF (флаг ловушки), бит 8. Установка флага TF переводит МП в пошаговый режим для отладки. Процессор автоматически генерирует исключение 1 после каждой команды, что позволяет проверить программу на исполнение каждой

команды. Когда флаг IF сброшен, то ловушка по исключению 1 возникает в точках адресов останова, загружаемых в регистры отладки DR0–DR3. Более подробная информация дана ниже при обсуждении регистров отладки.

SF (флаг знака), бит 7. Установлен, если установлен старший бит результата, иначе он сброшен. Для 8-, 16- и 32-разрядных операций этот флаг отражает состояние бита 7, 15 и 31 соответственно.

ZF (флаг нуля), бит 6. Установлен, если все биты результата равны нулю, иначе сброшен.

AF (флаг вспомогательного переноса), бит 4. Используется для упрощения сложения и вычитания упакованных двоично-десятичных чисел. Независимо от длины операнда (8, 16 или 32 бит) флаг AF установлен, если операция привела к заему из бита 3 при вычитании или переносу из бита 3 при сложении, иначе он сброшен. Напомним, что двоично-десятичные числа используют биты от 0 до 3 для представления десятичных цифр.

PF (флаг четности), бит 2. Установлен, если младшие восемь битов операнда содержат четное число единиц (проверка на четность), иначе сброшен. На этот флаг влияют только младшие восемь битов независимо от длины операнда.

CF (флаг переноса), бит 0. Установлен, если операция привела к переносу из старшего бита при сложении или к заему в старший бит при вычитании, иначе сброшен. Для 8-, 16- и 32-разрядных операций этот бит устанавливается при переносе из битов 7, 15 и 31 соответственно.

1.4.3. Регистры сегментов

Шесть 16-разрядных регистров содержат значения селекторов сегментов, которые указывают на текущие адресуемые сегменты памяти. Ниже перечислены эти регистры.

Регистр сегмента программы (CS). Указывает на сегмент, который содержит текущую последовательность исполняемых команд. Процессор выбирает все команды из этого сегмента, используя содержимое счетчика команд как относительный адрес. Содержимое регистра CS изменяется в результате выполнения внутрисегментных команд управления потоком, прерываний и исключений. Он не может быть загружен явным способом.

Регистр сегмента стека (SS). Вызовы подпрограмм, записи параметров и активизация процедур обычно требуют области памяти, резервируемой под стек. Все операции со стеком используют регистр SS при обращении к стеку. В отличие от регистра CS регистр SS может быть загружен явно с помощью команды программы.

Остальные четыре регистра являются регистрами сегментов данных (DS, ES, FS, GS), каждый из которых адресуется текущей исполняемой программой. Доступ к четырем раздельным областям данных имеет целью повысить эффективность программ, позволяя им обращаться к различным типам структур данных. Содержимое этих регистров может быть заменено под управлением программы.

При использовании регистров сегментов МП 80386 с каждым выбранным сегментом связывает базовый адрес. При адресации единицы данных внутри сегмента к базовому адресу сегмента добавляется 32-разрядный относительный адрес. Если сегмент выбран загрузкой селектора сегмента в регистр сегмента, то командам манипуляции данными нужен только этот относительный адрес.

Регистры, которые содержат дескрипторы сегментов, программно недоступны. Внутри МП 80386 регистр дескриптора связан с каждым программно доступным регистром сегмента. Каждый регистр дескриптора содержит 32-разрядный базовый адрес сегмента, 32-разрядную границу сегмента (поле, определяющее размер сегмента) и другие необходимые атрибуты.

Когда значение селектора загружается в регистр сегмента, процессор автоматически обновляет связанный с ним регистр дескриптора. В защищенном режиме базовый адрес, граница и атрибуты целиком обновляются содержимым дескриптора сегмента, выбранного селектором. В режиме реальных адресов обновляется только базовый адрес путем сдвига значения селектора на четыре бита влево, поскольку максимальная граница сегмента и его атрибуты в реальном режиме фиксированы.

1.4.4. Регистры управления сегментированной памятью

Эти регистры известны также как регистры системных адресов. Четыре регистра указывают структуры данных, которые управляют механизмом сегментированной памяти. Они определены для ссылок на таблицы или сегменты, поддерживаемые моделью защиты МП 80286 и 80386.

Адреса этих таблиц и сегментов хранятся в специальных регистрах системных адресов и системного сегмента. В гл. 2 подробно описаны таблица глобальных дескрипторов, таблица локальных дескрипторов и таблица дескрипторов прерываний.

Регистр таблицы глобальных дескрипторов (GDTR). Содержит 32-разрядный линейный адрес и 16-разрядную границу таблицы глобальных дескрипторов.

Регистр таблицы локальных дескрипторов (LDTR). Содержит 16-разрядный селектор для таблицы локальных дескрипторов. Так как эта таблица является специфичным для задачи сегментом, то она определяется значением селектора, хранимым в регистрах системного сегмента. Регистр дескриптора сегмента, связанный с этой таблицей, программно недоступен.

Регистр таблицы дескрипторов прерываний (IDTR). Указывает на таблицу точек входа в программы обработки прерываний. Регистр содержит 32-разрядный линейный базовый адрес и 16-разрядную границу таблицы дескрипторов прерываний (IDT).

Регистр задачи (TR). Указывает на информацию, необходимую процессору для определения текущей задачи. Регистр TR содержит 16-разрядный селектор дескриптора сегмента состояния задачи (см. подробно в гл. 2). Поскольку этот сегмент специфичен для задачи, то он определяется значениями селекторов, хранящихся в регистрах системного сегмента. Заметим, что с каждым регистром системных сегментов связан программно недоступный регистр дескриптора сегмента.

1.4.5. Указатель команд

Расширенный указатель команд (EIP) является 32-разрядным регистром. Он содержит относительный адрес следующей команды, подлежащей выполнению. Относительный адрес отсчитывается от начала (или базового адреса) сегмента текущей программы. Указатель команд непосредственно не доступен программисту, но он управляется явно командами управления потоком, прерываниями и исключениями.

Младшие 16 бит регистра EIP называются IP и могут быть использованы процессором независимо. Это свойство полезно при исполнении команд МП 8086 и 80286, которые имеют только регистр IP.

1.4.6. Регистры управления

Микропроцессор 80386 имеет три 32-разрядных регистра управления (CR0, CR2 и CR3, а CR1 зарезервирован фирмой Intel), в которых хранятся состояния машины или глобальные состояния. Глобальное состояние – это такое состояние, к которому

может получить доступ любой из логических блоков системы или которое управляет этими блоками. Вместе с регистрами системных адресов эти регистры хранят информацию о состоянии машины, которая влияет на все задачи в системе. Для доступа к регистрам управления определены команды их загрузки и сохранности содержимого.

Системным программистам регистры управления доступны только через варианты команды MOV, которые позволяют их загружать или сохранять в регистрах общего назначения. Формат четырех регистров управления показан на рис. 1.9.

Регистр CR0 содержит флаги, которые управляют или показывают условия, относящиеся к системе в целом, а не к отдельной задаче. Младшие 16 бит (биты с 0 по 15) этого регистра являются словом состояния машины (MSW) для совместимости с защищенным режимом МП 80286.

Бит 31 разбиения на страницы (PG) показывает, использует ли МП 80386 таблицы страниц для трансляции линейных адресов в физические.

Бит 4 расширения (ET) показывает тип сопроцессора (80287 или 80387), присутствующего в системе.

Бит 3 переключения задачи (TS) процессор устанавливает при включении каждой задачи. Он также проверяет этот бит, когда интерпретирует команды сопроцессора, так как состояние последнего требуется сохранять только в случае запроса сопроцессора новой задачей. Бит TS может быть очищен загрузкой в регистр CR0, а также и специальной командой CLTS очистки этого бита.

Бит 2 эмуляции (EM) показывает, нужно ли эмулировать функции сопроцессора. Эмуляция используется для имитации одной системы или процессора другой таким образом, что имитирующая система воспринимает те же самые данные,

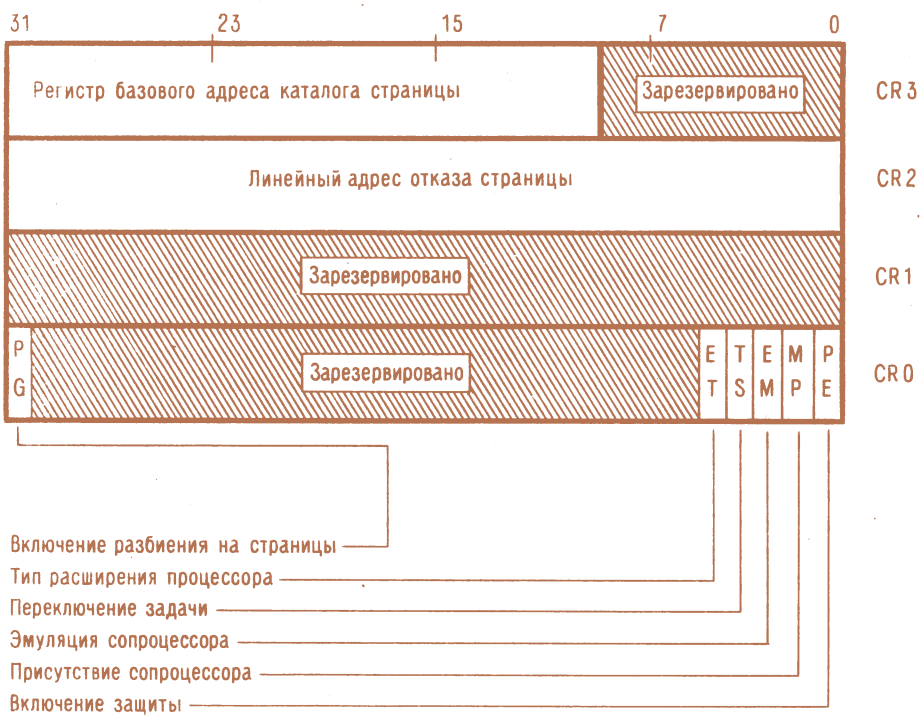


Рис. 1.9. Форматы регистров управления.

исполняет те же самые программы и достигает тех же самых результатов, что и имитируемая система или функция. Установка бита EM часто возникает в случае отсутствия сопроцессора в системе.

Бит 1 арифметического сопроцессора (MP) управляет функцией команды WAIT, которая используется для взаимодействия с сопроцессором.

Бит 0 включения защиты (PE): при установке процессор переходит в защищенный режим, а при очистке – в режим реальных адресов.

Регистр CR1 зарезервирован для будущих процессоров фирмы Intel. Регистр CR2 используется для обработки отказов страницы в случае, когда установлен бит PG в регистре CR0. В регистре CR2 процессор сохраняет тот линейный адрес, который вызвал отказ. Код ошибки, запоминаемый при возникновении отказа в стеке обработки отказов страницы, обеспечивает дополнительную информацию о состоянии при этом отказе.

Регистр CR3 используется, когда установлен бит PG в регистре CR0. Этот регистр позволяет процессору определить место расположения таблицы каталога страниц для текущей задачи. Он содержит физический базовый адрес таблицы каталога страниц. Таблица каталога страниц МП 80386 всегда выровнена на размер страницы в 4 Кбайт, поэтому младшие 12 бит регистра CR3 игнорируются при записи в него и не определены при его сохранении.

Включение задачи через сегмент состояния задачи TSS, который изменяет значение или явно загружает регистр CR3, отменяет все входы в кэш-таблицу страниц в блоке кэширования страниц. Заметим, что если не изменять значение регистра CR3, то входы в кэш-таблицу страниц не пропадают.

1.4.7. Регистры отладки

Шесть доступных программисту регистров отладки (DR0 – DR3, DR6 и DR7) расширяют возможности отладки в МП 80386, они устанавливают точки останова по данным и позволяют устанавливать точки останова по командам без модификации сегментов программ. Регистры DR0 – DR3 предназначены для четырех линейных точек останова. Регистры DR4 и DR5 зарезервированы фирмой Intel для будущих разработок. Регистр DR6 показывает текущее состояние точек останова, а регистр DR7 используется для установки точек останова.

1.4.8. Буфер ассоциативной трансляции

Буфер ассоциативной трансляции (TLB) – это кэш-память, используемая для трансляции линейных адресов в физические.

Предупреждение: механизм проверки TLB является уникальным для МП 80386 и может быть не реализован в том же виде в будущих процессорах. Программы, которые используют этот механизм в его нынешнем виде, могут оказаться несовместимыми с будущими процессорами.

Буфер TLB является четырехканальной ассоциативной по множеству памятью. Множество – это набор элементов, имеющих некоторое общее свойство или находящихся в определенном отношении друг с другом. Структура TLB показана на рис. 1.10. В буфере TLB имеются следующие элементы:

Ассоциативное запоминающее устройство (АЗУ). Адресуемое по содержимому ячейки АЗУ содержит 32 линейных адреса и связанные с ними так называемые *теговые* биты, которые используются для защиты данных и реализации кэш-памяти.

Запоминающее устройство с произвольной выборкой (ЗУПВ). Адресуемое по номеру (адресу) ячейки ЗУПВ содержит старшие 20 бит 32-разрядных физических адресов, которые соответствуют линейным адресам в АЗУ.

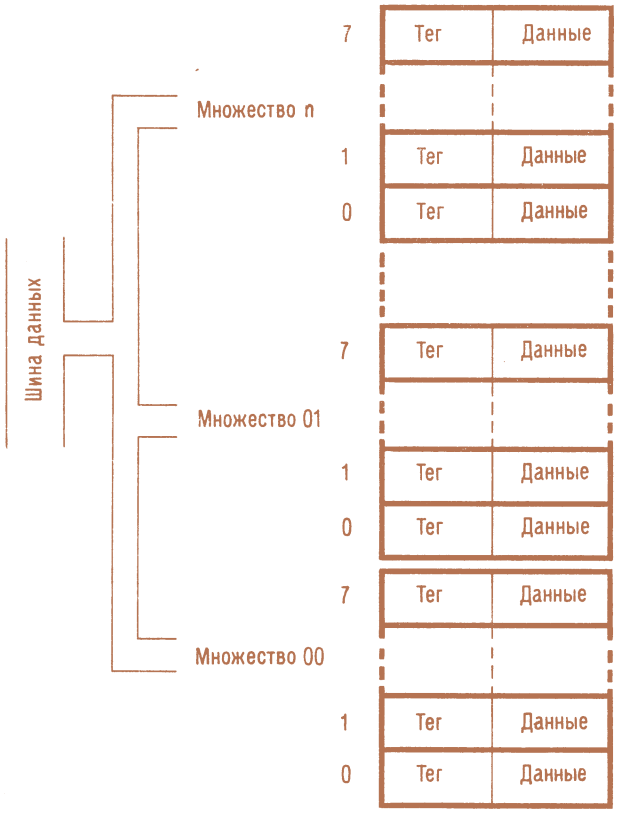


Рис. 1.10. Структура буфера трансляции адресов.

Аппаратная логика реализует четырехканальную кэш-память и включает двухразрядный указатель замены, определяющий для каждого из четырех каналов, к какому входу он присоединяется при записи в буфер TLB.

Адреса и команды записываются в буфер TLB через регистр команд, тогда как данные считываются из или записываются в буфер через регистр данных. Регистр проверки TR6 является регистром команд при доступе к буферу TLB, а регистр проверки TR7 используется как регистр данных.

Регистры проверки. Два регистра проверки используются для управления проверкой АЗУ или ЗУПВ в буфере TLB, а кэш-память – для хранения информации из таблиц страниц. Регистр TR6 является командным регистром проверки, а регистр TR7 – регистром данных, который содержит данные для проверки буфера TLB. (Регистры TR0 – TR5 не существуют.)

Эти регистры доступны в различных вариантах команды MOV, которые определены как для режима реальных адресов, так и для защищенного режима. В защищенном режиме команда MOV, которая обращается к этим регистрам, может исполняться только на уровне привилегий 0. Любая попытка считать из или записать в любой из этих регистров на любом другом уровне привилегий вызовет обработку исключения общей защиты. Формат регистров проверки показан на рис. 1.11.

Регистр TR6 имеет следующие поля:

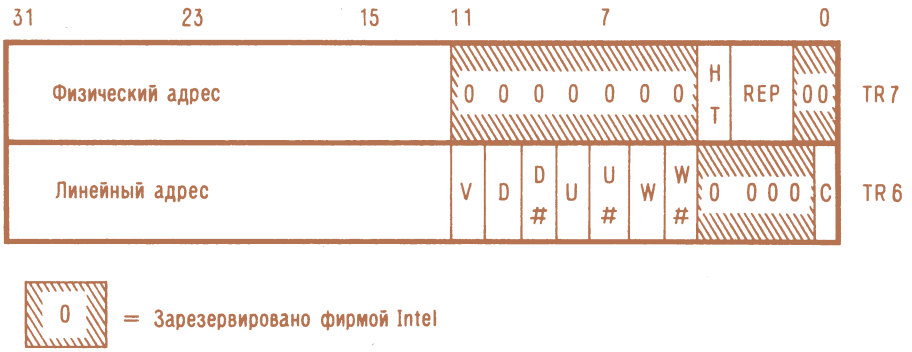


Рис. 1.11. Форматы тестовых регистров.

Линейный адрес – биты 12–31. При записи в буфер TLB для данного линейного адреса резервируется вход в буфер TLB. Остальные поля входа в буфер TLB устанавливаются в зависимости от значения регистра TR7 и значения, только что записанного в регистр TR6. При просмотре буфера TLB в нем ищется это значение. Если найден один и только один вход в буфер TLB, то остальные поля регистров TR6 и TR7 устанавливаются по найденному входу.

Бит достоверности V – бит 11. Буфер TLB использует этот бит для определения входов, содержащих достоверные данные. Достоверность означает, что была проведена успешная трансляция адресов. Входы в буфер TLB, которым не были присвоены определенные значения, имеют в этом бите нуль. Все биты достоверности могут быть очищены при записи в регистр управления CR3.

Биты мусора D, D # – биты 10 и 9. Бит 10 является битом мусора, показывающим, был ли изменен вход в буфер TLB, а бит 9 является его инверсией для/из входа в буфер TLB.

Биты доступности пользователю U, U # – биты 8 и 7. Бит 8 является битом доступности пользователю или системе, а бит 7 является его инверсией для/из входа в буфер TLB.

Биты чтения/записи W, W # – биты 6 и 5. Бит 6 управляет доступом по чтению/записи, а бит 5 является его инверсией для/из входа в буфер TLB.

Бит команды C – бит 0. В регистре TR6 имеется теговый бит команды и адреса. Для немедленного просмотра буфера TLB в регистр TR6 пересылается двойное слово, содержащее в этом бите 1. Для немедленной записи в буфер TLB в этот регистр пересылается двойное слово, содержащее в этом бите 0.

Регистр TR7 имеет следующие поля:

Физический адрес – биты 31–12. Эти биты являются полем данных буфера TLB. При записи в TLB вход в этот буфер, зарезервированный для линейного адреса в регистре TR6, устанавливается по значению этих битов. Если при просмотре буфера TLB обнаруживается, что бит HT установлен, то поле данных (физический адрес) считывается в это поле, в противном случае значение этого поля не определено.

Бит успеха просмотра HT – бит 4. Для записи в буфер TLB бит HT должен быть установлен. Если просмотр буфера TLB был успешным – HT = 1, если нет – HT = 0.

Биты блока с тегом REP – биты 3 и 2. При записи в буфер TLB должен быть записан один из четырех ассоциативных блоков. Эти биты показывают, какой именно. Если при чтении буфера TLB обнаруживается, что бит HT установлен, то эти биты показывают, в каком из четырех ассоциативных блоков был найден тег, иначе их значение не определено.

Глава 2

Селекторы и дескрипторы

В МП 80386 селекторы и дескрипторы снабжают аппаратные средства сведениями об ожидаемых программами местах расположения информации различного типа.

2.1. Селекторы

Селектор – это 16-разрядный указатель, который, будучи загружен в регистр или использован с определенными командами, выбирает определенные дескрипторы. В логическом адресе селекторная часть определяет индивидуальный дескриптор, во-первых, определяя таблицу дескрипторов, а во-вторых, указывая этот дескриптор в данной таблице. Общий формат селектора показан на рис. 2.1. Различные термины, такие как дескриптор и таблица глобальных дескрипторов, определяются позже в этой главе.

Селектор имеет следующие поля:

Индекс – биты 15–4. Индекс выбирает один из 8192 дескрипторов в таблице дескрипторов. Процессор умножает значение этого индекса на восемь (длину дескриптора) и добавляет результат к базовому адресу таблицы дескрипторов. Это обеспечивает правильный вход в таблицу.

Индикатор таблицы TI – бит 3. Определяет таблицу дескрипторов, на которую ссылается селектор: нуль означает таблицу глобальных дескрипторов (GDT), а единица – текущую таблицу локальных дескрипторов (LDT).

Запрашиваемый уровень привилегий RPL– биты 2–0. Используются механизмом защиты системы, более подробно рассматриваемым в гл. 6.

Первый вход в таблицу GDT процессором не используется. Селектор, имеющий нулевые индекс и индикатор таблицы, т. е. селектор, указывающий на первый вход в таблицу GDT, называется *нуль-селектором*. Загрузка нуль-селектора в любой регистр сегмента, отличный от регистра CS или SS, не вызывает в МП 80386 обработки исключения. Но использование этого регистра сегмента для доступа к памяти вызывает обработку исключения. Это удобно для выявления случайных ссылок.

Рис. 2.1. Формат селектора.

2.2. Дескрипторы сегментов

Дескрипторы – это те объекты, на которые показывают селекторы сегмента. Они представляют собой 8-байтовые блоки, содержащие атрибуты области линейных адресов, т. е. сегмента. Эти атрибуты включают 32-разрядный линейный базовый адрес сегмента, 20-разрядную длину и дробность сегмента, уровень защиты, привилегии по чтению, записи и выполнению, размер операндов по умолчанию (16- или 32-разрядные) и тип сегмента. Вся информация об атрибутах дескриптора содержится в 12 битах дескриптора сегмента. Сегменты МП 80386 в целом имеют три поля атрибутов: бит P присутствия, биты DPL уровня привилегий дескриптора и бит S дескриптора сегмента.

Дескрипторы сегментов хранятся либо в таблице GDT, либо в таблице LDT, которые МП 80386 адресует с помощью регистров GDTR или LDTR соответственно.

Дескриптор сегмента обеспечивает МП 80386 данными, которые ему нужны для отображения логического адреса в линейный адрес. Эти дескрипторы создаются не прикладными программами, а компиляторами, редакторами связей, загрузчиками или операционной системой. На рис. 2.2 показан общий формат дескриптора сегмента.

Дескриптор сегмента имеет следующие поля:

База. Определяет место сегмента внутри линейного 4-Гбайт адресного пространства. Процессор объединяет три фрагмента базового адреса для формирования одного 32-разрядного значения.

Граница. Определяет размер сегмента. Процессор связывает две части поля границы для формирования 20-разрядного результата. Затем он интерпретирует поле границы одним из двух способов в зависимости от состояния бита дробности:

1) в единицах байтов для определения границы до 1 Мбайт;

2) в единицах страниц по 4 Кбайт для определения границы до 4 Гбайт, при загрузке поле границы сдвигается влево на 12 бит и в младшие биты вставляются единицы.

Рис. 2.2. Общий формат дескриптора сегмента.

Бит дробности G. Определяет размер единиц, в которых интерпретируется поле границы. Если $G = 0$, то граница интерпретируется в байтах, иначе в единицах по 4 Кбайт.

Бит системного сегмента S. Определяет, является ли данный сегмент системным ($S = 0$) или же сегментом программы или данных ($S = 1$).

Тип. Интерпретация этого поля зависит от вида дескриптора. В дескрипторах сегментов программы или данных это поле содержит 3-разрядный тип и один бит флага обращения. Дескрипторы системных сегментов используют следующий набор значений в поле типа:

0: запрещено	8: запрещено
1: TSS 80286 доступен	9: TSS 80386 доступен
2: таблица	A: зарезервировано
3: TSS 80286 занят	B: TSS 80386 занят
4: вентиль вызова 80286	C: вентиль вызова 80386
5: вентиль задачи (286/386)	D: зарезервировано
6: вентиль прерывания 80286	E: вентиль прерывания 80386
7: вентиль ловушки 80286	F: вентиль ловушки 80386

Уровень привилегий дескриптора DPL. Используется механизмом защиты.

Бит присутствия сегмента P. Если этот бит очищен, то данный дескриптор не может быть использован при трансляции адресов. Когда селектор такого дескриптора загружается в регистр сегмента, процессор переходит к обработке исключения. На рис. 2.3 показан формат отсутствующего дескриптора.

Бит обращения A. Устанавливается в дескрипторах программы и данных при обращении к сегменту. Операционные системы, реализующие виртуальную память на уровне сегментов, могут следить за частотой использования сегмента путем периодической проверки и очистки этого бита.

В дополнение к значению селектора каждый регистр сегмента имеет связанный с ним «невидимый»[1] кэш-регистр дескриптора сегмента. Когда содержимое регистра сегмента изменяется, то 8-байтовый дескриптор, связанный с этим селектором, автоматически загружается в кэш-регистр. После загрузки этого регистра все ссылки на данный сегмент используют информацию кэшированного дескриптора вместо повторного обращения к самому дескриптору.

Для обеспечения совместимости операционных систем МП 80286 и 80386 последний поддерживает все дескрипторы сегментов МП 80286. Единственное

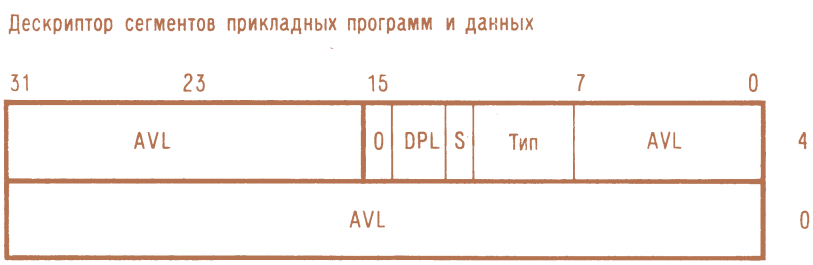


Рис. 2.3. Формат неприсутствующего дескриптора.

[1] То есть программно недоступный. - *Прим. перев.*

различие между двумя форматами состоит в значениях полей типа и в том, что поля границы и базового адреса в МП 80386 расширены.

Дескрипторы сегментов МП 80286 содержат 24-разрядный адрес и 16-разрядную границу. С другой стороны, дескрипторы сегментов МП 80386 имеют 32-разрядный базовый адрес, 20-разрядную границу и бит дробности *G*. Таким образом, если старшее слово дескриптора равно нулю, то это дескриптор для МП 80286.

Еще одно различие между дескрипторами МП 80286 и 80386 заключается в интерпретации поля счетчика слов вентиля вызова и бита *B*. Поле счетчика слов определяет количество 16-разрядных блоков при копировании для вентиля вызова МП 80286 и 32-разрядных блоков для вентиля вызова МП 80386. Бит *B* управляет размером блоков, загружаемых в стек при использовании вентиля вызова. Если $B = 0$, то это блок в 16 бит, иначе блок в 32 бит.

2.3. Таблицы дескрипторов

Таблицы дескрипторов определяют все сегменты, используемые МП 80386. Существуют три типа таблиц: таблица глобальных дескрипторов (GDT), таблица локальных дескрипторов (LDT) и таблица дескрипторов прерываний (IDT). Регистры, которые указывают на эти таблицы (и содержат 32-разрядный линейный базовый адрес и 16-разрядную границу каждой таблицы), называются GDTR, LDTR и IDTR соответственно. Эти регистры рассмотрены ранее в гл. 1.

Таблицы дескрипторов – это массивы памяти переменной длины с 8-байтовыми входами, которые содержат дескрипторы. В МП 80386 они имеют длину от 8 байт до 64 Кбайт и в каждой таблице может быть до 8192 дескрипторов. Старшие 13 битов селектора используются как индекс в таблице дескрипторов. Поскольку эти таблицы обслуживает операционная система, то команды загрузки таблицы дескрипторов являются привилегированными командами. Первый вход в таблицу GDT (с нулевым индексом) в МП 80386 не используется.

Процессор хранит информацию о дескрипторах в регистрах сегментов. Поэтому процессор не должен обращаться к таблице каждый раз, когда ссылается на информацию. Для того чтобы это сделать, существуют два класса команд: команды подразумеваемой загрузки, такие как CALL и JMP, и команды прямой загрузки, такие как POP, MOV, LDS, LFS.

Каждый регистр имеет программно доступную и программно недоступную части. В командах прямой или подразумеваемой загрузки программно доступная часть регистров адресации сегментов используется программами, поскольку это просто 16-разрядные регистры. Программно недоступная часть используется процессором: он выбирает базовый адрес, границу, тип и другую информацию из таблицы дескрипторов и загружает все это в «невидимую» часть регистра сегмента.

2.3.1. Таблица глобальных дескрипторов

Каждая система с МП 80386 содержит таблицу глобальных дескрипторов GDT, которая включает дескрипторы, доступные всем задачам в системе. Таблица GDT может содержать любой вид дескрипторов сегментов, кроме управляющих прерываниями и исключениями.

В общем случае таблица GDT содержит три вида дескрипторов: сегментов программы и данных, используемых операционной системой, дескрипторов для локальных дескрипторов в системе и сегментов состояния задачи. Первый вход в таблице GDT не используется; он соответствует нуль-селектору, который определяет нулевое значение указателя.

2.3.2. Таблицы локальных дескрипторов

Операционные системы часто конструируют так, чтобы каждая задача имела отдельную таблицу локальных дескрипторов LDT. Таблицы LDT обеспечивают способ изоляции сегментов программы и данных конкретной задачи от всего остального в операционной системе и содержат дескрипторы общих для всех задач сегментов. Таблица LDT связана с конкретной задачей и может содержать только дескрипторы программы, данных, стека, вентиля задачи и вентиля вызова.

Сегмент не может быть доступен задаче, если ее дескриптор сегмента не существует ни в текущей таблице LDT, ни в таблице GDT. Это и изолирует, и защищает сегменты конкретной задачи, позволяя все-таки иметь глобальные данные, разделяемые между задачами.

Регистры GDT и LDT содержат 16-разрядную границу и 32-разрядный адрес (16 плюс 32 дает 48 бит или 6 байт). В отличие от 6-байтовых регистров GDT и IDT, которые содержат и базовый адрес, и границу, в программно доступной части регистра LDT имеется только 16-разрядный селектор. Этот селектор ссылается на дескриптор таблицы LDT в таблице GDT.

2.3.3. Таблица дескрипторов прерываний

Таблица дескрипторов прерываний IDT содержит дескрипторы, которые указывают место расположения до 256 программ обработки прерываний. Эта таблица может содержать только вентили ловушки, вентили задачи и вентили прерываний. Таблица должна быть длиной не менее 256 байт, чтобы она могла вместить дескрипторы для 32 зарезервированных фирмой Intel прерываний. Каждое прерывание, используемое в системе, должно иметь вход в таблицу IDT.

Глава 3

Прерывания и исключения

Как аппаратные, так и программные прерывания могут изменить порядок исполнения программ МП 80386. Аппаратное прерывание появляется в ответ на появление сигнала на одном из входов прерываний процессора: входе NMI для немаскируемых прерываний или входе INTR для маскируемых прерываний. Программное прерывание (отказ, ловушка или выход из процесса) появляется в ответ на команду INT или исключение – программное условие, которое требует специальной обработки. Схематичное представление маскируемых и немаскируемых прерываний показано на рис. 3.1.

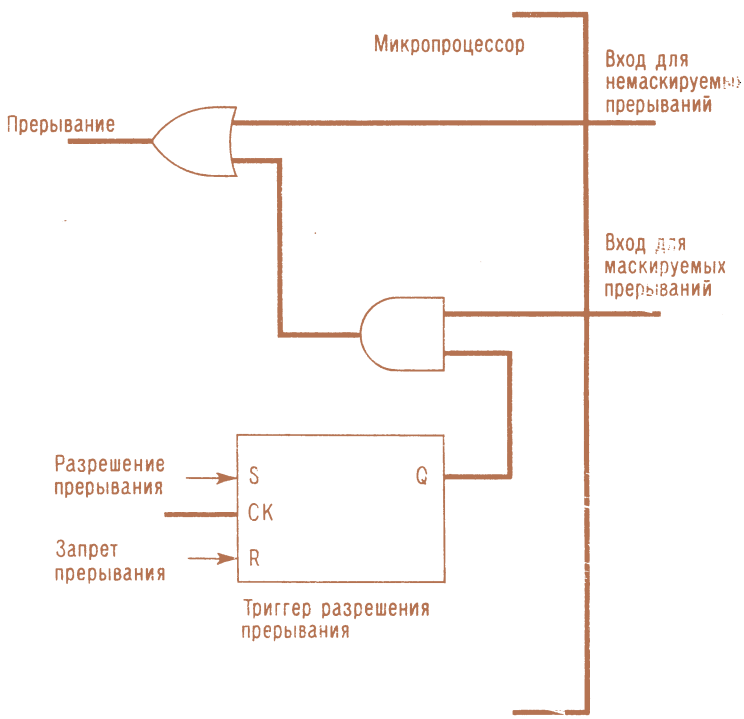


Рис. 3.1. Представление маскируемых и немаскируемых прерываний.

3.1. Маскируемые прерывания

Вход INTR позволяет внешним устройствам прервать исполнение программы МП 80386. Для уверенного опознавания прерывания на входе INTR должен поддерживаться сигнал высокого уровня до тех пор, пока МП 80386 не выдаст сигнал его опознания путем выполнения последовательности команд подтверждения прерывания. Вход INTR опрашивается в начале каждой команды. Сигнал на этом входе должен иметь высокий уровень в течение по крайней мере восьми периодов CLK2 перед командой для того, чтобы гарантировать опознание прерывания; CLK2 – это сигнал удвоенной тактовой частоты для синхронных операций. Соотношение между сигналами CLK и CLK2 показано на рис. 3.2. Такая процедура опознания прерываний уменьшает возможность ложных срабатываний от помех в виде коротких импульсов напряжения. Для опознания прерываний маскируемые прерывания должны быть также программно разрешены. Тактовый генератор системы подробно описан в гл. 7.

На вход INTR сигнал обычно подается через программируемый контроллер прерываний типа Intel 8259A. Контроллер в свою очередь подключается к устройствам, которые требуют обслуживания по прерыванию. С точки зрения МП 80386 контроллер 8259A выглядит как набор портов ввода-вывода. Он воспринимает запросы прерываний от подключенных к нему устройств, выдает сигнал на вход INTR, а затем по запросу МП выставляет соответствующий вектор обработки прерывания. Вектор обработки прерывания – это вход в таблицу IDT, который указывает на программу обработки прерывания. В данном случае используется тот же самый механизм, что и для команды INT.

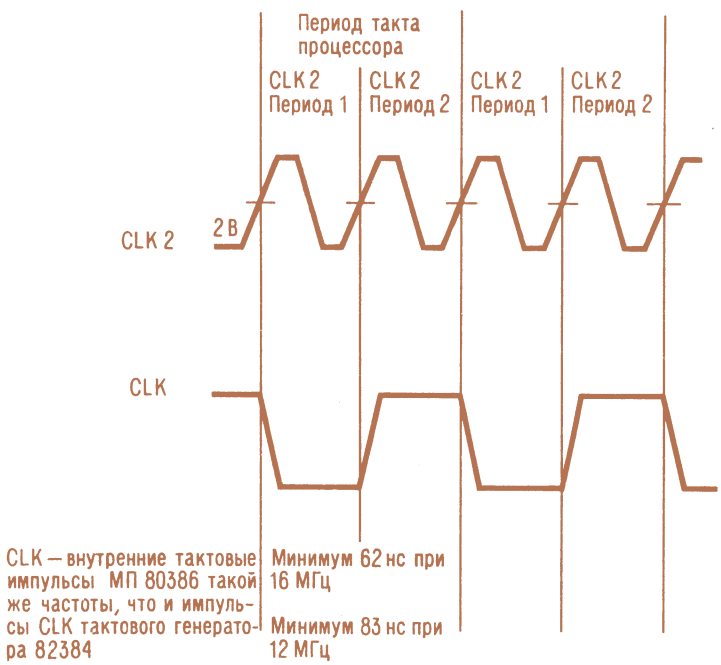


Рис. 3.2. Отношение между сигналами CLK и CLK2.

3.2. Немаскируемые прерывания

Вход NMI обычно используется для сигналов о катастрофических событиях, таких как возможность сбоя по питанию, ошибка памяти или ошибка четности по магистрали. Этот вход является асинхронным с запуском по фронту сигнала (переход от низкого уровня к высокому). Сигнал считается достоверным, если он был низкого уровня в течение восьми периодов CLK2 до перехода и поддерживался высоким в течение восьми периодов CLK2 после него.

Сигнал на входе NMI заставляет МП 80386 автоматически выполнять программу обработки прерывания, соответствующую входу 2 в таблицу IDT. Процессор не будет обслуживать последующие запросы по входу NMI до тех пор, пока не обработан текущий запрос прерывания. В реальном режиме МП игнорирует запросы по входу INTR, хотя немаскируемые прерывания могут быть повторно разрешены программой обработки. В защищенном режиме запрет на запросы по входу INTR зависит от вентиля по входу 2 таблицы IDT.

Немаскируемым прерываниям и опознаваемым процессором программным исключениям назначены определенные номера (векторы) в диапазоне от 0 до 31. Некоторые из них зарезервированы фирмой Intel для использования в будущем.

В случае маскируемых прерываний внешние контроллеры прерываний, такие как 8259A, определяют вектор и направляют его в процессор во время подтверждения прерывания. В ответ на запрос прерывания МП 80386 принимает и затем обрабатывает прерывание. Иными словами, он передает управление программе обработки прерывания. Любое число от 32 до 255 может быть назначено программе обработки прерывания. В табл. 3.1 показаны назначенные и неназначенные векторы прерываний.

Таблица 3.1. Векторы прерываний и исключений

Вектор	Описание
0	Деление на нуль
1	Исключения для отладки
2	Немаскируемое прерывание
3	Точка останова
4	Переполнение
5	Превышение границы индекса массива
6	Неразрешенный код операции
7	Сопроцессор отсутствует
8	Двойной отказ
9	Выход сопроцессора из сегмента
10	Неразрешенный сегмент состояния задачи
11	Сегмент отсутствует
12	Исключение по стеку
13	Общая защита
14	Отказ страницы
15	Зарезервирован
16	Ошибка сопроцессора
17–31	Зарезервированы
32–255	Векторы внешних маскируемых прерываний

3.3. Таблица дескрипторов прерываний

Дескрипторы точек входа в программы обработки прерываний или задачи прерываний хранятся в памяти в таблице, называемой таблицей дескрипторов прерываний (IDT). Таблица IDT связывает каждый вектор прерывания или исключения с дескриптором для команд, которые обрабатывают соответствующее событие.

Для доступа к конкретной программе обработки МП 80386 получает вектор, который является индексом позиции таблицы с дескриптором. Источник этого вектора зависит от типа прерывания. Если прерывание маскируемое (по входу INTR), то этот вектор передается контроллером прерываний 8259A. Если прерывание немаскируемое (по входу NMI), то для входа в таблицу IDT используется вектор 2.

Таблица IDT является массивом 8-байтовых дескрипторов. В отличие от таблиц GDT и LDT первый вход в таблицу может содержать дескриптор. Для правильного обращения к дескриптору процессор умножает значение вектора на восемь. Таблица IDT может содержать до 256 дескрипторов векторов и размещаться в любом месте памяти. Процессор получает к ней доступ через регистр IDTR с помощью команд LIDT и SIDT его загрузки и сохранения.

Команда LIDT загружает из памяти регистр IDTR значениями линейного базового адреса и границы таблицы IDT. Эта команда выполняется только при нулевом текущем уровне привилегий CPL. Обычно она используется во время инициализации операционной системы при создании таблицы IDT. Операционная система может также использовать эту команду для переключения с одной таблицы IDT на другую.

Команда SIDT копирует в ячейки памяти значения базы и границы, хранящиеся в регистре IDTR, и может исполняться при любом уровне привилегий.

3.4. Исключения

Исключения подразделяются на отказы, ловушки и выходы из процесса в зависимости от способа их возникновения и от того, возможен ли рестарт с вызвавшей их команды.

Отказы – это исключения, которые обнаруживаются либо перед исполнением, либо во время исполнения команды. При обнаружении во время исполнения команды сообщение об отказе возникает вместе с переводом машины в состояние, которое позволяет осуществить рестарт с данной команды. Значения регистра сегмента программы CS и указателя команд EIP, сохраняемые при возникновении отказа, указывают на вызвавшую отказ команду.

Ловушка возникает на границе команд сразу же после команды, вызвавшей это исключение. Значения регистров CS и EIP, сохраненные при срабатывании ловушки, указывают на команду, следующую за вызвавшей срабатывание командой. Запоминаемые значения регистров CS и EIP отражают изменения в нормальном порядке выполнения программы, если ловушка сработала на команде, изменяющей этот порядок. Например, если ловушка сработала на команде JMP, то в стеке запоминаются значения регистров CS и EIP, указывающие на ссылку команды JMP, а не на непосредственно следующую за командой JMP команду.

Выход из процесса не позволяет ни осуществить рестарг программы, ни точно локализовать команду, вызвавшую это исключение. Выходы из процесса используются для обнаружения крупных ошибок, таких как неразрешенные или несовместимые значения в системных таблицах или аппаратные сбои.

Время, которое проходит от запроса прерывания до его обработки, зависит от

нескольких факторов. Источник прерывания должен принимать во внимание эту задержку. На задержку влияет следующее:

- если прерывание маскируемое, то запрос по входу INTR не воспринимается до тех пор, пока прерывания не будут разрешены программно;
- если в данный момент обрабатывается немаскируемое прерывание, то запрос очередного немаскируемого прерывания не будет принят до тех пор, пока МП 80386 не исполнит команду возврата из прерывания IRET;
- сохранение регистра флагов и других регистров требует затрат времени;
- если обработка прерывания требует переключения задач, то на сохранение и восстановление состояния задачи также должно быть потрачено время;
- если в данный момент МП 80386 выполняет команду, то это выполнение должно быть завершено. С определенными исключениями (такими как пересылка строки, которая разрешает прерывания после пересылки каждого блока) запрос прерывания опознается только на границах команд.

Наибольшая задержка появляется в том случае, когда запрос прерывания возникает во время исполнения МП 80386 длительной по времени команды, такой как умножение, деление или переключение задачи в защищенном режиме. Если команда устанавливает флаг прерываний, разрешая тем самым прерывания, то прерывание не обрабатывается до конца следующей команды.

3.5. Контроллер прерываний

Программируемый контроллер прерываний 8259A управляет прерываниями в системе на базе МП 80386. Свойство программируемости позволяет использовать его различными способами для удовлетворения требований к прерываниям в конкретной системе. Один контроллер 8259A может воспринимать прерывания максимум от восьми внешних источников, а каскадное включение нескольких контроллеров позволяет обслуживать до 64 запросов прерывания.

Контроллер 8259A разрешает приоритеты между активными прерываниями, затем прерывает выполняемую процессором программу и посылает ему код идентификации источника прерывания.

3.6. Приоритеты прерываний и исключений

Если по завершении выполнения команды имеется более одного запроса прерывания или исключения, то процессор обслуживает их по одному в порядке убывания приоритета. Высшими в списке приоритетов являются отказы (кроме отладочных отказов), команды ловушек (INT0, INTn, INT3), отладочные ловушки для данной команды, отладочные отказы для следующей команды и немаскируемые прерывания. Самый низкий приоритет имеет маскируемое прерывание.

Когда процессор выбирает прерывание для обработки, то любые исключения с меньшим приоритетом задерживаются до достижения их приоритета. Низкоприоритетные исключения обслуживаются после того, как программа обработки прерывания возвратит управление в прерванную точку программы. Они запоминаются и ожидают обработки.

3.7. Задачи и процедуры прерываний

Прерывание или исключение может вызвать программу обработки, которая является процедурой или задачей. Вентиль прерывания или вентиль ловушки

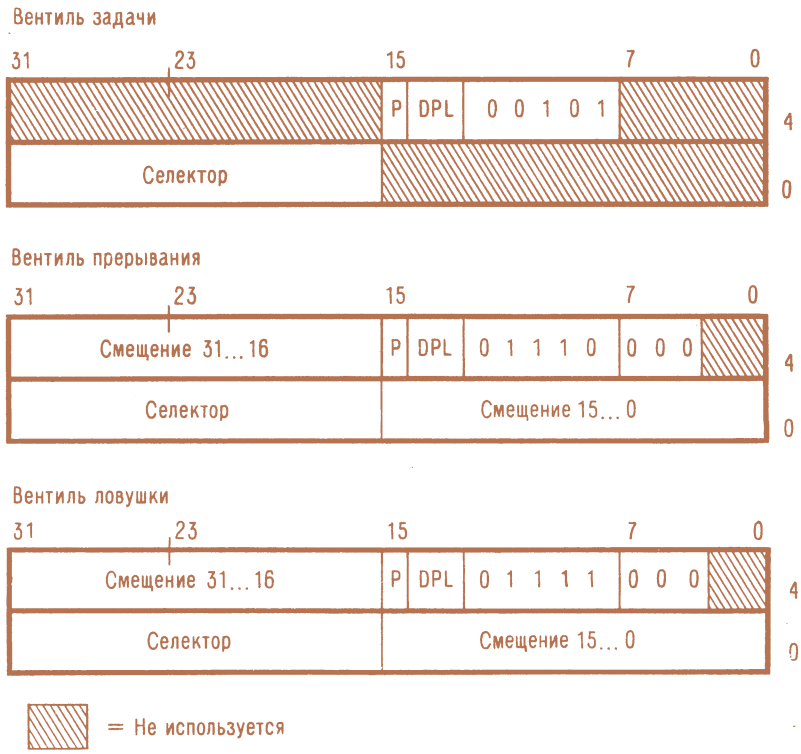


Рис. 3.3. Дескрипторы вентилей таблицы IDT для МП 80386.

косвенно указывает на процедуру, которая выполняется в контексте текущей исполняемой задачи. Селектор вентиля указывает на дескриптор исполнительного сегмента либо в таблице GDT, либо в текущей таблице LDT. Поле смещения вентиля указывает на начало программы обработки прерывания или исключения. Процессор вызывает эту программу обработки таким же образом, как и обычную процедуру (или подпрограмму). Форматы дескрипторов вентилей таблицы IDT показаны на рис. 3.3.

Процедура обработки прерывания отличается от обычной процедуры методом выхода из процедуры. Для выхода из процедуры обработки прерывания применяется команда возврата из прерывания IRET. Эта команда подобна простой команде возврата RET, за исключением того, что команда IRET увеличивает значение указателя стека ESP дополнительно на четыре, поскольку в стеке дополнительно сохраняется значение регистра флагов. Затем команда IRET восстанавливает из стека в регистре EFLAGS сохраненные значения флагов.

Прерывание, векторизующееся через вентиль прерывания, сбрасывает флаг IF разрешения прерываний. Это предотвращает вмешательство других прерываний в работу программы обработки текущего прерывания. Команда IRET в конце программы обработки прерывания устанавливает флаг IF по его значению в образе регистра EFLAGS в стеке. В противоположность этому прерывание, векторизующееся через вентиль ловушки, не изменяет значения флага IF.

Процессор не разрешает прерыванию передавать управление процедуре, расположенной в сегменте с меньшими привилегиями, т. е. с численно бóльшим

значением уровня привилегий, чем текущий уровень. Нарушение этого правила приведет к запуску исключения общей защиты. Поскольку предсказать появление прерывания трудно, то существуют ограничения, накладываемые на уровни привилегий, на которых могут исполняться процедуры обработки прерывания или исключения. Используйте один из следующих методов для того, чтобы быть уверенным в отсутствии нарушения уровней привилегии:

- располагайте процедуру обработки в сегменте с нулевым уровнем привилегий;
- располагайте процедуру обработки в согласованном сегменте. Этот метод подходит программам обработки определенных исключений, таких как ошибка деления на нуль. Подобная программа должна использовать только данные из стека. Если же она нуждается в информации из сегмента данных, то такой сегмент должен иметь уровень привилегий 3, что делает его незащищенным.

3.8. Коды ошибок

Когда процессор обнаруживает исключение, связанное с конкретным сегментом, то он запоминает соответствующий код ошибки в стеке программы обработки. Формат кода ошибки подобен формату селектора, за исключением того, что он имеет два однобитовых поля вместо трехбитового поля RPL. Формат кода ошибки показан на рис. 3.4.

Код ошибки имеет следующие поля:

Индекс селектора – указывает место в таблицах обработки ошибок.

Индикатор таблицы TI – бит 2. Если бит 1 очищен и TI = 0, то этот бит показывает, что код ошибки ссылается на таблицу GDT. Если же бит 1 очищен и TI = 1, то код ошибки ссылается на таблицу LDT.

Бит прерывания I – бит 1. Процессор устанавливает этот бит, если индексная часть кода ошибки ссылается на дескриптор вентиля в таблице IDT.

Бит внешнего события EXT – бит 0. Процессор устанавливает этот бит, если данное исключение было вызвано событием, внешним по отношению к текущей программе.

Микропроцессор 80386 распознает шестнадцать определенных кодов ошибок.

Прерывание 0 – ошибка деления на нуль. Эта ошибка возникает во время исполнения команды беззнакового (DIV) или знакового (IDIV) деления в том случае, если делитель равен нулю.

Прерывание 1 – исключения для отладки. Процессор генерирует это общее прерывание для пяти условий:

1) ловушка точки останова при переключении задачи;
2) отказ общего типа;
3) ловушка точки останова по адресу данных;
4) отказ точки останова по адресу команды;
5) ловушка пошагового (покомандного) исполнения.

Рис. 3.4. Формат кода ошибки.

Будет ли это исключение ловушкой или отказом, зависит от условия. Процессор не запоминает в стеке кода ошибки для этого исключения. Напишите собственную программу обработки, которая проверяет регистры отладки DR0–DR7 для определения конкретного условия, вызвавшего данное исключение.

Прерывание 3 – точка останова. Эта ловушка вызывается командой INT 3 (вызов процедуры прерывания). Команда INT 3 имеет длину в один байт, что позволяет легко заменять код операции в исполняемом сегменте на код операции точки останова. Запоминаемое в стеке содержимое регистров CS и EIP указывает на байт, следующий за точкой останова.

Прерывание 4 – переполнение. Это прерывание запускается в том случае, если процессор выполняет команду INTO (вызов процедуры прерывания по переполнению) и при этом флаг переполнения установлен. Как знаковая, так и беззнаковая арифметика используют одни и те же команды. Процессор не может определить, какая из них имелась в виду в конкретной ситуации. Поэтому он не вызывает исключения по прерыванию автоматически. Он устанавливает флаг в случае выхода результата из диапазона чисел со знаком.

Прерывание 5 – нарушение границы. Этот отказ возникает при исполнении команды BOUND (проверка индекса массива на соответствие границе), если операнд оказывается больше, чем значение границы. Программа может также использовать команду BOUND для проверки индекса массива со знаком на выход за границы со знаком, которые определены в блоке памяти.

Прерывание 6 – неразрешенный код операции. Этот отказ возникает, если в исполнительный блок поступает неразрешенный код операции. Он не обнаруживается до тех пор, пока процессор не попытается исполнить команду с таким кодом операции. Иными словами, предварительная выборка такого кода не вызывает данного исключения. В стеке код ошибки также не запоминается.

Это исключение может вызвать и другая операция. Отказ возникает, если тип операнда не соответствует данному коду операции. Например, прерывание возникает, если команда LES (загрузить полный указатель в регистр ES) использует в качестве операнда-источника регистр.

Прерывание 7 – сопроцессор отсутствует. Это исключение возникает в одной из двух ситуаций:

1) процессор обнаруживает либо команду WAIT, либо команду ESC (сигнал, что это команда сопроцессора), и оба бита MP и TS в регистре CR0 установлены;

2) процессор обнаруживает команду ESC, и бит EM регистра CR0 установлен.

Прерывание 8 – двойной отказ. Обычно МП 80386 обрабатывает исключения последовательно. Если он не может этого сделать, то возникает двойной отказ. Для того чтобы определить, является ли данная ошибка условием двойного отказа, процессор разделяет исключения на три класса: незначительные исключения, сопутствующие исключения и отказы страниц. Код ошибки всегда запоминается в стеке программы обработки двойного отказа. Этот код всегда равен нулю. Рестарт с вызвавшей отказ команды невозможен. Если во время попытки процессора обработать двойной отказ возникает другое исключение, то процессор прекращает работу. Три класса двойных отказов показаны на рис. 3.5.

Прерывание 9 – выход сопроцессора из сегмента. Это исключение возникает, если процессор в защищенном режиме обнаруживает нарушение страницы или сегмента при передаче средней части операнда процессора в NPX.

Прерывание 10 – неразрешенный сегмент состояния задачи. Это исключение вызывает попытка переключиться на неразрешенный сегмент состояния задачи (TSS). Сегмент TSS может считаться неразрешенным при двенадцати условиях, показанных на рис. 3.6. Код ошибки заносится в стек и помогает определить условие, вызвавшее исключение. Бит EXT показывает, было ли это исключение

Класс	Вектор	Описание
Незначительные исключения	1	Исключения для отладки
	2	Немаскируемое прерывание
	3	Точка останова
	4	Переполнение
	5	Превышение границы индекса массива
	6	Неразрешенный код операции
	7	Сопроцессор отсутствует
	16	Ошибка сопроцессора
Сопутствующие исключения	0	Деление на нуль
	9	Выход сопроцессора из сегмента
	10	Неразрешенный сегмент состояния задачи
	11	Сегмент отсутствует
	12	Исключение по стеку
	13	Общая защита
Отказ страницы	14	Отказ страницы

Рис. 3.5. Три класса двойных отказов.

Код ошибки	Условия
TSS id + EXT	Граница в дескрипторе TSS меньше, чем 103
LDT id + EXT	Неразрешенный селектор или отсутствует таблица LDT
SS id + EXT	Селектор сегмента стека вне границы таблицы
SS id + EXT	Сегмент стека не доступен для записи
SS id + EXT	Уровень DPL сегмента стека не соответствует новому уровню CPL
SS id + EXT	Уровень RPL сегмента стека не равен уровню CPL
CS id + EXT	Селектор сегмента программы вне границы таблицы
CS id + EXT	Селектор сегмента программы не ссылается на сегмент программы
CS id + EXT	Уровень DPL несогласованного сегмента программы не равен новому уровню CPL
CS id + EXT	Уровень DPL согласованного сегмента программы больше, чем новый уровень CPL
DS/ES/FS/GS id + EXT	Селектор сегмента DS, ES, FS или GS вне границы таблицы
DS/ES/FS/GS id + EXT	Сегмент DS, ES, FS или DS не доступен для чтения

Рис. 3.6. Условия неразрешенного сегмента TSS.

вызвано внешним по отношению к программе условием, таким как внешнее прерывание через вентиль задачи, который осуществил переключение на неразрешенный сегмент TSS. Для того чтобы обеспечить правильный сегмент TSS для обработки этого условия, программа обработки исключения должна вызываться через вентиль задачи.

Прерывание 11 – сегмент отсутствует. Это исключение появляется, когда МП 80386 обнаруживает, что бит *P* дескриптора очищен, после этого отказа возможен рестарт. Программа обработки исключения делает сегмент присутствующим и осуществляет возврат. Затем возобновляется выполнение прерванной программы. Обычно операционная система использует это исключение для реализации виртуальной памяти на уровне сегментов.

Прерывание 12 – исключение по стеку. Это прерывание появляется при выполнении одного из двух условий:

1) когда процессор пытается загрузить регистр SS дескриптором, который помечен как отсутствующий, а в остальном является допустимым. Такая ситуация может возникнуть во внутреннем уровне вызова или возврата при переключении задачи, при исполнении команд LSS или MOV и POP со ссылкой на регистр SS;

2) при любой операции со ссылкой на регистр SS и нарушением границы. Эти операции включают стек-ориентированные команды, такие, как ENTER, LEAVE, POP, PUSH. Сюда также относятся другие обращения к памяти, предусматривающие использование сегмента SS.

Во всех случаях возможен рестарт с вызвавшей это прерывание команды, на которую указывает указатель возврата, запоминаемый в стеке программы обработки этого исключения.

Прерывание 13 – исключение общей защиты. Это исключение является «всеядным». Все те нарушения защиты, которые не вызывают другого исключения, запускают исключение общей защиты. Причины его следующие:

1) нарушение границы сегмента при ссылке на таблицу дескрипторов;

2) запись в доступный только чтению сегмент данных или в сегмент программы;

3) считывание из доступного только по исполнению сегмента;

4) загрузка регистров DS, ES, FS, GS или SS дескриптором системного сегмента;

5) переключение на занятую задачу;

6) нарушение правил привилегий;

7) загрузка регистра CR0 битом PG = 1 (включено разбиение на страницы) и битом PE = 0 (защита не включена);

8) нарушение границы сегмента при использовании регистров CS, DS, ES, FS, GS;

9) передача управления сегменту, который не является исполнительным.

Прерывание 14 – отказ страницы. Это прерывание возникает, когда осуществляется разбиение на страницы (PG = 1) и процессор обнаруживает при трансляции линейного адреса в физический адрес одно из следующих условий:

1) текущая процедура не имеет привилегий, достаточных для доступа к указанной странице памяти;

2) вход в таблицу страниц или в каталог страниц, который нужен для трансляции адресов, имеет сброшенный бит присутствия.

Для программы обработки отказа страницы МП 80386 делает доступными два вида информации с целью облегчения диагностики ошибки и ее возможной коррекции.

1) Регистр управления CR2. Процессор сохраняет линейный адрес, использованный при вызвавшем это исключение доступе, в регистре CR2. Программа обработки исключения использует этот регистр для локализации соответствующих входов в таблицу страниц и в каталог страниц.

2) Код ошибки в стеке. Этот код имеет формат, отличающийся от формата для других исключений. Он сообщает, во-первых, обусловлено ли исключение отсутствием страницы или же нарушением прав доступа, во-вторых, был ли вызвавший исключение доступ к памяти чтением или записью и, в-третьих, находился ли МП 80386 в момент исключения на уровне пользователя или супервизора.

Прерывание 15 – зарезервировано фирмой Intel для использования в будущем.

Прерывание 16 – ошибка сопроцессора. Процессор вызывает это прерывание, если он обнаруживает сигнал от арифметического сопроцессора 80287 или 80387 на выводе ERROR#. Процессор проверяет этот вывод дважды: в начале определенных команд ESC и когда он обнаруживает команду WAIT при нулевом бите EM в слове состояния машины MSW (т.е. эмуляция сопроцессора не нужна). На рис. 3.7 приведена сводная информация об ошибках для каждого из исключений.

Вектор	Описание	Возврат к команде, вызвавшей отказ	Вызывается командой
0	Деление на нуль	○	DIV, IDIV
1	Исключения для отладки	○	Любой
3	Точка останова	×	INT 3 (однобайтовая)
4	Переполнение	×	INTO
5	Превышение границы	○	BOUND
6	Неразрешенный код операции	○	Любым из неразрешенных кодов
7	Сопроцессор отсутствует	○	ESC, WAIT
8	Двойной отказ	○	Любой, вызывающей исключение
9	Выход сопроцессора из сегмента	×	Любым операндом ESC, выходящим за сегмент
10	Неразрешенный сегмент состояния задачи	○	JMP, CALL, IRET или любым прерыванием
11	Сегмент отсутствует	○	Любым изменением регистра сегмента
12	Исключение по стеку	○	Любым доступом к памяти через регистр SS
13	Общая защита	○	Любым доступом к памяти или выборкой команды
14	Отказ страницы	○	Любым доступом к памяти или выборкой команды
16	Ошибка сопроцессора	○	ESC, WAIT
0—255	Двухбайтовое программное прерывание	×	INT n (двухбайтовая)

○ — выполняется; × — не выполняется.

Рис. 3.7. Сводная информация по исключениям.

3.8.1. Коды ошибок сопроцессора

Процессор может работать с арифметическими сопроцессорами 80287 и 80387. Бит ET в регистре CR0 показывает, какой именно из них присутствует. Значение этого бита устанавливается во время системного сброса RESET по уровню на входе ERROR#. При необходимости этот бит может быть установлен или сброшен загрузкой регистра CR0 командой MOV. Если бит ET = 1, то МП 80386 использует 32-разрядный протокол обмена с сопроцессором 80387, иначе он использует 16-разрядный протокол обмена с сопроцессором 80287.

Процессор идентифицирует код 110011 в первых пяти битах команды как код операции, предназначенный для сопроцессора. Команды с таким кодом называются командами ESCAPE, или ESC. Когда МП 80386 обнаруживает код ESC, то он делает следующее: проверяет флаг EM (режим эмуляции) для того, чтобы определить, нужно ли эмулировать функции сопроцессора программным способом, проверяет флаг TS для того, чтобы определить, имело ли место контекстное изменение с момента выполнения последней команды ESC, и для некоторых таких команд проверяет вывод ERROR# для того, чтобы определить, обнаружил ли сопроцессор ошибку в предыдущей команде ESC.

Биты EM и MP регистра CR0 управляют реакцией МП 80386 на команды сопроцессора. Бит EM показывает, нужно ли эмулировать функции сопроцессора. Если МП 80386 обнаруживает, что бит EM установлен при выполнении команды ESC, то возникает исключение 7, которое дает возможность программе обработки этого исключения эмулировать команду ESC. Бит MP показывает, подключен ли к магистрали сопроцессор, и управляет функцией команды WAIT. Если МП 80386 обнаруживает, что бит MP установлен во время выполнения команды WAIT, то он проверяет флаг TS. Если флаг TS установлен, возникает исключение 7.

Значения битов EM и MP могут быть изменены с помощью команды MOV, использующей в качестве приемника регистр CR0. Они могут быть считаны командой MOV, использующей регистр CR0 в качестве операнда-источника. Эти формы команды MOV могут выполняться только на нулевом уровне привилегий.

Бит TS в регистре CR0 помогает определить, соответствует ли контекст сопроцессора текущей задаче. Процессор устанавливает бит TS каждый раз при переключении задачи, вызванном либо программным, либо аппаратным способом. Если МП 80386 обнаруживает, что бит TS уже установлен при интерпретации одной из команд ESC, то он запускает исключение 7. Команда WAIT также вызывает исключение 7, если установлены оба бита TS и MP. Разрешенная только на нулевом уровне привилегий команда CLTS сбрасывает флаг TS.

К сопроцессору имеют отношение три исключения: прерывание 7 (сопроцессор отсутствует), прерывание 9 (выход сопроцессора из сегмента) и прерывание 16 (ошибка сопроцессора). Эти три прерывания включены в общие прерывания (см. выше). Здесь они рассматриваются несколько подробнее.

Прерывание 7 – сопроцессор отсутствует. Это исключение возникает в одном из двух случаев:

1) процессор обнаруживает либо команду WAIT, либо команду ESC, когда оба бита MP и TS установлены. Если это произошло, то программе обработки исключения следует при необходимости обновить состояние сопроцессора;

2) процессор обнаруживает команду ESC при установленном флаге EM эмуляции функций сопроцессора. В этом случае программе обработки следует эмулировать вызвавшую это исключение команду. Бит TS также может быть установлен.

Прерывание 9 – выход сопроцессора за пределы сегмента. Это исключение возникает в защищенном режиме в следующих случаях:

1) как первый, так и последний байты операнда (при рассмотрении оборачивания адресов[1]) расположены по адресам сегмента в присутствующих и доступных страницах и адрес оборачивается;

2) когда операнд команды сопроцессора оборачивается вокруг адресной границы: 0FFFFH – для малых сегментов, 0FFFFFFFFH – для больших сегментов и нуль – для расширяющихся вниз сегментов. Операнд может оборачиваться вокруг адресной границы, если граница сегмента находится около адресной границы и операнд расположен по адресу вблизи максимально разрешенного адреса в сегменте. Начальный и конечный адреса такого операнда будут находиться вблизи противоположных концов сегмента благодаря оборачиванию адреса;

3) когда операнд находится по адресу, доступ к которому невозможен.

Адрес арифметической команды, инициировавшей отказ, и операнд-данные могут быть утеряны. Команда FSTENV не возвращает достоверных адресов. Выход сопроцессора из сегмента следует обрабатывать исполнением команды FNINIT, что эквивалентно команде FINIT без предшествующей команды WAIT. Адрес возврата в стеке не обязательно указывает на команду, вызвавшую отказ, или на следующую за ней команду. Рестарт с такой команды невозможен.

Прерывание 16 – ошибка сопроцессора. При выполнении команд сопроцессоры 80287 и 80387 обнаруживают шесть различных условий исключения. Если это исключение не маскировано соответствующим битом в управляющем слове, сопроцессор посылает сигнал процессору о возникновении ошибки через вывод ERROR#. Процессор вызывает прерывание 16 при следующей проверке вывода ERROR#, которую он осуществляет только для определенных команд ESC и в начале последующей команды WAIT. Если исключение маскировано, то сопроцессор обрабатывает это исключение с помощью расположенной на его плате аппаратной логики. В этом случае он не посылает сигнал процессору по выводу ERROR#.

---

[1] Двоичный счетчик-регистр, содержащий максимальное для его формата число, при очередном инкрементировании начинает счет с нулевого значения. Если этот регистр используется как указатель адреса, то в таком случае говорят, что адрес оборачивается – adress wrap around. – *Прим. перев.*

Глава 4

Память

Память является основным компонентом компьютерной системы; в ней хранится информация для использования в процессе работы ЭВМ. Запоминающее устройство должно не только хранить большой объем информации, его организация должна обеспечивать быстрый доступ к любой конкретной части этой информации. Быстродействие, объем и стоимость являются решающими критериями для любого запоминающего устройства. Эта глава расширяет сведения о МП 80386 и описывает организацию памяти, интерфейс с памятью и кэш-памятью.

4.1. Организация памяти

Внутренняя память физически составляет единое целое с компьютером и управляется процессором (с некоторой помощью средств прямого доступа к памяти). Организационные методы распределения памяти имеют своей целью предоставить программистам возможности эффективного использования всей компьютерной системы. Ниже рассматриваются две общие организационные модели – сплошная и сегментированная.

4.1.1. Сегментация

Сегментация – это разделение памяти на логические блоки с целью эффективного управления пространством логических адресов. Сегменты используются для объединения областей памяти, которые имеют общие атрибуты. Иначе говоря, память организуется как один или более сегментов переменной длины, размером от одного байта до четырех гигабайт. Каждая задача в МП 80386 может иметь до 16383 сегментов, размером до четырех гигабайт каждый, таким образом, общий объем виртуальной памяти составляет 64 Тбайт. Любая из этих областей линейного адресного пространства (сегмент физической памяти) имеет несколько связанных с ней атрибутов. Они включают ее размер, расположение, тип (стек, программа или данные) и характеристики защиты.

Видимая программистами модель организации памяти определяется при проектировании системы. Архитектура МП 80386 дает проектировщикам свободу выбора модели для каждой задачи. Наиболее распространенными моделями являются следующие.

Сплошная память: адресное пространство, состоящее из одного массива длиной до 4 Гбайт, которое процессор с помощью механизма трансляции адресов отображает в пространство физических адресов. Указатель является 32-разрядным числом, значение которого может изменяться от 0 до $2^{32} - 1$. Перемещение модулей, скомпилированных по отдельности, может быть выполнено системным программным обеспечением, таким как загрузчики, редакторы связей и т. п.

Сегментированная память: адресное пространство, состоящее из набора до 16383 линейных адресных пространств, размером до 4 Гбайт каждое. Полное пространство, видимое программой, может быть до 2^{46} байт (64 Тбайт). С помощью механизма трансляции адресов процессор отображает пространство логических адресов в пространство физических адресов.

Программисты воспринимают сегментированную модель (называемую также пространством логических адресов) как набор до 16383 одномерных подпространств определенной длины. Каждое из этих подпространств – это сегмент, блок сплошного адресного пространства. Полный указатель в этом адресном пространстве состоит из двух частей: селектора сегмента 16-разрядного поля, идентифицирующего сегмент, и относительного 32-разрядного адреса, указывающего на байт внутри сегмента.

4.1.2. Адресное пространство

Физическая память организована в виде последовательности 8-разрядных байтов. Каждому байту присвоен уникальный адрес, который может изменяться от 0 до $2^{32} - 1$, или 4 Гбайт. Процессор 80386 имеет три различных адресных пространства: физическое, логическое и линейное.

Физические адреса – это реальные адреса, используемые для выбора микросхем физической памяти, содержащих данные. Логический адрес состоит из селектора сегмента и относительного адреса внутри сегмента. Линейный адрес – это адрес, сформированный добавлением относительного адреса к базовому адресу сегмента. С точки зрения программы в модели сегментированной памяти пространство логических адресов является наибольшим и может содержать до 2^{46} байт, или 64 Тбайт. С помощью механизма трансляции адресов МП 80386 отображает логическое пространство в физическое.

Процессор 80386 преобразует логические адреса в физические в два этапа: сначала он выполняет трансляцию сегмента, преобразуя логический адрес, состоящий из селектора сегмента и относительного адреса, в линейный, а затем – трансляцию страниц, преобразуя линейный адрес в физический. Второй этап не обязателен и зависит от того, как была спроектирована система.

4.1.3. Разбиение на страницы и их трансляция

Разбиение на страницы – это другой способ управления памятью, удобный для многозадачных операционных систем. Разбиение на страницы возможно только в защищенном режиме и обеспечивает управление очень большими сегментами МП 80386. С помощью этого способа программы разделяются на одинаковые по размеру страницы в отличие от сегментации, которая представляет программы и данные в виде сегментов переменной длины. Фактически разбиение на страницы – это способ более низкого уровня, чем сегментация. Это означает, что механизм разбиения на страницы транслирует защищенный линейный адрес, который поступает от блока сегментации в физический адрес. Механизм разбиения на страницы МП 80386 иллюстрируется рис. 4.1.

Во второй фазе трансляции адреса МП 80386 преобразует линейный адрес в физический адрес, определяя таблицу страниц, страницу внутри этой таблицы и относительный адрес внутри данной страницы. Этап трансляции страниц не является обязательным. Трансляция страниц возможна, только когда установлен бит PG регистра CR0. Этот бит обычно устанавливается операционной системой во время программной инициализации. Он должен быть установлен, если операционная система предполагает использовать множественные задачи виртуального МП

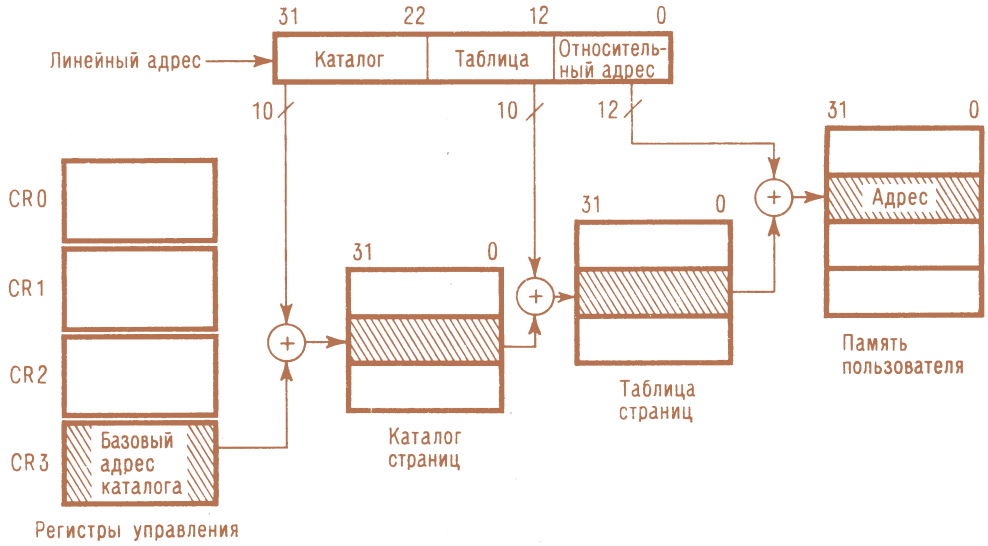


Рис. 4.1. Механизм разбиения памяти на страницы.

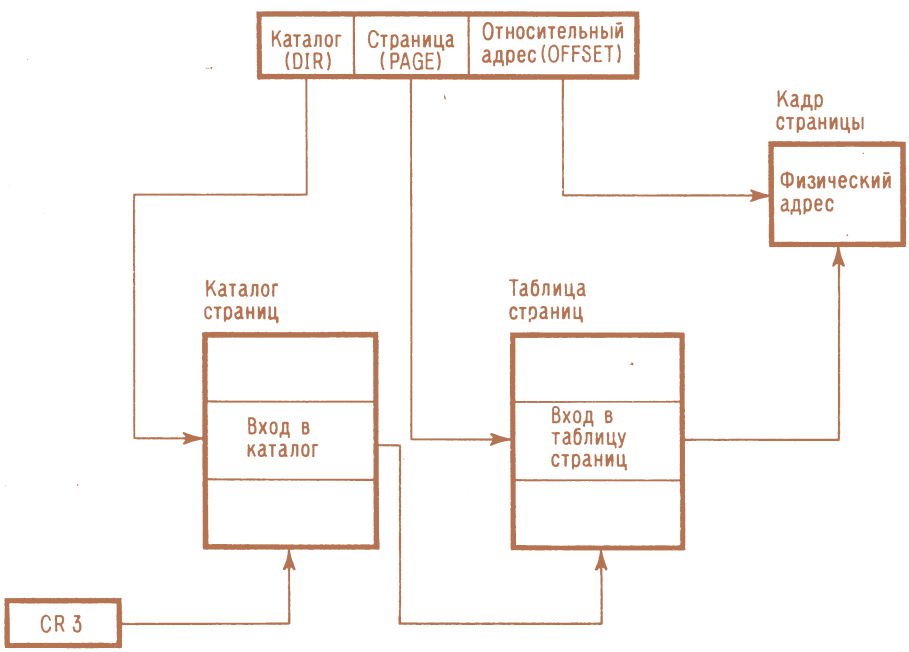


Рис. 4.2. Трансляция страницы.

8086, странично-ориентированную виртуальную память или странично-ориентированную защиту.

Линейный адрес состоит из трех полей: DIR (*каталог*), PAGE (*страница*) и OFFSET (*относительный адрес*). На рис. 4.2 показано, как МП 80386 преобразует линейный адрес в физический. Механизм адресации использует поле DIR в качестве индекса в каталоге страниц, а поле OFFSET для указания байта внутри страницы, определяемой таблицей страниц.

Кадр страницы – это блок в 4 Кбайт физической памяти. Страницы начинаются на границах 4-Кбайт областей памяти и фиксированы по размеру. Страницы не имеют непосредственного отношения к логической структуре программы. Адрес кадра страницы указывает физический начальный адрес страницы и его младшие 12 битов всегда равны нулю. В каталоге страниц адрес кадра страницы – это адрес таблицы страниц. В таблице второго уровня адрес кадра страницы является адресом кадра той страницы, которая содержит нужный операнд в памяти.

Таблицы страниц. Это массив 32-разрядных спецификаторов страниц. Таблица также является страницей и занимает 4 Кбайт памяти или до 1 К 32-разрядных входов. Формат обращения к таблице страниц (входа) показан на рис. 4.3.

Этот формат имеет следующие поля:

Бит мусора D(6) и бит доступа A(5). Эти биты содержат информацию об использовании страницы и устанавливаются аппаратно. Процессор устанавливает биты доступа в таблицах страниц обоих уровней перед операцией чтения из или записи в страницу. Затем процессор устанавливает бит мусора в таблице страниц второго уровня перед операцией записи по адресу, соответствующему входу в эту таблицу. Бит мусора входов каталога страниц является неопределенным.

В операционной системе, которая поддерживает страничную виртуальную память, эти биты используются для определения тех страниц, которые могут быть удалены из физической памяти в случае, когда требуется больший объем, чем размер доступной в данный момент физической памяти. Ответственность за проверку и очистку этих двух битов лежит на операционной системе.

Бит пользователя/супервизора U/S (2) и бит чтения/записи R/W (1). Используются для защиты на уровне страниц, которую МП 80386 выполняет в то же самое время, что и трансляцию адреса.

Бит присутствия P, бит 0. Этот бит показывает, может ли вход в таблицу страниц использоваться при трансляции адреса. Если может, то P = 1. Если же P = 0 в таблице страниц любого уровня, то этот вход нельзя использовать для трансляции адреса и остальные поля входа доступны для использования программой.

Если P = 0 в таблице страниц любого уровня и сделана попытка использовать вход в таблицы страниц для трансляции адреса, то МП 80386 вызывает исключение по странице.

Трансляция страниц и комбинации сегментов. Если МП 80386 используется для выполнения программ, спроектированных для специальных архитектур, не имеющих сегментов, то может оказаться целесообразным выключить сегментацию. Процессор не имеет специального режима для подавления сегментации. Однако тот же самый эффект достигается путем начальной загрузки регистров сегментов

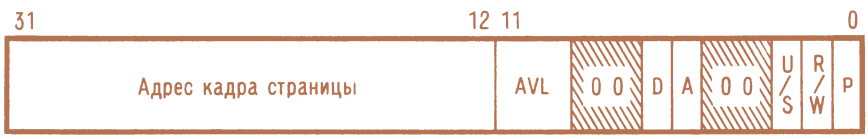


Рис. 4.3. Формат входа в таблицу страниц.

селекторами дескрипторов, которые покрывают полное 32-разрядное пространство линейных адресов. После загрузки таких дескрипторов регистры сегментов не изменяются. Относительные 32-разрядные адреса команд МП 80386 адресуют все линейное адресное пространство.

Сегменты могут быть больше или меньше, чем страница размером в 4 Кбайт. Если требуемая структура данных больше, чем страница, и операционная система поддерживает страничную виртуальную память, то операционная система разделяет эту структуру на страницы, любое количество которых может присутствовать в любой данный момент времени. Это прозрачно для прикладного программиста. Если требуемая информация меньше, чем страница, то операционная система может быть спроектирована так, чтобы комбинировать несколько структур данных внутри одной страницы.

Архитектура МП 80386 допускает несоблюдение прямого соответствия между границами страниц и сегментов. Таким образом, сегмент может содержать конец одной страницы и начало другой. Аналогично страница может содержать конец одного сегмента и начало другого. Программы управления памятью, однако, можно сделать проще, если использовать некоторую взаимосвязь между границами страниц и сегментов. Например, логика для резервирования страниц и сегментов может быть комбинированной, если сегменты резервируются только целыми страницами. В этом случае логика для учета частично используемых страниц не нужна.

4.1.4. Контроллер прямого доступа к памяти

Контроллер прямого доступа к памяти (ПДП) служит для передачи данных между основной памятью и устройствами ввода-вывода, обычно жестким диском, гибким диском или каналом связи. При передаче с ПДП большой блок данных может быть скопирован из одного места в другое без вмешательства ЦП.

Усовершенствованный контроллер ПДП типа 82258 имеет четыре канала и вырабатывает все сигналы, необходимые для передач с ПДП. Особенности контроллера ПДП:

- один из высокоскоростных каналов может быть заменен на 32 низкоскоростных мультиплексных канала;
- возможность соединения команд в цепочку для исполнения сложных команд;
- возможность соединения данных в цепочку для распределения данных по отдельным местам памяти – например, отдельным страницам – и для объединения данных из отдельных мест памяти;
- выполнение функций сравнения, трансляции и верификации;
- автоматическая сборка и разборка для преобразования данных 16-разрядной памяти, передаваемых в 8-разрядное устройство ввода-вывода, и наоборот.

4.2. Интерфейс с памятью

Для достижения потенциальной производительности МП 80386 система должна использовать относительно быстродействующую память. Однако чем выше быстродействие памяти, тем дороже она стоит. Задача улучшения технико-экономических показателей может быть решена путем разделения функций и комбинирования быстрой и медленной памяти. Путем возложения наиболее часто используемых функций на быструю память, а всех других – на медленную память высокая производительность для большинства операций может быть достигнута за значительно меньшую стоимость, чем подсистемы, целиком состоящей из быстрой памяти.

Основная память является главной частью микрокомпьютера и используется для хранения команд программы и данных. Обычно основная память имеет произвольный доступ и представляет собой совокупность слов, каждое из которых состоит из нескольких битов. Каждому слову присваивается адрес для возможности независимого доступа к нему.

4.2.1. Интерфейс с основной памятью

Построение системы памяти из отдельных блоков можно представить как заполнение пространства памяти. Это пространство можно рассматривать как прямоугольную область, которая разделена на ряды, соответствующие возможным адресным комбинациям. Каждый ряд состоит из нескольких битов, образующих слово данных. После размещения блоков памяти в пространстве остается определить необходимые взаимосвязи между ними. Каждая из линий данных блока памяти подключается к соответствующей линии шины данных процессора. Линии адреса используются сначала для выбора блока памяти или параллельной группы блоков, а затем для определения слова внутри выбранного блока или блоков. Структурная схема на рис. 4.4 показывает, какие сигналы управления формирует логика управле-

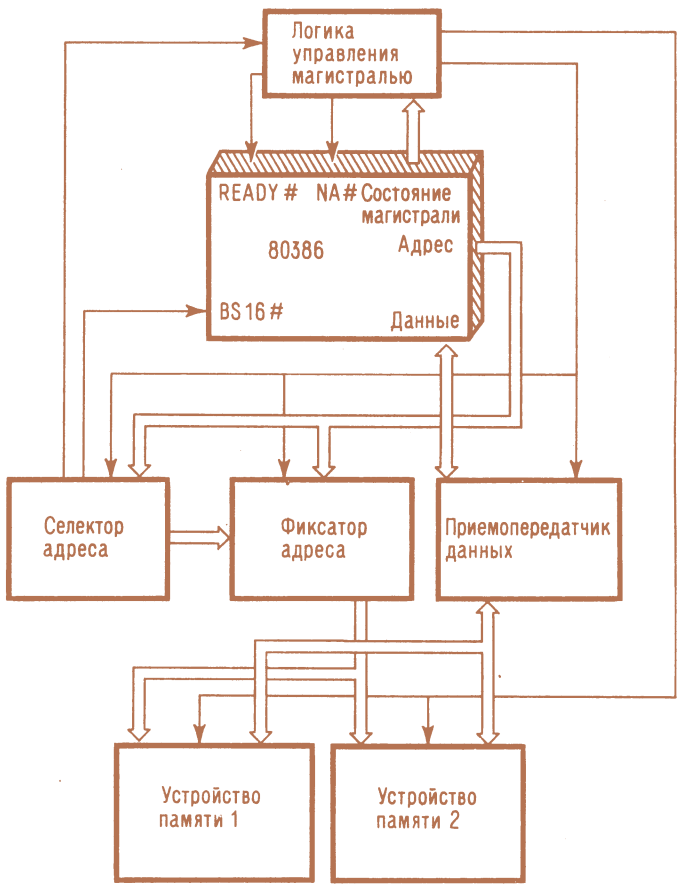


Рис. 4.4. Структурная схема основного интерфейса с памятью.

ния магистралью для буферов данных, фиксаторов адреса памяти и устройств памяти. Она также формирует активный сигнал READY# для завершения цикла магистрали МП 80386 и сигнал NA# для управления конвейеризацией адреса. На основе адресных выходов МП 80386 селектор адреса генерирует сигналы выбора микросхем памяти и сигналы BS16#.

4.2.2. Интерфейс со статическим ОЗУ

Статическое ОЗУ (СОЗУ) сохраняет информацию до выключения питания. Элемент СОЗУ состоит из двух транзисторных каскадов с перекрестными связями, которые находятся в одном из двух стабильных состояний – включено или выключено.

Две дополнительные линии данных в каждом столбце микросхемы памяти доставляют информацию к элементу или от него в выбранной строке. Элементы памяти в каждом столбце связаны с линиями данных нормально запертыми *n*-канальными МОП-транзисторами. Затворы транзисторов каждой строки подключены к соответствующей линии выбора строки. Линии выбора строки являются выходными линиями селектора адреса, поэтому отпираются связывающие транзисторы только в выбранной строке (как определено селектором адреса). Таким образом, информация канализуется между линиями данных и выбранным элементом в каждом столбце.

4.2.3. Интерфейс с динамическим ОЗУ

Подсистемы памяти с динамическим ОЗУ (ДОЗУ) обеспечивают относительно малые времена доступа при низкой стоимости бита. В то же время ДОЗУ требует короткого нерабочего времени между циклами обращения для подзаряда или регенерации данных. Регенерация осуществляется путем выполнения операции чтения и/или записи. Каждая операция регенерации восстанавливает содержимое нескольких слов в блоке памяти. Если это время не отводится, то данные, хранящиеся в ДОЗУ, могут быть утеряны. В случае когда требуются два последовательных обращения к одному и тому же участку ДОЗУ, второе обращение задерживается на время подразряда. Чтобы избежать этой задержки, каждое последующее обращение к памяти должно, по возможности, направляться в различные участки ДОЗУ.

В большинстве ДОЗУ периодическая активация каждого сигнала адреса строки внутренне регенерирует данные для каждого столбца строки. Обычно в ДОЗУ цикл регенерации осуществляется только при помощи сигнала RAS (выбор адреса строки). Временная диаграмма этого цикла такая же, что и у цикла чтения, разница лишь в том, что вместо сигнала CAS (выбор адреса столбца) активен сигнал RAS и все выводы данных находятся в высокоимпедансном состоянии.

Частота регенерации и количество регенерируемых строк зависят от типа ДОЗУ. Для ДОЗУ с организацией 64 K × N и более в цикле регенерации должны использоваться только восемь младших мультиплексированных адресных битов (A0–A7, 256 строк). Старшие биты адреса игнорируются. ДОЗУ большей емкости обычно требуют регенерации каждые 4 мс. После инициализации системы поддерживаются целостность данных ДОЗУ и их состояние (даже во время состояния останова МП 80386 или аппаратного системного сброса), так как все функции системы ДОЗУ выполняются аппаратурой, внешней по отношению к процессору.

Некоторые ДОЗУ требуют нескольких циклов «разогрева» перед тем, как они смогут работать нормально. Обычно это осуществляется двумя способами, во-первых, с помощью внешней логики активируется сигнал RFRQ на определенный

период времени, что инициирует работу аппаратуры управления ДОЗУ в течение нескольких циклов регенерации; во-вторых, как часть процесса инициализации МП 80386 выполняется несколько фиктивных обращений к ДОЗУ, для чего производится установка регистров МП 80386 и выполнение команды REP LODS.

4.3. Кэш-память в МП 80386

Кэш-память располагают между основной памятью и процессором для улучшения эффективной скорости взаимодействия с памятью и увеличения быстродействия процессора. Название «кэш» (в переводе с английского – тайник) говорит о том, что данный механизм прозрачен для пользователя. Обычно кэш-память реализуют на полупроводниковых устройствах, быстродействие которых сравнимо с быстродействием процессора, тогда как основная память использует более дешевые и медленно действующие устройства.

Концепция кэш-памяти основана на предвосхищении наиболее вероятного использования процессором данных из основного устройства памяти путем копирования их в кэш-памяти. Эта концепция распространяется на данные, соседние с текущими данными. Для этого обычно осуществляется передача из основной памяти в кэш-память блока из нескольких слов, даже если в данный момент требуется только одно слово. Если требуемое слово является частью потока последовательных команд, то выборка последующих команд производится на этом первом этапе работы, поскольку благодаря этому отпадает необходимость повторных обращений к основной памяти (рис. 4.5).

Все данные хранятся в основной памяти. С помощью системы с кэш-памятью некоторые из этих данных копируются в кэш-память. Когда процессору нужно считать или записать данные в память, то сначала он проверяет их наличие в кэш-памяти. Если необходимые ему данные находятся там (кэш-попадание), то МП 80386 может легко и быстро их использовать, так как кэш-память реализуется на самых быстродействующих элементах. В противном случае (кэш-промах) эти данные извлекаются из основной памяти и переписываются в кэш-память.

Эффективность кэш-попаданий выражается отношением успешных обращений к общему количеству обращений к кэш-памяти. На это отношение влияют размер, физическая организация и алгоритм кэш-памяти, а также выполняемая программа. Эффективной кэш-память считается в том случае, если ее организация увеличивает процент кэш-попаданий.

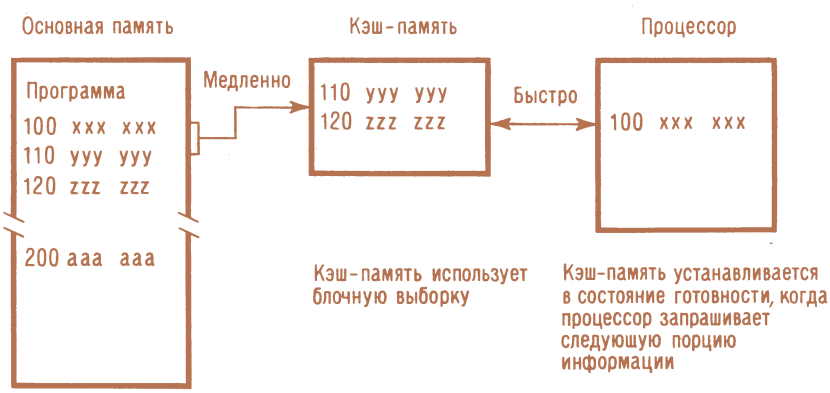


Рис. 4.5. Использование кэш-памяти для просмотра вперед.

Если кэш-память организована так, что часто необходимые процессору команды и данные уже находятся в ней, то это значительно уменьшает среднее время доступа к памяти. Программы выполняются быстрее, если большинство операций – это передачи в наиболее быструю кэш-память и из нее.

Большие компьютеры используют кэш-память уже давно. Появление быстрых 32-разрядных процессоров вызвало необходимость в таком средстве и в области малых машин. Микропроцессор 80386 работает с внешней кэш-памятью. Описываемая здесь кэш-память состоит из 10 микросхем. Кэш-память физического адреса не требует накладных расходов в отличие от кэширования логического адреса, реализованного процессором с расположенной на его плате кэш-памятью. Целью применения кэш-памяти является поддержка быстродействия от 3,5 до 4 миллионов команд в секунду, обеспечиваемого следующим: 32-Мбайт шина, двухтактовые циклы магистрали, МП 80386 с тактовой частотой 16 МГц, арифметический сопроцессор 80387 и 32-Кбайт кэш-память физического адреса для команд и данных (рис. 4.6).

Предсказание адреса следующего обращения в память было бы почти невозможно, если бы программы обращались к памяти абсолютно произвольным образом. На самом же деле программы имеют тенденцию обращаться к памяти по адресу, близкому к адресу предыдущего обращения. Этот принцип называется локальностью программы или локальностью ссылок. Локальность программы и есть то, что делает систему с кэш-памятью эффективной. Например, загрузка и выгрузка стека осуществляется с одного конца, так что несколько соседних обращений делаются в окрестностях вершины стека.

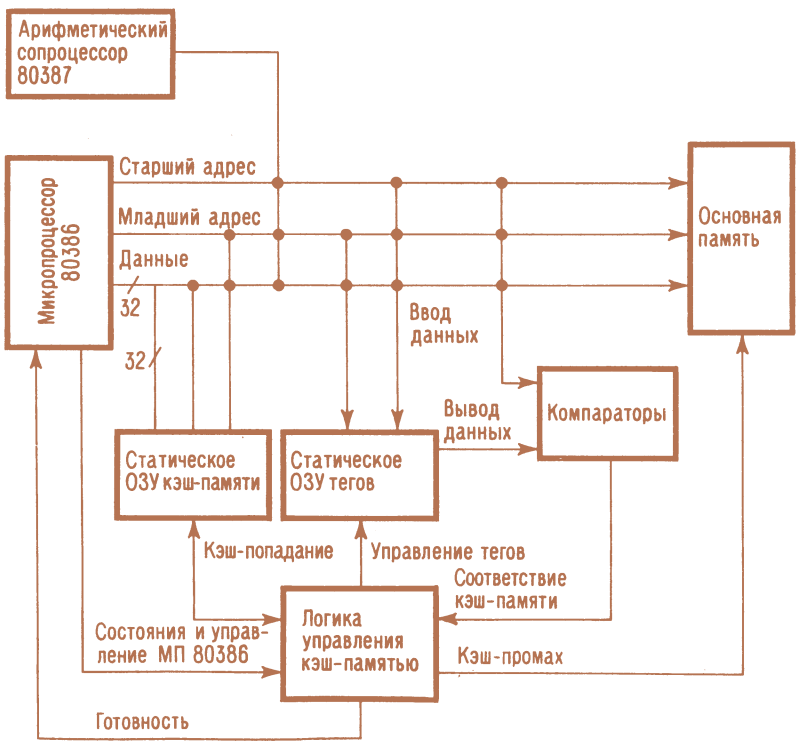


Рис. 4.6. Структурная схема МП 80386 и кэш-памяти.

Для увеличения процента кэш-попаданий процессор использует блочную выборку. Кэш-контроллер разбивает основную память на блоки (известные также как длина строки) длиной обычно в 2, 4, 8 или 16 байт. Размер блока является одним из наиболее важных параметров при проектировании системы с кэш-памятью. Если блок слишком мал, то уменьшается эффективность выборки из основной памяти и процент кэш-попаданий. Слишком большой блок уменьшает количество блоков, помещающихся в кэш-памяти. При таком подходе требуется более широкая разрядность шины между основной и кэш-памятью, а также больший объем статической и динамической памяти, что приводит к увеличению стоимости.

Обычно 32-разрядный процессор использует два или четыре слова на блок. В случае кэш-промаха кэш-контроллер пересылает весь содержащий нужное слово блок из основной памяти в кэш-память. Блочная выборка может принести данные, расположенные перед требуемым байтом (просмотр назад), данные, расположенные после него (просмотр вперед), или же как те, так и другие. Когда к адресам обращаются в возрастающем порядке (например, при выборке команд), то доступ к первому байту блока приводит к выборке с просмотром вперед. Когда к адресам обращаются в убывающем порядке (подобно загрузке в стек), то осуществляется просмотр назад.

4.3.1. Организация кэш-памяти

В большинстве программ есть обращения к сегментам программ, подпрограммам, спискам, буферам, расположенным в различных местах адресного пространства, и к различным стекам. Эффективная кэш-память должна поддерживать блоки данных, не являющиеся непрерывными.

Полностью ассоциативная кэш-память. Полностью ассоциативная кэш-память определяет, какие блоки хранятся в ней в любой момент времени. В такой кэш-памяти не существует однозначного соответствия между всеми адресами блоков, и поэтому в ней должны храниться полный адрес каждого блока так же, как и сам блок. Когда процессору нужны данные из основной памяти, то кэш-контроллер сравнивает адреса этих данных с каждым из адресов в кэш-памяти. Это относительно медленно и дорого и нарушает основной принцип ограничения количества требуемых сравнений адресов.

Кэш-память с прямым отображением. Каждый адрес в такой памяти имеет две части. Первая часть (поле кэш-индекса) содержит достаточно битов для того, чтобы определить место блока внутри кэш-памяти. Вторая часть (поле тега) содержит достаточно битов для того, чтобы отличить отдельный блок от других блоков, которые могут храниться в конкретном месте кэш-памяти.

Большое количество сравнений адресов в полностью ассоциативной кэш-памяти объясняется тем, что любой блок из основной памяти может храниться в любом месте кэш-памяти. Именно это заставляет проводить сравнения адресов всех блоков. С другой стороны, кэш-память с прямым отображением уменьшает требуемое количество сравнений, позволяя каждому блоку из основной памяти занимать только одно место в кэш-памяти.

Кэш-память с прямым отображением имеет и свои недостатки. Если процессор делает частые запросы из двух мест, для которых области памяти отображены в одну и ту же кэш-память, то контроллер должен часто обращаться к основной памяти, поскольку только одна из этих областей может в данный момент времени находиться в кэш-памяти.

Ассоциативная по множеству кэш-память. Эта кэш-память является компромиссом между полностью ассоциативной кэш-памятью и кэш-памятью с прямым отображением. Ассоциативная по множеству кэш-память имеет несколько групп

блоков прямого отображения, которые работают параллельно. Для каждого кэш-индекса существует несколько разрешенных мест расположения блока, по одному на каждое множество или группу. Блок поступающих данных может направляться в любое место для блока из своего множества.

Чрезмерно частое обращение к основной памяти, которое является недостатком кэш-памяти с прямым отображением, уменьшается, а процент кэш-попаданий увеличивается, поскольку ассоциативная по множеству кэш-память имеет несколько мест для блоков с одинаковыми кэш-индексами в их адрес. Ассоциативная по множеству кэш-память, однако, более сложна, чем кэш-память с прямым отображением.

В двухканальной ассоциативной по множеству кэш-памяти имеются два места, где может храниться каждый блок. Контроллер должен сделать два сравнения для того, чтобы определить, в каком из блоков находятся требуемые данные и есть ли они там вообще. Ассоциативная по множеству кэш-память нуждается также в более широком поле тега и в большем пространстве статического ОЗУ для хранения тегов. Кроме того, кэш-контроллер должен решить, какой блок кэш-памяти перезаписать при исполнении блочной выборки, так как существует несколько мест, в которых могут храниться данные из основной памяти. Наконец, когда информация сохраняется в ассоциативной по множеству кэш-памяти, то требуется дополнительная логика, которая должна решить, что должно быть сделано и в каком блоке следует хранить эту информацию.

4.3.2. Обновление кэш-памяти

Работая с кэш-памятью, необходимо помнить, что одновременно могут существовать две копии одних и тех же данных: одна в кэш-памяти, а другая в основной памяти. Если изменяется только одна из этих копий, то по одному и тому же адресу будут существовать два набора информации. Кэш-память должна содержать систему обновления для предотвращения использования устаревших данных в качестве достоверных. Рис. 4.7 иллюстрирует, как данные могут стать устаревшими и неправильными. Для согласования содержимого кэш-памяти с основной памятью используются три общих метода: сквозная запись, буферизованная сквозная запись и обратная запись.

Сквозная запись. При обновлении кэш-памяти методом сквозной записи кэш-контроллер одновременно обновляет содержимое основной памяти (рис. 4.8). Иначе говоря, основная память отражает текущее содержимое кэш-памяти. Быстрое обновление позволяет перезаписывать любой блок в кэш-памяти в любое время без потери данных. Система со сквозной записью проста, но время, требуемое для записи в основную память, снижает производительность и увеличивает количество обращений по магистрали (что особенно заметно в мультипроцессорной системе).

Буферизованная сквозная запись. В схеме обновления в буферизованной сквозной записью любая запись в основную память буферизуется, т. е. информация задерживается в кэш-памяти перед записью в основную память (схемы кэш-памяти управляют доступом к основной памяти асинхронно по отношению к работе процессора). Затем процессор начинает новый цикл до завершения цикла записи в основную память. Если за записью следует чтение, то это кэш-попадание, так как чтение может быть выполнено в то время, когда кэш-контроллер занят обновлением основной памяти. Эта буферизация позволяет избежать снижения производительности, характерного для системы со сквозной записью (рис. 4.9).

У этого метода есть один существенный недостаток. Так как обычно буферизуется только одиночная запись, то две последовательные записи в основную

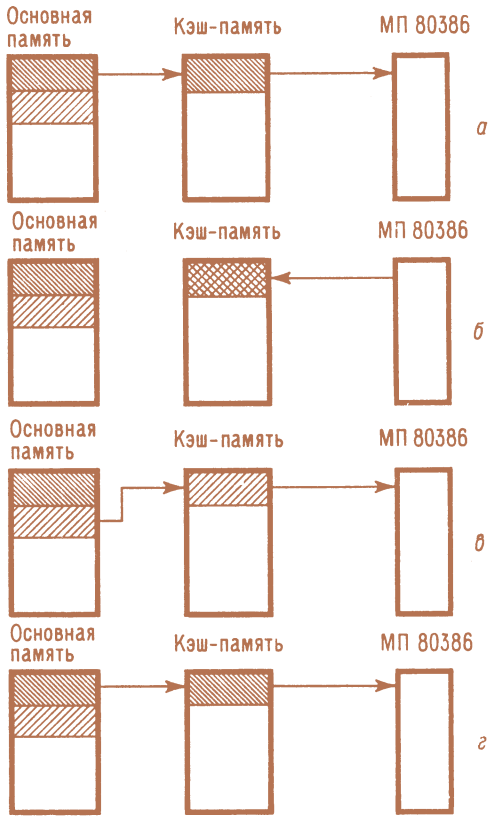


Рис. 4.7. Создание устаревших или неправильных данных. *а* – МП 80386 требует данные и фиксирует кэш-промах. Данные копируются из основной памяти и пересылаются в процессор. *б* – МП 80386 модифицирует данные и отсылает их для записи. *в* – позже МП 80386 запрашивает новые данные. Данные в кэш-памяти, помеченные перекрестной штриховкой в п. *б*, перекрываются новыми данными. *г* – МП 80386 запрашивает модифицированные данные, однако получает исходные, так как данные были модифицированы только в кэш-памяти, а в основную память переписаны не были.

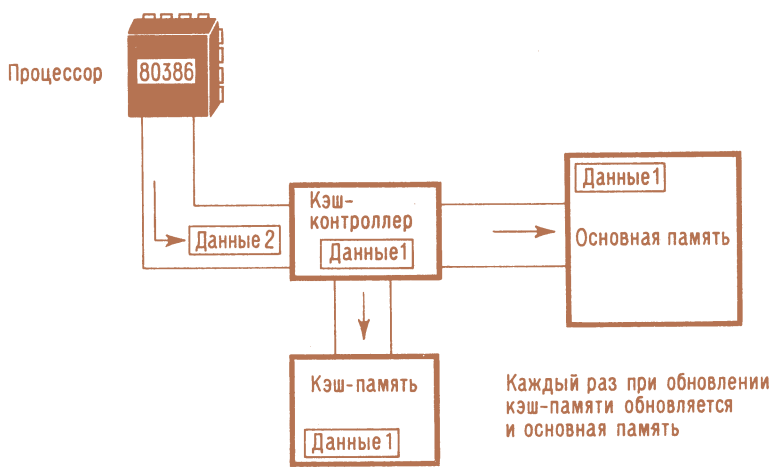


Рис. 4.8. Пример сквозной записи.

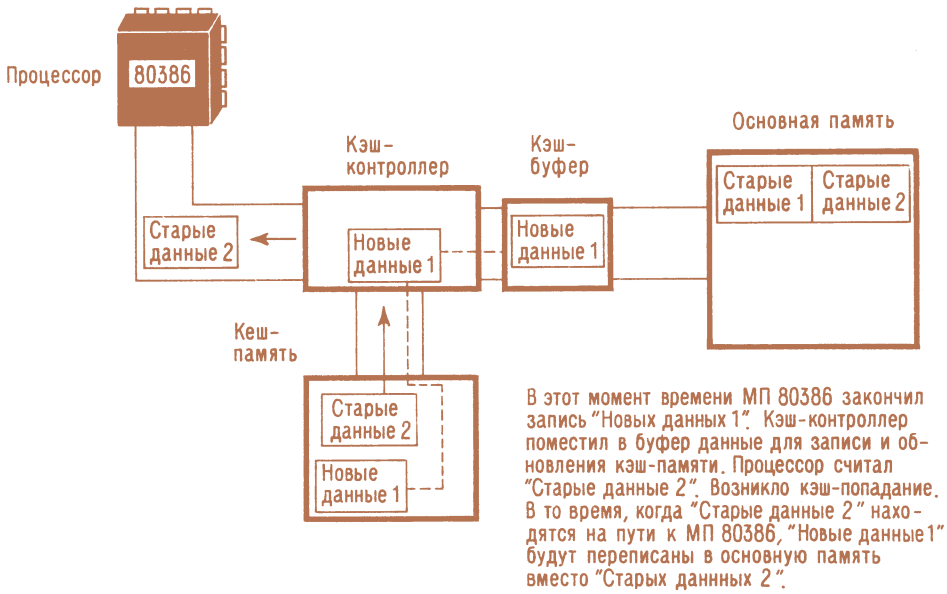


Рис. 4.9. Пример буферизованной сквозной записи.

память требуют цикла ожидания процессора. Кроме этого, запись с пропущенным последующим чтением также требует ожидания процессора. Состояние ожидания – это внутреннее состояние, в которое входит процессор при отсутствии синхронизирующих сигналов. Состояние ожидания используется для синхронизации процессора с медленной памятью.

Обратная запись. В схеме обновления с обратной записью используется бит «изменения» в поле тега. Этот бит устанавливается, если блок был обновлен новыми данными и является более поздним, чем его оригинальная копия в основной памяти. Перед тем как перезаписать блок в кэш-памяти, контроллер проверяет состояние этого бита. Если он установлен, то контроллер переписывает данный блок в основную память перед загрузкой новых данных в кэш-память.

Обратная запись быстрее сквозной, так как обычно число случаев, когда блок изменяется и должен быть переписан в основную память, меньше числа случаев, когда эти блоки считываются и перезаписываются.

Однако обратная запись имеет несколько недостатков. Во-первых, все измененные блоки должны быть переписаны в основную память перед тем, как другое устройство сможет получить к ним доступ. Во-вторых, в случае катастрофического отказа, например отключения питания, когда содержимое кэш-памяти теряется, но содержимое основной памяти сохраняется, нельзя определить, какие места в основной памяти содержат устаревшие данные. Наконец, кэш-контроллер для обратной записи содержит больше (и более сложных) логических микросхем, чем контроллер для сквозной записи. Например, когда система с обратной записью осуществляет запись измененного блока в память, то она формирует адрес записи из тега и выполняет цикл обратной записи точно так же, как и вновь запрашиваемый доступ.

Согласованность кэш-памяти. Устаревшие данные в основной памяти, вызванные операциями записи, виртуально уничтожаются сквозной и обратной записью. Существует категория устаревших данных, появляющаяся когда кэш-память ис-

пользуется в системе с доступом к памяти более чем из одного устройства. Например, пусть два устройства копируют раздел памяти и устройство 1 обновляет его первым. Когда устройство 2 производит запись в память, оно разрушит любые изменения, сделанные устройством 1. При этом говорят, что нарушается согласованность кэш-памяти. В общем случае существуют три подхода к обеспечению согласованности кэш-памяти.

Очистка кэш-памяти. При этой операции измененные данные записываются в основную память, а кэш-память очищается. Если все кэш-памяти в системе очищаются перед операцией записи устройствами в память совместного пользования, то возможность появления устаревших данных в любой кэш-памяти исключается. Преимуществом этого метода является простая аппаратная реализация. Главный недостаток в том, что следующий за очисткой доступ в память должен по определению быть кэш-промахом до тех пор, пока кэш-память не заполнится новыми данными.

Аппаратная прозрачность. Аппаратное решение проблемы устаревших данных дает уверенность, что все обращения к основной памяти (т.е. к той ее части, что отображается в кэш-память) проводятся через кэш-память. Это может быть сделано либо (1) путем копирования всех кэш-записей, как в основную память, так и во все другие кэш-памяти, либо (2) пересылкой всех кэш-записей через единственную кэш-память. На рис. 4.10 показаны различные аппаратные подходы.

Некэшируемая память. Другой метод решения проблемы согласованности кэш-памяти – это использование общей памяти как некэшируемой. В системе, основанной на этом методе, все обращения к общей памяти являются кэш-промахами, поскольку эта память никогда не копируется в кэш. Некэшируемая память может быть идентифицирована путем использования старших битов адреса или с помощью логики выбора микросхем памяти.

Конструкторы систем имеют тенденцию применять некоторую комбинацию этих трех методов для уверенности в достоверности данных. Например, одна система может использовать некэшируемую память для медленных задач ввода-вывода, таких как вывод на печать, и аппаратную прозрачность для критичных по времени операций ввода-вывода, подобных разбиению на страницы.

4.3.3. Кэш-память в МП 80386

Кэш-памятью прототипа является кэш-память прямого отображения объемом 64 Кбайт с 16384 входами. Каждый вход содержит 32 бит данных вместе с 16 бит теговой информации. Длина строки (единица передачи данных между кэш-памятью и ДОЗУ) составляет 32 бит. Десять микросхем памяти включают восемь статических ОЗУ данных с организацией 8К × 8 и два ДОЗУ тегового поля с организацией 8К × 8. Для управления кэш-памятью используются два компаратора и две программируемые логические матрицы. Интерфейс и буферы выполнены на восьми дополнительных логических микросхемах. Биты присутствия не обязательны, так как всегда предполагается, что каждый вход содержит 32-разрядное двойное слово достоверной информации. Подсистема кэш-памяти использует буферизованную сквозную запись для обновления ДОЗУ. Для МП 80386 комбинация кэш-память – ДОЗУ обеспечивает отсутствие тактов ожидания при записи. Процессор использует около четырех тактов ожидания во время кэш-промахов при чтении.

В кэш-памяти прямого отображения каждый элемент в ДОЗУ отображается в один элемент внутри кэш-памяти. Отображение осуществляется путем разделения 30-разрядного физического адреса процессора на две части: 16-разрядный тег и 14-разрядный индекс. Тег соответствует старшему адресу строк (A16–A31), а индекс – младшему адресу строк (A2–A15). Так как каждый элемент ДОЗУ с

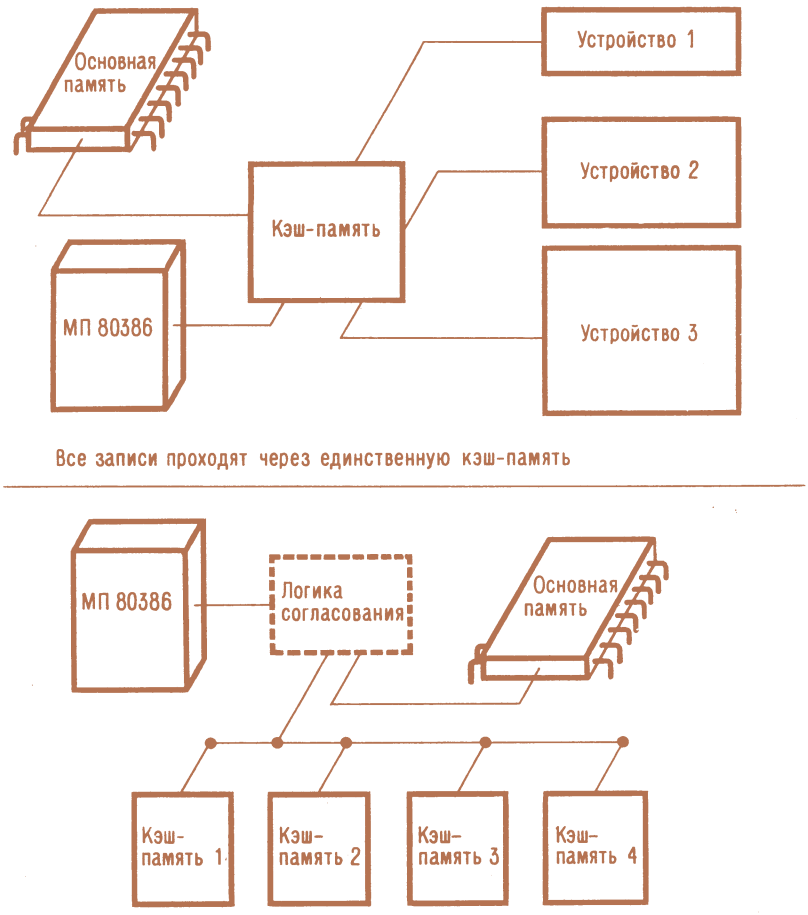


Рис. 4.10. Использование аппаратуры кэш-памяти.

одинаковыми значениями A2–A15 отображается в соответствующее место в кэш-памяти, то лишь один из этих элементов может быть представлен в кэш-памяти в любой момент времени.

Поля тега и индекса совместно определяют место двойного слова в основной памяти. В дополнение к этому индекс выбирает одно из 16384 мест в кэш-памяти. Самые младшие два бита адреса кодируются так, что становятся доступными все четыре байта слова, бит A31 определяет различия между некэшируемыми адресами ЭППЗУ и кэшируемыми адресами ДОЗУ. Если требуемые данные находятся в кэш-памяти, то чтение может быть выполнено за цикл в 2 такта. В противном случае чтение требует шести тактов. Ключ к успеху в конструировании системы с кэш-памятью лежит в увеличении числа двухтактовых циклов магистрали относительно шеститактовых. Это достигается увеличением объема кэш-памяти с 16 до 64 Кбайт.

Кэш-память использует буферизованную сквозную запись для обновления ДОЗУ. Во время циклов записи новые данные записываются в кэш-память и

одновременно в ДОЗУ. При использовании буфера цикл записи в ДОЗУ перекрывается по времени со следующим циклом магистрали. Это перекрытие позволяет выполнить запись только за три такта.

Буферизованная сквозная запись реализована путем отделения режима работы машины с кэш-памятью от режима работы с ДОЗУ. Поскольку как кэш-память, так и ДОЗУ имеют свои собственные контроллеры, то операция с перекрытием этих функций возможна. В режиме работы с кэш-памятью программируемая логическая матрица определяет, когда необходим цикл обращения к памяти, тогда как в режиме работы с ДОЗУ этот запрос проверяется только в случае готовности к запуску нового цикла.

Глава 5

Интерфейс: локальная магистраль, ввод и вывод

Отличительными особенностями МП 80386 являются 32-разрядные шины передачи внешних и внутренних данных и шесть 32-разрядных регистров общего назначения. Процессор непосредственно выдает 32-разрядный адрес в физическую память объемом 4 Гбайт. В этой главе более подробно рассматриваются подсистемы, которые окружают МП 80386, в частности интерфейс локальной магистрали, ввод-вывод и интерфейс с устройствами ввода-вывода.

5.1. Интерфейс с локальной магистралью

Магистраль – это путь, по которому передаются данные от любого из нескольких источников к любому из нескольких приемников. Она работает как общий канал связи между рядом блоков, такими как процессор, память и периферийные устройства. На рис. 5.1 показана структурная схема интерфейса с общей магистралью.

Процессор выполняет множество магистральных операций в ответ как на внутренние, так и внешние запросы, например на требования обслуживания по прерыванию. Он обменивается информацией с внешней памятью, устройствами ввода-вывода и другими устройствами через интерфейс с параллельной магистралью. Этот интерфейс организуется с помощью двунаправленной шины данных, отдельной шины адреса, пяти сигналов состояния магистрали и трех сигналов управления магистралью. Работа происходит следующим образом.

Двунаправленная шина данных. Имеются 32 вывода (D31–D0), по которым за один раз могут передаваться 8, 16, 24 или 32 бит.

Шина адреса. Эта шина генерирует 32-разрядные адреса и состоит из 30 выводов адреса (A31–A2) и четырех выводов выбора байтов (BE3#–BE0#). Каждый из выводов выбора байтов соответствует одному из передаваемых байтов 32-разрядной шины данных. Выводы адреса идентифицируют область из 4 байтов. Выводы выбора байтов выбирают активные байты внутри этой области.

Состояние магистрали. Эти выводы устанавливают тип выполняемого цикла магистрали и показывают следующие условия:

1) сигнал LOCK# – блокированный цикл магистрали;
2) состояние адреса (ADS#) – выходы шины адреса достоверны;
3) данные/управление (D/C#) – цикл данных или управления;
4) память/ввод-вывод (M/IO) – цикл памяти или ввода-вывода;
5) запись/чтение (W/R#) – цикл записи или чтения.

Управление магистралью. Эти выводы позволяют внешней логике управлять циклом магистрали на основе принципа «цикл за циклом». Назначение входов:

1) сигнал READY # – завершает текущий цикл магистрали;
2) следующий адрес (NA#) – этот вывод позволяет организовать конвейер

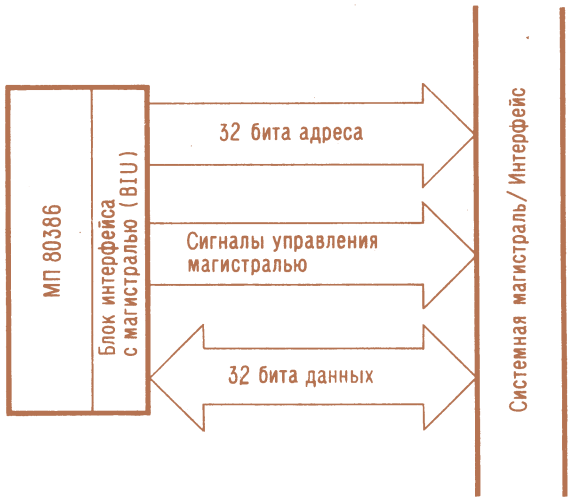


Рис. 5.1. Общий интерфейс с магистралью.

адресов, т. е. сигнал на этом выводе заставляет процессор генерировать сигналы адреса и состояния для следующего цикла магистрали в текущем цикле;

3) 16-разрядная шина данных (BS16#) – активизация этого вывода говорит о том, что ширина магистрали составляет 16 разрядов и будет передано 16 бит информации. Данные передаются по младшим 16 бит шины данных. Передача более 16 бит обеспечивается дополнительными циклами магистрали.

Следующие семь выводов управляют выполнением команд в МП 80386 и интерфейсом с внешними задатчиками магистрали. Процессор обеспечивает как стандартный интерфейс для взаимодействия с другими задатчиками магистрали, так и поддержку специального интерфейса с арифметическим сопроцессором.

1) RESET. Этот вход используется для установки МП 80386 в начальное состояние.

2) INTR (маскируемое прерывание) и NMI (немаскируемое прерывание). Эти входы служат для прерывания выполнения текущего потока команд процессором и инициирования выполнения программы обработки прерывания.

3) CLK2. На этот вход поступает сигнал двойной тактовой частоты для синхронных операций. При тактовой частоте 16 МГц МП 80386 использует сигнал 32 МГц CLK2, т. е. сигнал CLK внутри процессора делится на два, так что тактовая частота МП 80386 составляет половину от частоты сигнала CLK2.

4) BUSY#, ERROR# и PEREQ (запрос сопроцессора). Эти сигналы образуют интерфейс с внешним арифметическим сопроцессором. Сигналы ERROR# и BUSY# являются сигналами состояния, поступающими от сопроцессора. Сигнал PEREQ позволяет сопроцессору запрашивать данные от МП 80386.

5) HOLD. Этот сигнал может генерироваться другим задатчиком магистрали как требование к МП 80386 передать управление магистралью. При передаче управления локальной магистралью МП 80386 активирует сигнал HLDA (подтверждение захвата).

На рис. 5.2 показано назначение сигнальных выводов МП 80386. Полный список выводов МП 80386 приведен в гл. 7.

Существуют семь типов магистральных операций: чтение памяти, запись в память, чтение ввода-вывода, запись в ввод-вывод, выборка команды, подтверждение прерывания и состояние останова.

Название сигнала	Назначение	Активный уровень	Вход-выход	Синхронный или асинхронный по отношению к CLK2	Третье состояние при активном HLDA
CLK2	Тактовая частота	—	Вход	—	—
D0—D31	Шина данных	Высокий	Двунаправленный	Синхронный	Да
BE0#—BE3#	Выбор байтов	Низкий	Выход	—	«
A2—A31	Шина адреса	Высокий	Выход	—	«
W/R#	Запись/чтение	Высокий	Выход	—	«
D/C#	Данные/управление	Высокий	Выход	—	«
M/IO#	Память/ввод-вывод	Высокий	Выход	—	«
LOCK#	Блокировка цикла	Низкий	Выход	—	«
ADS#	Состояние адреса	Низкий	Выход	—	«
NA#	Следующий адрес	Низкий	Вход	Синхронный	—
BS16#	Магистраль в 16 бит	Низкий	Вход	«	—
READY#	Готовность	Низкий	Вход	«	—
HOLD#	Захват магистрали	Высокий	Вход	«	—
HLDA#	Подтверждение захвата	Высокий	Выход	—	Нет
PEREQ#	Запрос сопроцессора	Высокий	Вход	Асинхронный	—
BUSY#	Сопроцессор занят	Низкий	Вход	«	—
ERROR#	Ошибка сопроцессора	Низкий	Вход	«	—
INTR	Маскируемое прерывание	Высокий	Вход	«	—
NMI	Немаскируемое прерывание	Высокий	Вход	«	—
RESET	Сброс	Высокий	Вход	Синхронный	—

Рис. 5.2. Назначение сигнальных выводов МП 80386.

Каждый цикл магистрали начинается, когда на шине адреса присутствует достоверный адрес и выводы состояния магистрали установлены в те состояния, которые соответствуют типу цикла магистрали. Циклы чтения и записи в память могут быть заблокированы для предотвращения использования локальной магистрали другим задатчиком, а также для обеспечения целостности операций «чтение–модификация–запись». (Более подробная информация по задатчикам/исполнителям магистрали MULTIBUS содержится в гл. 12 и приложении 4.)

Конвейер адресов позволяет циклам магистрали перекрываться по времени, что увеличивает интервал времени, отводимый памяти или устройствам ввода-вывода для отклика. Первый цикл после незанятого состояния магистрали никогда не

конвейеризуется. Для инициализации конвейера адресов этот цикл должен быть расширен по крайней мере на один такт CLK так, чтобы сигналы адреса и состояния могли быть выведены перед завершением цикла. Последующие циклы могут быть конвейеризованы до тех пор, пока не встретится незанятый цикл магистрали.

5.1.1. Состояния магистрали

Для формирования тактового сигнала CLK процессор использует тактовый сигнал двойной частоты CLK2. Внутренний тактовый сигнал соответствует внешнему сигналу CLK тактового генератора 82324. Допускается некоторое отставание сигнала CLK генератора 82324 от сигнала CLK2, но ни в коем случае его опережение. Кроме того, он может быть использован как индикатор фаз состояния.

Каждый цикл магистрали состоит по меньшей мере из двух состояний магистрали: T1 и T2. Каждое состояние магистрали состоит в свою очередь из двух тактов CLK2, которые могут быть представлены как фаза 1 и фаза 2 состояния магистрали. Во время первого состояния магистрали (T1) активизируются как выводы адреса, так и выводы состояния. Во время второго состояния (T2), которое совпадает с концом второго такта CLK2, цикл магистрали завершается. Если процессор не нуждается в циклах магистрали, МП 80386 остается в состоянии незанятой магистрали (T1).

Максимальная скорость передачи для магистральных операций определяется внутренней тактовой частотой МП 80386 и составляет 32 бит на каждые два такта CLK, т.е. 32 Мбайт/с (частота сигнала CLK2 = 32 МГц, частота внутреннего сигнала CLK = 16 МГц). Удобным свойством МП 80386 является то, что управление синхронизацией выходов адреса и состояния может осуществляться так, что эти выходы становятся достоверными до завершения предыдущего цикла магистрали. Такое перекрытие позволяет организовать конвейер из циклов магистрали. Конвейер адресов увеличивает пропускную способность магистрали без уменьшения времени доступа к памяти или устройствам ввода-вывода. При тактовой частоте 16 Мгц МП 80386 передает данные со скоростью 32 Мбайт/с, и это позволяет иметь время доступа к адресу в три такта CLK (187,5 нс при частоте 16 МГц, пренебрегая задержками сигналов). Без конвейера адресов время доступа составляет только два такта CLK (125 нс при частоте 16 МГц).

Процессор адресует до 4 Гбайт (2^{32} байт, адреса от 00000000H до 0FFFFFFFFH) физической памяти и 64 Кбайт (2^{16} байт, адреса от 00000000H до 0000FFFFH) устройств ввода-вывода. Он поддерживает раздельные адресное пространство физической памяти и адресное пространство устройств ввода-вывода.

Пространство памяти и пространство устройств ввода-вывода являются последовательностями 32-разрядных двойных слов (2^{30} 32-разрядных ячеек памяти и 2^{14} 32-разрядных портов ввода-вывода максимум). Каждое двойное слово начинается с физического адреса, кратного четырем. Это слово имеет также четыре индивидуально адресуемых байта по последовательным адресам. Выводы A31–A2 процессора соответствуют старшим битам физического адреса. Заметьте, что эти биты адресуют двойные слова памяти.

В каждом магистральном цикле передачи данные могут быть переданы группами по 8, 16, 24 или 32 бит. Для этого необходимо два цикла магистрали, которые автоматически запускаются МП 80386. Определенные комбинации сигналов BE3# – BE0# никогда не выставляются, поскольку МП 80386 оперирует только байтами, словами и двойными словами. Например, цикла магистрали с активными только сигналами BE0# и BE2# не бывает никогда, так как это была бы передача двух несмежных байтов в одно и то же время.

5.1.2. Циклы чтения и записи

В интерфейсе с локальной магистралью используются циклы чтения двух типов: с конвейеризацией адресов и без конвейеризации адресов. В первом случае сигналы шины адреса и шины состояния выводятся перед началом цикла для увеличения времени доступа к памяти. В цикле без конвейеризации адресов сигналы шины адреса и шины состояния становятся достоверными в первом такте CLK этого цикла.

Подобно циклам чтения циклы записи также бывают двух типов: с конвейеризацией и без конвейеризации адресов.

5.1.3. Прерывания

Порядок выполнения программы в МП 80386 может быть изменен как аппаратными, так и программными прерываниями. Аппаратное прерывание возникает при активизации одного из входов запроса прерываний МП 80386 (NMI – для немаскируемых прерываний и INTR – для маскируемых прерываний). Программное прерывание возникает, когда МП 80386 обнаруживает команду прерывания INT или особое программное условие (исключение), которое нуждается в обслуживании.

Время реакции прерывания – это время, проходящее с момента запроса прерывания до его обслуживания. На время реакции прерывания влияют следующие причины:

- задержка начала обслуживания прерывания в случае, если программа обработки прерывания сохраняет те регистры, которые автоматически не сохраняются процессором;
- сохранение регистра флагов EFLAGS, а также регистра сегмента программы CS и указателя команд EIP, которые содержат адрес возврата;
- маскирование прерывания, что приводит к опознанию сигнала на входе INTR только после повторного разрешения прерываний;
- выполнение процессором текущей команды должно быть завершено. Запрос прерывания опознается только на границах команд с единственным исключением для команд повтора строк, которые могут прерываться после каждой итерации;
- если обслуживание прерывания нуждается в переключении задачи, одни регистры должны быть сохранены, а другие – восстановлены.

Цикл подтверждения прерывания. Являясь частью цикла подтверждения, немаскированное прерывание принуждает МП 80386 временно прекратить выполнение текущей программы и выполнять некоторые команды другой программы. Эта вторая программа называется программой обработки прерывания (или обработчиком).

Процессор выполняет один за другим два цикла подтверждения прерывания в ответ на активный сигнал на входе INTR. Это специальные циклы магистрали, активирующие программируемый контроллер прерываний 8259A, который в свою очередь выдает вектор прерывания в битах D0–D7 шины данных.

Системная логика должна задержать сигнал READY# для того, чтобы удлинить цикл согласно требованиям контроллера 8259A к минимальной длительности импульса. Для соответствия времени отклика контроллера 8259A процессор дополнительно вставляет между циклами как минимум 160 нс холостого времени магистрали (четыре состояния T1).

Цикл BS16#. Процессор обеспечивает передачу данных как по 16-разрядной, так и по 32-разрядной шинам. Вход управления BS16# позволяет определять ширину шины для каждого цикла магистрали. Динамическое изменение ширины шины позволяет МП 80386 обрабатывать данные различной разрядности.

Если необходима максимальная внутренняя скорость обмена информацией, то может быть использовано прямое подключение к магистрали. Для некоторых систем целесообразно использовать 16-разрядную шину данных. Память с 16-разрядной организацией в отличие от 32-разрядной требует меньше микросхем поддержки, например буферов, фиксаторов и т. п. Более того, адреса устройств ввода-вывода, расположенные по границам слов (а не двойных слов), могут быть программно совместимы с другими системами, которые используют 16-разрядные микропроцессоры.

По сигналу на входе BS16# МП 80386 передает данные по 16-разрядной шине данных (D15–D0). Если же сигнала на этом входе нет, процессор передает 32-разрядные данные. Процессор автоматически выполняет два или три цикла для передачи данных, разрядность которых больше 16 бит, и для невыровненных на границы 16-разрядных слов. Сигнал на вход BS16# приходит от внешней аппаратуры: либо непосредственно от адресуемого устройства, либо через микросхему селектора адреса. Вход BS16# опрашивается в начале фазы 2 (такт CLK) при условии, что вход ADS# не активен. Если сигналы на входах BS16# и READY# имеют низкий уровень в том же самом такте CLK, то МП 80386 считает шину данных 16-разрядной.

Состояние входа BS16# влияет на скорость передачи данных только в тех случаях, когда сигналы BE0# и BE1# активны и в то же самое время активен сигнал BE2# или сигнал BE3#. При этих передачах МП 80386 выполняет два цикла магистрали, используя только младшую половину шины данных. Если необходим еще один цикл магистрали, то МП 80386 оставляет на магистрали текущий адрес для второго цикла. Поскольку адресный конвейер требует, чтобы следующий адрес был сформирован на магистрали перед окончанием текущего магистрального цикла, то его нельзя использовать в циклах BS16#. Вот почему оба сигнала опрашиваются в одном и том же временном окне. Сигнал BS16# должен быть активным перед (или одновременно с) сигналом следующего адреса NA# для того, чтобы гарантировать 16-разрядную операцию. Если при опросе было установлено, что сигнал NA# был активен, а в то же время сигнал BS16# не был активен, то вход BS16# блокируется внутри процессора.

Цикл останова/прекращения работы. Процессор останавливается в ответ на команду HLT. Кроме этого, условие прекращения работы появляется, когда МП 80386 обнаруживает нарушение защиты при обработке двойного отказа. Процессор не может выйти из такой ситуации и прекращает работу. С точки зрения внешних устройств цикл прекращения работы отличается от цикла останова результирующим конечным состоянием выходов адресной шины. Ниже описаны шаги, выполняемые процессором.

- Сигналы M/IO# и W/R# переходят в состояние высокого уровня, сигнал D/C# переходит в состояние низкого уровня, что идентифицирует цикл останова.
- Все выходы шины адреса переходят на низкий уровень, причем
  - для условия останова активен сигнал BE2#;
  - для условия прекращения работы активен сигнал BE0#.
- Для завершения цикла останова или прекращения работы должен быть активизирован сигнал READY#. Процессор остается в состоянии останова или прекращения работы до тех пор, пока:
  - сигнал NMI не перейдет в состояние высокого уровня, тогда МП 80386 обслуживает прерывание;
  - сигнал RESET не перейдет в состояние высокого уровня, тогда МП 80386 повторно инициализуется.

Если маскируемые прерывания разрешены, то в состоянии останова активный сигнал на входе INTR принуждает МП 80386 завершить цикл останова и обслужи-

вать это прерывание. Однако этого не происходит в состоянии прекращения работы. Процессор может обслуживать запросы процессорного расширения по входу PEREQ и запросы на захват магистрали по входу HOLD как в состоянии останова, так и в состоянии прекращения работы.

5.1.4. Блокированные циклы и захват магистрали

Блокированные циклы должны использоваться в тех ситуациях, когда настоятельно требуется, чтобы два и более циклов магистрали неразрывно следовали один за другим, особенно в системе, в которой локальной магистралью могут управлять более одного устройства. Если блокирование не применяется, то эти циклы могут быть разделены циклом от другого задатчика магистрали.

В качестве непреложного правила нужно принять, что любые магистральные циклы, которые должны выполняться один за другим (без вмешательства циклов от другого задатчика магистрали), следует блокировать. Сигнал LOCK# процессора оповещает других задатчиков магистрали о том, что они не могут получить управление магистралью. Точно так же МП 80386 с активным сигналом LOCK# не будет опознавать запрос по входу HOLD на захват магистрали от другого задатчика.

Префикс LOCK в определенных командах активирует сигнал LOCK#. Команда XCHG, обновление дескриптора, циклы подтверждения прерывания и обновление таблицы страниц автоматически активизируют сигнал LOCK#.

Максимальное время сигнала LOCK# зависит от исполняемой команды и количества состояний ожидания в цикле. Наиболее длительный период в реальном режиме имеет длительность в два цикла магистрали плюс около двух тактов. Это случается при выполнении команды XCHG и блокированных операций «чтение-модификация-запись». Наиболее длительный период в защищенном режиме имеет длительность в пять циклов магистрали и еще около 15 тактов. Он используется при возникновении аппаратного или программного прерывания, когда МП 80386 выполняет следующие блокированные операции: чтение вентиля в таблицу дескрипторов прерываний IDT, чтение нужного дескриптора и запись бита доступа в нужный дескриптор.

Запрашивающий управление локальной магистралью задатчик активизирует вход HOLD процессора. Процессор завершает выполнение своей задачи, а затем устанавливает все свои выходы (кроме HLDA) в третье (отключенное) состояние для того, чтобы освободить магистраль. Затем МП 80386 активизирует выход HLDA для оповещения запрашивающего задатчика магистрали о том, что тот может взять управление на себя. Этот задатчик должен поддерживать сигнал HOLD активным все то время, пока он нуждается в магистрали. Процессор ожидает переход сигнала HOLD на низкий уровень, после чего МП 80386 выставляет низкий уровень на выходе HLDA и возобновляет цикл магистрали (если таковой был прерван).

Процессор не простаивает во время активности сигнала HOLD – он продолжает выполнять команды из своей очереди предвыборки. Программа останавливается, если нужен цикл чтения в то время, когда МП 80386 не может получить доступа к локальной магистрали. Ожидая доступа к магистали, МП 80386 ставит во внутреннюю очередь один цикл записи. Если же нужно более одного цикла записи, то МП 80386 должен задержать дальнейшее выполнение, пока все записи не будут сделаны.

Хотя сигнал HOLD и имеет приоритет над большинством циклов магистрали, существуют три случая, когда он не опознается: во-первых, между двумя циклами подтверждения прерывания, во-вторых, во время блокированных циклов и, в-третьих, между двумя повторяющимися циклами BS16#.

Процессор игнорирует все входы в то время, когда он находится в состоянии захвата HOLD, за исключением следующих трех случаев: во-первых, один запрос по входу NMI опознается и запоминается. Он будет обслужен после снятия сигнала HOLD. Во-вторых, сигнал RESET имеет приоритет над сигналом HOLD. Активный вход RESET повторно инициализирует МП 80386. В-третьих, вход HOLD опрашивается для определения того момента, когда МП 80386 может получить управление магистралью.

5.2. Интерфейс с устройствами ввода-вывода

Микропроцессор 80386 поддерживает 8-, 16- и 32-разрядные устройства ввода-вывода, где ввод-вывод может быть отображен либо в 4-Гбайт пространство физических адресов памяти с использованием обычных команд манипуляции операндами либо в 64-Кбайт пространство адресов устройств ввода-вывода с использованием специальных команд ввода-вывода. Отображение в пространство ввода-вывода и отображение в пространство памяти имеют три главных отличия:

1. Отображение в пространство памяти обеспечивает больше гибкости по защите, чем отображение в пространство ввода-вывода, поскольку отображенные устройства защищаются механизмом защиты и управления памятью. В зависимости от того, в какое место памяти отображается устройство, это устройство может быть недоступным для задачи, может быть видимо, но защищено, или же быть полностью доступным. Разбиение на страницы дает те же самые уровни защиты для каждой страницы в 4 Кбайт и показывает, можно или нет выполнять запись в данную страницу.

2. Селектор адреса, необходимый для выработки сигналов выбора микросхем для отображенных в пространство ввода-вывода устройств, обычно проще, чем для устройств, отображенных в пространство памяти. К тому же отображенные в пространство ввода-вывода устройства находятся в 64-Кбайт пространстве, тогда как отображенные в пространство памяти устройства находятся в намного большем, 4-Гбайт пространстве памяти, что заставляет использовать больше адресных линий.

3. К отображенным в пространство памяти устройствам можно получить доступ с помощью любой команды МП 80386. Это позволяет эффективно программировать передачи из устройства в память, из памяти в устройство и между двумя устройствами ввода-вывода. К отображенным в пространство ввода-вывода устройствам можно получить доступ с помощью четырех команд: IN, INS, OUT, OUTS. Все обмены с устройствами ввода-вывода осуществляются через регистры AL (8-разрядный), AX (16-разрядный) или EAX (32-разрядный). Первые 256 байт пространства ввода-вывода адресуемы прямо, а полное 64-Кбайт пространство ввода-вывода адресуемо косвенно через регистр DX.

Интерфейс с периферийными устройствами зависит не только от разрядности данных, но также и от требований устройства к сигналам, и от его места в пространстве памяти или ввода-вывода. Независимо от того, отображено ли данное устройство в пространство памяти или в пространство ввода-вывода, необходима селекция адреса для выработки правильных сигналов выбора микросхем. Внутри пространства ввода-вывода или памяти устройствам могут быть присвоены произвольные адреса. Эти адреса следует выбирать так, чтобы минимизировать количество необходимых адресных линий.

Процессор имеет изолированное пространство адресов ввода-вывода, которое отличается от физической памяти и используется для внешних 16-разрядных устройств. Адресное пространство ввода-вывода образуют до 2^{16} (64К) индиви-

дуально адресуемых 8-разрядных портов. С любыми соседними 8-разрядными портами с четным адресом можно обращаться, как с 16-разрядным портом, а с любыми соседними 8-разрядными портами с адресом двойного слова можно обращаться, как с 32-разрядным портом. Это означает, что полное адресное пространство ввода-вывода вмещает до 64К 8-разрядных портов (пронумерованных от 0 до 65535), до 32К 16-разрядных портов (пронумерованных 0, 2, 4 и так до 65534) или 16К 32-разрядных портов (пронумерованных 0, 4, 8 и так до 65532).

Отображение в пространство памяти устройств ввода-вывода дает гибкость в программировании. Любая команда со ссылкой в память может использоваться для доступа к порту ввода-вывода в пространстве памяти. Доступ к таким устройствам осуществляется с использованием полного набора команд и всех режимов адресации для выбора требуемого устройства ввода-вывода. Вместе с тем ссылка на отображенное в пространство памяти устройство ввода-вывода, подобно любой другой ссылке в память, остается объектом защиты и управления доступом во время исполнения в защищенном режиме.

5.2.1. Команды ввода-вывода

Команды ввода-вывода МП 80386 обеспечивают доступ к портам ввода-вывода процессора для обмена данными с периферийными устройствами. Эти команды в качестве единственного операнда имеют адрес порта в пространстве ввода-вывода. Существуют два класса команд ввода-вывода: передающие строки объектов, расположенных в памяти (известные как команды ввода-вывода строк или блочные команды ввода-вывода), и передающие единственные байт, слово или двойное слово, расположенные в регистре.

Блочные команды ввода-вывода INS и OUTS пересылают блоки данных между портами ввода-вывода и пространством памяти. Эти команды используют регистр DX для указания порта в пространстве адресов ввода-вывода. 8-разрядные порты нумеруются от 0 до 65535, 16-разрядные порты нумеруются 0, 2, 4 и так до 65534, 32-разрядные порты нумеруются 0, 4, 8 и так до 65532.

Команды IN и OUT пересылают данные между портами ввода-вывода и регистрами AL (для 8-разрядного ввода-вывода), AX (для 16-разрядного ввода-вывода) и EAX (для 32-разрядного ввода-вывода). Команды IN и OUT адресуют порты ввода-вывода либо прямо (адресом одного из 256 портов, закодированном в команде), либо косвенно (используя регистр DX для указания адреса одного из 64К портов).

5.2.2. Защита, привилегии и ввод-вывод

Уровень привилегий ввода-вывода (ВВ) обеспечивает защиту, разрешая задаче доступ ко всем устройствам ввода-вывода (УВВ) или же запрещая задаче доступ к любому УВВ. В режиме виртуального МП 8086 для выбора уровня привилегий любой комбинации байтов ВВ может использоваться битовая карта разрешения ВВ.

Защита ВВ обеспечивается двумя способами: битовая карта разрешения ВВ сегмента TSS определяет право использовать порты в адресном пространстве ВВ; уровень привилегий ВВ IOPL в регистре EFLAGS определяет право использовать связанные с ВВ команды. Эти механизмы работают только в защищенном режиме, в который входит также и режим виртуального МП 8086. Они не работают в реальном режиме. В реальном режиме любая процедура исполняет команды ВВ и любой порт ВВ может быть адресован любой из этих команд.

Битовая карта разрешения ввода-вывода. IN, INS, OUT, OUTS – это команды, которые прямо используют адреса из пространства ВВ. Процессор может селектив-

но выявить ссылки на определенные адреса ВВ. Механизм, который позволяет осуществить это выявление,– это битовая карта разрешения ВВ в сегменте TSS. На рис. 5.3 показано, где именно в сегменте TSS появляется эта карта.

Эта карта является вектором битов, и ее размер и размещение в TSS могут изменяться. Процессор размещает битовую карту посредством 16-разрядного поля базы карты в фиксированной части таблицы TSS, которая содержит относительный адрес начала карты. Верхняя граница карты ввода-вывода – это то же самое, что и граница сегмента TSS. Поскольку карта разрешения ВВ находится в таблице TSS, то различные задачи могут иметь различные карты. Таким образом, операционная система может зарезервировать порты для задачи путем изменения карты разрешения ВВ в таблице TSS задачи.

Каждый бит в этой карте соответствует адресу байта порта ВВ. То есть бит для порта 41 находится по адресу: база карты ВВ + 5, относительный адрес бита равен 1. Процессор проверяет все биты, которые соответствуют адресам портов ВВ, участвующим в операции ВВ. То есть при операции с двойным словом будут проверяться четыре бита, которые соответствуют адресам четырех соседних байтов. Если любой из проверяемых битов установлен, то МП 80386 вызывает обработку исключения по защите. Если же все проверяемые биты очищены, то операция ввода-вывода продолжается.

Если МП 80386 обнаруживает команду ввода-вывода в защищенном режиме, то сначала он проверяет, не является ли текущий уровень привилегий CPL меньшим или равным уровню IOPL. Если это верно, то продолжается операция ввода-выво-

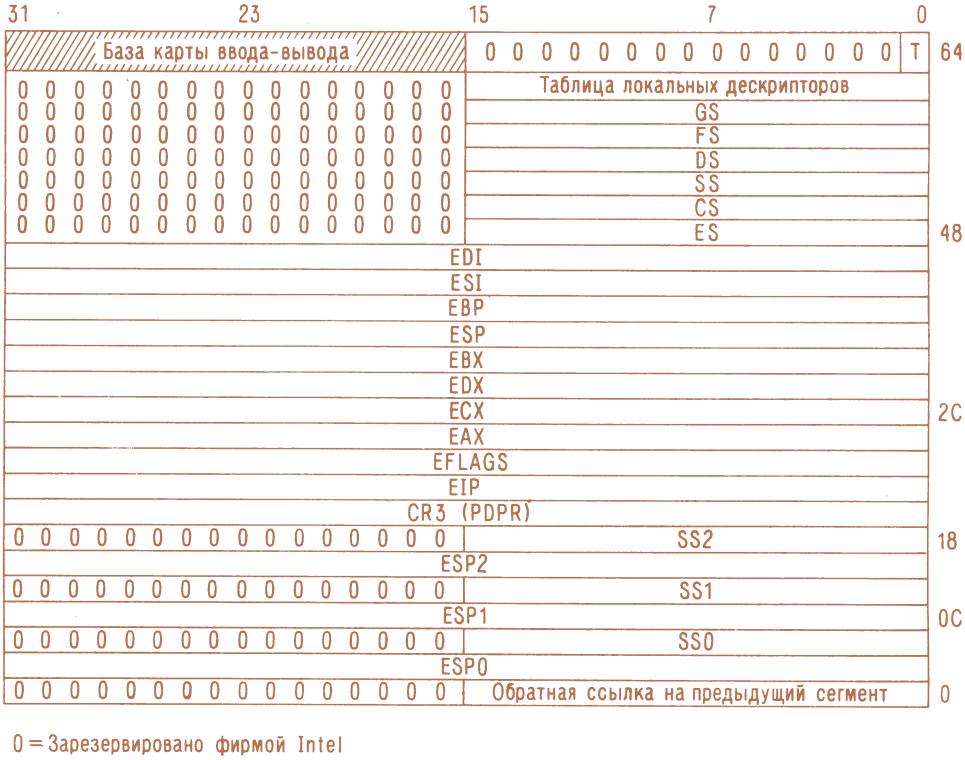


Рис. 5.3. Таблица TSS с картой ввода-вывода.

да, иначе МП 80386 проверяет карту разрешения ввода-вывода. (Заметим, что в режиме виртуального МП 8086 МП 80386 проверяет эту карту безотносительно к уровню IOPL.) Если база карты ввода-вывода больше или равна границе таблицы TSS, то сегмент TSS не имеет карты разрешения ввода-вывода и все команды ввода-вывода вызывают обработку исключения, когда уровень CPL больше уровня IOPL.

Необходимо, чтобы на команды, имеющие отношение к вводу-выводу, не только налагались ограничения, но и чтобы они были исполняемыми процедурами на уровнях привилегий, отличных от нуля. Для обеспечения этого МП 80386 использует два бита регистра флагов для хранения уровня IOPL. Уровень IOPL определяет уровень привилегий, необходимый для того, чтобы выполнять связанные с ВВ команды. Следующие шесть команд могут выполняться только в том случае, если уровень CPL меньше или равен уровню IOPL. Эти команды называются «чувствительными», так как они чувствительны к значению, хранимому в битах IOPL: IN (ввод), INS (ввод строки), OUT (вывод), OUTS (вывод строки), CLI (очистить флаг разрешения прерываний), STI (установить флаг разрешения прерываний).

Для того чтобы использовать чувствительные команды, процедура должна исполняться на уровне привилегий, по крайней мере таком же, как и уровень, хранимый в битах IOPL. Любая попытка менее привилегированной процедуры использовать одну из этих шести команд приводит к обработке исключения по общей защите.

Каждая задача в системе имеет свою уникальную копию регистра флагов, поэтому она может иметь свой уровень IOPL. Задача может изменить значения битов IOPL только командой POPF. Такие изменения являются привилегированными. Ни одна процедура не может модифицировать биты IOPL, если только эта процедура не исполняется на нулевом уровне привилегий. Любая менее привилегированная команда или процедура, пытающаяся изменить биты IOPL, не приводит к обработке исключения; уровень IOPL остается неизмененным.

5.2.3. Основной интерфейс ввода-вывода

В типичной системе на базе МП 80386 ряд исполнителей – УВВ – управляются через тот же самый интерфейс локальной магистрали. Другие УВВ, особенно те, которые могут управлять локальной магистралью, требуют более сложных интерфейсов.

Производительность и гибкость интерфейса локальной магистрали МП 80386 плюс увеличивающаяся доступность программируемой и полузаказной логики делают возможным конструирование заказной логики управления магистралью, которая удовлетворяет требованиям любой конкретной системы. Основной интерфейс ВВ показан на рис. 5.4.

С этим интерфейсом совместимы следующие компоненты:

8041, 8042	Универсальный периферийный интерфейс
8237	Контроллер ПДП (удаленный узел)
8253, 8254	Программируемый интервальный таймер
8255	Программируемый периферийный интерфейс
8259A	Программируемый контроллер прерываний
8272	Контроллер гибкого диска
8274	Мультипротокольный последовательный контроллер
82062, 82064	Контроллер жесткого диска
82258	Улучшенный контроллер ПДП (удаленный узел)

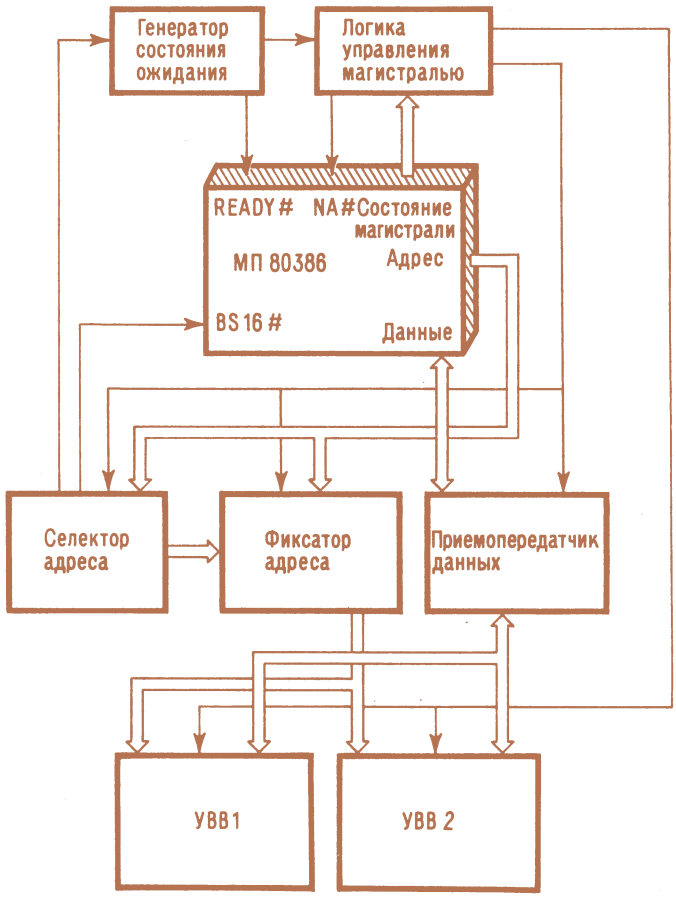


Рис. 5.4. Структурная схема основного интерфейса ввода-вывода.

8-, 16- и 32-разрядный ввод-вывод. Восьмиразрядные УВВ могут быть подключены к любой из четрых 8-разрядных частей шины данных. Если в цикле записи сигналы BE3# и/или BE2# (старшая часть двойного слова) активны, а сигналы BE1# или BE0# не активны, то значение старшей половины шины данных воспроизводится на ее младшей половине. Если адреса лежат в границах одного и того же двойного слова (адреса двух устройств отличаются только значением сигналов BE3# – BE0#), то для формирования сигнала выбора микросхемы нужно декодировать сигналы BE3# – BE0#. Этот сигнал при записи в одно устройство предотвращает от выполнения ошибочной записи в другое. Выбор микросхемы может быть сделан с помощью либо логики ТТЛ, либо селектора адреса на ПЛМ.

Другой метод сопряжения с 8-разрядными периферийными устройствами показан на рис. 5.5. Шина данных в 32 разряда представляет собой объединение 8-разрядных шин данных, тем самым обеспечивается байт-ориентированный прямой доступ к памяти или блочный обмен данными с отображенными в память устройствами ввода-вывода. Поскольку в любой момент времени включается только одна 8-разрядная часть шины данных, то адреса, назначенные подключенным устройствам, могут располагаться рядом.

Для того чтобы избежать лишних циклов магистрали и упростить выбор

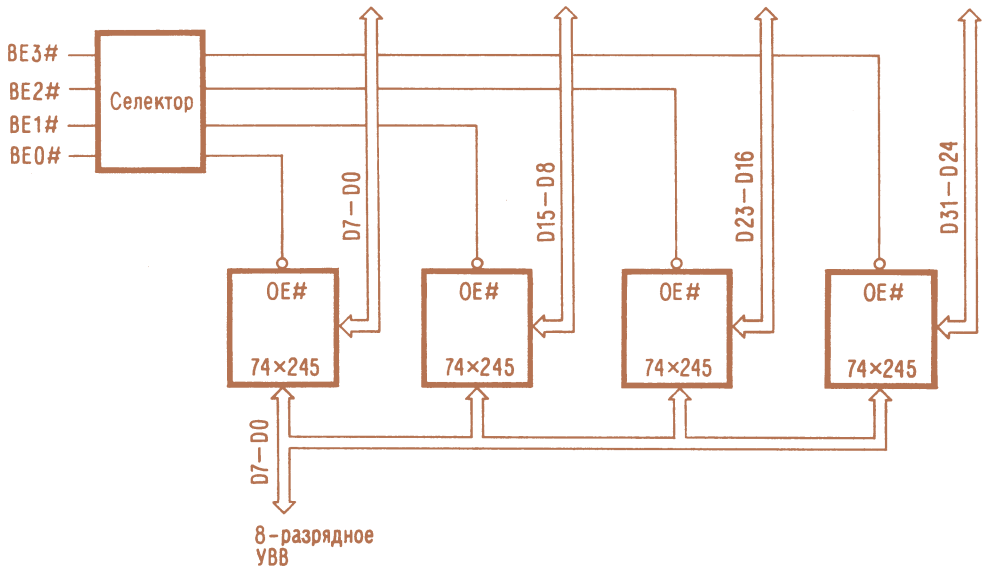


Рис. 5.5. Преобразование 32-разрядной магистрали в 8-разрядную.

устройства, 16-разрядным УВВ следует назначать четные адреса. Если адреса расположены на границах соседних слов, то при их декодировании нужно генерировать сигнал BS16# с тем, чтобы МП 80386 выполнял цикл 16-разрядной магистрали. Если же адреса расположены на границах слов, но с пропуском одного слова (т. е. по адресам каждого двойного слова), то цикл BS16# не нужен.

В этих же целях 32-разрядным УВВ следует назначать адреса, кратные четырем. Сигнал выбора такого 32-разрядного устройства должен быть активизирован сигналами включения всех байтов BE3# – BE0#.

Системы с 14 или менее портами ввода-вывода, которые располагаются в пространстве ввода-вывода или которые требуют более одного активного сигнала выбора, для доступа к УВВ могут использовать линейный выбор. Фиксируемые адресные линии A15–A2 подключаются непосредственно ко входам выбора УВВ.

Фиксация адреса. Фиксаторы – это схемы, которые сохраняют свое содержимое на определенный период времени. Фиксаторы используются для сохранения адреса ввода-вывода в течение магистрального цикла. Вход фиксации микросхемы типа SN 74 × 373 управляется сигналом фиксации адреса ALE, который приходит от логики управления магистралью. Сигнал фиксации становится активным в начале каждого магистрального цикла. Выход микросхем 74 × 373 (OE #) всегда активен.

Селектор адреса. Селектор адреса состоит из двух селекторов 1-из-4, один из которых декодирует адреса памяти, а другой – адреса ввода-вывода. На рис. 5.5 выход селектора адресов памяти для доступа к устройствам ввода-вывода активирует селектор адресов ввода-вывода. Благодаря способу распределения адресов УВВ для формирования правильного сигнала выбора нужны только адресные биты A4 и A5.

Показанный на рис. 5.5 селектор адреса преобразует также адрес МП 80386 в сигналы выбора микросхем. Селектор расположен перед фиксаторами адреса, хотя он может быть расположен и после них. В первом случае достоверность сигнала выбора микросхемы устанавливается как можно раньше, но он должен быть зафиксирован вместе с адресом. Это означает, что количество требуемых фиксато-

ров адреса определяется местом расположения селектора адреса, так же как и количеством адресных битов и сигналами выбора микросхем, гребуемых для интерфейса. Сигналы выбора микросхем посылаются в логику управления магистралью, которая устанавливает текущее количество состояний ожидания для устройства, к которому осуществляется доступ.

Приемопередатчики данных. Стандартные 8-разрядные приемопередатчики (рис. 5.5) обеспечивают изоляцию и дополнительную нагрузочную способность для шины данных МП 80386. Они предотвращают конфликт на шине данных, который возникает, если некоторые медленно действующие устройства не удаляют с магистрали считываемые данные после цикла чтения. Если за циклом чтения следует цикл записи, то МП 80386 может занять шину данных до того, как медленно действующее устройство полностью удалит с нее свои данные. Это может привести к конфликту на шине.

Интерфейс с магистралью включает приемопередатчики в количестве, достаточном для сопряжения устройства с максимальным количеством входов или выходов с шиной данных. Например, если максимальная разрядность устройств равна 16 бит и адреса ввода-вывода расположены так, что все устройства подключаются только к младшей половине шины данных, требуются только два 8-разрядных приемопередатчика. Обычно в системе с МП 80386 используется 32-разрядная память, которая требует четырех 8-разрядных приемопередатчиков.

Логика управления магистралью. Логика управления магистралью для основного интерфейса ввода-вывода такая же, как и для интерфейса с памятью. Контроллер магистрали декодирует выходы состояния МП 80386 (W/R#, M/IO#, D/C#) и активизирует командный сигнал для требуемого магистрального цикла. Контроллер магистрали управляет также входом READY# МП 80386, который завершает каждый цикл магистрали. ПЛМ-2 (ПЛМ управления магистралью) подсчитывает число состояний ожидания и возвращает сигнал готовности READY# в активное состояние после того числа состояний ожидания, которое требуется устройству для осуществления соответствующего доступа.

Глава 6

Привилегии и защита, конвейеризация, многозадачность и мультиобработка

В этой главе описывается, как МП 80386 работает в режиме конвейера, в многозадачном режиме с его связанными сегментами состояния задачи (TSS) и дескрипторами и в режиме мультиобработки. Сначала дадим определение некоторым дополнительным терминам, используемым в главе.

Привилегии. Это свойство (обычно устанавливаемое при проектировании системы), которое определяет, какие компьютерные операции разрешаются в любой момент времени и какие доступы к памяти законны. Привилегии используются для обеспечения безопасности в компьютерной системе.

Уровень привилегии PL. Один из четырех уровней привилегий в МП 80386. Уровень 0 является наиболее привилегированным уровнем, а уровень 3 – наименее привилегированным.

Уровень привилегий инициатора запроса RPL. Инициатор запроса – это программа или устройство, которое хочет использовать системные ресурсы. Уровень RPL – это уровень привилегий, запрашиваемый абонентом селектора. Уровень RPL определяется двумя младшими битами селектора.

Текущий уровень привилегий CPL. Это уровень привилегий, на котором в данный момент исполняется задача. Уровень CPL равен обычно уровню привилегий исполняемого сегмента программы. Он может быть определен также проверкой двух младших битов регистра CS, за исключением случаев согласованных сегментов программы.

Уровень привилегий дескриптора DPL. Это наименее привилегированный уровень, на котором задача может получить доступ к данному дескриптору и сегменту, связанному с этим дескриптором. Уровень DPL определяется битами 6 и 5 в байте прав доступа дескриптора.

Эффективный уровень привилегий EPL. Это наименее привилегированный уровень из уровней RPL и DPL. Заметим, что меньшие значения уровня привилегий соответствуют бóльшим привилегиям. Уровень EPL – это максимальное значение из чисел RPL и DPL.

6.1. Привилегии и защита

Привилегии и защита являются средствами управления доступом к программам операционной системы и к данным. Безопасность системы поддерживается на уровне целостности информации. Привилегии и защита стали жизненно необходимы программистам в условиях множественного использования оборудования.

6.1.1. Привилегии

Концепция привилегий является центральной для нескольких аспектов защиты. В приложении к процедурам привилегии являются степенью доверия данной процедуры, при которой не делаются ошибки, могущие повлиять на другие процедуры или данные. В приложении к данным привилегии являются той степенью защиты, которой следует обладать этой структуре, чтобы не пострадать от процедур с меньшей степенью доверия.

6.1.2. Уровни и правила привилегий

Микропроцессор 80386 имеет четыре уровня защиты для оптимизации поддержки многозадачности. Привилегии реализуются путем присвоения значения от нуля до трех ключевым объектам, которые опознаются процессором. Это значение называется уровнем привилегий. Значение нуль соответствует наибольшим привилегиям, тогда как значение три – наименьшим. Ниже перечислены эти ключевые объекты.

Уровень привилегий дескриптора DPL. Дескрипторы содержат поле, называемое DPL. Это наименьшие привилегии, которые должна иметь задача для доступа к дескриптору.

Уровень привилегий инициатора запроса RPL. Уровень RPL представляет собой уровень привилегий, запрашиваемый процедурой, которая является источником селектора.

Текущий уровень привилегии CPL. Обычно уровень CPL равен сегментному уровню DPL того сегмента программы, который в данный момент исполняет процессор. Уровень CPL изменяется, когда управление передается сегментам с различными уровнями DPL.

На рис. 6.1 показано, как четыре уровня привилегий интерпретируются в виде колец защиты. Центр – уровень 0 – предназначен для сегментов, содержащих наиболее критичные программы (обычно ядро операционной системы). Внешние кольца предназначены для сегментов с менее критичными программами или данными. Использование всех четырех уровней привилегий не является необходимым. Существующие программы, спроектированные с меньшим количеством уровней, могут просто игнорировать другие допустимые уровни. Рекомендуется, чтобы одноуровневая система использовала нулевой уровень привилегий. Двухуровневой системе следует использовать уровни привилегий нуль и три. Если у вас появятся

Рис. 6.1. Уровни привилегий.

новые соображения, то вы сможете перемещаться вверх на уровень ядра или вниз на уровень прикладных программ, расширяя проект до трех или четырех уровней.

Микропроцессор 80386 автоматически проверяет право процедуры на доступ к другому сегменту путем сравнения уровня CPL процедуры с одним или более уровнями привилегий. Эта проверка выполняется во время загрузки селектора дескриптора в регистр сегмента.

6.1.3. Привилегированные команды

Привилегированные команды, которые влияют на системные структуры данных, могут использоваться, только когда уровень CPL равен нулю. Если процессор обнаруживает одну из этих команд в то время, когда уровень CPL больше нуля, то он вызывает обработку исключения по общей защите. Этими командами, более подробно описанными в гл. 10, являются:

CLTS	Сброс флага переключения задачи
HLT	Останов
LGDT	Загрузка таблицы глобальных дескрипторов
LIDT	Загрузка таблицы дескрипторов прерываний
LLDT	Загрузка таблицы локальных дескрипторов
LMSW	Загрузка слова состояния машины
LTR	Загрузка регистра задачи
MOV to/from CRn	Пересылка в/из регистра управления CRn
MOV to/from DRn	Пересылка в/из регистра отладки DRn
MOV to/from TRn	Пересылка в/из регистра проверки TRn

6.1.4. Чувствительные команды

На команды, относящиеся к вводу-выводу, необходимо налагать ограничения для того, чтобы защитить программы операционной системы и различные уровни данных. Однако при необходимости должна существовать возможность их выполнения процедурами, которые исполняются на уровнях привилегий, отличных от нуля. Это может уменьшить необходимость в переключениях состояния.

6.1.5. Защита

Для облегчения обнаружения и идентификации ошибок МП 80386 сконструирован со встроенными методами защиты. Основной единицей защиты является сегмент, и дескрипторы сегментов хранят необходимые параметры защиты. На рис. 6.2 показаны имеющие отношение к защите поля дескрипторов сегментов.

Поля типа сегментов данных и исполняемой программы содержат биты, которые в дальнейшем определяют цель использования данного сегмента. Бит 9 интерпретируется по-разному в сегменте исполняемой программы и в сегменте данных.

1. *Бит разрешения чтения.* В сегменте исполняемой программы бит 9 показывает, разрешается ли командам считывать данные из этого сегмента. Чтение из исполнительного сегмента может быть осуществлено либо через регистр CS путем использования префикса «CS», либо загрузкой селектора его дескриптора в один из регистров сегмента данных (DS, ES, FS, GS).

2. *Бит разрешения записи.* В сегменте данных бит 9 определяет, могут ли команды осуществлять запись в этот сегмент.

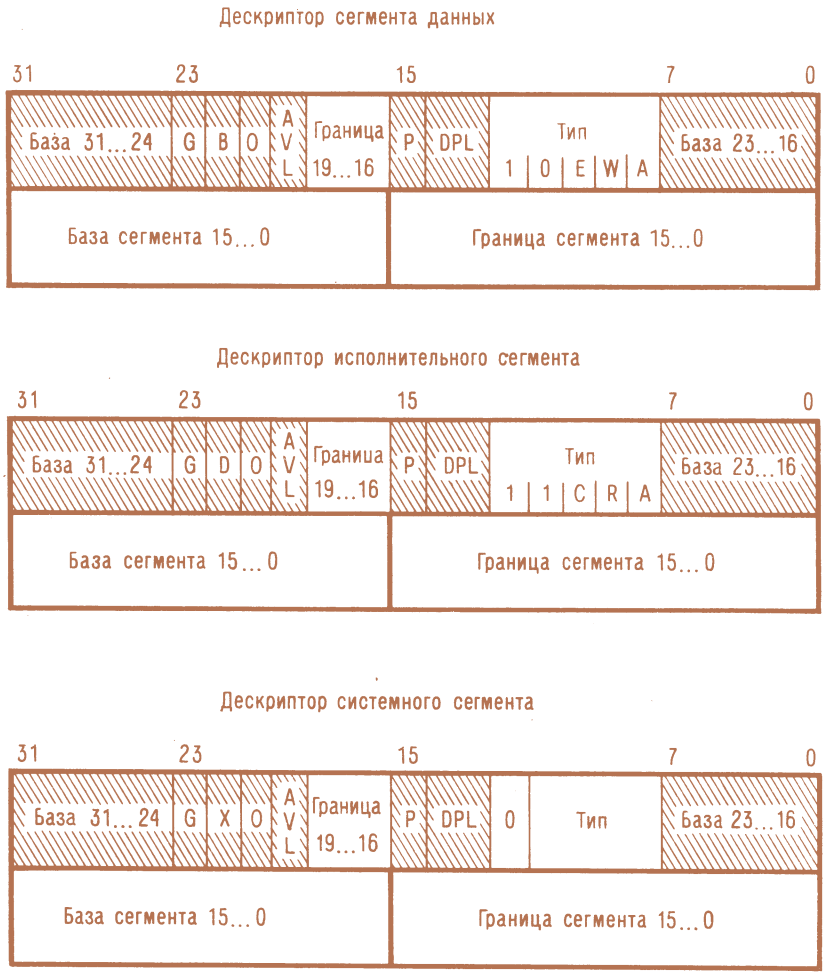


Рис. 6.2. Поля защиты дескрипторов сегментов.

Перед началом цикла обмена с памятью аппаратным путем проверяется каждая ссылка в память для верификации того, что эта ссылка удовлетворяет критерию защиты. Любое нарушение предотвращает начало цикла обмена и приводит к обработке исключения. Так как эти проверки и трансляция адреса осуществляются одновременно друг с другом, то потери производительности малы или отсутствуют.

Параметры защиты запоминаются в дескрипторах сегментов во время их создания системными программами. Когда программа загружает селектор в регистр сегмента, то МП 80386 загружает не только базовый адрес сегмента, но также информацию о защите. Каждый регистр сегмента в своей невидимой программисту части имеет биты для хранения базы, границы, типа и уровня привилегий. Любые последовательные проверки защиты того же самого сегмента не требуют дополнительных циклов обращения к памяти.

Защита имеет восемь аспектов: проверку типа, проверку границы, ограничение доступа к данным, ограничение точек входа в процедуру, ограничение набора

команд, проверку достоверности указателя и дескриптора, защиту страницы и сегмента.

Проверка поля TYPE. Поле TYPE (тип) дескриптора различно для дескрипторов разного формата и определяет желаемый способ использования сегмента. Проверка типа может быть использована для обнаружения программных ошибок, которые представляют собой попытки использовать сегмент способом, неожиданным для программиста. Процессор проверяет информацию о типе в двух случаях.

1. Если команда явно или неявно обращается к регистру сегмента. Некоторые сегменты могут использоваться командами только предопределенными способами. Например:
   - если бит разрешения записи не установлен, то ни одна команда не может записывать в сегмент данных;
   - до тех пор, пока не установлен бит разрешения чтения, ни одна команда не может читать из исполнительного сегмента;
   - ни одна команда не может записывать в исполнительный сегмент.
2. Если селектор дескриптора загружается в регистр сегмента. Некоторые регистры сегментов могут содержать дескрипторы только определенных типов. Например:
   - в регистр SS могут быть загружены селекторы сегментов данных только с разрешением записи;
   - регистр CS может загружаться селектором только исполнительного сегмента;
   - селекторы исполнительных сегментов с защитой от чтения не могут быть загружены в регистры сегментов данных.

Проверка границы. Проверка границы обнаруживает такие программные ошибки, как неверные вычисления значения указателя адреса или выход за допустимые пределы индексов массива. Эти ошибки обнаруживаются в момент возникновения, что существенно облегчает определение их причины. Без проверки границы такие ошибки могли бы исказить другие модули. Это искажение обнаружилось бы позднее по аномальной работе искаженного модуля, но идентифицировать ошибку в этот момент уже намного труднее.

В дескрипторе сегмента процессор использует поле LIMIT (граница) для того, чтобы предотвратить адресацию из программы в область вне сегмента. При интерпретации поля LIMIT процессор использует бит дробности G. Для сегментов данных процессор использует также бит направления расширения E и бит большого сегмента B. В табл. 6.1 приведены некоторые комбинации этих битов.

Если бит G = 0, границей является значение 20-разрядного поля границы дескриптора. Граница может изменяться от 0 до 0FFFFFH ($2^{20} - 1$ или 1 Мбайт). Если бит G = 1, то МП 80386 присоединяет 12 младших единичных битов к значению поля границы. В этом случае реальная граница может изменяться от 0FFFH ($2^{12} - 1$ или 4 Кбайт) до 0FFFFFFFH ($2^{32} - 1$ или 4 Гбайт).

Процессор использует поле LIMIT в таблицах дескрипторов для того, чтобы предотвратить выбор программами входа в таблицу, расположенного вне таблицы. Поле LIMIT определяет последний достоверный байт последнего дескриптора в этой таблице. Поскольку дескриптор имеет в длину восемь байтов, то значение границы для таблицы, содержащей N дескрипторов, равно (N * 8 – 1).

Свойство расширения вниз позволяет расширить стек путем копирования его в сегмент большего размера без необходимости обновления также и внутристековых указателей.

Для всех типов сегментов, кроме расширяемых вниз сегментов данных, значение границы на единицу меньше размера сегмента в байтах. В МП 80386 вызывается обработка исключения в любом из следующих случаев:

Таблица 6.1. Комбинации битов E, G, B

Случай	1	2	3	4
Направление расширения:	Вверх	Вверх	Вниз	Вниз
Бит G	0	1	0	1
Бит B	X	X	0	1
Нижняя граница:				
0	X	X		
LIMIT + 1			X	
shl(LIMIT, 12, 1) + 1				X
Верхняя граница:				
LIMIT	X			
shl(LIMIT, 12, 1)		X		
64K – 1			X	
4Г – 1				X
Максимальный размер сегмента:				
64K	X			
64K – 1			X	
4Г – 4K		X		
4Г				X
Минимальный размер сегмента:				
0	X		X	
4K		X		X

Примечание: shl(x, 12, 1) = сдвиг x влево на 12 бит со вставкой справа единичных битов. X – любое состояние бита.

– попытка обращения к двойному слову памяти по адресу, большему или равному значению LIMIT минус два;
– попытка обращения к слову памяти по адресу, большему или равному значению LIMIT;
– попытка обращения к байту памяти по адресу, большему значения LIMIT.

Для расширяемых вниз сегментов данных поле LIMIT имеет ту же самую функцию, но интерпретируется по-другому. Для этих сегментов данных диапазон достоверных адресов простирается от значения (LIMIT + 1) вплоть до 64K – 1 или 4Г – 1 в зависимости от значения бита B. Расширяемый вниз сегмент достигает максимального размера, когда значение LIMIT равно нулю. Бит B управляет также форматом обмена со стеком по умолчанию. Если B = 1, то используется 32-разрядное значение, иначе – 16-разрядное.

Ограничение доступа к данным. Перед тем как процессор сможет адресовать операнды в памяти, программа МП 80386 загружает селектор сегмента данных в один из регистров сегмента данных (DS, ES, FS, GS или SS). Затем МП 80386 сравнивает уровни привилегий для того, чтобы оценить возможность доступа к этим данным. Как показано на рис. 6.3, в эту проверку входят три различных уровня привилегий: текущий уровень привилегий (CPL), уровень привилегий инициатора запроса (RPL) селектора, использованного для спецификации целевого сегмента, и уровень привилегий дескриптора (DPL) целевого сегмента.

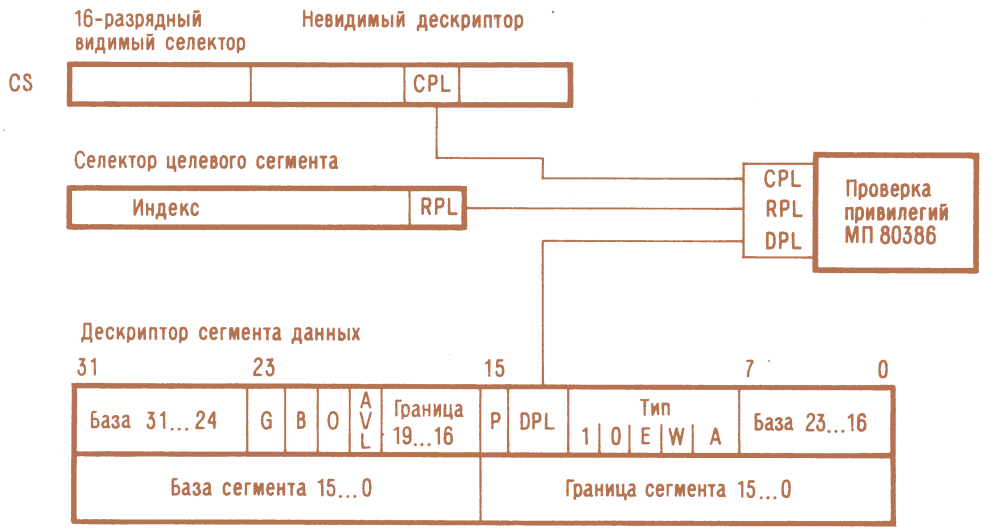


Рис. 6.3. Проверка привилегий доступа к данным.

Командам разрешается загружать регистр сегмента данных и получать доступ к соответствующему сегменту только в том случае, если уровень DPL целевого сегмента численно больше (меньший уровень привилегий) или равен большему из значений уровня CPL и уровня RPL селектора. Это может быть использовано для предотвращения изменения или чтения таблиц операционной системы прикладными программами.

Менее распространенным является использование сегмента программы для хранения данных. Сегменты программы могут содержать константы, однако невозможно записать что-либо в сегмент, определенный как сегмент программы. Существуют три способа чтения данных из сегмента программы

1. Использовать префикс «CS» для чтения разрешенного к чтению исполнительного сегмента, чей селектор уже загружен в регистр CS.

2. Загрузить один из регистров сегмента данных селектором несогласованного, разрешенного к чтению исполнительного сегмента.

3. Загрузить один из регистров сегмента данных селектором согласованного, разрешенного к чтению исполнительного сегмента.

Способ 1 можно использовать всегда, так как по определению уровень DPL сегмента программы по регистру CS равен уровню CPL. Способ 2 основан на тех же правилах, что и для доступа к данным. Способ 3 также всегда пригоден, так как уровень привилегий сегмента, чей бит согласованности установлен, совпадает с уровнем CPL независимо от уровня DPL.

Передачи управления. В МП 80386 управление передается путем использования исключений, прерываний, а также с помощью команд CALL, JMP, INT, IRET, RET. Исключения и прерывания являются специальными случаями и рассмотрены в гл. 3. В этом же разделе обсуждаются формы команд. Существуют как «ближние», так и «дальние» формы команд CALL, JMP, RET.

«Ближняя» форма передает управление внутри текущего сегмента программы и поэтому является предметом проверки только на превышение границы. Процессор просто удостоверяется, что приемник команды находится в пределах текущего исполнительного сегмента. Так как эта граница кэшируется в регистр CS, то

проверки защиты для ближних передач не требуют дополнительных тактов генератора.

«Дальние» команды CALL и JMP передают управление в другие сегменты, из-за чего МП 80386 проверяет уровни привилегий. Существуют два способа, которыми команда CALL или JMP может ссылаться на другой сегмент: операнд выбирает дескриптор вентиля вызова или дескриптор другого исполнительного сегмента.

Два разных уровня привилегий входят в проверку привилегий для передачи управления, которая не использует вентиль вызова: текущий уровень привилегий CPL и уровень привилегий дескриптора DPL целевого сегмента. Обычно уровень CPL равен уровню DPL исполняемого в данный момент сегмента. Однако уровень CPL может быть больше уровня DPL, если установлен бит согласованности в дескрипторе исполняемого сегмента.

Исполнительный сегмент, в дескрипторе которого бит согласованности (бит 10) установлен, называется согласованным сегментом. Согласованный сегмент позволяет использовать процедуры, которые могут вызываться с различных уровней привилегий, но должны исполняться на уровне привилегий вызывающей процедуры. Если управление передается в согласованный сегмент, уровень CPL не изменяется. Это единственный случай, когда уровень CPL может быть не равным уровню DPL исполняемого в данный момент сегмента.

Процессор позволяет осуществить командами CALL или JMP соответственно вызов или переход в другой сегмент, только если: установлен бит согласованности в дескрипторе целевого сегмента программы и уровень DPL целевого сегмента меньше или равен уровню CPL; или уровень целевого сегмента равен уровню CPL.

Большинство сегментов программы являются несогласованными. Для этих сегментов управление может быть передано к исполнительному сегменту только на том же самом уровне привилегий. Для передачи управления к численно меньшим уровням привилегий команда CALL используется с дескрипторами вентиля вызова. Команда JMP никогда не может передать управление к несогласованному сегменту, чей уровень DPL не равен уровню CPL.

С помощью дескрипторов вентиля МП 80386 обеспечивает защиту передач управления между исполнительными сегментами с различными уровнями привилегий. Существуют четыре типа вентилей: вентили задачи, вентили ловушки, вентили прерывания и вентили вызова. Вентили задачи используются для переключения задач и рассмотрены в другом разделе. Вентили ловушки и вентили прерывания используются исключениями и прерываниями и описаны в соответствующих разделах. В этом разделе мы рассмотрим только вентили вызова.

Вентиль вызова используется для описания уровня привилегий и определения точки входа в процедуру. Дескрипторы вентиля вызова используются командами CALL и JMP таким же способом, что и дескрипторы сегмента программы. Когда аппаратно выявляется, что селектор приемника ссылается на дескриптор вентиля, то команда исполняется в зависимости от содержимого вентиля вызова.

При использовании защиты гарантируется, что селектор и относительный адрес образуют указатель точки входа процедуры. Все передачи управления из другого сегмента направляются в достоверную точку входа, а не в середину процедуры или в середину команды, что возможно. Дело в том, что операционная система управляет всеми действиями в целом и может удостовериться, что будет исполняться программа, имеющая необходимую степень доверия. Это позволяет правильным образом предоставлять увеличивающиеся привилегии.

Процессор проверяет четыре уровня привилегий для санкционирования передачи управления через вентиль вызова: текущий уровень привилегий CPL, уровень привилегий дескриптора DPL вентиля, уровень привилегий инициатора запроса

RPL селектора, который определяет вентиль вызова, и уровень DPL дескриптора целевого исполнительного сегмента.

Из всех команд только CALL может использовать вентили для передачи управления к меньшим уровням привилегий. Командой JMP вентиль может быть использован только для передачи управления к исполнительному сегменту с тем же самым уровнем привилегий или же к согласованному сегменту.

Межуровневая передача запрашивается в том случае, если сегмент программы, являющийся пунктом назначения вентиля вызова, имеет уровень привилегий, отличный от уровня CPL. Для поддержания целостности системы каждый уровень привилегий имеет отдельный стек. Эти стеки гарантируют, что для вызовов процессов с меньших уровней привилегий зарезервирована достаточная по объему область стека. Без такого стека процедура с достаточным уровнем доверия работала бы неправильно, в случае если вызываемой процедуре не обеспечивается достаточное пространство в стеке вызывающей процедуры.

Процессор находит эти стеки через сегмент TSS. Поскольку каждая задача имеет отдельный сегмент TSS, то она соответственно имеет и отдельный стек. Системные программы создают сегменты TSS и располагают в них правильные указатели стеков; начальные указатели являются значениями, разрешенными только к чтению. Процессор никогда не изменяет их во время исполнения программы. Системные программы гарантируют также, что каждый стек обладает пространством, достаточным для размещения полного старого указателя стека SS:ESP (указателя сегмента стека и расширенного указателя стека), адреса возврата, всех параметров и локальных переменных, которые могут быть нужны для вызова процесса.

Если для изменения уровня привилегий используется вентиль вызова, то новый стек выбирается путем загрузки значения указателя из сегмента TSS. Процесс использует уровень DPL целевого сегмента программы (который является новым уровнем CPL) для присвоения начальному указателю стека уровня привилегий 0, 1 или 2. Сегмент TSS не может иметь указателя стека с уровнем привилегий 3 потому, что этот уровень не может быть вызван ни одной процедурой с другим уровнем привилегий. Если уровень DPL нового сегмента стековых данных не равен уровню CPL, то возникает обработка исключения по стеку.

Для облегчения предоставления привилегий МП 80386 копирует параметры в новый стек. Он использует поле счетчика в вентиле вызова для указания, сколько двойных слов (максимум равен 31) нужно скопировать из стека вызывающей процедуры в новый стек. Если поле счетчика равно нулю, то копирование параметров не производится. Процедуры, вызванные с другого уровня привилегий и требующие копирования более 31 двойного слова, должны использовать сохраняемую связь в виде указателя стека SS:ESP для того, чтобы получить доступ ко всем параметрам за последним скопированным двойным словом.

«Ближние» формы команды RET передают управление внутри текущего сегмента программы. Поэтому они являются предметом проверки только на превышение границы. Для этого из стека выбирается относительный адрес команды, следующей за командой CALL, и процессор проверяет, превышает или нет этот адрес границу текущего исполняемого сегмента.

«Дальняя» форма команды RET в свою очередь выбирает из стека указатель возврата, который был в него загружен предыдущей «дальней» командой CALL. Обычно этот указатель возврата бывает достоверным благодаря его связи с предыдущей командой CALL или INT. Процессор, несмотря на это, выполняет проверку привилегий из-за того, что текущая процедура может неверно обслуживать стек или изменить его указатель. Уровень RPL селектора сегмента программы, выбираемый из стека по команде возврата, определяет уровень привилегий вызываемой процеду-

ры. Межсегментная команда возврата может изменить уровни привилегий, но только в направлении процедур с меньшими привилегиями.

Ограничение набора команд. Определенные команды могут повлиять на механизм защиты и вмешаться в обычный порядок работы системы. Эти команды могут исполняться только процедурами с достаточным уровнем доверия. Процессор имеет два класса команд ограниченного применения: привилегированные команды и чувствительные команды. Более подробно эти команды описаны в предыдущем разделе, касающемся привилегий.

Проверка достоверности указателя. Проверка достоверности сводится к нахождению границ, в которых могут находиться данные. Достоверность указателей является важной частью выявления программных ошибок. Она необходима также для поддержания изоляции между уровнями привилегий и состоит из следующих трех шагов: 1) проверки соответствия типа сегмента его ожидаемому использованию, 2) проверки превышения указателем границы сегмента, 3) проверки полномочий задатчика указателя обращаться к данному сегменту.

Процессор автоматически выполняет первый и второй шаги проверки во время исполнения команды, но в выполнении третьего шага ему должны помочь программные средства. Это делает непривилегированная команда ARPL настройки поля RPL селектора. Выполнить первый и второй шаги проверки можно также и программными средствами, а не ожидать обработки исключения. Сделать это помогают непривилегированные команды LAR загрузки байта прав доступа, LSL загрузки границы сегмента, VERR верификации сегмента для чтения, VERW верификации сегмента для записи. Команда LAR верифицирует, что указатель ссылается на сегмент с соответствующими уровнями привилегий и типов. Команда LSL позволяет программно проверить границу дескриптора. Команды VERR и VERW определяют, указывает ли селектор на сегмент, из которого можно считывать или в который можно записывать на текущем уровне привилегий. Ни одна из этих команд не вызывает обработки исключения по защите в случае отрицательного результата проверки.

Уровень привилегий инициатора запроса RPL может иметь своей целью предотвращение ложного использования указателей, которые с меньшего уровня привилегий могут исказить данные или работу программы с большими привилегиями. Поле RPL позволяет присвоить селектору атрибут привилегий, который обычно показывает уровень привилегий программы, порождаемой этим селектором. Процессор автоматически проверяет поле RPL любого селектора, загружаемого в один из регистров сегмента, для верификации того, что уровень RPL позволяет осуществить доступ к этому сегменту.

Защита страниц и каталогов. Каждый вход в таблицу страниц имеет два бита, связанных с типом защиты: U/S и R/W. Бит присутствия может быть использован для ограничения адресуемой области. На уровне адресации страниц определены два типа доступа: доступ только по чтению (R/W = 0) и доступ по чтению и по записи (R/W = 1).

Если процессор работает на уровне супервизора (уровень CPL меньше 3), состояния битов U/S и R/W игнорируются. Все страницы разрешены как по чтению, так и по записи. На уровне пользователя (уровень CPL равен 3) записывать можно только те страницы, которые принадлежат уровню пользователя (U/S = 1) и помечены как доступные по записи (R/W = 1). Страницы, принадлежащие уровню супервизора, не доступны с уровня пользователя ни по чтению, ни по записи. «Право собственности» на страницы устанавливается через ограничение доступа к страницам.

Ограничение доступа к страницам реализовано присвоением каждой странице одного из двух уровней.

1. Уровень пользователя (бит 2 входа в таблицу страниц, бит U/S = 1). Этот уровень предназначен для прикладных программ и данных.

2. Уровень супервизора (бит 2 входа в таблицу страниц, бит U/S = 0). Этот уровень предназначен для операционной системы и других системных программ, а также связанных с ними данных.

Текущий уровень (пользователя или супервизора) относится к уровню CPL. Если уровень CPL равен 0, 1 или 2, процессор работает на уровне супервизора. Если же уровень CPL равен 3, процессор работает на уровне пользователя. На уровне супервизора все страницы адресуемы, на уровне пользователя адресовать можно только страницы, принадлежащие этому уровню.

Процессор вычисляет эффективные атрибуты защиты путем проверки атрибутов защиты как в каталоге страниц, так и в таблице страниц. Для любой страницы атрибуты защиты при обращении к каталогу страниц могут отличаться от атрибутов при обращении к таблице страниц.

Существуют возможности обойти защиту страниц. Определенные доступы проверяются так, как будто они являются ссылками с уровнем привилегий 0, даже если уровень CPL равен 3. К таким доступам относятся любые ссылки на таблицы LDT, GDT, IDT, TSS или же доступ к внутреннему стеку в кольце чередующихся команд CALL и INT.

В режиме разбиения на страницы процессор сначала проверяет защиту сегментов, а затем оценивает защиту страниц. Если МП 80386 обнаруживает нарушение защиты либо на уровне сегментов, либо на уровне страниц, он запрещает дальнейшее выполнение запрашиваемой операции и вызывает обработку исключения по защите.

6.2. Конвейеризация с целью повышения производительности

Производительность системы определяется тем, насколько быстро микропроцессорная система выполняет данную задачу или набор команд. Благодаря увеличенной скорости обработки использование МП 80386 как центрального узла системы может увеличить общую производительность. Одним из главных преимуществ конвейеризации команд является прозрачность ее для пользователя. При правильной реализации производительность команд может быть существенно увеличена без затрагивания уже написанных программ. Дополнительная информация по конвейеризации приведена в гл. 14.

Так как система может включать в себя устройства с меньшим быстродействием по сравнению с циклом магистрали МП 80386, то производительность системы в целом часто меньше, чем позволяет потенциал МП 80386. Поэтому для более эффективного использования медленных устройств применяют два метода: конвейеризация адресов и уменьшение частоты системного тактового генератора.

6.2.1. Конвейеризация адресов

Фактически исполнение команды разделяется на четыре этапа. Во-первых, команда считывается из памяти. Во-вторых, она интерпретируется или декодируется. В-третьих, из памяти считываются операнд или операнды. Наконец, команды выполняются. Если все эти операции выполняются друг за другом для последовательных команд (как на сборочной линии), то следующая команда выбирается из памяти в то время, когда первая команда декодируется и т. д. Можно добиться 100%-ной эффективности конвейера, так что дополнительное время для завершения

исполнения команды может быть уменьшено почти только до времени исполнения команды.

Конвейер адресов увеличивает время, которое отводится памяти для отклика на один такт CLK без затягивания цикла магистрали. Этот дополнительный такт CLK исключает выходную задержку на выходах адреса и состояния МП 80386. С точки зрения устройства памяти, к которому осуществляется доступ, конвейер адресов перекрывает по времени момент выдачи адреса и сигналов состояния следующего цикла магистрали. Обращение к памяти, которое требует двух состояний ожидания без конвейера адресов, с конвейером адресов требует только одного состояния ожидания.

Конвейер адресов является преимуществом для большинства циклов магистрали, но, если следующий адрес не известен до окончания текущего цикла, МП 80386 не может его обрабатывать в режиме конвейера. В этом случае последовательность событий на магистрали идентична циклу магистрали без конвейера. Точно так же первый цикл после занятого состояния магистрали не включается в конвейер потому, что нет предыдущего цикла, в котором ранее был бы сформирован адрес. Если следующий цикл должен быть конвейеризован, то первый цикл необходимо удлинить по крайней мере на одно состояние ожидания для того, чтобы на шину можно было выдать новый адрес до окончания этого цикла.

Конвейер адресов менее эффективен для УВВ, требующих нескольких состояний ожидания. Чем больше требуется состояний ожидания, тем менее существенным становится сокращение на одно состояние ожидания в конвейере адресов. Этот факт вместе с относительно нечастыми обращениями УВВ означает, что конвейер адресов для УВВ оказывает незначительное влияние на производительность системы.

В МП 80386 возможны как конвейеризированные, так и неконвейеризированные магистральные циклы. Если выбирается конвейер адресов, МП 80386 перекрывает по времени циклы магистрали, что позволяет получить большее время доступа. Поскольку обращение к кэш-памяти не требует состояний ожидания, то часто предпочитают неконвейеризированные циклы. Использование таких циклов минимизирует время между запросом процессором информации из внешнего мира и получением достоверных данных на выводах МП 80386. Так как выигрыш в производительности из-за применения кэш-памяти происходит благодаря ее способности обеспечивать передачу 32-разрядного двойного слова за 2 такта генератора, то применение конвейера адресов рекомендуется только для систем с медленными устройствами памяти, которые в противном случае потребовали бы одного или нескольких состояний ожидания.

6.2.2. Замедление микропроцессора

Кроме конвейеризации команд, которая является одним из способов, позволяющим медленным устройствам подстраиваться под быстрый МП 80386, существует менее распространенный способ подстраивания под быстродействие памяти – это уменьшить рабочую частоту МП 80386, т. е. замедлить его. Тактовая частота – это периодические импульсы, которые используются для упорядочения операций компьютера. Тактовые генераторы вырабатывают тактовые и фазирующие сигналы. Частота, с которой генерируются эти сигналы, определяет частоту, с которой переключаются логические вентили и происходит перемещение информации в компьютерной системе.

Поскольку более низкая тактовая частота увеличивает длительность цикла магистрали, то для конкретного устройства памяти может потребоваться меньше состояний ожидания. С другой стороны, производительность системы непосредст-

Таблица 6.2. Относительное количество состояний ожидания как функция рабочей частоты

Количество состояний ожидания	Без конвейера на частотах		С конвейером на частотах	
	16 МГц	12,5 МГц	16 МГц	12,5 МГц
0	1,00	0,78	0,91	0,71
1	0,81	0,64	0,76	0,59
2	0,66	0,52	0,63	0,49
3	0,57	0,45	—	—

венно зависит от тактовой частоты. Время исполнения пропорционально периоду тактового генератора. При тактовой частоте 12,5 МГц МП 80386 требуется почти на 33% больше времени для исполнения программы, чем при частоте 16 МГц с одинаковым количеством состояний ожидания (табл. 6.2).

6.3. Многозадачность

Многозадачность – это метод, с помощью которого управляют компьютерной системой, выполняющей несколько видов работ, таких как редактирование файла, компиляции программы или выполнение внутрисистемных передач. Отдельные задачи выполняются так, как если бы они выполнялись на специализированных процессорах с общей памятью. Другими словами, за исключением пауз для взаимодействия или синхронизации с другими задачами, каждая задача работает параллельно с другими задачами. В МП 80386 имеются аппаратные средства для поддержания эффективной многозадачности.

Для эффективного переключения с задачи на задачу МП 80386 использует специальные быстродействующие устройства, благодаря которым для такого переключения требуется всего лишь единственная команда или прерывание. При работе на тактовой частоте 16 МГц процессор может сохранить состояние одной задачи (все регистры), загрузить состояние другой задачи и возобновить исполнение всего лишь за 16 мкс.

Для управления многозадачностью МП 80386 не использует специальных команд. Вместо этого он по-другому интерпретирует обычные команды передачи управления в тех случаях, когда они ссылаются на специальные структуры данных. Регистрами и структурами данных, которые поддерживают многозадачность, являются сегмент состояния задачи TSS, дескриптор TSS, регистр задачи и дескриптор вентиля задачи.

В дополнение к простому переключению задач МП 80386 обеспечивает два других свойства управления задачами.

1. При каждом переключении задачи процессор может переключиться на другую таблицу LDT и на другой каталог страниц. Таким образом, каждая задача может иметь различные отображения логического адреса в линейный и линейного адреса в физический. Благодаря этому свойству задачи могут быть изолированы друг от друга и предотвращено их взаимное влияние.

2. Если по проекту системы необходимо, то прерывания или исключения могут вызвать переключение задачи. Процессор не только переключается на задачу, которая обслуживает это прерывание или исключение, но и автоматически пере-

ключается обратно после того, как данное прерывание или исключение было обработано.

В действительности многозадачность моделирует многие процессоры, обеспечивая каждую задачу виртуальным процессором. То есть в каждом конкретном случае операционная система назначает реальный процессор одному из виртуальных процессоров, который затем выполняет задачу данного виртуального процессора. Для того чтобы сделать это, МП 80386 использует сегменты TSS и команды, которые переключают задачи.

6.3.1. Сегмент состояния задачи

Сегмент TSS – это структура данных, которая характеризует состояние виртуального процессора задачи. Сегмент TSS состоит из двух частей. Первой частью является динамический набор, обновляемый процессором при каждом переключении задачи. Этот набор включает поля, содержимое которых следующее:

- регистры сегментов – GS, FS, DS, SS, CS, ES;
- регистры общего назначения – EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX;
- регистр флагов – EFLAGS;
- указатель команд – EIP;
- селектор сегмента TSS предыдущей исполняемой задачи. Он обновляется только тогда, когда ожидается возврат.

Второй частью сегмента TSS является статический набор, который процессор считывает, но не изменяет. Этот набор включает поля, которые хранят следующее:

- базу карты ввода-вывода;
- бит ловушки для отладки, бит T, который принуждает МП 80386 выполнять обработку исключения для отладки при возникновении переключения задачи;
- селектор таблицы LDT задачи;
- определение стека для программ обработки прерывания на уровнях 0, 1 или 2, которые должны исполняться в среде задачи;
- регистр управления CR3, который содержит базовый адрес каталога страниц (PDBR) задачи. Он считывается только в том случае, когда включено разбиение на страницы.

Сегмент TSS может располагаться в любом месте пространства линейных адресов. Осторожность следует соблюдать в том случае, когда сегмент TSS пересекает границу страниц и страница со старшими адресами отсутствует. При этом МП 80386 вызывает обработку исключения, если обнаруживает неприсутствующую страницу при чтении TSS во время переключения задачи. В последнем случае, когда обе страницы отсутствуют, программа обработки отказа по странице делает обе страницы присутствующими перед рестартом команды, которая вызвала отказ по странице. На рис. 6.4 показан 32-разрядный сегмент состояния задачи МП 80386.

При создании новой задачи операционная система создает сегмент TSS и присваивает ему значения, которые должна иметь задача в начале своего исполнения. При изменении любого из этих значений информация обновляется.

6.3.2. Дескриптор сегмента состояния задачи

Подобно всем сегментам, сегмент TSS определяется дескриптором. Этот дескриптор находится только в таблице GDT. Попытка идентифицировать сегмент TSS селектором, который имеет бит TI = 1 (флаг индикатора таблицы, означающий текущую таблицу LDT), вызывает обработку исключения. Точно так же, даже если процедура имеет доступ к дескриптору TSS, она не имеет права считывать или

31 — 23	15 — 7	0	
База карты ввода-вывода	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	T	64
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	LDT		60
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	GS		5C
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	FS		58
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	DS		54
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SS		50
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	CS		4C
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	ES		48
EDI			44
ESI			40
EBP			3C
ESP			38
EBX			34
EDX			30
ECX			2C
EAX			28
EFLAGS			24
Указатель команд (EIP)			20
CR3 (PDBR)			1C
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SS2		18
ESP2			14
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SS1		10
ESP1			0C
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SS0		8
ESP0			4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Обратная ссылка на предыдущий сегмент		0

Рис. 6.4. Сегмент состояния задачи МП 80386.

31 — 23					15		7		
База 31...24	G	0	0	AVL	Граница 19...16	P	DPL	Тип 0 1 0 B 1	База 7...0	4
База 15...0					Граница15...0					0

Рис. 6.5. Дескриптор сегмента состояния задачи.

модифицировать сегмент TSS. Чтение или изменение может быть сделано только с помощью другого дескриптора, который переопределяет сегмент TSS как сегмент данных. Попытка загрузить дескриптор TSS в один из регистров сегментов (CS, DS, ES, FS, GS, SS) вызывает обработку исключения. На рис. 6.5 показан формат дескриптора TSS.

Задачи не являются реентерабельными, так как и селектор LDT, и регистр CR3 для задачи запоминаются в сегменте TSS. Бит занятости B поля типа позволяет

процессору обнаруживать попытку переключения на задачу, которая еще занята. Код 9 в поле типа показывает, что задача не занята. Код 11 в поле типа показывает, что она занята.

Поля базы, границы и уровня привилегий дескриптора TSS, а также биты дробности G и присутствия P имеют функции, подобные их аналогам в дескрипторах сегментов данных. База определяет место расположения сегмента в пространстве линейных адресов. Граница определяет размер сегмента. В дескрипторе TSS граница должна содержать значение 103 или больше, потому что сам сегмент TSS требует 104 байт. Попытка переключиться на задачу, у которой граница меньше 103, вызывает обработку исключения.

6.3.3. Переключение задач

Процессор составляет расписание задач и выполняет задачи на основе набора приоритетов операционной системы. Для этого МП 80386 использует регистр задачи TR, в котором содержит селектор и дескриптор для сегмента TSS текущей задачи. Регистр TR содержит как видимую, так и невидимую части. Его видимая и изменяемая части могут быть прочитаны и модифицированы командами. Невидимая часть обслуживается процессором с тем, чтобы соответствовать изменяемой части, и не может быть считана ни одной командой.

Две команды (загрузка регистра задачи LTR и сохранение регистра задачи STR) позволяют считывать и модифицировать изменяемую часть регистра TR. Обе команды требуют одного операнда, которым является 16-разрядный селектор, расположенный в памяти или регистре общего назначения.

Команда LTR загружает изменяемую часть регистра задачи селектором-оператором, который должен выбрать дескриптор TSS в таблице GDT. При исполнении команды LTR загружается также невидимая часть регистра информацией из дескриптора TSS, выбранного операндом. Команда LTR является привилегированной, т. е. может исполняться только тогда, когда уровень CPL равен нулю. Обычно команда LTR используется во время инициализации системы для установки начального значения регистра задачи. После этого содержимое регистра TR изменяется операциями переключения задачи.

Команда STR сохраняет изменяемую часть регистра TR в регистре общего назначения или в слове памяти, она не является привилегированной.

Дескриптор вентиля задачи дает косвенную, защищенную ссылку на сегмент TSS. В дополнение к дескрипторам TSS МП 80386 использует вентили задачи для достижения следующих трех целей.

1. Поскольку бит занятости хранится в дескрипторе TSS, то каждая задача должна иметь только один такой дескриптор. Однако вентилей задачи может быть несколько, они выбирают единственный дескриптор TSS.

2. С помощью вентилей задачи системные программы могут ограничивать право вызывать переключения на специфичные задачи.

3. Вентили задачи могут также располагаться и в таблице IDT, что позволяет вызывать переключение задачи прерываниями и исключениями.

Для переключения задач операционная система выдает команды перехода JMP или вызова CALL, операндом которых является селектор для сегмента TSS или для вентиля задачи новой задачи.

Процессор сначала проверяет, возможно ли переключение текущей задачи на другую задачу. В случаях команд JMP или CALL применяются правила привилегий доступа к данным. Уровень DPL дескриптора TSS или вентиля задачи должен быть меньше или равен максимальному из уровней CPL и RPL селектора вентиля.

Затем проверяется дескриптор TSS с целью выяснения, является ли этот сегмент

присутствующим и имеет ли он достоверную границу. Обнаруживаемая вплоть до этой точки ошибка возникает в контексте прекращаемой задачи. После этих ошибок возможен рестарт, и они могут быть обработаны способом, который делает их прозрачными для прикладных программ.

Потом процессор исполняет команду JMP перехода к новому сегменту TSS, запоминая сначала текущее содержимое регистров в текущем сегменте TSS. Указатель команд EIP загружается адресом команды, следующей за той, которая вызвала переключение задачи. Затем он загружает регистр TR селектором, определенным в команде JMP, отмечает занятым дескриптор TSS включаемой задачи и устанавливает бит переключения задачи TS в слове состояния машины MSW. Так как теперь действует новый сегмент TSS, то МП 80386 загружает свои регистры значениями из этого нового сегмента TSS. Исполнение продолжается с команды, на которую показывает указатель команд новой задачи. Любая обнаруживаемая на этом шаге ошибка возникает в контексте включаемой задачи.

Для программы обработки исключения ошибка появляется так, как будто первая команда новой задачи еще не исполнялась. Программы, ориентированные на обработку исключений во включаемой задаче, не должны содержать действий, которые могли бы загрузить селектор, способный вызвать исключение. До тех пор, пока этот обработчик не проверит селектор и не разрешит любые потенциальные проблемы, такое действие может вызвать другое исключение.

Каждое переключение задачи устанавливает бит TS в слове MSW. Флаг TS полезен, когда используется сопроцессор, например арифметический. Бит TS показывает, что контекст сопроцессора может не соответствовать текущей задаче МП 80386.

Для возобновления выполнения старой задачи операционная система выдает команду JMP перехода к сегменту TSS старой задачи. Процесс запоминания текущего содержимого регистров, загрузки регистров новым содержимым и продолжения выполнения повторяется. Переключение задачи занимает по времени около 17 мкс.

Уровень привилегий, на котором осуществляется рестарт во включаемой задаче, не ограничен уровнем привилегий прекращаемой задачи. Задачи изолированы своими отдельными адресными пространствами и сегментами TSS, а для предотвращения неправильного доступа к сегментам TSS используются правила привилегий доступа. Таким образом, для поддержания определенных отношений между уровнями CPL отдельных задач не требуется никаких специальных правил привилегий. Новая задача просто начинает исполнение на уровне привилегий, определенном значением поля RPL селектора программного сегмента, загружаемого из сегмента TSS.

Команды JMP, CALL, IRET, а также прерывания и исключения – это все обычные механизмы, которые могут быть использованы, когда не требуется переключение задачи. Стандартный механизм отличается от механизма, который вызывает переключение задачи, либо типом ссылки дескриптора, либо битом вложенной задачи NT в слове флагов.

6.4. Мультиобработка

Мультиобработка – это параллельное выполнение нескольких программ или программных сегментов так, что процессор в каждый момент выполняет одну программу с помощью общих ресурсов, таких как память и устройства ввода-вывода.

Компоненты обычной мультиобрабатывающей системы МП 80386 включают сигнал LOCK#, командный префикс LOCK, программно управляющий сигналом

LOCK#, и автоматическую установку сигналов процессором при неявных обновлениях памяти.

6.4.1. Префикс LOCK и сигнал LOCK#

Командный префикс LOCK и его выходной сигнал LOCK# используются для предотвращения прерывания операции перемещения данных со стороны других системных элементов. Префикс LOCK имеет ограничения, по которым он может использоваться только со следующими шестнадцатью командами МП 80386. Использование префикса LOCK перед любой командой, отличной от приведенных ниже, приводит к обработке исключения по неразрешенной команде:

- команды обмена XCHG;
- команды проверки и изменения битов BT, BTC, BTR, BTS;
- однооперандные арифметические и логические команды DEC, INC, NEG, NOT;
- двухоперандные арифметические и логические команды ADD, ADC, AND, OR, SBB, SUB, XOR.

Блокировка от доступа процессора гарантируется только для области памяти, определяемой операндом с заданным начальным адресом и заданной длиной. Префикс LOCK может заблокировать и бóльшую область памяти. Например, конфигурации МП 8086 и МП 80286 полностью блокируют пространство физической памяти.

Когда МП 80386 инициирует активность на шине данных самостоятельно и хочет, чтобы она правильно завершилась, он выдает сигнал LOCK#. Обычно это происходит в следующих случаях.

1. Исполнение команд XCHG. Процессор всегда выдает сигнал LOCK# во время исполнения команды XCHG, даже если префикс LOCK не используется.

2. Загрузка дескрипторов. Когда МП 80386 копирует содержимое таблицы дескриптора в один из регистров сегмента, он выдает сигнал LOCK# с тем, чтобы дескриптор не модифицировался во время его загрузки.

3. Подтверждение прерывания. После запроса прерывания контроллер прерывания посылает в МП 80386 по шине данных идентификатор источника прерывания. Процессор выдает в это время сигнал LOCK# для того, чтобы отключить от этой шины другие источники данных.

4. Установка бита занятости дескриптора TSS. Процессор проверяет и затем устанавливает бит занятости в поле типа дескриптора TSS. Для того чтобы предотвратить от переключения на одну и ту же задачу двух различных процессоров, МП 80386 выдает сигнал LOCK# во время проверки и установки этого бита.

5. Обновление бита обращения A и бита мусора D таблицы страниц. Процессор выдает сигнал LOCK# при обновлении битов A и D входов в таблицу страниц. Заметим, что МП 80386 обходит кэш-память таблицы страниц и обновляет эти биты непосредственно в основной памяти.

Глава 7

Аппаратные средства и подсистемы

При оптимальном проектировании систем большое значение имеет интерфейс с аппаратурой. Часто знание программистами аппаратных средств и их внутреннего функционирования позволяет выбрать лучшую структуру программы или набор команд для увеличения эффективности системы. Зная работу аппаратных средств, программист может быстро определить, вызвана ли ошибка работой оборудования или же она находится в программном обеспечении, использующем это оборудование. В этой главе обсуждаются не рассмотренные ранее концепции и требования к аппаратным средствам.

Микропроцессор 80386 смонтирован в корпусе с высокой плотностью упаковки, так называемом корпусе с матричным расположением штырьковых выводов (рис. 7.1). Он имеет керамическое основание с шагом выводов 0,100 дюйм.

7.1. Контроллеры

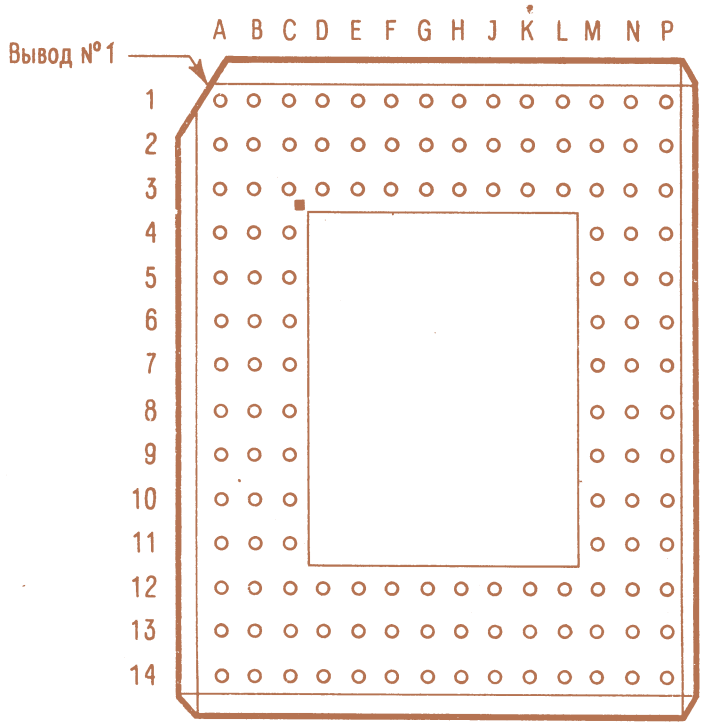
Контроллер – это элемент (или группа элементов) в компьютерной системе, который управляет последовательностью операций и посылает соответствующие сигналы к другим схемам компьютера для выполнения команд. В наше обсуждение включены контроллер прерываний 8259A, контроллер последовательной связи 8274 и контроллер прямого доступа к памяти типа 82258.

7.1.1. Контроллер последовательной связи 8274

Микросхема 8274 является мультипротокольным контроллером последовательной связи. Он сконструирован для интерфейса с высокоскоростными последовательными линиями связи, использующими различные протоколы связи: асинхронный, бисинхронный IBM и HDLC/SDLC (высокоуровневое управление передачей данных или управление синхронной передачей данных). Контроллер 8274 имеет два независимых дуплексных канала и может служить в качестве высокопроизводительного аналога двух универсальных синхронно-асинхронных приемопередатчиков типа 8251A. Более подробная информация по приемопередатчикам приведена в гл. 12. Контроллер 8274 допускает меньшее время между последовательными доступами по сравнению с основной аппаратурой интерфейса с вводом-выводом.

7.1.2. Контроллер прерываний 8259A

Программируемый контроллер прерываний 8259A управляет прерываниями в системе на основе МП 80386. Один контроллер 8259A может принимать прерывания от восьми внешних источников, а с помощью каскадирования нескольких таких



Список выводов по номерам

Вывод	Сигнал	Вывод	Сигнал	Вывод	Сигнал	Вывод	Сигнал
A1	V_{CC}	B1	V_{SS}	C1	A8	D1	A11
A2	V_{SS}	B2	A5	C2	A7	D2	A10
A3	A3	B3	A4	C3	A6	D3	A9
A4	Не подключен	B4	Не подключен	C4	A2	D12	V_{CC}
A5	V_{CC}	B5	V_{SS}	C5	V_{CC}	D13	NA#
A6	V_{SS}	B6	Не подключен	C6	Не подключен	D14	HOLD
A7	V_{CC}	B7	INTR	C7	Не подключен		
A8	ERROR#	B8	NMI	C8	PEREQ		
A9	V_{CC}	B9	BUSY#	C9	RESET	E1	A14
A10	V_{CC}	B10	W/R#	C10	LOCK#	E2	A13
A11	D/C#	B11	V_{SS}	C11	V_{SS}	E3	A12
A12	M/IO#	B12	Не подключен	C12	V_{CC}	E12	BE0#
A13	BE3#	B13	BE2#	C13	BE1#	E13	Не подключен
A14	V_{CC}	B14	V_{SS}	C14	BS16#	E14	ADS#
F1	A15	G1	A16	H1	A17	J1	A20
F2	V_{SS}	G2	V_{CC}	H2	A18	J2	V_{SS}
F3	V_{SS}	G3	V_{CC}	H3	A19	J3	V_{SS}
F12	CLK2	G12	V_{CC}	H12	D0	J12	V_{SS}
F13	Не подключен	G13	READY#	H13	D1	J13	V_{SS}
F14	V_{SS}	G14	V_{CC}	H14	D2	J14	D3

Рис. 7.1. Назначение выводов МП 80386.

		M1	A26	N1	A27	P1	A30
K1	A21	M2	A29	N2	A31	P2	V_{CC}
K2	A22	M3	V_{CC}	N3	V_{SS}	P3	D30
K3	A25	M4	V_{SS}	N4	V_{CC}	P4	D29
K12	D7	M5	D31	N5	D27	P5	D26
K13	D5	M6	D28	N6	D25	P6	V_{SS}
K14	D4	M7	V_{CC}	N7	V_{CC}	P7	D24
		M8	V_{SS}	N8	D23	P8	V_{CC}
L1	A23	M9	D20	N9	D21	P9	D22
L2	A24	M10	V_{SS}	N10	D17	P10	D19
L3	A28	M11	D15	N11	D16	P11	D18
L12	V_{CC}	M12	D10	N12	D12	P12	D14
L13	O8	M13	V_{CC}	N13	D11	P13	D13
L14	O6	M14	HLDA	N14	D9	P14	V_{SS}

Список выводов по функциям

Сигнал	Вывод	Сигнал	Вывод	Сигнал	Вывод	Сигнал	Вывод
A31	N2	D31	M5	ADS#	E13	V_{CC}	A1
A30	P1	D30	P3	BE3#	A13	V_{CC}	A5
A29	M2	D29	P4	BE2#	B13	V_{CC}	A7
A28	L3	D28	M6	BE1#	C13	V_{CC}	A10
A27	N1	D27	N5	BE0#	E12	V_{CC}	A14
A26	M1	D26	P5	BS16#	C14	V_{CC}	C5
A25	K3	D25	N6	BUSY#	B9	V_{CC}	C12
A24	L2	D24	P7	D/C#	A11	V_{CC}	D12
A23	L1	D23	N8	ERROR#	A8	V_{CC}	G2
A22	K2	D22	P9	LOCK#	C10	V_{CC}	G3
A21	K1	D21	N9	M/IO#	A12	V_{CC}	G12
A20	J1	D20	M9	NA#	O13	V_{CC}	G14
A19	H3	D19	P10	READY#	C13	V_{CC}	L12
A18	H2	D18	P11	W/R#	B10	V_{CC}	M3
A17	H1	D17	N10			V_{CC}	M7
A16	C1	D16	N11	CLK2	F12	V_{CC}	M13
A15	F1	D15	M11	HOLD	D14	V_{CC}	N4
A14	E1	D14	P12	HLDA	M14	V_{CC}	N7
A13	E2	D13	P13	INTR	B7	V_{CC}	P2
A12	E3	D12	N12	NMI	B8	V_{CC}	P8
A11	D1	D11	N13	PEREQ	C8	V_{SS}	A2
A10	D2	D10	M12	RESET	C9	V_{SS}	A6
A9	D3	D9	N14			V_{SS}	A9
A8	C1	D8	L13			V_{SS}	B1
A7	C2	D7	K12	Не подключен	A4	V_{SS}	B5
A6	C3	D6	L14	Не подключен	B4	V_{SS}	B11
A5	B2	D5	K13	Не подключен	B6	V_{SS}	B14
A4	B3	D4	K14	Не подключен	B12	V_{SS}	C11
A3	A3	D3	J14	Не подключен	C6	V_{SS}	F2
A2	C4	D2	H14	Не подключен	C7	V_{SS}	F3
		D1	H13	Не подключен	E13	V_{SS}	F14
		D0	H12	Не подключен	F13	V_{SS}	J2
						V_{SS}	J3
V_{SS}	M4	V_{SS}	M10	V_{SS}	P6	V_{SS}	J12
V_{SS}	M8	V_{SS}	N3	V_{SS}	P14	V_{SS}	J13

микросхем может быть обработано до 64 запросов прерывания. Контроллер выполняет функцию арбитра на основе приоритетов одновременных прерываний, останавливает процессор и выдает в МП 80386 код идентификации источника прерывания.

Одиночный и каскадированный контроллер прерываний. Когда возникает прерывание, то контроллер 8259А активирует свой собственный выход прерывания (INT). Этот выход подключается ко входу запроса прерывания МП 80386 (INTR). Процессор автоматически выполняет два следующих один за другим цикла подтверждения прерывания. Требования контроллера 8259А к синхронизации:

- между двумя циклами подтверждения должны быть вставлены четыре незанятых цикла магистрали, холостые циклы МП 80386 вставляет автоматически;
- каждый цикл подтверждения прерывания должен быть расширен по крайней мере на одно состояние ожидания, это расширение должно обеспечиваться логикой генератора состояний ожидания.

Несколько контроллеров 8259А могут каскадироваться для обработки до 64 запросов прерывания. В такой системе один контроллер назначается главным. Он принимает сигналы от других контроллеров, называемых подчиненными. Каждый подчиненный контроллер определяет наиболее приоритетный запрос из восьми запросов прерывания и передает его в главный контроллер. Главный контроллер в свою очередь определяет наиболее приоритетный запрос среди запросов подчиненных и передает его в МП 80386.

Для каскадированных контроллеров так же, как и для одиночного контроллера 8259А, существенной является синхронизация интерфейса. Во время первого цикла подтверждения прерывания все контроллеры 8259А замораживают состояния своих входов запроса прерывания. Когда главный контроллер выводит адрес каскада для выбора подчиненного контроллера с наивысшим приоритетом, последний выдает МП 80386 вектор прерывания. Это происходит во время второго цикла подтверждения прерывания МП 80386.

Если система на основе МП 80386 нуждается в более чем 64 линиях запроса прерывания, то может быть добавлен третий уровень контроллеров 8259А. При этом, когда один из подчиненных контроллеров третьего уровня принимает запрос прерывания, он делает активным один из входов запроса прерывания второго уровня. После разрешения приоритетов контроллер второго уровня посылает в МП 80386 вектор подпрограммы обслуживания. Подпрограмма обслуживания должна содержать команды опроса контроллеров прерывания третьего уровня для того, чтобы определить источник запроса прерывания. Дополнительные микросхемы 8259А и логика их выбора – это единственные средства, требуемые для обработки более 64 прерываний. Для максимальной производительности рекомендуется, чтобы контроллеры прерывания третьего уровня использовались лишь для некритичных, нечасто используемых прерываний.

7.1.3. Контроллер прямого доступа к памяти 82258

Контроллер прямого доступа к памяти (ПДП) управляет передачей данных методом ПДП между основной памятью и одним из устройств ввода-вывода (как правило, жестким диском), гибким диском или каналом связи. Передача методом ПДП происходит без вмешательства центрального процессора (ЦП). Выполнение функций ЦП другим системным элементом улучшает производительность системы в целом, так как ЦП не нужно контекстно переключаться при каждом обмене с памятью.

Усовершенствованный контроллер ПДП типа 82258 имеет четыре канала. Он вырабатывает все сигналы, необходимые для передач с ПДП. Его особенности:

- один из четырех высокоскоростных каналов может быть заменен за 32 низкоскоростных мультиплексных канала;
- автоматическое преобразование данных для передачи из 16-разрядной памяти в 8-разрядное устройство ввода-вывода и наоборот;
- функции сравнения, трансляции и верификации;
- возможность соединения команд в цепочку для исполнения множественных команд;
- возможность соединения данных в цепочку для распределения по отдельным местам памяти (например, отдельным страницам) и сбора их из этих мест.

В паре с контроллером магистрали 82288 контроллер 82258 может быть задатчиком общей локальной магистрали. Кроме того, его можно использовать и в качестве контроллера системной магистрали МП 80386. В первом случае необходим интерфейс с локальной магистралью, а во втором – интерфейс с МП 80386 для непосредственного взаимодействия с центральным процессором.

В большинстве случаев контроллер ПДП взаимодействует с МП 80386 через память. Иногда МП 80386 выполняет магистральные циклы непосредственно с контроллером 82258, например, когда осуществляет непосредственный доступ к нему для установки регистра общего режима во время инициализации. Этот доступ происходит асинхронно через интерфейс режима исполнителя контроллера 82258, который содержит четыре типа выводов: 1) выбор микросхемы CS#, который включает интерфейс режима исполнителя; 2) данные (D15 – D0), которые служат для передачи данных в и из контроллера 82258; 3) управление (RD# и WR#), которое показывает тип цикла магистрали; 4) адрес регистра (A7 – A0), который выбирает один из внутренних регистров контроллера 82258.

7.2. Сопроцессоры

Многие прикладные программы работают быстрее и более эффективно, если компьютер использует специализированные сопроцессоры (рис. 7.2). Сопроцессоры способны выполнять функции, которые в их отсутствии должны быть эмулированы программными средствами, к тому же они расширяют набор команд основного процессора.

7.2.1. Арифметические сопроцессоры 80287 и 80387

Интерфейс с арифметическим сопроцессором в МП 80386 поддерживает арифметические сопроцессоры 80287 и 80387, которые программно совместимы с ранней моделью 8087. В приложениях, где требуются вычисления с целыми числами высокой точности и с числами с плавающей точкой, оба сопроцессора полностью совместимы со стандартом IEEE для представления чисел с плавающей точкой.

МП 80386 предоставляет разработчикам систем выбор между низкой стоимостью или высокой производительностью арифметических решений. На частоте 16 МГц сопроцессор 80387 работает более чем в шесть раз быстрее сопроцессора 80287 на частоте 5 МГц, хотя последний является достаточно быстрым для большинства приложений и эффективным по стоимости решением для многих проектов.

Если МП 80386 обнаруживает команду, предназначенную для выполнения сопроцессором 80287, он связывается с сопроцессором, выполняет все необходимые магистральные циклы с памятью и передает данные в сопроцессор 80287 или получает данные из него по младшей половине шины данных. Процессор автоматически преобразует 32-разрядные данные в 16-разрядные и наоборот.

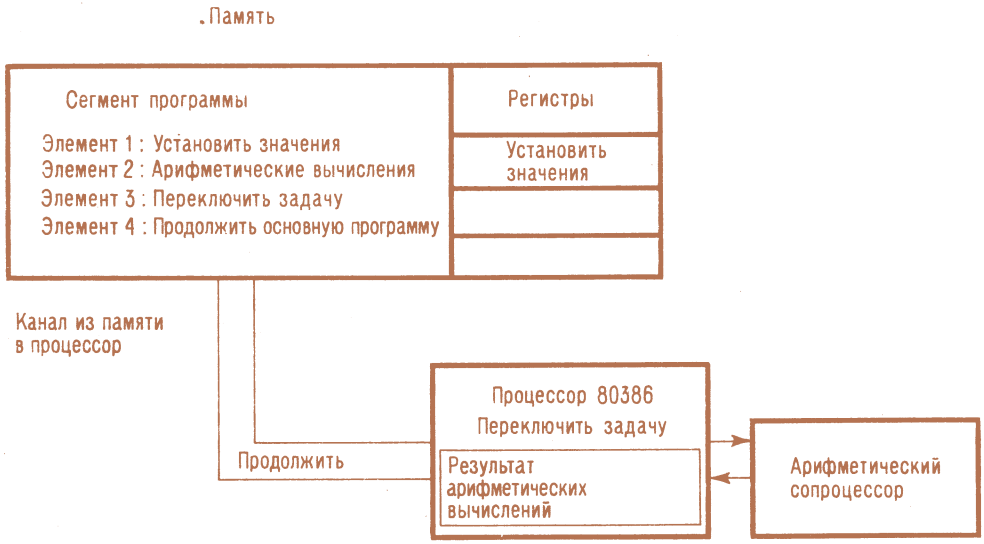


Рис. 7.2. Передача процессором арифметических вычислений сопроцессору. Процессор передал арифметическую операцию сопроцессору. В описании результата процессор выполняет следующий шаг программы.

Когда МП 80386 работает с сопроцессором 80387, он выполняет все необходимые магистральные циклы с памятью и передачи данных в 80387 и из него. Все передачи с сопроцессором 80387 являются 32-разрядными. Циклы чтения (передачи от сопроцессора 80387 к МП 80386) требуют по крайней мере одного состояния ожидания, тогда как циклы записи в сопроцессор 80387 не требуют состояний ожидания.

Сопроцессор 80387 имеет более высокую производительность и бóльшие возможности команд по сравнению с сопроцессором 80287. Он работает на внутренней тактовой частоте до 16 МГц. Для получения высокой производительности используется синхронный интерфейс с МП 80386 с полной 32-разрядной шиной данных. Сопроцессор 80387 сконструирован для полностью синхронной, либо псевдосинхронной работы с МП 80386. В псевдосинхронном режиме интерфейсная логика сопроцессора 80387 работает от тактового сигнала МП 80386, тогда как его внутренняя логика работает от другого тактового сигнала. На рис. 7.3 показано, как МП 80386 подключается к сопроцессору 80287, а именно:

- выходы BUSY#, ERROR#, PEREQ сопроцессора 80287 подключаются к соответствующим выводам МП 80386;
- вход RESET сопроцессора 80287 подключается к выходу RESET тактового генератора 82384;
- входы выбора микросхемы арифметического сопроцессора 80287 (NPS1#, NPS2) подключаются к фиксируемым выходам M/IO# и A31 соответственно;
- входы команды сопроцессора 80287 (CMD1 и CMD2) различают данные от команд. Эти входы подключаются к земле и к фиксируемому выходу A2 соответственно; (Примечание: МП 80386 выводит адрес 800000F8H при записи команды и адрес 800000FCH при записи или считывании данных.)
- младшая половина шины данных подключается к 16 разделам шины данных сопроцессора 80287. МП 80386 передает данные к и от сопроцессора только по линиям D15 – D0;

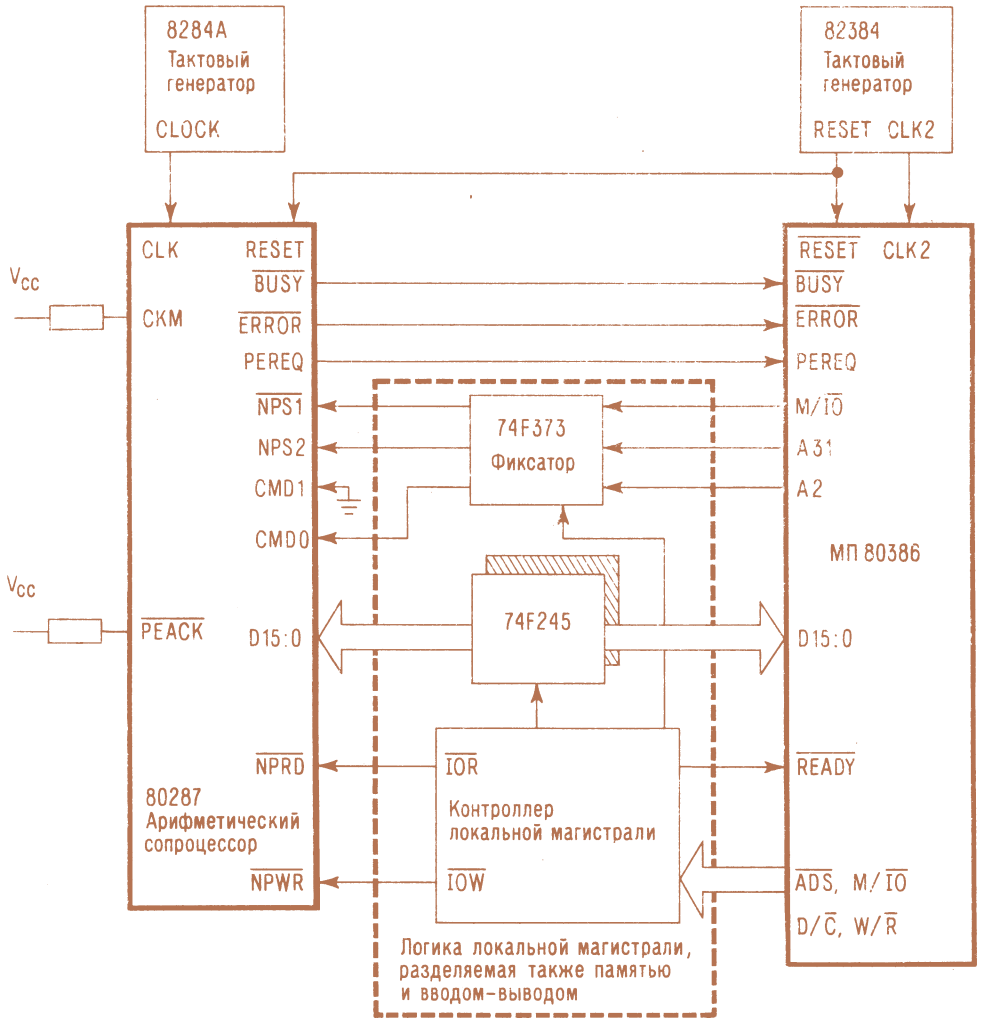


Рис. 7.3. Система на МП 80386 с сопроцессором 80287.

– входы чтения (NPRD#) и записи (NPWR#) арифметического сопроцессора 80287 воспринимают сигналы чтения и записи ввода-вывода логики управления локальной магистралью;
– на входе подтверждения расширения процессора (PEAK#) сопроцессора 80287 устанавливается высокий уровень. В системе на основе МП 80286 сигнал PEAK# генерируется МП 80286 для того, чтобы через выход PEREQ сопроцессора 80287 не передавались лишние данные. В системе с МП 80386 сигнал PEAK# не нужен, поэтому он не используется.

Между МП 80386 и сопроцессором 80387 имеются следующие соединения (рис. 7.4):

– выходы BUSY#, ERROR#, PEREQ сопроцессора 80387 подключаются к соответствующим выводам МП 80386;

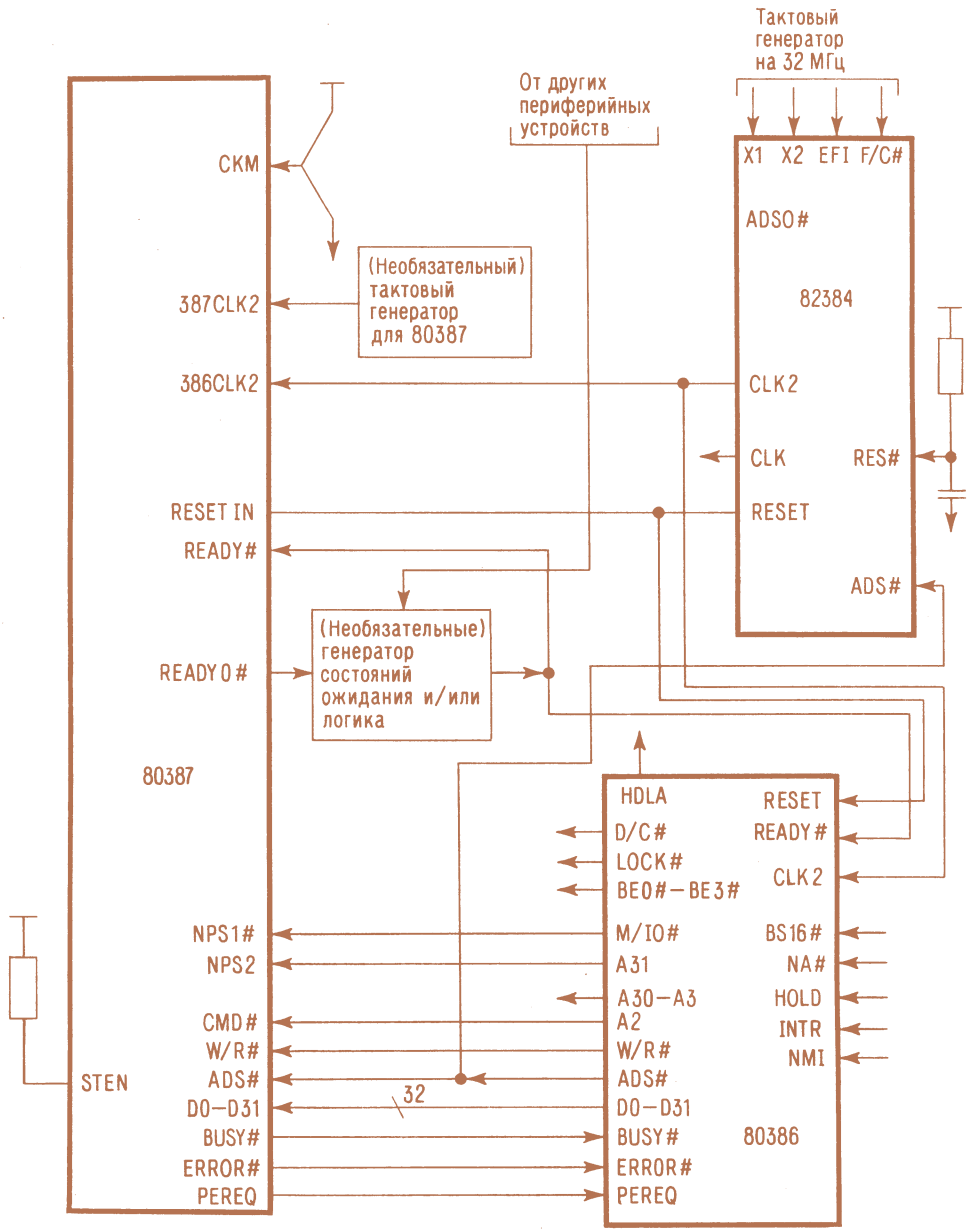


Рис. 7.4. Система на МП 80386 с сопроцессором 80387.

- вход RESETIN сопроцессора 80387 подключается к выходу RESET тактового генератора 82384;
- входы выбора микросхемы арифметического сопроцессора 80387 (NPS1#, NPS2) непосредственно подключаются к выходам M/IO# и A31 соответственно. Для циклов с сопроцессором сигнал A31 всегда имеет высокий уровень, а сигнал M/IO# – всегда низкий уровень напряжения;

- вход команды сопроцессора 80387 (CMD0#) отделяет данные от команд. Этот вход подключается к выходу A2. МП 80386 выводит адрес 800000F8H при записи команды, и адрес 800000FCH – при записи или считывании данных;
- все 32 линии (D31 – D0) шины данных МП 80386 непосредственно подключаются к шине данных сопроцессора 80387. Из-за этого все приемопередатчики локальной магистрали должны отключаться, когда МП 80386 считывает данные от сопроцессора 80387;
- входы READY#, ADS#, W/R# сопроцессора 80387 подключаются к соответствующим выводам МП 80386. Сопроцессор 80387 использует сигналы READY# и ADS# для отслеживания активности магистрали и того момента времени, когда могут быть опрошены сигналы W/R#, NPS1#, NPS2 и сигнал разрешения состояния (STEN);
- сигнал STEN является сигналом выбора микросхемы 80387. Он может быть установлен на высоком уровне напряжения. Если же один и тот же МП 80386 работает с несколькими сопроцессорами 80387, то сигнал STEN может использоваться для активизации одного из них в конкретный момент времени;
- выход готовности (READY0#) является дополнительным выходом. Он может быть использован для генерации состояния ожидания, требуемого сопроцессором. Состояние ожидания может генерироваться также внешней логикой (см. рис. 7.4).

Если МП 80386 обнаруживает команду сопроцессора, он автоматически генерирует один или более циклов ввода-вывода по адресам 800000F8H и 800000FCH. Процессор выполняет все необходимые магистральные циклы с памятью и передачи к сопроцессору 80387 и от него. Все обмены с сопроцессором 80387 являются 32-разрядными. Если по какой-либо причине используется 16-разрядная подсистема памяти, то МП 80386 автоматически выполняет преобразования перед передачей данных к арифметическому сопроцессору или от него.

Передача данных от сопроцессора 80387 к МП 80386 требует по крайней мере одного состояния ожидания. Циклы записи в сопроцессор 80387 не требуют состояний ожидания. Поскольку интерфейс с МП 80386 всегда синхронный, то вывод CLK2 сопроцессора 80387 должен быть подключен ко входу CLK2 МП 80386. Состояние входа CKM сопроцессора 80387 определяет, в каком из двух режимов он работает.

1. В псевдосинхронном режиме сигнал CKM имеет низкий уровень. Ко входу CLK2 сопроцессора 80387 должен быть подключен источник тактовой частоты. Синхронной с МП 80386 является только интерфейсная логика сопроцессора 80387. Внутренняя логика сопроцессора 80387 работает от источника тактовой частоты со входа 80387 CLK2. Его частота может быть от 10/16 до 16/10 частоты основного тактового сигнала CLK2.

2. В синхронном режиме сигнал CKM имеет высокий уровень и вход 80387 CLK2 никуда не подключается. Сопроцессор 80387 работает от основного тактового сигнала CLK2. Операции арифметического сопроцессора полностью синхронны с работой МП 80386.

Сопроцессоры обоих типов 80387 и 80287 используют два метода для взаимодействия с МП 80386. Первый используется, когда операции с сопроцессором инициирует МП 80386. Это происходит во время исполнения команды сопроцессора (команды ESC) и осуществляется под контролем программы. Взаимодействие по второму методу осуществляется, когда сопроцессор использует сигнал PEREQ для запроса, чтобы МП 80386 начал передачу операндов к системной памяти или от нее. Эти передачи осуществляются по требованию сопроцессора, поэтому они асинхрон-

ны с выполнением команд МП 80386. Передачи данных для сопроцессора имеют тот же самый приоритет на магистрали, что и передачи данных под управлением программы.

Для передачи каждого операнда может быть необходимо два, три, четыре или пять циклов магистрали. Эти циклы включают один цикл сопроцессора плюс следующие:

- один цикл с памятью для выровненного на границу операнда;
- два цикла с памятью для невыровненного операнда; два или три цикла с 16-разрядной памятью для невыровненного 32-разрядного операнда;
- четыре цикла с 16-разрядной памятью для невыровненного 64-разрядного операнда.

7.2.2. Сопроцессор локальной вычислительной сети 82586

Сопроцессор локальной вычислительной сети (ЛВС) 82586 является интеллектуальным высокопроизводительным контроллером связи, спроектированным для выполнения большинства задач, требуемых для управления доступом к ЛВС. Обычно контроллер 82586 является устройством управления связью станции, подключенной к ЛВС. Этот сопроцессор выполняет все функции, относящиеся к передаче данных между разделяемой памятью и каналом ЛВС, включая фильтрацию адреса, управление каналом, разбиение данных на кадры, обнаружение ошибок, кодирование данных, управление сетью, прямой доступ к памяти и формирование цепочки буферов. На рис. 7.5 показана структурная схема станции с сопроцессором ЛВС.

Этот тип станции обычно включает главный процессор, блок последовательного интерфейса, разделяемую память, приемопередатчик и канал ЛВС. Сопроцессор 82586 имеет два интерфейса: интерфейс от сети к блоку последовательного интерфейса и интерфейс от магистрали к локальной магистрали МП 80386.

Сопроцессор 82586 является задатчиком на локальной магистрали МП 80386. Он взаимодействует непосредственно с МП 80386 с помощью сигналов внимания к каналу (CA) и прерывания (INT).

На основе процессора 80186 или 80188 можно спроектировать высокопроизводительный дорогостоящий интерфейс управления сопроцессором ЛВС 82586. Этот процессор выполняет программы канального уровня, а иногда сетевого, транспортного или сеансового уровней. (Эти уровни являются четырьмя стандартными уровнями в архитектуре сетей передачи данных.) Специализированный процессор избавляет МП 80386 от необходимости выполнения сетевых задач.

7.3. Тактовый генератор 82384

Микропроцессорные системы предъявляют особые требования к тактовым импульсам для синхронизации выполнения различных функций. Тактовый генератор – это «сердце» системы, так как он генерирует фиксированные во времени импульсы, которые управляют логическими подсистемами.

Тактовый генератор является многофункциональным устройством, обеспечивающим синхронизацию операции МП 80386 и его компонентов. Он делает это следующим образом.

- Генерирует сигналы CLK2 и CLK. CLK2 – это сигнал удвоенной тактовой частоты для МП 80386 и некоторых устройств поддержки. CLK является системным тактовым сигналом для других устройств поддержки. Фаза сигнала CLK генератора 82384 соответствует фазе сигнала CLK, генерируемого внутри МП 80386.

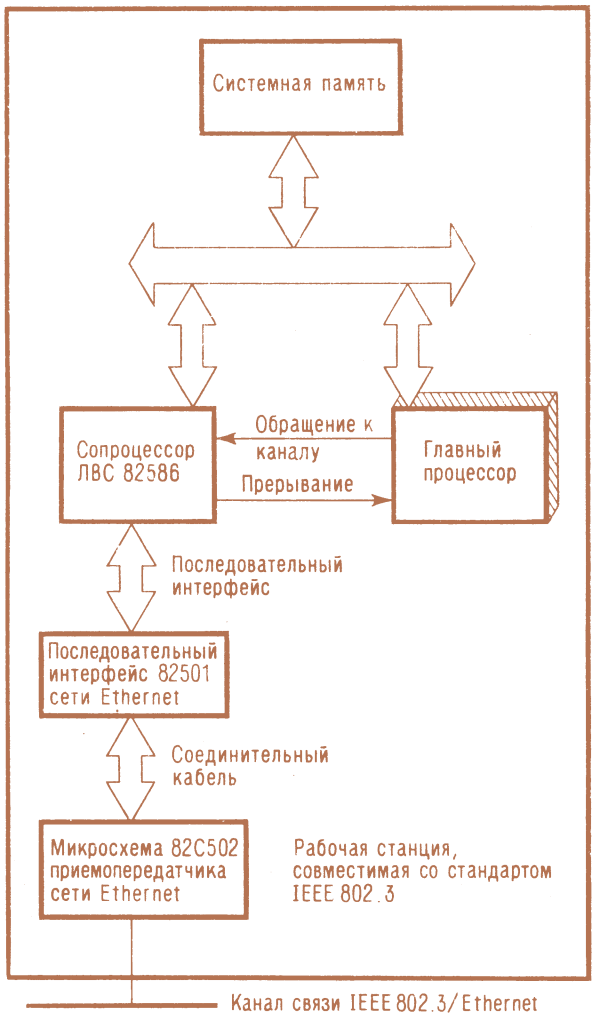


Рис. 7.5. Структурная схема рабочей станции ЛВС.

- Генератор 82384 использует выход ADS# МП 80386, который обеспечивает установку и время удержания сигнала CLK2 для того, чтобы генерировать сигнал ADSO#, функционально эквивалентный сигналу ADS#, но, кроме того, гарантирующий установку и время удержания по отношению к сигналу CLK.
- На вход RES (сигнал RESET) генератора 82384 поступает асинхронный сигнал от RC-схемы или подобного источника, этот сигнал синхронизируется с сигналом CLK, а затем выдается активный (высокий уровень) сигнала RESET к МП 80386 и другим системным компонентам.

Генератор 82384 может работать как от внешнего источника частоты, так и от кварцевого резонатора с третьей гармоникой. Источник сигнала определяется входом F/C#. Если на входе F/C# установлен низкий уровень, источником частоты является кварцевый резонатор, подключенный к выводам X1 и X2 генератора 82384. Если же на входе F/C установлен высокий уровень, то генератор 82384 принимает

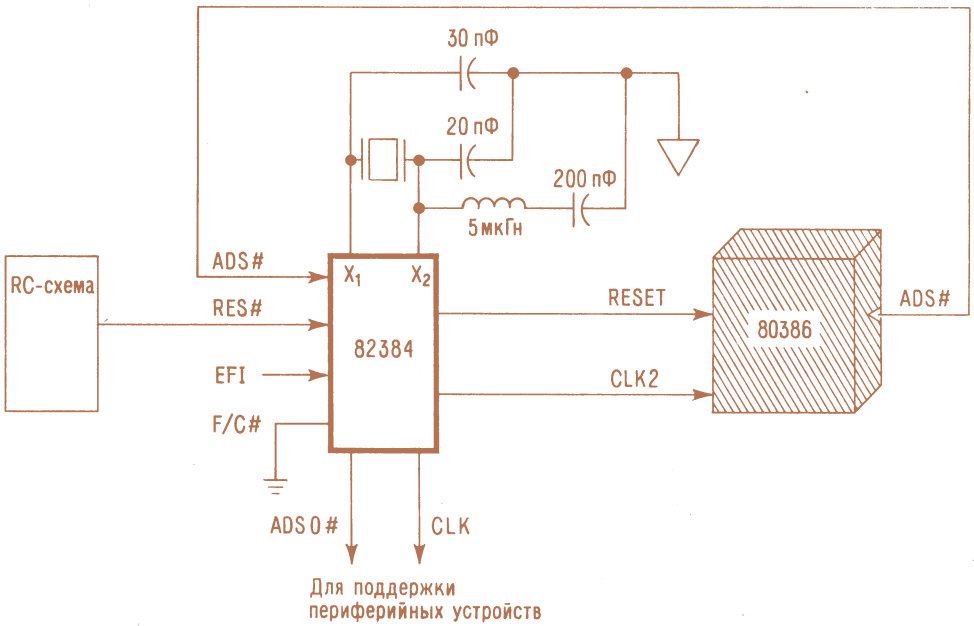


Рис. 7.6. Подключение тактового генератора 82384 к МП 80386.

сигнал от внешнего источника частоты (вывод EFI). На рис. 7.6 показана схема подключения генератора 82384 к МП 80386.

7.3.1. Тактовые частоты и синхронизация

Тактовая частота ограничена быстродействием используемых в системе микросхем. Тактовые генераторы весьма разнообразны: от простых местных устройств до весьма развернутых и сложных систем. При прочих равных условиях чем выше тактовая частота, тем больше функций может быть выполнено в единицу времени.

Сигналы CLK и CLK2 генератора 82384 являются выходными сигналами МОП-схем с напряжением высокого уровня, равным V_{cc} – 0,6 В, и могут питать ТТЛ-входы. Частота сигнала CLK2 является удвоенной частотой сигнала CLK. Внутренний сигнал CLK МП 80386 соответствует сигналу CLK на выходе генератора 82384 после перехода сигнала RESET из высокого в низкий уровень.

Задержка между сигналами CLK2 и CLK генератора 82384 поддерживается в диапазоне 0–16 нс независимо от тактовой частоты. Для интерфейсов с быстрой синхронизацией периферийные устройства должны быть синхронизированы от сигнала CLK2. Устройства, которые не могут работать на удвоенной тактовой частоте, должны использовать сигнал CLK.

7.3.2. Подача тактовых импульсов и оконечная нагрузка

Тактовый генератор 82384 обеспечивает синхронизацию для МП 80386 и поддерживающих его компонентов. Генератор 82384 вырабатывает как стандартный тактовый сигнал (CLK2), так и сигнал половинной частоты (CLK), предназначенный для обозначения внутренней фазы процессора. Тактовые импульсы генератора 82384

Рис. 7.7. Формы импульсов сигнала CLK2.

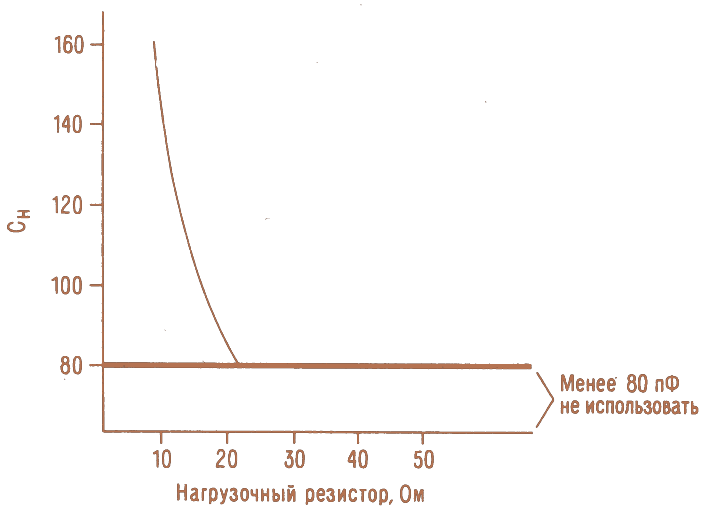


Рис. 7.8. Нагрузочный резистор для линии сигнала CLK2.

управляют также совместимыми с МП 80386 устройствами, которые могут быть включены в систему.

Для работы на высокой частоте тактовый сигнал CLK2 должен быть свободен от шумов и удовлетворять требованиям спецификации МП 80386. Обычно эти требования выполнимы, если придерживаться следующих рекомендаций:

1. Для того чтобы избежать отражения сигналов, к одному проводнику подключайте не более двух нагрузок.
2. Используйте осциллограф для сравнения формы сигнала с теми, которые показаны на рис. 7.7.
3. Для получения сигналов CLK и CLK2 используйте тактовый генератор 82384.
4. Получайте чистый сигнал путем установки нагрузочного резистора на линии сигнала CLK2. Вычисляйте значение сопротивления этого резистора по рис. 7.8

путем измерения полной емкостной нагрузки на линии CLK2. Если ее значение меньше 80 пФ, то добавьте конденсатор для устранения разницы. Из-за высокой частоты сигнала CLK2 рекомендуется применять композиционные резисторы.

Емкость нагрузки определяется по формуле

$$Сн = Свх(80386) + Свх(80387) + Свх(ПЛМ) + \ldots + Смон,$$

где монтажная емкость Смон вычисляется исходя из параметров печатного монтажа и печатной платы: толщины, диэлектрической проницаемости, расстояния до проводников заземления и питания.

Нагрузочный резистор должен быть малоиндуктивного типа.

7.4. Требования к питанию и заземлению

Так как рассеиваемая мощность зависит главным образом от частоты, то МП 80386, выполненный по технологии CHMOS III, в статическом режиме почти не рассеивает мощности. Потребление мощности МП 80386 носит преимущественно емкостной характер. Внутреннее потребление зависит от рабочей частоты и до некоторой степени от состояний ожидания и от программного обеспечения.

Мощность, потребляемая оконечными каскадами ввода-вывода, составляет примерно одну пятую от полной рассеиваемой мощности и изменяется с изменением частоты и напряжения. Она также зависит от емкостной нагрузки шины и быстродействие будет уменьшаться, если эта нагрузка превышает рекомендованный максимум. Хотя МП 80386 потребляет меньше мощности, чем МП 80286, возможность всплесков мощности увеличивается из-за большой частоты и большего количества выводов.

С целью минимизации наводок в системе с МП 80386 для питания и заземления должны быть использованы широкие плоские проводники (вместо обычных печатных проводников). Так как шины питания и заземления обладают внутренними индуктивностью и емкостью, то при конструировании, предусматривающем их использование, должно быть учтено сопротивление $Z = (LC)^{1/2}$. У МП 80386 имеется по 20 выводов питания V_{cc} и заземления V_{ss}. Все выводы питания и заземления должны объединяться соответствующими плоскими проводниками. МП 80386 лучше всего располагать в центре платы для получения наибольшего эффекта от использования этих проводников.

На высоких частотах должны быть учтены характеристики прохождения сигнала по линиям передачи в схеме. Отражения, помехи и шумы могут вызвать ложные прохождения сигнала, нарушения уровней входного напряжения и ошибки в данных. Эти ошибки обнаруживаются и устраняются с трудом.

Нарушения уровней входного напряжения обычно обусловлены импульсами напряжения, которые выходят за максимальный или минимальный допустимые уровни входного напряжения. Такие уровни напряжения вызывают всплеск тока во входных вентилях, что может привести к полному отказу устройства.

Помеха является результатом электрической активности в одном проводнике, вызывающей появление наведенного напряжения в другом проводнике. Существуют два типа помех: электромагнитные и электростатические. Электромагнитная помеха (называемая также перекрестной помехой) вызывается магнитным полем, которое существует вокруг любого проводника с током. Электростатическая помеха вызывается емкостной связью двух соседних проводников. Эти проводники действуют друг на друга как пластины конденсатора, т. е. заряд, накопленный на одном из них, наводит заряд противоположного знака на другом.

7.5. Тепловые характеристики

Допустимый уровень нагрева является критичным фактором для правильной работы любого электронного компонента, особенно процессора. Температура на поверхности кристалла называется температурой перехода[1]. Эту температуру можно определить внешними измерениями, используя известные тепловые характеристики корпуса электронного компонента. Для вычисления температуры перехода используйте следующие уравнения:

$$T_j = T_a + (Q_{ja} * PD),$$
$$T_j = T_c + (Q_{jc} * PD),$$

где
T_j – температура перехода,
T_a – температура окружающей среды,
T_c – температура корпуса,
Q_{ja} – температурный коэффициент переход – окружающая среда,
Q_{jc} – температурный коэффициент переход – корпус,
PD – рассеиваемая мощность.
(Уравнения предложены фирмой Intel и опубликованы с ее согласия.)

Вычисление температуры перехода по температуре корпуса имеет преимущество перед вычислениями по температуре окружающей среды. Из-за локализации измерения в единственной точке, в центре верхней части корпуса, температуру его легче вычислить или измерить, чем температуру окружающей среды. К тому же температура перехода для наихудшего случая (обозначенная T_j в уравнениях) меньше для уравнения с температурой корпуса, потому что тепловой коэффициент переход – корпус (Q_{jc}) меньше, чем тепловой коэффициент переход – окружающая среда Q_{ja}. Это означает, что вычисленная температура перехода менее зависит от изменения рассеиваемой мощности (PD).

Температура окружающей среды – это температура воздуха, окружающего источник питания или компонент. Температура циркулирующего воздуха, например в камере с вентилятором, не является правильно измеренной температурой окружающей среды, так как компоненты охлаждаются. При использовании тепловых характеристик корпуса компонента разработчик системы может либо задаться какой-либо температурой окружающей среды, либо использовать вентиляторы для управления температурой корпуса. Вместо вентиляторов могут быть использованы охлаждение с теплопереносом или тепловые радиаторы.

Просмотрите текущие справочные данные для определения значения коэффициента Q_{ja} воздушного потока и температуры окружающей среды в системе; с их помощью можно рассчитать желаемую температуру корпуса компонента.

7.6. Изготовление и отладка аппаратуры

Разработчик сначала рассматривает систему в целом, при этом сознавая, что ее сердцевина должна быть испытана до реального воплощения этой системы. Разработка и построение системы с МП 80386 по шагам является ключевым моментом для ее реализации за кратчайший период времени. Если базовая система разработана, изготовлена и, наконец, отлажена, к ней можно добавлять все больше программных средств и расширений.

---

[1] Имеются в виду p–n-переходы компонентов, составляющих процессор. – *Прим. ред.*

7.6.1. Рекомендации по шаговому построению системы

Инженеры-разработчики фирмы Intel предлагают следующие рекомендации вместе с последними справочными данными по аппаратным средствам.

1. Сначала смонтируйте тактовый генератор 82384 и его кварцевый резонатор. Убедитесь, что на входе CLK2 сигнал чистый и подключите его к МП 80386.

2. Подключите выход RESET генератора 82384 ко входу RESET МП 80386. С поступающим сигналом CLK2 убедитесь, что состояние RESET у МП 80386 правильное.

3. Подайте на выходы INTR, NMI и HOLD МП 80386 низкий уровень напряжения. До окончания первого магистрального цикла подайте на вывод READY# высокий уровень напряжения. Сбросьте МП 80386. Убедитесь, что процессор выдает правильные сигналы для выполнения выборки своей первой команды из физического адреса FFFFFFF0H. Затем подключите фиксатор адреса и убедитесь, что этот адрес появляется на ее выходах.

4. Подключите схемы выбора адреса к МП 80386. После сброса убедитесь, что процессор пытается выбрать ЭППЗУ, которое содержит начальную программу для исполнения.

5. Подключите к системе приемопередатчики данных. После сброса убедитесь, что выводы управления приемопередатчиками устанавливаются для цикла чтения. Подключите все адресные выводы разъемов ЭППЗУ. После сброса убедитесь, что эти выводы принимают правильный адрес для цикла выборки первой команды.

После монтажа ЭППЗУ подайте на вход READY# сигнал высокого уровня для того, чтобы МП 80386 начал первый магистральный цикл после сброса и продолжал добавлять состояния ожидания. С помощью вольтметра или осциллографа проверьте состояния сигналов.

Если МП 80386, фиксатор адреса, селектор адреса, генератор 82384, кварцевый резонатор, приемопередатчики и логика формирования сигнала READY# (которая служит для формирования состояний ожидания) работают правильно, МП 80386 может выполнять программы, хранимые в ЭППЗУ. Для верификации того, что части системы работают вместе, может быть выполнена простая программа отладки (как предлагается фирмой Intel), показанная на рис. 7.9. Если эта программа не выполняется, с помощью логического анализатора определите причину этого.

Если смонтировано ЭППЗУ с зашитой диагностической программой и МП 80386 ее выполняет, светодиодный индикатор для сигнала ADS# (при наличии его в системе) светится, так как сигнал ADS# генерируется для каждого магистрального цикла. Убедитесь, что микросхемы ЭППЗУ выбираются в каждом цикле выборки команды.

```
                        ASSUME CS:SIMPLEST_CODE

0000                    SIMPLEST_CODE   SEGMENT
FFF0                                    ORG 0FFF0H

FFF0 90                 START:          NOP
FFF1 90                                 NOP
FFF2 EB FC                              JMP START
  .
FFF4                    SIMPLEST_CODE   ENDS
                                        END
```

Рис. 7.9. Образец 4-байтовой диагностической программы.

Тестовая программа является замкнутым на себя бесконечным циклом. Запустите программу и проверьте результаты для случая, когда вход READY подключен к логике формирования сигнала READY#.

Убедитесь, что фиксаторы адреса фиксируют адрес с первого раза и селектор адреса формирует сигнал выбора микросхем ЭППЗУ. Эти ЭППЗУ должны выдавать первые четыре байта кода операции первой выполняемой команды: 90H, 90H, EBH, FCH для четырехбайтовой программы на рис. 7.9. Код операции должен проходить через приемопередатчики данных на выводы данных МП 80386.

7.6.2. Рекомендации по отладке

Микропроцессор 80386 может остановиться по любой из следующих причин.

(1) Обнаруживает одно из предопределенных условий прекращения работы. В реальном режиме состояние прекращения работы обычно показывает, что МП 80386 обнаружил мусор на шине данных.

(2) Переходит в состояние останова, обусловленное командой HALT.

(3) Сигнал READY#, предназначенный для завершения магистрального цикла, не активируется ни одним из исполнителей.

Для того чтобы показать состояние останова или прекращения работы, МП 80386 выдает коды на выводах W/R#, D/C#, M/IO# и выходах адреса. Если МП 80386 останавливается в состоянии прекращения работы, проверьте содержимое ЭППЗУ, монтаж проводников адреса и данных и приемопередатчики данных. Для проверки системы используйте четырехбайтовую диагностическую программу. Прежде чем пробовать более сложные программы, попытайтесь запустить эту программу.

Если МП 80386 останавливается, проверьте схему формирования сигнала READY#. Возможно, что она не активирует сигнал READY# для завершения цикла магистрали. Процессор добавляет к циклу состояния ожидания, ожидая, что сигнал READY# станет активным. Проверьте эту схему логическим анализатором.

7.7. Вопросы производительности

По своей сущности производительность системы измеряется тем, насколько быстро микропроцессор выполняет задачу или набор команд. Поскольку система часто включает в себя устройства, реакция которых медленнее реакции процессора, то производительность системы в целом меньше потенциальной. При оценке того, какое устройство наиболее важно для улучшения производительности, имейте в виду, что влияние быстродействия памяти на производительность намного сильнее, чем быстродействия ввода-вывода, так как большинство программ обращается чаще к памяти, чем к вводу-выводу.

Первым методом, применяемым обычно для сопряжения медленных устройств с быстрыми процессорами, является использование состояния ожидания. Состояния ожидания – это дополнительные такты, добавление к циклу магистрали. Для МП 80386 некоторая внешняя логика генерирует ожидание, задерживая сигнал на входе READY#. Это означает, что при работе МП 80386 на частоте 16 МГц одно состояние ожидания добавляет 62,5 нс ко времени, отведенному памяти для отклика.

Вторым методом является организация конвейера адресов. В отличие от первого метода конвейер адресов увеличивает время, необходимое памяти для отклика, на один такт без удлинения цикла магистрали. Этот дополнительный такт виртуально убирает выходную задержку на выходах адреса и состояния. Конвейер адресов выгодно использовать для большинства циклов магистрали. Однако если

следующий адрес не известен до окончания текущего цикла, то МП 80386, очевидно, не может конвейеризовать этот следующий адрес, так как он еще не сформирован. В этом случае синхронизация магистрали идентична циклу магистрали без конвейера. Заметим также, что первый цикл после незанятого состояния магистрали всегда является неконвейеризованным, потому что предыдущего цикла, в котором ранее выводился бы адрес, нет. И если следующий цикл должен быть конвейеризован, то первый цикл должен быть удлинен по крайней мере на одно состояние ожидания для того, чтобы новый адрес мог быть выдан на магистраль до окончания этого цикла.

Конвейер адресов менее эффективен для устройств ввода-вывода, требующих нескольких состояний ожидания. Действительно, чем больше требуется состояний ожидания, тем менее значимо сокращение на одно состояние ожидания из-за конвейера адресов. Свяжите этот факт с относительно нечастым обращением к устройствам ввода-вывода и станет ясно, что конвейер адресов для устройств ввода-вывода оказывает незначительное влияние на производительность системы.

Третьим и интересным методом приспособления к малому быстродействию памяти является уменьшение тактовой частоты МП 80386. Этот подход требует очень точной настройки, так как если более низкая тактовая частота увеличивает длительность цикла магистрали, то для конкретного устройства памяти может потребоваться меньше состояний ожидания. Заметим, что в то же самое время производительность системы в целом непосредственно зависит от тактовой частоты. Время выполнения увеличивается в прямой пропорции с увеличением периода тактового генератора. При тактовой частоте 12,5 МГц МП 80386 требуется почти на 33% больше времени для выполнения программы, чем при частоте 16 МГц с одинаковым количеством состояний ожидания.

7.8. Применение программируемых логических матриц

Программируемые логические матрицы (ПЛМ) благодаря отлаженной технологии плавких перемычек позволяют пользователям «зашить в кремнии» нужные цифровые функции. ПЛМ заменяют четыре или пять корпусов микросхем и находятся по сложности между логическими схемами малой (или средней), и большой степени интеграции.

7.8.1. Интерфейс на основе ПЛМ

Программируемые логические приборы (ПЛП) – это микросхемы, которые могут быть запрограммированы для выполнения определенных логических функций. Большинство ПЛП состоят из двух массивов логических вентилей: массива вентилей И, за которым следует массив вентилей ИЛИ. Входные сигналы ПЛП должны сначала пройти через массив вентилей И, где формируются специальные комбинации входных сигналов. В терминологии ПЛП каждая группа комбинаций И называется линией произведения. Линии произведения затем суммируются в массиве вентилей ИЛИ.

Существуют три основных типа ПЛП: программируемые постоянные запоминающие устройства (ППЗУ, PROM), программируемые логические массивы (PLA) и программируемые логические матрицы (ПЛМ, PAL). В ППЗУ матрица вентилей И фиксирована, а матрица вентилей ИЛИ программируема. В устройствах PLA программируемы обе матрицы. С другой стороны, ПЛМ имеют программируемую матрицу вентилей И, а матрица вентилей ИЛИ фиксирована. В пределах заданного числа входов, выходов и термов уравнений ПЛМ могут быть реализованы любые суммы произведений путем определения правильных соединений в матрице вентилей И.

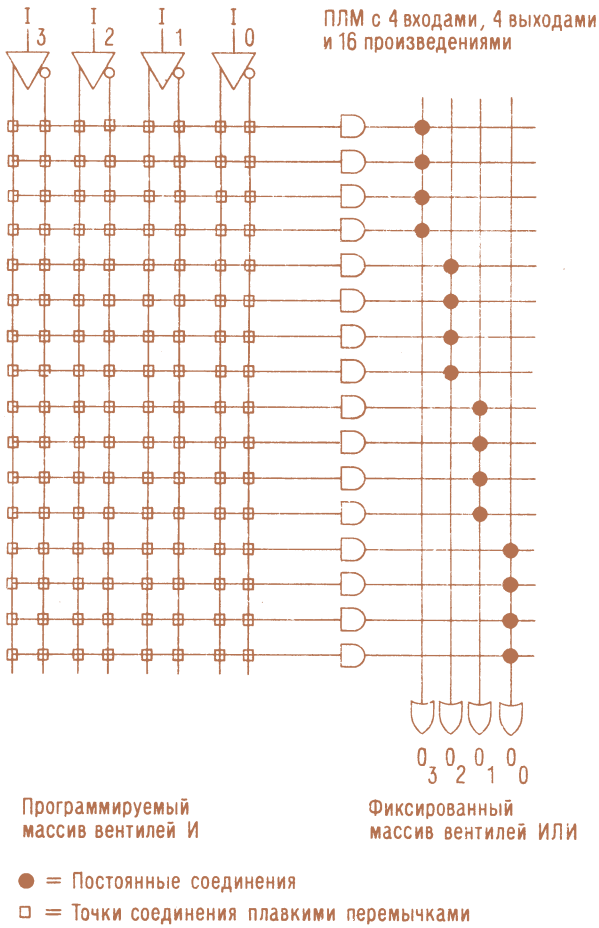


Рис. 7.10. Упрощенная схема устройства ПЛМ.

Обладая программируемым массивом вентилей И, пользователь может программировать только желаемые входные комбинации. Фиксация массива вентилей ИЛИ требует, чтобы определенные линии произведений были поданы на определенные выходы, обычно по восемь линий произведений на один выход. Упрощенная структурная схема ПЛМ с программируемым массивом вентилей И и фиксированным массивом вентилей ИЛИ показана на рис. 7.10.

Ранние модели ПЛМ, такие как 10LB и 14LB (которые имели 10 и 14 входов и 8 и 4 выхода соответственно), были комбинационными схемами и имели неплотные массивы. ПЛМ обладают различным быстродействием. Обычно суффикс при номере изделия говорит о быстродействии и потребляемой мощности. Изделия со стандартным быстродействием (50 нс) и потребляемой мощностью (0,2 Вт) не имеют суффикса.

ПЛМ может быть использована как машина состояний или дешифратор сигнала. К достоинствам ПЛМ относятся: программируемость, которая позволяет легко изменять функции ПЛМ, что упрощает разработку прототипа; ПЛМ недороги по сравнению со специализированными контроллерами магистрали; разводку вы-

водов ПЛМ определяет разработчик, поэтому он может упростить печатный монтаж платы путем назначения определенных сигналов требуемым выводам.

С другой стороны, ПЛМ имеют три недостатка: во-первых, нагрузочная способность ПЛМ может оказаться недостаточной для некоторых применений, где требуется буферизация; во-вторых, количество выводов или быстродействие ПЛМ не позволяет создавать любые функции и, в-третьих, большинство ПЛМ не имеет регистров сохранения состояния, поэтому при реализации машины состояния к выходным выводам нужно подключать регистры для сохранения текущего состояния.

Прослеживается тенденция к изготовлению генетических ПЛМ, которые могут заменить большинство других. Генетическая ПЛМ может иметь на своем выходе макроклетку (логический блок между массивом плавких перемычек и выходным выводом, который может быть программированием определенных перемычек установлен для работы любым из нескольких способов), запрограммированную для эмуляции выходной структуры любой из существующих 20-выводных ПЛМ.

Некоторые ПЛМ стираемы. Они полезны при отладке логических уравнений перед воплощением конечного продукта в однократно программируемой ПЛМ. Другим достоинством является то, что они позволяют изготовителю испытать логическое устройство и при необходимости стереть его для того, чтобы убедиться в его работоспособности.

Сначала логические функции ПЛМ описываются в виде набора булевых уравнений. Эти уравнения транслируются в карту плавких перемычек компилятором, который знает структуру целевой ПЛМ. Результирующая карта плавких перемычек выводится в виде файла в стандарте JEDEC (Joint Electron Device Engineering Council – Объединенный промышленный совет по электронным приборам), который воспринимается почти всем оборудованием для программирования ПЛМ. Файл JEDEC загружается в программатор ПЛМ, и плавкие перемычки пережигаются.

Многие изготовители поставляют ПЛМ с перемычками блокировки, которые, будучи разрушены, лишают пользователя возможности читать или верифицировать содержимое массива плавких перемычек ПЛМ, не затрагивая в то же время функцию ПЛМ в схеме.

Времена установки, удержания и задержки распространения сигнала для каждого входа и выхода ПЛМ могут быть определены из справочных данных по ПЛМ, имеющихся у изготовителя и большинства организаций по продаже. Изготовитель оставляет за собой право вносить необходимые изменения в эти и другие характеристики аппаратуры и обновлять соответствующие справочные данные. Рис. 7.11 иллюстрирует уравнения ПЛМ. Эти уравнения предназначены для контроллера регенерации ДОЗУ и приведены с разрешения фирмы Intel.

7.8.2. ПЛМ управления локальной магистралью

Фирма Intel реализовала контроллер локальной магистрали в виде двух ПЛМ. Первая, ПЛМ-1, отслеживает циклы магистрали процессора и генерирует все сигналы синхронизации цикла магистрали. Вторая, ПЛМ-2, генерирует большинство сигналов управления магистралью. Обе ПЛМ тактируются сигналом CLK2, поскольку выбор ПЛМ ограничен только 20-выводными ПЛМ серии B. Сигнал CLK2 дает следующие преимущества над сигналом CLK:

- в ПЛМ можно обеспечивать задержку шагами по 31 нс, а не ограничиваться шагами по 62 нс;
- задержка между тактовым и командным сигналами уменьшается, что позволяет достичь большего быстродействия с медленными устройствами;

Уравнения контроллера регенерации ДОЗУ

```
module refresh;
flag'— r3';
title 'DRAM refresh controller Intel Corporation';

pall  device 'p16 r8';
h, 1, c, x  =  1, 0,. C.,. x.;

gnd pin 10;
vcc pin 20;

clk 2 pin 1;          80386 CLK2
clk pin 2;            82384 CLK
refreq pin 3;         Высокий уровень для запроса цикла регенерации
refout pin 5;         Низкий уровень для активизации всех линий RAS в ДОЗУ
refgate pin 6;        Высокий уровень для включения регенерации, если необходимо
oe pin 11;

tc0 pin 14;           Бит 0 таймера регенерации
tc1 pin 15;           Бит 1 таймера регенерации
tcout pin 16;         Перенос из битов 0 и 1 таймера регенерации
q0 pin 17;            Переменная состояния q0
q1 pin 18;            Переменная состояния q1
refin pin 19;         Низкий уровень для запроса цикла регенерации ДОЗУ

Машина состояний регенерации

state_diagram [refin, q0, q1];
state [1, 1, 1]:      Холостой цикл
   if (refreq & clk) then [1, 0, 1]
   else [1, 1, 1];

state [1, 0, 1]:      Проверить включение регенерации
   if (refout & ! refgate & clk) then [0, 1, 1]
   else [1, 0, 1]

state [0, 1, 1]:      Запрос регенерации
   if (!refreq & clk) then [1, 1, 0]
   else [0, 1, 1];

state [1,1,0]:        Восстановление
   if (! refreq & clk) then [1,1,1]
   else [1,1,0];

state [0,0,0]: goto [1,1,1];   Неразрешенное состояние
state [0,0,1]: goto [1,1,1];   Неразрешенное состояние
state [0,1,0]: goto [1,1,1];   Неразрешенное состояние
state [1,0,0]: goto [1,1,1];   Неразрешенное состояние

   Младшие 2 бит таймера регенерации

state-diagram [tc1, tc0];
state [0,0]:
   if (clk) then [0,1]
   else [0,0];

state [0,1]:
   if (clk) then [1,0]
   else [0,1];

state [1,0]:
   if (clk) then [1,1]
   else [1,0];
```

Рис. 7.11. Уравнения контроллера регенерации ДОЗУ (с разрешения фирмы Intel).

```
state [1,1]:
   if (clk) then [0,0]
   else [1,1];
```

Перенос таймера регенерации

```
state-diagram [tcout];
state 0: if (tc1 & ! tc0 & clk) then 1
         else 0;
state 1: if (clk) then 0
         else 1;
test__ vectors ([clk2,clk,refreq,refout,refgate] → [refin])
        [c,h,1,h,1] → [h];      Холостой цикл
        [c,h,1,h,1] → [h];      Холостой цикл
        [c,h,1,h,1] → [h];      Холостой цикл
        [c,h,h,1,1] → [h];      Ожидание
        [c,h,h,1,1] → [h];          —«—
        [c,h,h,1,1] → [h];          —«—
        [c,h,h,1,1] → [h];      Регенерировать
        [c,h,h,h,1] → [1];          —«—
        [c,h,h,h,1] → [1];          —«—
        [c,h,h,1,1] → [h];          —«—
        [c,h,1,1,1] → [h];          —«—
        [c,h,1,h,1] → [h];          —«—
        [c,h,1,h,1] → [h];      Холостой цикл
        [c,h,1,h,1] → [h];          —«—
test__ vectors ([clk2,clk] → [tc1, tc0, tcout])
        [c,h] → [h,1,1];        Счет 2
        [c,h] → [h,h,h];        Счет 3, перенос
        [c,h] → [1,1,1];        Счет 0
        [c,h] → [1,h,1];        Счет 1
        [c,h] → [h,1,1];        Счет 2
        [c,h] → [h,h,h];        Счет 3, перенос
        [c,h] → [1,1,1];        Счет 0
        [c,h] → [1,h,1];        Счет 1
```

Примечание: этот набор уравнений ПЛМ был отлажен только «на бумаге» и приведен здесь для ссылки.

Рис. 7.11. (Продолжение).

- сигналы ADS# и READY# процессора могут быть опрошены непосредственно.

Функции ПЛМ-1. В реализации ПЛМ-1 является устройством с двумя состояниями. Первое состояние, BUSSTATE (состояние магистрали) используется для отслеживания циклов магистрали МП 80386. Оно определяется состоянием двух сигналов IDLE (холостой) и PIPE (конвейер) и отслеживает состояние магистрали МП 80386 путем опроса сигналов ADS# и READY#. Второе состояние, LOCALSTATE отслеживает состояние локальной магистрали. Сигналы L0, L1, L2 определяют состояние LOCALSTATE. Для определения того, какой тип цикла должен выполняться, состояние LOCALSTATE использует сигнал процессора W/R# и различные входы выбора микросхем.

Функции ПЛМ-2. Большинство сигналов управления магистралью генерирует ПЛМ-2. Сюда входят все пять командных сигналов, сигнал READY# и сигналы фиксации и включения приемопередатчиков данных. Входами ПЛМ-2 являются три сигнала состояния LOCALSTATE (см. ПЛМ-1) и три сигнала определения типа

магистрального цикла МП 80386: W/R#, M/IO#, D/C#. Это делается для отслеживания состояния локальной магистрали. ПЛМ-2 вводит также сигнал выбора микросхемы нулевого состояния ожидания, чтобы ускорить установку выходных сигналов для нулевых состояний ожидания.

7.8.3. ПЛМ состояния ДОЗУ

ПЛМ управления состоянием ДОЗУ определяет, когда нужно выполнять новый цикл ДОЗУ. Она также отслеживает состояние ДОЗУ во время этого цикла. На входы этой ПЛМ поступают запросы от МП 80386 к ДОЗУ, а также запросы на регенерацию. Выходы ПЛМ сохраняют информацию и генерирует два сигнала выбора адреса строки.

Если сигналы выбора адреса строки нужно запоминать во внутренних регистрах ПЛМ, используется ПЛМ типа 16R8. Если же применяются внешние регистры, то используется ПЛМ типа 16R6. Для систем на ПМ 80386 с частотой 16 МГц требуются ПЛМ с быстродействием серии B.

7.8.4. ПЛМ управления ДОЗУ

Большинство сигналов управления схемой ДОЗУ генерирует ПЛМ управления ДОЗУ. На входы этой ПЛМ поступают сигналы выходов включения байтов и сигнал W/R# МП 80386, а также сигналы состояния от ПЛМ состояния ДОЗУ. На выходах ПЛМ генерирует четыре сигнала выбора адреса столбца, сигналы для логики процессорных сигналов READY# и NA# и два сигнала управления приемопередатчиками.

Если сигналы выбора адреса столбца нужно сохранять во внутренних регистрах, то используется ПЛМ типа 16R8. Когда для этих сигналов применяются внешние регистры, то используется ПЛМ типа 16R4. Для систем на МП 80386 с частотой 16 МГц требуются ПЛМ с быстродействием серии B.

7.8.5. ПЛМ счетчика интервалов регенерации

ПЛМ счетчика интервалов регенерации периодически генерирует запросы на регенерацию для ПЛМ состояния ДОЗУ. ПЛМ счетчика интервалов регенерации работает как счетчик, уменьшающий свое значение с каждым тактом CLK. Когда этот счетчик достигает предустановленного значения, он сбрасывает свое значение на 255 и активирует свой сигнал запроса регенерации. Этот сигнал остается активным до тех пор, пока не будет обнаружено активное состояние одновременно обоих входов подтверждения регенерации.

7.8.6. ПЛМ счетчика адресов регенерации

ПЛМ счетчика адресов регенерации отслеживает адрес следующей строки ДОЗУ, которую нужно регенерировать. После каждого цикла регенерации ПЛМ увеличивает на 1 значение этого адреса. Поскольку большинство ДОЗУ нуждается только в 8 или менее битах для адреса регенерируемой строки, то может быть использована ПЛМ типа 16R8. Если для адреса строки требуется 10 бит, то можно использовать ПЛМ типа 20 × 10. Для систем с МП 80386, работающих на любой частоте, быстродействия стандартных ПЛМ вполне достаточно.

Глава 8

Программирование МП 80386

В данной главе обсуждается стратегия развития программного обеспечения в целом, а также среда виртуального МП 8086 (V86), защищенный режим МП 80286 и режим реальных адресов МП 80386. В гл. 9 даны общие указания по программированию и перечислены особенности программирования МП 80386. В гл. 10 описана полная система команд МП 80386, представляющая собой надмножество команд всех предыдущих поколений микропроцессоров, от 8086/88 до 80286.

8.1. Разработка программных комплексов

Разработка программного обеспечения представляет собой приложение принципов, навыков и тонкого искусства к проектированию и составлению программ и программных комплексов. Средой этих программ является вычислительная система, на которой они выполняются.

Для программных комплексов характерны следующие особенности.

- Программа (или программы) выполняется под единым управлением, чаще всего под управлением операционной системы, работающей на главном, центральном процессоре.
- Взаимное управление отдельными программами комплекса является подчиненным по отношению к этому единому управлению, которое обычно реализуется на другом уровне приоритета и/или уровнях привилегий.
- Внутренние уровни операционной системы скрыты от конечных пользователей вычислительного комплекса. Эти внутренние уровни «священны» и не могут быть изменены конечными пользователями.

Проектирование программного комплекса заключается в создании его полного и точного описания. Это описание представляет собой комплект подописаний программных и аппаратных компонентов и их взаимодействий. Полное описание программного компонента – это программа, написанная на хорошо определенном языке программирования. Семантика языка программирования должна быть известна еще до фактического начала программирования, чтобы заранее убедиться в возможности правильной реализации всех программных компонентов с помощью данного языка.

Разработчики программного обеспечения стремятся достичь четырех целей: заданного функционирования, правильности, производительности и надежности.

Заданное функционирование – это получение требуемого и правильного выходного ответа на каждый входной запрос. Вход включает в себя всю информацию, воспринимаемую программным комплексом. Выход – это выдаваемая информация. *Правильность* определяет, насколько точно программный комплекс отвечает целям, поставленным при проектировании. Под *производительностью* понимается скорость работы и эффективность программного комплекса при потреблении им ресурсов в

процессе достижения поставленной при проектировании цели. Наконец, *надежность* – это способность программного комплекса правильно выполнять свои функции, несмотря на отказы компонентов компьютерной системы. Под отказом здесь понимается временное или постоянное изменение характеристик компонентов, приводящее к нарушению их функций. Большинство программистов не считают, что, например, программное обеспечение отказывает – оно просто может быть *неправильным*.

«Модульный» – значит «собранный с использованием стандартизованных элементов или размеров ради гибкости и универсальности использования». Применительно к программной инженерии под модульностью понимается компоновка программных комплексов путем объединения частей, называемых *программными модулями*. Большой программный комплекс представляет систему, собранную из многих взаимодействующих частей, или модулей.

8.2. Элементы языка

Грубо говоря, язык, или система команд, состоит из тех элементов (мнемоник), которые разработчики сочли необходимым использовать для выполнения заданного круга задач. Например, если нет необходимости использовать вычисления с плавающей точкой, в языке не будет и соответствующих команд.

Ограничения и правила внутри языка или системы команд были определены в процессе разработки. Одно из основных требований к МП 80386 заключалось в необходимости поддержки системы команд предыдущих поколений. Другим важным обстоятельством, оказавшим влияние на систему команд МП 80386, было требование реализации уровней привилегий; по мере усложнения программных комплексов их защита приобретает все большее значение. В-третьих, мощная операционная система должна быть защищена от прикладных программ; защищенная система – это управляемая среда, в которой предусмотрены точно определенные границы и четкие правила, ограничивающие связи и пересылки.

8.3. Режимы адресации

Некоторые команды обращаются к памяти с целью получения или сохранения данных, используемых при выполнении команды. В других командах обращение к памяти происходит с целью определения ячейки, на которую осуществляется переход. В любом случае в команде должен быть указан адрес ячейки памяти, к которой выполняется обращение. Часть команды, определяющая местоположение ячейки памяти, называется *адресным полем*. Адресное поле содержит *указанный адрес*, а адрес ячейки памяти, к которой происходит обращение, называется *исполнительным* (эффективным).

В случае *прямой адресации* исполнительный адрес указывается в команде. Если все ячейки памяти компьютера адресуются прямо, команда должна состоять из нескольких слов. В случае *непосредственной* адресации операнд команды является частью самой команды. Ячейка с операндом расположена непосредственно за ячейкой, содержащей операционную часть команды.

Полезным и важным является режим *косвенной адресации*, при котором указанный в адресном поле команды адрес представляет собой указатель ячейки памяти, содержащей исполнительный адрес. В некоторых микропроцессорах предусмотрена возможность сложения адреса, указанного в команде, с содержимым указанного в ней же регистра. Такой способ адресации называется *индексным*, а используемый

регистр – *индексным регистром*. К индексной адресации примыкает *относительная*, при которой исполнительный адрес образуется сложением указанного в команде адреса с содержимым указателя команд. Указанный адрес рассматривается как величина со знаком, поэтому относительная адресация в команде перехода дает возможность осуществлять переход как вперед, так и назад (относительно ячейки, на которую указывает указатель команд) на величину, указанную в адресном поле команды.

В МП 80386 предусмотрены команды 9 типов: арифметические, операций над битами, пересылки данных, поддержки языка высокого уровня, логические и сдвига (линейного и циклического), управления процессором, управления программой, защиты и, наконец, операций над строками. Все эти команды с разбивкой по типам перечислены ниже. Кроме того, для обеспечения программной совместимости с МП 8086, МП 80386 может выполнять 16-разрядные команды в реальном и защищенном режимах.

Процессор самостоятельно определяет действующий по умолчанию размер операнда выполняемых им команд по D-биту дескриптора сегмента программы. Если D = 0, длина всех операндов и исполнительные адреса предполагаются 16-разрядными. Если D = 1, операнды и адреса имеют длину 32 разряда. Однако в реальном режиме действующий по умолчанию размер операнда и адреса составляет 16 разрядов (дескрипторы МП 80386 в реальном режиме не используются).

Независимо от размера, действующего по умолчанию, два префикса (префикс размера операнда и префикс длины адреса) отменяют значение бита D для конкретной команды. Эти префиксы автоматически добавляются ассемблерами фирмы Intel.

8.4. Обзор системы команд

Команды, входящие в систему команд, можно рассматривать, как орудия труда. Каждое орудие было создано для выполнения конкретной задачи, удовлетворяющей определенным требованиям. Высококачественные, полезные орудия являются залогом получения качественного конечного продукта.

Система команд МП 80386 является расширением наборов команд МП 8086/88 и 80286. Расширенная система МП 80386 имеет следующие четыре основные отличия от систем команд предыдущих моделей.

- Для команд с обращением к памяти добавлен индексный режим адресации с масштабированием. Этот режим дает возможность масштабировать (умножать на 1, 2, 4 или 8) содержимое индексного регистра перед сложением его с базой. Масштабирование обеспечивает эффективную индексацию при обращении к массивам многобайтовых данных.
- Команды типа умножения (MUL) оптимизированы путем использования алгоритма быстрого выхода, который, обнаружив, что старшие биты множителя равны нулю, завершает команду. В результате команда выполняется быстрее.
- Для обоих режимов – реальных адресов и защищенного – предусмотрена поддержка 32-разрядных операндов и адресов. В реальном режиме с целью совместимости с МП 8086/88 и 80286 размер операнда и адреса составляет по умолчанию 16 разрядов. Размер, действующий по умолчанию, можно заменить с помощью префиксного байта. В защищенном режиме размеры операнда и адреса определяются битом в дескрипторе сегмента, действие которого можно отменить префиксом команды.
- 64-разрядная схема сдвига в исполнительном блоке (EU) оптимизирует операции сдвига, деления и умножения. С помощью этой схемы операции многораз-

рядных сдвигов могут выполняться за один машинный цикл. Например, команды SHRD и SHLD используют два 32-разрядных регистра для сдвига 64-разрядной строки на несколько позиций за один машинный цикл.

МП 80386, обеспечивая скорость выполнения между 3 и 4 млн. операций в секунду (MIPS), имеет высокую системную производительность. Для этого в нем используются конвейерное выполнение команд (описание конвейерной обработки см. в гл. 6), аппаратная трансляция адреса и широкополосная магистраль.

Система команд МП 80386 обеспечивает работу с 8-, 16- и 32-разрядными данными. Как уже отмечалось выше, команды МП 80386 группируются в 9 функциональных категорий. Ниже следует список всех команд; в гл. 10 каждая команда будет определена и детально описана[1)].

Арифметические команды

Сложение

ADD	Сложение операндов
ADC	Сложение с переносом
INC	Инкрементирование операнда на 1
AAA	ASCII-коррекция для сложения
DAA	Десятичная коррекция для сложения

Вычитание

SUB	Вычитание
SBB	Вычитание с займом
DEC	Декрементирование операнда на 1
NEG	Изменение знака операнда
CMP	Сравнение операндов
DAS	Десятичная коррекция для вычитания
AAS	ASCII-коррекция для вычитания

Умножение

MUL	Умножение с двойной/одинарной точностью
IMUL	Целочисленное умножение
AAM	ASCII-коррекция для умножения

Деление

DIV	Беззнаковое деление
IDIV	Целочисленное деление
AAD	ASCII-коррекция для деления

Операции над битами

Операции над отдельными битами

BT	Проверка бита
BTS	Проверка и установка бита
BTR	Проверка и сброс бита
BTC	Проверка и инверсия бита
BSF	Прямое сканирование битов
BSR	Обратное сканирование битов

[1)] Расшифровка мнемоники команд будет дана в гл. 10.– *Прим. перев.*

Пересылка данных

Команды общего назначения

MOV	Пересылка операнда
PUSH	Запись операнда в стек
POP	Чтение операнда из стека
PUSHA	Запись в стек содержимого всех регистров
POPA	Чтение из стека содержимого всех регистров
XCHG	Обмен операнда с содержимым регистра

Преобразование

MOVZX	Пересылка байта, слова, двойного слова с расширением нуля
MOVSX	Пересылка байта, слова, двойного слова с расширением знака
CBW	Преобразование байта в слово в AX
CWDE	Преобразование слова в двойное слово в EAX
CWD	Преобразование слова в двойное слово в DX, AX
CDQ	Преобразование двойного слова в EDX, EAX

Ввод/вывод

IN	Ввод операнда из пространства ввода/вывода
OUT	Вывод операнда в пространство ввода/вывода

Адресация объекта

LEA	Загрузка исполнительного адреса
LDS	Загрузка указателя в регистр сегмента D
LES	Загрузка указателя в регистр сегмента E
LFS	Загрузка указателя в регистр сегмента F
LGS	Загрузка указателя в регистр сегмента G
LSS	Загрузка указателя в регистр сегмента S (сегмента стека)

Операции над флагами

LAHF	Загрузка флагов в регистр AH
SAHF	Запись содержимого регистра AH в регистр флагов
PUSHF	Запись флагов в стек
POPF	Чтение флагов из стека
PUSHFD	Запись содержимого EFLAGS в стек
POPFD	Чтение содержимого EFLAGS из стека
CLC	Сброс флага переноса
CLD	Сброс флага направления
CMC	Инвертирование флага переноса
STC	Установка флага переноса
STD	Установка флага направления

Поддержка языка высокого уровня

BOUND	Проверка границ массива
ENTER	Создание блока параметров для входа в процедуру
LEAVE	Выход из процедуры
SETcc	Установка байта по условию

Логические команды, сдвиг и циклический сдвиг

Логические

NOT	Операция НЕ над операндом
AND	Операция И над операндом
OR	Операция ВКЛЮЧАЮЩЕЕ ИЛИ над операндом
XOR	Операция ИСКЛЮЧАЮЩЕЕ ИЛИ над операндом
TEST	Проверка операнда

Сдвиги

SHL/SHR	Логический сдвиг влево или вправо
SAL/SAR	Арифметический сдвиг влево или вправо
SHLD/SHRD	Двойной сдвиг влево или вправо

Циклические сдвиги

ROL/ROR	Циклический сдвиг влево/вправо
RCL/RCR	Циклический сдвиг через перенос влево/вправо

Управление процессором

ESC	Расширение процессора
HLT	Останов
LOCK	Блокировка магистрали-префикс команды
MOV	Пересылка содержимого управляющих регистров
NOP	Холостая команда
WAIT	Ожидание сброса сигнала BUSY #

Управление программой

Условные переходы

JA/JNBE	Переход если выше/не ниже и не равно
JAE/JNB	Переход если выше или равно/не ниже
JB/JNAE	Переход если ниже/не выше и не равно
JBE/JNA	Переход если ниже или равно/не выше
JC/JNC	Переход если есть перенос/нет переноса
JE/JZ	Переход если равно/нуль
JG/JNLE	Переход если больше/не меньше и не равно
JGE/JNL	Переход если больше или равно/не меньше
JL/JNGE	Переход если меньше/не больше и не равно
JLE/JNG	Переход если меньше или равно/не больше
JNE/JNZ	Переход если не равно/не нуль
JNO	Переход если нет переполнения
JNP/JPO	Переход если нет четности/нечетность
JNS	Переход если нет знака
JO	Переход если есть переполнение
JP/JPE	Переход если есть четность/четность
JS	Переход если есть знак

Безусловные переходы

CALL	Вызов процедуры или задачи
RET	Возврат из процедуры
JMP	Переход

Управление итерациями

LOOP	Цикл
LOOPE/LOOPZ	Цикл если равно/нуль
LOOPNE/LOOPNZ	Цикл если не равно/не нуль
JCXZ	Переход если CX = 0
JECXZ	Короткий переход если ECX = 0

Прерывания

INT	Прерывание
INTO	Прерывание по переполнению
IRET	Возврат из прерывания/задачи
CLI	Сброс разрешения прерывания
STI	Установка разрешения прерывания

Защита

CLTS	Сброс флага переключения задачи
SGDT	Запись таблицы глобальных дескрипторов
SIDT	Запись таблицы дескрипторов прерываний
STR	Запись регистра задачи
SLDT	Запись таблицы локальных дескрипторов
LGDT	Загрузка таблицы глобальных дескрипторов
LIDT	Загрузка таблицы дескрипторов прерываний
LTR	Загрузка регистра задачи
LLDT	Загрузка таблицы локальных дескрипторов
ARPL	Коррекция затребованного уровня привилегий
LAR	Загрузка прав доступа
LSL	Загрузка границы сегмента
VERR/VERW	Проверка сегмента на чтение/запись
LMSW	Загрузка слова состояния машины
SMSW	Запись слова состояния машины

Операции над строками

MOVS	Пересылка байта, слова, двойного слова строки
INS	Ввод строки из пространства ввода-вывода
OUTS	Вывод строки в пространство ввода-вывода
CMPS	Сравнение байта, слова, двойного слова строки
SCAS	Сканирование байта, слова, двойного слова строки
LODS	Загрузка байта, слова, двойного слова строки
STOS	Запись байта, слова, двойного слова строки
REP	Префикс повторения
REPE/REPZ	Префикс повторения, пока равно/нуль
REPNE/REPNZ	Префикс повторения, пока не равно/не нуль
XLAT	Перекодирование

Средняя длина команды составляет 3,2 байт. Поскольку МП 80386 использует 16-байтовую очередь команд, в среднем 5 команд оказываются предварительно выбранными из памяти. Команды содержат 0, 1, 2 или 3 операнда. Операнды могут храниться в регистре, в самой команде или в памяти. Большинство безоперандных команд занимают лишь 1 байт, в то время как двухоперандные команды обычно имеют длину 2 байт. Однако к любой команде может быть добавлен префикс замены длины операнда, действующей по умолчанию.

Операнды могут быть длиной 8, 16 и 32 разряда. Обычно при выполнении

программы, написанной для 32-разрядного МП 80386, операнды имеют длину 8 или 32 разряда. При выполнении программы, написанной для МП 80286 или 8086, используются операнды длиной 8 или 16 разрядов. Для замены длины операнда, действующей по умолчанию, к операндам можно добавить префикс. Это дает возможность использовать 32-разрядные операнды в программе, написанной для 16-разрядной машины, и наоборот.

Для обращения к памяти требуется 16- или 32-разрядный адрес. Каждая команда, обращающаяся к памяти, имеет атрибут размера адреса 16 или 32 разряда.

Для 16-разрядного адреса характерно использование 16-разрядного смещения в команде и вычисление 16-разрядного исполнительного адреса, т. е. образование 16-разрядного адресного смещения, или относительного адреса в сегменте. 32-разрядные адреса используют 32-разрядное смещение; при этом образуется 32-разрядный относительный адрес. Любая команда, читающая или записывающая 16-разрядное слово или 32-разрядное двойное слово, имеет атрибут размера операнда 16 или 32 разряда.

Команды, использующие стек в явном виде, например POP EAX, также имеют атрибут размера адреса стека 16 или 32 разряда. Для формирования адреса вершины стека 16-разрядные адреса используют 16-разрядный регистр SP (регистр указателя стека). Команды с атрибутом размера адреса стека 32 разряда используют 32-разрядный ESP (расширенный регистр указателя стека). Атрибут размера адреса стека указывает B-битом дескриптора сегмента данных в регистре сегмента стека SS. Нулевое значение B-бита выбирает атрибут размера адреса стека 16; значение 1 выбирает атрибут 32.

Программы, выполняемые в реальном или виртуальном режиме МП 8086, используют по умолчанию 16-разрядные адреса и операнды. С другой стороны, в защищенном режиме D-бит (в дескрипторах исполняемого сегмента) определяет действующий по умолчанию атрибут размера и для операндов, и для адресов. Эти значения атрибутов действуют при выполнении всех команд в сегменте. Если D-бит равен 0, по умолчанию действует размер адреса и операнда 16 разрядов. Если D-бит равен 1, размер по умолчанию составляет 32 разряда.

Атрибут сегмента, действующий по умолчанию, можно заменить. Внутренний код команды может включать двухбайтовые префиксы: префикс размера адреса (67H) и префикс размера операнда (66H). В табл. 8.1 показано, как каждый префикс влияет на размеры операндов и адресов.

Таблица 8.1. Атрибуты размеров

Бит определения размера по умолчанию	0	0	0	0	1	1	1	1
Префикс размера операнда (66H)	Н	Н	Д	Д	Н	Н	Д	Д
Префикс размера адреса (67H)	Н	Д	Н	Д	Н	Д	Н	Д
Размер операнда	16	16	32	32	32	32	16	16
Размер адреса	16	32	16	32	32	16	32	16

Н – нет, этот префикс команды отсутствует.
Д – да, этот префикс команды присутствует.

Для облегчения программирования МП 80386 предусмотрены средства проверки и отладки. Они включают самотестирование и прямую адресацию к кэш-памяти трансляции страниц. В дополнение к этому МП 80386 содержит четыре регистра точек останова, с помощью которых реализуются ловушки в точках останова при выполнении команд или обращении к данным.

8.5.Виртуальная среда МП 8086

Прикладные программы, написанные для МП 8086/88 и 80186/188, могут выполняться на МП 80386 в реальном режиме без всяких изменений. Операционная система защищенного режима так же может выполнять эти программы в неизменном виде. Прикладные программы и операционные системы МП 8086 выполняются в защищенном режиме как часть виртуальной задачи (V86), использующей аппаратные средства многозадачности, предоставляемые в защищенном режиме.

Задача V86 образует «виртуальную машину», состоящую из аппаратных средств МП 80386 и системного программного обеспечения. Системные программы контролируют внешний интерфейс V86, прерывания и ввод-вывод. К аппаратным средствам относятся сегмент состояния задачи (TSS) с виртуальными регистрами, а также виртуальная память, занимающая первый мегабайт линейного адресного пространства. Аппаратные средства реализуют также выполнение команд с обращением к регистрам и адресному пространству.

8.5.1. Вход и выход из виртуального режима МП 8086

Системные программы МП 80386 не работают непосредственно с флагом VM (виртуальный режим) в регистре EFLAGS. Они изменяют логическую копию, образ EFLAGS, хранящийся в сегменте состояния задачи или в стеке. Монитор V86, создавая задачу V86, устанавливает бит VM в образе EFLAGS. Обработчики исключений и прерываний проверяют бит VM в стеке. Если VM установлен, это значит, что прерванная процедура выполнялась в режиме V86, и обработчику прерываний может понадобиться активизировать монитор V86.

МП 80386 может войти в виртуальный режим МП 8086 двумя способами.

- При возврате с помощью команды IRET из процедуры задачи МП 80386, которая загружает регистр EFLAGS из стека. Если флаг VM установлен, МП 80386 возвращает управление процедуре МП 8086. При выполнении IRET текущий уровень привилегий (CPL) должен быть равен нулю, в противном случае МП 80386 не изменяет VM.
- В процессе переключения на задачу МП 80386, когда регистр EFLAGS загружается из нового TSS. Этот новый сегмент состояния должен принадлежать МП 80386, а не 80286, потому что сегмент состояния задачи МП 80286 не сохраняет старшее слово регистра EFLAGS, содержащее бит VM. Если VM равен 1, новая задача выполняет команды МП 8086 и МП 80386 образует базовые адреса так, как если бы он был процессором 8086.

МП 80386 выходит из виртуального режима при обработке прерывания или исключения. Здесь могут быть два случая.

- Если в результате прерывания или исключения происходит векторный переход на процедуру с нулевым уровнем привилегий, МП 80836 сохраняет текущее состояние EFLAGS в стеке и сбрасывает флаг VM. В результате обработчик прерываний или исключений выполняется как собственная программа МП 80386 в защищенном режиме. Если прерывание или исключение передает управление по вектору согласованному сегменту или уровню привилегий, отличному от 3, МП 80386 фиксирует исключение – ошибку общей защиты. Код ошибки этого исклю-

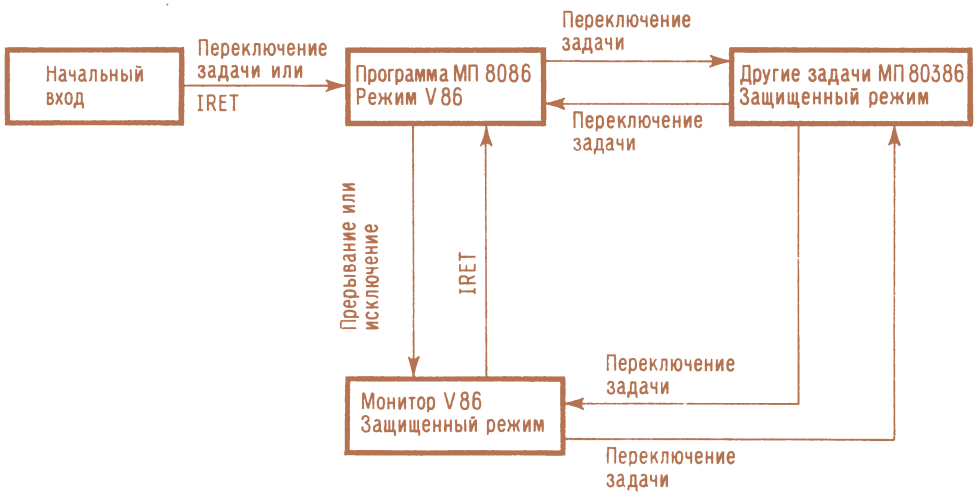


Рис. 8.1. Вход и выход МП 80386 из виртуального режима МП 8086.

чения представляет собой селектор исполняемого сегмента, на который была сделана попытка перехода.

• Прерывание или исключение вызывают переключение задачи. При переключении задачи V86 на любую другую происходит загрузка регистра EFLAGS из сегмента состояния новой задачи. Если новый TSS является сегментом МП 80386 и бит VM равен нулю, а также если новый TSS принадлежит МП 80286, процессор сбрасывает флаг VM, загружает регистры сегмента из нового TSS, используя процедуру формирования адресов МП 80386, и начинает выполнять команды новой задачи в соответствии с защищенным режимом МП 80386.

На рис. 8.1 показано, как МП 80386 входит и выходит из программы МП 8086.

Переключение с задачи V86 или на нее вызывается несколькими обстоятельствами. Это может быть прерывание с векторным переходом на вентиль задачи, возврат из прерывания по IRET с установленным флагом NT (вложенная задача) или операция планировщика операционной системы МП 80386. Независимо от причины МП 80386 переключает флаг VM в регистре EFLAGS нового TSS. Если этот новый сегмент состояния принадлежит МП 80286, процессор очищает VM, потому что старшее слово регистра не входит в TSS. МП 80286 корректирует VM перед загрузкой регистров сегмента из их образов в новом TSS. В зависимости от нового значения флага VM МП 80386 интерпретирует новые образы регистров сегмента как селекторы МП 80386/80286 или МП 8086.

Вентили прерываний и ловушек.

Если МП 80386 выходит из режима V86 в результате исключения или прерывания по вектору через вентиль ловушки или прерывания на процедуру с нулевым уровнем привилегий, обработчик исключения или прерывания возвращает управление программе МП 8086, выполняя команду IRET. Поскольку эта команда была предназначена для исполнения процессором 8086, программа МП 8086 содержит таблицу прерываний, начинающуюся с линейного адреса 0. МП 80386 не использует эту таблицу; векторный переход для всех прерываний и исключений, возникающих в режиме V86, происходит через таблицу дескрипторов прерываний. Элемент этой таблицы для прерываний и исключений, возникающих в режиме V86, должен содержать вентиль ловушки, вентиль ловушки МП 80386 (тип 14) или вентиль

прерываний МП 80386 (тип 15). Вентили типов 14 и 15 должны указывать на несогласованный сегмент программы с нулевым уровнем привилегий.

Если прерывание должно отражаться на обработчик прерываний задачи виртуального МП 8086, обработчик прерываний МП 80386 эмулирует прерывание задачи так, как это требуется для МП 8086.

Прерывания и исключения, использующие вентили ловушек или прерываний МП 80386 в таблице дескрипторов прерываний, осуществляют векторный переход на процедуру соответствующего обработчика прерываний на нулевом уровне привилегий. МП 80386 сохраняет содержимое сегментных регистров МП 8086 в стеке PL0 (нулевого уровня привилегий). Затем, приступая к выполнению процедуры обработчика прерывания, МП 80386 заполняет все регистры сегмента нулями. Процедуры прерываний, ожидающие найти какие-либо значения в регистрах сегмента, или возвращающие через них значения должны использовать образы регистров, сохраненные в стеке PL0. Для определения того, произошло ли прерывание в режиме V86, обработчики прерываний анализируют флаг VM в сохраненном содержимом EFLAGS.

Для возврата к программе МП 8086 после обработки прерывания или исключения МП 80386 должен выполнить следующие шаги:

1. Обратившись к вектору прерываний МП 8086, найти соответствующую процедуру обработчика.
2. Сохранить состояние программы МП 8086 в стеке PL3 (уровня привилегий 3).
3. Модифицировать указатель возврата на стеке PL0 так, чтобы он указывал на процедуру обработчика прерываний на уровне PL3.
4. Выполнением команды IRET передать управление обработчику прерываний.
5. После того как команда IRET обработчика на уровне PL3 передаст управление через ловушку монитору V86, восстановить указатель возврата на уровне PL0, чтобы он указывал на первоначальную процедуру обработки прерываний на уровне привилегий 3.
6. Выполнить команду IRET для передачи управления назад в прерванную процедуру.

8.5.2. Адресация в режиме виртуального МП 8086

Если флаг VM (виртуальный режим) в регистре EFLAGS установлен, МП 80386 работает в режиме V86. Процессор проверяет флаг VM при декодировании команд, чтобы определить, какие команды чувствительны к IOPL (уровню привилегий ввода-вывода), и при загрузке регистров сегмента для выяснения необходимости использовать формирование адресов по правилам МП 8086.

В режиме V86 доступны все регистры МП 80386, включая сегментные регистры FS и GS, регистры управления и тестирования, а также регистры отладки. Кроме того, в программе V86 могут использоваться новые команды МП 80386, выполняющие операции над регистрами FS и GS в явном виде, а также префиксы замены дескрипторов сегмента, что вызывает обращение к регистрам FS и GS при вычислении адресов в процессе выполнения команд. К этим командам относятся: LSS, LFS и LGS, сканирование битов, двойной сдвиг, установка байта по условию, условные переходы с длинным смещением, пересылка с расширением нуля или знака и обобщенное умножение.

8.5.3. Формирование адреса

Находясь в режиме V86, МП 80386 не использует селекторы МП 8086 для обращения к дескрипторам, а формирует линейный адрес так, как это делается в МП 8086. Селектор сдвигается влево на 4 бит с образованием 20-разрядного базового адреса. Исполнительный адрес дополняется слева четырьмя нулевыми битами и прибавляется к базовому адресу. При этом возникает возможность переноса, поэтому результирующий линейный адрес может иметь до 21 значащего бита. Программа МП 8086, выполняемая в режиме V86, может генерировать линейные адреса в диапазоне от 0 до 10FFEFH, что соответствует 1 Мбайт плюс приблизительно 64 Кбайт.

Задачи V86 фактически генерируют 32-разрядные линейные адреса. Хотя программа МП 8086 использует из этого 32-разрядного адреса только младшие 21 бит, линейный адрес может быть отображен с помощью таблиц страниц на 32-разрядный физический адрес. В отличие от процессоров 8086 и 80286 использование префикса размера адреса дает возможность генерации 32-разрядных исполнительных адресов. Если, однако, значение 32-разрядного адреса превышает 65535, возникает исключение (прерывание 12 или 13 без кода ошибки).

8.5.4. Задача виртуального МП 8086

Задача виртуального МП 8086 должна быть представлена сегментом состояния задачи МП 8086, который используется процессором 80386 для выполнения программы МП 8086 перед возвратом в защищенный режим, где выполняется задача МП 80386. Задача V86 состоит из двух частей: предназначенной для выполнения программы МП 8086, а также программы МП 80386, играющей роль монитора виртуальной машины.

Монитор V86, фактически представляющий программу МП 80386, выполняемую на нулевом уровне привилегий в защищенном режиме, состоит в основном из процедур инициализации и обработки исключений. Монитор, как и в случае любой другой программы МП 80386, использует дескрипторы исполняемого сегмента, которые должны находиться в таблице глобальных дескрипторов GDT или таблице локальных дескрипторов задачи LDT. Монитору могут также понадобиться дескрипторы сегмента данных, с помощью которых монитор получает доступ к таблице векторов прерываний или другим частям программы МП 8086, находящимся в первом мегабайте адресного пространства.

Средства операционной системы могут оставаться частью программы МП 8086, либо эмулироваться монитором V86. Основной довод в пользу первого варианта состоит в том, что прикладная программа модифицирует операционную систему и группа разработки может не иметь в запасе достаточно времени, чтобы реализовать операционную систему МП 8086 в виде программы для МП 80386.

Некоторые достоинства реализации или эмуляции системы в мониторе V86 заключаются в том, что функции операционной системы МП 8086 можно зачастую легко эмулировать с помощью вызовов (командами CALL) процедур операционной системы МП 80386. Кроме того, функции операционной системы (и системные ресурсы, например жесткие диски и печатающие устройства) проще распределяются между несколькими задачами V86. Полезно заметить, что независимо от способа реализации режима V86, различные задачи V86 могут использовать разные операционные системы МП 8086, что открывает перед системными программистами дополнительные возможности.

Разбиение на страницы, не являясь обязательным при выполнении одной задачи V86, тем не менее полезно, так как дает возможность:

- перенаправлять или захватывать ссылки на устройства ввода-вывода, отображаемые в пространство памяти;
- совместно использовать операционную систему МП 8086 или программы, записанные в ПЗУ, и общие для нескольких одновременно выполняемых программ МП 8086;
- создавать виртуальное адресное пространство, превышающее по размеру физическое адресное пространство;
- создавать множество задач V86, в котором каждая задача отображает нижний мегабайт линейных адресов на различные физические ячейки;
- эмулировать циклический возврат через 1 Мбайт, присущий МП 8086.

При выполнении программ МП 8086 процессор 80386 не обращается к дескрипторам и, следовательно, не использует реализуемый с помощью дескрипторов механизм защиты. Для защиты системных программ в задаче V86 от программ МП 8086 можно использовать два подхода. Разработчик программного обеспечения может использовать бит U/S элемента таблицы страниц для защиты монитора виртуальной машины и других системных программ, располагаемых в пространстве каждой задачи V86. Когда МП 80386 работает в режиме V86, текущий уровень привилегий (CPL) равен 3, что дает программе МП 8086 лишь привилегии пользователя. Если страницы монитора виртуальной машины имеют привилегии супервизора, они недоступны для программы МП 8086.

Другой способ защиты системы V86 от прикладных программ V86 заключается в резервировании в начале линейного пространства адресов каждой задачи 1 Мбайта плюс 64 кбайт для программ МП 8086. Задачи МП 8086 не могут генерировать адреса за пределами этого диапазона.

8.5.5. Чувствительные команды в режиме виртуального МП 8086

Когда МП 80386 работает в режиме V86, команды IN, INS, OUT и OUTS не чувствительны к битам IOPL в регистре EFLAGS, хотя в защищенном режиме чувствительность имеет место. Есть также несколько команд, которые, наоборот, приобретают чувствительность к битам IOPL. К ним относятся CLI, INTn, IRET, LOCK, POPF, PUSHF и STI. Этим командам придано свойство чувствительности к IOPL для того, чтобы монитор V86 мог эмулировать их функции.

В режиме виртуального МП 8086 текущий уровень привилегий всегда равен 3. Если IOPL меньше, эти 7 команд вызывают исключение – ошибку общей защиты.

Обращение ко многим операционным системам МП 8086 осуществляется путем занесения параметров в стек и выполнения затем команды INTn. Команда INTn чувствительна, поэтому монитор V86 может перехватывать обращения к операционной системе МП 8086, если IOPL меньше 3. Затем монитор V86 может эмулировать требуемую функцию операционной системы МП 8086, либо направить прерывание назад в операционную систему МП 8086.

Команды POPF, PUSHF и IRET чувствительны к IOPL, так что монитор V86 может контролировать любые изменения флага разрешения прерываний (IF). Команды CLI и STI чувствительны к IOPL как в программах МП 8086, так и в программах МП 80386.

8.5.6. Ввод-вывод в виртуальном режиме

Программы, написанные для МП 8086, могут быть предназначены для однозадачной системы и содержать прямые обращения к устройствам ввода-вывода. В мультизадачной среде эти программы действуют разрушительно. Разработчики

системы вместо прямых обращений могут использовать другие подходы. Выбор метода управления вводом-выводом зависит от того, отображаются ли порты ввода-вывода в общее пространство памяти или в отдельное пространство ввода-вывода.

Некоторые возможности управления:

- выборочно захватывать и затем эмулировать ссылки задачи на конкретные порты вода-вывода;
- захватывать или перенаправлять ссылки на адреса устройств ввода-вывода, отображенные на память;
- эмулировать операционную систему МП 8086, как программу МП 80386 и потребовать от нее выполнения операций ввода-вывода с помощью системных программных прерываний. Захватывать все попытки прямого ввода-вывода.

Ввод-вывод с отдельным адресным пространством в режиме V86 имеет одно отличие от защищенного режима. Механизм защиты не анализирует IOPL при выполнении команд IN, INS, OUT и OUTS; только битовая карта разрешения ввода-вывода контролирует выполнение задачами V86 этих команд ввода-вывода. Битовая карта разрешения ввода-вывода выборочно захватывает команды ввода-вывода в зависимости от адреса, на который они ссылаются. Поскольку в каждой задаче есть своя битовая карта разрешения ввода-вывода, захваченные задачами адреса могут различаться.

Каждая задача, выполняющая ввод-вывод в адресном пространстве, общем с основной памятью, должна иметь одну или несколько страниц для пространства адресов устройств ввода-вывода. Монитор V86 может в этом случае контролировать ввод-вывод либо 1) с помощью ловушек монитора, вызывающих отказ страницы, отображенной на память (страницы с доступом только для чтения будут захватывать операции записи, а отсутствующие страницы – как операции записи, так и чтения), либо 2) назначением странице, отображенной на память, соответствующих физических адресов. Для некоторых устройств ввода-вывода вмешательство монитора может оказаться чрезмерным. В таком случае отказ страницы может быть вызван при первой операции ввода-вывода. Затем монитор убеждается в том, что устройство имеет монопольный доступ к устройству, и устанавливает в состоянии страницы биты «присутствие» и «чтение-запись». Это позволяет последующим операциям ввода-вывода выполняться без задержек.

8.5.7. Различия виртуального МП 8086 и МП 80386

Большая часть программ выполняется одинаково на процессорах 8086 и МП 80386 в виртуальном режиме. Следует, однако, иметь в виду некоторые различия.

- Коды операций, не определенные для процессоров 8086 и 8088, вызывают исключение 6. Если, однако, неопределенный для МП 8086/88 код операции транслируется в код операции, определенный для МП 80386, он будет выполнен так, как это предусмотрено в МП 80386.
- Поскольку МП 80386 требует для выполнения большинства команд меньше машинных тактов, чем МП 8086 или 8088, в двух случаях могут возникнуть проблемы: при отработке запланированных задержек в программах, использующих параллельную работу МП 8086/88 и арифметического сопроцессора 8087, а также при управлении устройствами ввода-вывода, для которых необходима задержка между операциями ввода-вывода.
- В случае переполнения при выполнении команды IDIV МП 80386 выдает в качестве частного максимальное отрицательное число. МП 8086/88 вызывает исключение 0.
- В МП 8086 и 8088 в случае исключения 0 при делении указатель команд

(CS:IP) указывает на команду, *следующую* за несостоятельной. В МП 80386 CS:IP указывает прямо на несостоятельную команду.

- В МП 8086/88 отсутствуют ограничения на длину команды. МП 80386 устанавливает предел длины команды 15 байт и при нарушении этого предела возникает исключение 13. Единственным способом превышения предела на МП 80386 является использование перед командой префиксов удлинения.
- Различается расположение битов флагов с 12 по 15 в зависимости от того, были ли они записаны в реальном режиме или в режиме V86. МП 8086 записывает в них 1. V86 *всегда* записывает 0 в бит 15, а в битах 14...12 оставляет предыдущие значения.
- Сигнал INT МП 8087 использует контроллер прерываний, в то время как сигнал ошибки МП 80387, направленный в МП 80386, через контроллер прерываний не проходит. Может потребоваться удаление команд сопроцессора, использующих контроллер прерываний.
- В системах МП 8086/88 сохраненное значение CS:IP указывает на саму команду ESC. В МП 80386 значение CS:IP (сохраненное для обработки исключений сопроцессора) указывает на любые префиксы *перед* командой ESC.
- Если в системе с МП 8086/88 для обработки прерываний от МП 8087 используется вектор, отличный от 16, оба вектора должны указывать на программу обработки исключений – ошибок сопроцессора. Любая система с МП 80386 и сопроцессором должна использовать вектор 16 для обработки ошибок сопроцессора.
- Команда PUSH в МП 8086/88 заносит в стек значение SP *после* выполнения инкремента. В МП 80386 значение SP записывается в стек *до* его инкремента в процессе выполнения команды PUSH.
- МП 80386 маскирует в счетчиках любых линейных и циклических сдвигов все биты, кроме младших пяти. Эта операция ограничивает величину сдвигов 31 битом и, следовательно, ограничивает задержку отклика на прерывание, ожидающее окончания выполнения команды.
- В МП 8086 относительный адрес 0 (например, занесение в стек слова командой PUSH, когда SP = 1) или попытка обращения к операнду в памяти с превышением относительного адреса 65535 (например, пересылка слова командой MOV по относительному адресу 65535) приводит к циклическому возврату относительного адреса по модулю 65536. МП 80386 в этих случаях возбуждает исключение (с номером 12, если сегмент является сегментом стека и 13 в случае сегмента данных).
- Если при последовательном обращении происходит переход через относительный адрес 65535, в МП 8086 байт следующей команды выбирается по относительному адресу 0 того же сегмента. МП 80386 возбуждает в этом случае исключение 13.

В МП 80286 функция блокирования магистрали реализуется не так, как в МП 80386. Это отличие может влиять или не влиять на программы МП 8086 в зависимости от того, как монитор обрабатывает префикс LOCK. Если программы МП 8086 выполняют LOCK непосредственно, то при использовании специфичных для МП 8086 форм блокирования памяти эти программы, возможно, не будут правильно выполняться в системе с МП 80386.

8.6. Выполнение программы в защищенном режиме МП 80286

Программы, предназначенные для выполнения в защищенном режиме на МП 80286, выполняются без изменений на МП 80386, поскольку, согласно требованиям разработки, МП 80386 является расширением МП 8086 и 80286.

Входящие в систему МП 80286 дескрипторы, исполняемые сегменты, вентили задачи и таблицы локальных дескрипторов используются как в МП 80286, так и в МП 80386. Для МП 80386 разработан новый вариант дескрипторов сегмента состояния задачи, но оба варианта дескрипторов (для МП 80286 и МП 80386) могут одновременно использоваться в одной и той же системе. Работая с общими дескрипторами, МП 80386 анализирует последнее слово и, если обнаруживает в нем нули, рассматривает дескриптор как принадлежащий МП 80286.

Перенося на МП 80386 программы МП 80286 следует иметь в виду два возможных случая.

Случай 1. Прикладные программы МП 80286 переносятся в среду МП 80386, в которой используются принадлежащие МП 80386 системный компоновщик, загрузчик и операционная система.

Сегменты состояния задачи, представляющие МП 80286, следует заменить сегментами МП 80386. Изменять объектные модули МП 80286 нет необходимости. Сегменты состояния задачи обычно строятся операционной системой, загрузчиком или системным компоновщиком.

Случай 2. Переносится целиком система МП 80286 вместе с операционной системой МП 80286, загрузчиком и системным компоновщиком.

Все задачи работают с сегментами состояния задачи МП 80286. МП 80386 функционирует как быстрый МП 80286.

8.6.1. Ограничения префикса LOCK

Префикс LOCK и соответствующий ему выходной сигнал следует использовать для предотвращения прерывания операций пересылки данных другими задатчиками магистрали. В типичных конфигурациях с МП 8086 и 80286 LOCK блокировал все физическое пространство памяти. В МП 80386 гарантированно блокируется доступ к памяти процессора, выполняющего блокируемую команду, если она адресуется к *той же самой* области памяти, т. е. содержит операнд с тем же стартовым адресом и той же длины.

LOCK можно использовать только со следующими командами, когда они модифицируют память: BT, BTS, BTR, BTC, XCHG, INC, DEC, NOT, NEG, ADD, ADC, SUB, SBB, AND, OR или XOR. При использовании LOCK перед любой другой командой генерируется неопределенный код операции.

Функция блокировки магистрали реализована по-разному в МП 80286 и 80386. Программы, использующие специфичные для МП 80286 формы блокирования памяти и перенесенные в прикладное программное обеспечение МП 80386, могут выполняться неправильно.

8.6.2. Циклический возврат адреса в МП 80286

При использовании МП 80286, комбинация базы и относительного адреса, приводящая к адресации за пределы 16 Мбайт, вызывает циклический возврат (оборачивание) адреса к первому мегабайту адресного пространства МП 80286. В МП 80386, имеющем большее физическое адресное пространство, эти адреса попадают на 17-й мегабайт. Если в программе используется циклический возврат, реализуемый в МП 80286, того же эффекта можно добиться с помощью разбиения на страницы. Для этого следует отобразить первые 64 кбайт 17-го мегабайта логических адресов на физические адреса в первом мегабайте.

8.7. Режим реальных адресов МП 80386

В режиме реальных адресов МП 80386 выполняет объектные программы, предназначенные для режима реального адреса МП 8086, 8088, 80186 и 80286. Архитектура МП 80386 почти идентична архитектуре МП 8086/88 и МП 80186/188. С точки зрения программиста, МП 80386, работающий в режиме реальных адресов, представляет собой быстродействующий МП 8086 с некоторым расширением состава регистров и команд.

8.7.1. Вход и выход из режима реальных адресов

Режим реальных адресов устанавливается автоматически после сигнала RESET. Даже если систему предполагается использовать в защищенном режиме, начальная программа (загрузки или инициализации системы) выполняется во временно инициализированном режиме реальных адресов.

Единственный способ выхода из режима реальных адресов – переключение в защищенный режим. МП 80386 входит в защищенный режим, когда команда пересылки MOV в регистр CR0 устанавливает в CR0 бит PE (включение защиты). Для обеспечения совместимости с МП 80286 бит PE можно устанавливать с помощью команды LMSW.

Если программа с помощью команды пересылки в CR0 сбрасывает бит PE, МП 80386 входит в режим реальных адресов. При этом обязательно выполнение определенных шагов.

1. Если включено разбиение на страницы:
   - Обеспечьте равенство линейных адресов физическим. .
   - Очистите бит PG (разбиение на страницы) в CR0.
   - Заполните CR3 нулями, чтобы очистить кэш-память страниц.
2. Передайте управление сегменту с границей 64K. Это загрузит регистр CS требуемой для реального режима границей.
3. Загрузите в сегментные регистры (SS, DS, ES, FS и GS) селектор дескриптора, содержащего соответствующие реальному режиму значения:
   - база = любое значение;
   - граница = 64K;
   - байтовая дробность: G = 0;
   - расширение вверх: E = 0;
   - доступ для записи: W = 1;
   - присутствие: P = 1.
4. Запретите маскируемые прерывания INTR, например, командой CLI. Немаскируемые прерывания NMI можно запретить внешней электрической цепью.
5. Очистите бит PE.
6. Выполните команду перехода в точку программы, где требуется реальный режим. Используйте команду дальнего перехода, которая очищает очередь команд и помещает правильные значения прав доступа в CS.
7. Обеспечьте установку векторов прерываний МП 8086 от адреса 0 до 4* (число источников прерываний) – 4.
8. Разрешите прерывания.
9. Загрузите сегментные регистры в соответствии с требованиями программы реального режима.

8.7.2. Формирование физического адреса в реальном режиме

МП 80386 предоставляет программам МП 8086 байтовое пространство памяти размером 1 Мбайт плюс 64 кбайт. Настройка сегментов выполняется, как и в МП 8086: 16-разрядное значение, записанное в селекторе сегмента, сдвигается влево на 4 бит (с добавлением нулей справа) и образует базовый адрес сегмента. Исполнительный адрес расширяется влево четырьмя нулями и прибавляется к базовому, образуя линейный адрес. Поскольку разбиение на страницы в режиме реального адреса отсутствует, линейный адрес эквивалентен физическому. При образовании линейного адреса может возникнуть перенос. В МП 8086 этот перенос усекается. В МП 80386 бит переноса записывается в разряд 20 линейного адреса.

В отличие от МП 8086 и 80286 МП 80386 может генерировать 32-разрядные исполнительные адреса с помощью префикса размера адреса. Значение этого адреса должно быть в диапазоне от 0 до 65536, в противном случае возникает исключение: прерывание 12 или 13 без кода ошибки. Это делается для совместимости с МП 80286.

8.7.3. Новые исключения МП 80386

Подробное описание всех исключений и прерываний, распознаваемых МП 80386, дано в гл. 3. Введено 8 новых исключений.

Исключение 5 – выполнялась команда BOUND со значением операнда в регистре, выходящим за значение границы.

Исключение 6 – был обнаружен неопределенный код операции, либо префикс LOCK был некорректно использован перед командой, с которой он не согласуется.

Исключение 7 – встретилась команда ESC, в то время как установлен бит EM (эмуляция сопроцессора) в слове состояния машины MSW. Это исключение возникает также, если встретилась команда WAIT и установлен бит TS (переключение задачи).

Исключение 8 – в результате прерывания или исключения произошел векторный переход на вход в таблицу прерываний с адресом, превышающим значение границы для таблицы прерываний в регистре IDTR. Это может произойти, только если команда LIDT изменила значение границы 3FFH, которое устанавливается по умолчанию и обеспечивает хранение всех 256 дескрипторов прерываний.

Исключение 12 – операнд пересекает границу сегмента стека, например выполняется команда MOV по адресу 0FFFFH или команды PUSH, CALL или INT с SP = 1.

Исключение 13 – операнд пересекает границу какого-либо сегмента, не являющегося сегментом стека. Или сделана попытка при последовательном выполнении команд выйти за относительный адрес 0FFFFH. Или команда (включая префиксы) длиннее 15 байт.

Исключение 14 – при включенном разбиении на страницы попытка трансляции адреса привела к обращению к каталогу страниц или входу в таблицу страниц, которые помечены отсутствующими, либо это обращение нарушает привилегии доступа.

Исключение 16 – в начале выполнения определенных команд ESC обнаружен сигнал ошибки сопроцессора, установленный МП 80287/80387 на выводе ERROR # МП 80386.

8.7.4. Различия реального режима и МП 8086

Работая в режиме реального адреса, МП 80386 обычно правильно выполняет программы для МП 8086/8088 и 80186/80188, записанные в ПЗУ.

Ниже приведен обзор отличий, с которыми может столкнуться программист.

- Коды операций, не определенные в МП 8086 и 8088, вызывают исключение 6. Если неопределенный в МП 8086/88 код операции транслируется в код, определенный в МП 80386, он выполняется, как это предусмотрено в МП 80386.
- МП 80386 требует для выполнения большинства команд меньше машинных тактов, чем МП 8086 или МП 8088. В двух случаях из-за этого могут возникнуть проблемы: при отработке запланированных задержек в программах, использующих параллельную работу МП 8086/88 и арифметического сопроцессора 8087, а также при управлении устройствами ввода-вывода, для которых необходима задержка между операциями ввода-вывода.
- В случае переполнения при выполнении команды IDIV МП 80386 выдает в качестве частного максимальное отрицательное число, МП 8086/88 возбуждает исключение 0.
- В МП 8086 и 8088 в случае исключения при делении (DIV) указатель команд (CS:IP) указывал на *следующую* команду *за* несостоятельной. В МП 80386 CS:IP указывает прямо на несостоятельную команду.
- В МП 8086/88 отсутствуют ограничения на длину команды. МП 80386 устанавливает предел длины команды 15 байт, и при нарушении этого предела возникает исключение 13. Единственным способом превышения предела на МП 80386 является использование перед командой префиксов удлинения.
- Различается расположение битов флагов с 12 по 15 в зависимости от того, были ли они записаны процессором 8086 или виртуальным МП 8086. МП 8086 записывает в них 1. V86 *всегда* записывает 0 в бит 15, а в битах 14...12 оставляет предыдущие значение.
- Сигнал INT МП 8087 использует контроллер прерываний, в то время как сигнал ошибки МП 80387, направленный в МП 80386, через контроллер прерываний не проходит. Может потребоваться удаление команд сопроцессора, использующих контроллер прерываний.
- В системах МП 8086/88 сохраненное значение CS:IP указывает на саму команду ESC. В МП 80386 значение CS:IP (сохраненное для обработки исключений сопроцессора) указывает на любые префиксы *перед* командой ESC.
- Если в системе с МП 8086/88 для обработки прерываний от МП 8087 используется вектор, отличный от 16, оба вектора должны указывать на обработчик исключений - ошибок сопроцессора. Любая система с МП 80386 и сопроцессором должна использовать вектор 16 для обработки ошибок сопроцессора.
- Дополнительные 6 исключений возникают, только если программа для МП 8086 содержит скрытую ошибку. Для интерпретации этих исключений как неправильных операций в систему надо включить программы обработки исключений. Поскольку обычно такие прерывания не возникают, эти дополнительные программы не влияют существенным образом на существующие программы для МП 8086. Заметьте, что упомянутые программы обработки прерываний не могли быть включены в программное обеспечение МП 8086, так как они попадают в диапазон, зарезервированный фирмой Intel.
- Команда PUSH в МП 8086/88 заносит в стек значение SP *после* выполнения инкремента. В МП 80386 она записывает в стек значение SP *до* его инкремента в процессе выполнения команды PUSH.
- МП 80386 маскирует в счетчиках любых линейных и циклических сдвигов все биты, кроме младших пяти. Эта операция ограничивает величину сдвигов 31 битом

и, следовательно, ограничивает задержку отклика на прерывание, ожидающее окончания выполнения команды.

- В МП 8086 относительный адрес 0 (например, занесение слова в стек командой PUSH, когда SP = 1) или попытка обращения к операнду в памяти с превышением относительного адреса 65535 (например, пересылка слова командой MOV по относительному адресу 65535) приводят к циклическому возврату относительного адреса по модулю 65536. МП 80386 в этих случаях возбуждает исключения: 12, если сегмент является сегментом стека, и 13 в случае сегмента данных.
- Если при последовательном обращении происходит переход через относительный адрес 65535, в МП 8086 байт следующей команды выбирается по относительному адресу 0 того же сегмента. МП 80386 возбуждает в этом случае исключение 13.
- Если МП 80386 обнаруживает немаскируемое прерывание, оно блокируется до выполнения команды IRET.
- МП 80386 всегда устанавливает сигнал LOCK в процессе выполнения команды XCHG с обращением к памяти, даже если префикс LOCK отсутствует. Исключение неопределенного кода (прерывание 6) возникает при использовании префикса LOCK перед любой командой, кроме ADD, ADC, AND, BTS, BTR, BTC, DEC, INC, NEG, NOT, OR, SBB, SUB, XCHG или XOR.
- В семействе процессоров 8086 невозможно адресовать пространство, большее 1 Мбайт. МП 8086 может формировать адреса не длиннее 20 бит. В адресах большей длины старшие биты усекаются, что приводит к циклическому возврату слишком длинных адресов в диапазон 00000H ...0FFEFH.

8.7.5. Различия реального режима и МП 80286

Различия МП 80386 в режиме реального адреса и МП 80286 вряд ли повлияют на выполнение существующих программ для МП 80286, кроме, возможно, инициализации системы. Всего имеются 4 различия.

1. После получения сигнала RESET процессором 80386 некоторые регистры общего назначения могут содержать иные значения, чем в МП 80286. Проблема несовместимости, однако, не возникает, так как эти регистры не используются в МП 80286.

2. В МП 80286 в регистр MSW (CR0 в МП 80386) записывается начальное значение FFF0H. В МП 80386 начальное значение в регистре составляет 0000H. Это может привести к ошибкам в программах, читающих содержимое MSW. Однако в МП 80286 эти биты не определены и не должны использоваться.

3. В МП 80386 и 80286 по-разному реализуется блокирование магистрали. Программы для МП 80286, использующие специфичные для этого процессора формы блокирования памяти, могут при переносе на МП 80386 выполняться неправильно. В типичных конфигурациях с МП 8086 и 80286 блокируется все физическое пространство памяти. В МП 80386 гарантированно блокируется доступ к памяти процессора, выполняющего блокируемую команду, если она адресуется к той же самой области памяти, т. е. содержит операнд с тем же начальным адресом и той же длины.

4. В МП 80286 в качестве стартовой используется ячейка с адресом 0FFFFF0H, отстоящая на 16 байт от границы 24-разрядного адресного пространства. В МП 80386 стартовая ячейка имеет адрес 0FFFFFFF0H и отстоит на 16 байт от границы 32-разрядного адресного пространства. Большинство программ инициализации МП 80286, записанных в ПЗУ, будет работать правильно. В некоторых случаях придется переопределить внешние сигналы на линиях A_{31-20}.

8.8. Сброс и инициализация

МП запускается или перезапускается сигналом RESET. Обнаружив на линии RESET переход из низкого в высокое состояние, процессор прекращает работу. Когда сигнал RESET снова становится низким, происходит инициализация МП 80386 и переход на выполнение команд со стартового адреса.

Сигнал RESET генерируется микросхемой 82384 (генератором тактовых сигналов). Для обеспечения инициализации уровень сигнала на входе RESET МП 80386 должен оставаться высоким по крайней мере в течение 15 периодов CLK2, в случае же самотестирования – в течение по крайней мере 80 периодов CLK2. После того как установились напряжение питания V_{cc} и частота сигналов CLK2, сигнал RESET должен оставаться в высоком состоянии по крайней мере 1 мс. МП 80386 не выполняет внутренних запросов магистрали перед выборкой первой команды, поэтому при получении запроса HOLD процессор освобождает магистраль. МП 80386 осуществляет выборку первой команды по линейному адресу 0FFFFFFF0H где-то между 350-м и 450-м тактами CLK2 после перехода сигнала RESET из высокого в низкое состояние. Обычно по этому адресу расположена команда JMP, осуществляющая переход на программу начальной загрузки.

По сигналу RESET на выводах шины данных устанавливается выключенное (высокоомное) состояние, на других выводах – состояние высокого или низкого уровней:

D31 – D0	Выключено
W/R #, M/IO #, HLDA, BE3 # – BE0 #	Низкий
LOCK #, D/C #, ADS #, A31 – A2	Высокий

8.8.1. Регистры после сброса

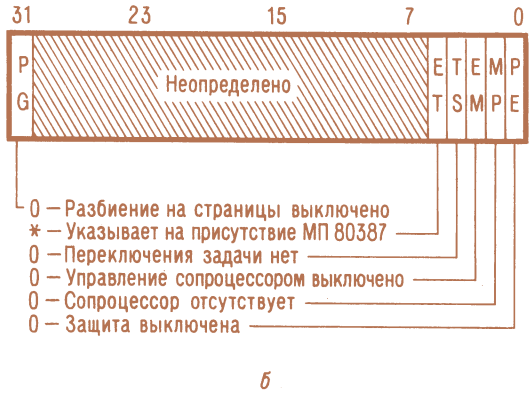
Содержимое регистра EAX зависит от результатов выполнения процедуры самотестирования при включении питания. Если тестирование МП 80386 прошло успешно, в EAX заносится 0. Отличное от 0 значение в EAX после самотестирования говорит о возможной неисправности этого МП.

На рис. 8.2 показано, как в регистре DX размещаются идентификатор изделия и номер модификации. Поле DH содержит число 3, указывающее на тип микропроцессора: МП 80386. В поле DL заносится номер модификации.

На рис. 8.2 показано также содержимое регистра управления (CR0). Бит ET установлен, что говорит о наличии в системе сопроцессора 80387. Если бит ET сброшен, система либо содержит сопроцессор 80287, либо арифметический сопроцессор вообще отсутствует; программа должна различать эти возможности. Содержимое остальных регистров и состояние флагов указано ниже, причем предполагается, что МП 80386 начинает работать в режиме реального адреса с запрещенными прерываниями. Регистры, не вошедшие в перечень, считаются неопределенными.

Регистры	Содержимое
Селектор CS	0000H
Селектор DS	0000H
Селектор ES	0000H
Селектор FS	0000H
Селектор GS	0000H
Селектор SS	0000H

а



б

Рис. 8.2. Содержимое EDX (*а*) и CR0 (*б*) после сигнала RESET.

Регистр EFLAGS	00000002H
Указатель команд IP	0000FFF0H
Регистр таблицы дескрипторов прерываний IDTR	База = 0, граница = 03FFH

8.8.2. Программная инициализация режима реальных адресов

Перед тем как запускать программы в реальном режиме, следует инициализировать некоторые структуры. Для того чтобы можно было выполнять команды с обращением к стеку, надо загрузить регистр сегмента стека (SS). Он должен указывать на область оперативной памяти.

Сигнал RESET автоматически устанавливает на адресных линиях A_{20-31} высокий уровень. Это позволяет инициализировать систему с помощью ПЗУ, расположенного в конце адресного пространства. Автоматическая установка адресных линий вместе с начальным значением CS:IP приводит к выполнению команды с физическим адресом FFFFFFF0H. С помощью внутрисегментных (ближних) форм команд передачи управления можно осуществить переход на другие адреса, принадлежащие верхним 64 кбайт адресного пространства. Первая же межсегментная (дальняя) команда JMP или CALL снимает высокий уровень с линий A_{31-20}. МП 80386 продолжает выполнять команды в нижнем мегабайте физической памяти.

Пока МП 80386 находится в начальном состоянии, прерывания запрещены. Процессор, однако, при возбуждении немаскируемого прерывания или исключения

пытается обратиться к таблице прерываний. В программе инициализации должно быть предусмотрено одно из следующих действий:

– заполните все позиции IDT, которые могут использоваться прерываниями или исключениями, указателями работоспособных программ обработки прерываний;

– занесите в IDTR указатель таблицы прерываний;

– измените значение границы в IDTR на 0, это останавливает систему при возникновении исключения или немаскируемого прерывания.

8.8.3. Программная инициализация защищенного режима

Как правило, большая часть действий по инициализации защищенного режима выполняется либо перед переключением в этот режим, либо непосредственно после переключения. Если инициализация осуществляется после переключения, выполняемые затем процедуры не должны использовать не инициализированные еще средства защищенного режима.

Процедура инициализации может выполняться в защищенном режиме перед инициализацией регистра задачи. Однако перед первым переключением задачи необходимо обеспечить следующее.

1. Регистр задачи должен указывать на область сохранения состояния текущей задачи. После первого переключения информация, занесенная в эту область, теряет свое значение и область можно использовать с другими целями.

2. Должен существовать достоверный TSS новой задачи. Указатель стека в TSS (для уровней привилегий, численно меньших или равных начальному CPL) должен указывать на достоверный сегмент стека.

Регистр сегмента стека (SS) можно загружать и в защищенном, и в реальном режимах. Если SS загрузить в реальном режиме, он и после переключения в защищенный режим продолжает указывать на тот же самый линейный базовый адрес.

Форматы таблицы прерываний отличаются в защищенном и реальном режимах. Поскольку невозможно одновременно переключить режим и изменить формат таблицы прерываний, неизбежно, что в течение некоторого времени IDTR будет определять таблицу прерываний с неправильным форматом. Прерывание или исключение в течение этого времени приведут к непредсказуемым результатам. Чтобы избежать этого, следует запретить прерывания до тех пор, пока не будут размещены все программы обработки прерываний и не будет создана правильная таблица дескрипторов прерываний IDT в защищенном режиме.

Таблицы страниц и регистр базы каталога страниц (PDBR) в CR3 можно инициализировать и в защищенном, и в реальном режимах. Бит PG в CR0 нельзя устанавливать, пока МП 80386 не переключился в защищенный режим. Перед установкой бита PG инициализируйте PDBR в CR3 физическим адресом достоверного каталога страниц. Выберите один из двух способов обеспечения согласованной адресации до и после включения разбиения на страницы:

- Непосредственно за установкой PG должна следовать команда JMP.
- Исполняемая в настоящий момент страница должна отображаться на те же физические адреса как до, так и после установки PG.

Регистр таблицы глобальных дескрипторов (GDTR) должен указывать на достоверную таблицу GDT перед тем, как содержимое какого-либо регистра изменяется в защищенном режиме. Таблицы GDT и LDT следует размещать в оперативной памяти, потому что МП 80386 модифицирует бит доступа в дескрипторах. Инициализацию GDT и GDTR можно провести в реальном режиме.

8.9. Тестирование и отладка

Процессор способен выполнять внутреннее тестирование двух видов: автоматическое самотестирование и тестирование буфера ассоциативной трансляции (TLB). Самотестированием управляет процессор. Разработчику требуется только инициировать тестирование и проверить результаты. Что же касается теста TLB, его надо разрабатывать и вводить в процессор извне. МП 80386 предоставляет интерфейс, упрощающий этот вид тестирования.

8.9.1. Самотестирование

В МП 80386 предусмотрены средства проверки функционирования трех основных программируемых логических матриц (ПЛМ) – точки входа, управления и тестирования, а также содержимого управляющего ПЗУ (CROM). Самотестирование запускается, когда в процессе инициализации становится активным вход BUSY #. Результат тестирования заносится в регистр EAX. Если соответствующий элемент процессора успешно проходит тестирование, содержимое EAX равно 0. Если в EAX не 0, это значит, что самотестирование обнаружило неисправность.

При частоте 16 МГц на самотестирование уходит около 2^{19} тактов или приблизительно 33 мс. Тестирование обнаруживает однобитовые дефекты с вероятностью 100%. Статистически однобитовые дефекты составляют значительную долю всех неисправностей.

8.9.2. Буфер ассоциативной трансляции (TLB)

Буфер ассоциативной трансляции детально обсуждается в гл. 1. Здесь следует подчеркнуть его значение в процедуре тестирования. В этот буфер заносятся данные кэш-памятью дескрипторов страниц, входящей в состав микросхемы МП 80386. В TLB хранятся значения, необходимые для отображения линейных адресов на физические и относящиеся к самым последним обращениям к памяти. Это обеспечивает быструю трансляцию для всех последующих обращений к тем же ячейкам. TLB включает следующие компоненты:

– ассоциативное ЗУ (АЗУ), в котором хранятся 32 линейных адреса (только поля каталога страниц и таблицы страниц) и дополнительные биты признаков (теги), используемые для защиты данных и реализации кэш-операций.

– оперативное ЗУ (ОЗУ), в котором хранятся 32 физических адреса (только старшие 20 разрядов), соответствующие линейным адресам в АЗУ.

Логическая схема, которая реализует четырехвходовую кэш-память и содержит двухразрядный указатель подстановки, определяет, на какой из четырех входов направляются новые данные в процессе записи в TLB.

Преобразуя линейный адрес в физический, МП 80386 сравнивает поля каталога страниц и таблицы страниц линейного адреса с элементом в АЗУ. Если произошло кэш-попадание, соответствующие 20 бит физического адреса извлекаются из ОЗУ и прибавляются к содержимому 12-разрядного поля относительного адреса в линейном адресе. В результате образуется 32-разрядный физический адрес. Если же произошел кэш-промах, МП 80386 записывает в TLB из памяти значения каталога страниц и таблицы страниц.

Имеются два механизма тестирования TLB. Первый заключается в заполнении ячеек TLB, второй осуществляет поиск в TLB. При заполнении в TLB ячейки, определяемой регистром данных, в нее записываются физический адрес из регистра данных и линейный адрес вместе с дополнительными битами признаков из регистра управления. Поиск в TLB осуществляется по линейному адресу (вместе с битами

признаков), содержащемуся в регистре управления. Если произошло кэш-попадание, запишите значение соответствующего физического адреса в регистр данных и установите в нем бит попадание/промах. Если произошел кэш-промах, сбросьте бит попадание/промах. В этом случае физический адрес в регистре данных не определен.

Используются два 32-разрядных регистра тестирования (TR6 и TR7). TR6 – это «регистр управления тестированием», TR7 – «регистр данных тестирования». Можно написать программу для МП 80386, которая будет генерировать тестовые комбинации для проверки TLB. Однако в процессе тестирования TLB механизм разбиения на страницы МП 80386 должен быть выключен, чтобы в TLB записывались только тестовые данные.

Глава 9

Замечания по программированию МП 80386

Как правило, программное обеспечение, разработанное для одного поколения микропроцессоров семейства Intel, можно использовать без всяких модификаций в системах следующего, более высокого уровня. В этой главе приводятся сведения, которые следует иметь в виду, если вы создаете программы или проектируете системы с МП 80386.

9.1. Указания программисту

Прикладные программы, разработанные для МП 8086/8088, прозрачно выполняются в режиме виртуального МП 8086. Виртуальная среда служит мостиком для обеспечения совместимости снизу вверх с существующим прикладным программным обеспечением, предоставляя в то же время новую среду с расширенными функциями и более высокой производительностью. Следует иметь в виду, что производительность прикладных программ, выполняемых в режиме виртуального МП 8086, обычно оказывается ниже, чем в реальном режиме того же процессора. Это происходит из-за вмешательства операционной системы, которая обрабатывает прерывания и эмулирует некоторые команды. К этим издержкам операционной системы добавляется время выполнения программных строк, сохраняющих и восстанавливающих состояние машины и эмулирующих команды. Издержки особенно заметны в программах, насыщенных прерываниями, поскольку чисто программные строки выполняются в виртуальном режиме почти без замедления. Пока для МП 80386 нет работоспособной операционной системы, при разработке программных продуктов, предназначенных для выполнения на новых системах с МП 80386, следует руководствоваться следующими указаниями.

- Если программы будут выполняться на процессорах разных поколений, пишите их для наинизшего семейства, именно 8086/8088.
- Избегайте какого бы то ни было явного или неявного использования битов в регистрах, флагов или структур данных, объявленных неопределенными или зарезервированными для будущих разработок Intel.
- Программы, написанные специально для МП 80386, не должны быть чувствительны к состоянию бита PE (включение защиты) в слове состояния машины в CR0. Установка PE заставляет процессор перейти в защищенный режим. Видимость бита PE при выполнении команды записи в слово состояния машины (SMSW) может привести к неприятностям, если программа пытается выполняться по-разному, в зависимости от того, работает ли МП 80386 в реальном или защищенном режиме.
- Значения некоторых регистров и флагов после запуска отличаются для разных процессоров. Не пользуйтесь начальным, после включения, состоянием

конкретного процессора. Программа должна загружать требуемые значения явным образом.

- Не используйте коды операций, не описанные явным образом в документации. Код операции, не являющийся составной частью поддерживаемой процессором системы команд, может быть определен в следующем поколении иначе, даже если он, казалось, выполнял определенные функции в ранних моделях процессоров.
- Для МП 80386 характерно значительно более быстрое выполнение конкретных последовательностей команд, чем для ранних моделей процессоров. Любая программа, взаимодействующая с событиями реального времени или основывающая свои действия на времени исполнения команд, должна использовать отсчет времени, независимый от тактовой частоты МП 80386.

9.2. Программирование МП 80386

МП 80386 обеспечивает совместимость прикладного программного обеспечения, разработанного для ранних моделей процессоров Intel, и в то же время предоставляет программную среду с большим 32-разрядным линейным адресным пространством. Имеется ряд соображений, которые стоит учитывать при программировании специально для среды МП 80386 и которые не следуют прямо из документации. Эти соображения приведены ниже.

9.2.1. Память

1. Формирование физического адреса в реальном режиме. В противоположность МП 8086 в МП 80386 линейный адрес, получающийся при вычислении исполнительного, может содержать до 21 значащего разряда. При сложении базового и исполнительного адресов возникает возможность переноса. В МП 8086 бит переноса усекается. В МП 80386 бит переноса сохраняется в разряде 20 линейного адреса.

2. В отличие от МП 8086 и 80286 32-разрядный исполнительный адрес может быть образован с помощью префикса размера адреса. Однако значение 32-разрядного адреса в реальном режиме не должно превышать 65536, иначе возбуждается исключение.

3. В МП 80286 любая комбинация базового и относительного адресов, приводящая к адресации за пределы 16 Мбайт, вызывает циклический возврат к первому мегабайту адресного пространства МП 80286. В МП 80386, имеющем большее физическое пространство адресов, такие адреса попадают в 17-й мегабайт. В случае когда программное обеспечение использует это явление, тот же эффект может быть получен на МП 80386 с помощью разбиения на страницы. Для этого следует отобразить первые 64 кбайт 17-го мегабайта логических адресов на физические адреса в первом мегабайте.

4. Для достижения максимальной гибкости при оптимальном размещении в памяти структур данных *отсутствует* требование выравнивания слов по четным адресам. Точно так же двойные слова *не должны* выравниваться по адресам, кратным четырем. Однако при использовании системы с 32-разрядной шиной фактическая передача данных между процессором и памятью осуществляется двойными словами, начинающимися с адресов, кратных четырем. Невыровненные слова увеличивают число циклов памяти, требуемых для выборки данных, что ухудшает производительность.

5. В тех случаях, когда на МП 80386 выполняются программы, разработанные для специальных архитектур, не имеющих сегментов, может оказаться целесообразным выключить сегментацию. МП 80386 не обеспечивает такого режима.

Однако того же эффекта можно добиться, загрузив первоначально сегментные регистры селекторами дескрипторов, охватывающих все 32-разрядное линейное пространство адресов. После того как дескрипторы загружены, содержимое сегментных регистров не изменяется. 32-разрядные относительные адреса в командах МП 80386 охватывают теперь все линейное пространство адресов.

6. В цикле записи (с 8-разрядными устройствами ввода-вывода), если установлены сигналы BE3# и/или BE2#, но не BE1# или BE0#, записываемые данные, присутствующие на верхней половине шины данных, копируются на ее нижнюю половину. Если адреса двух устройств отличаются только значениями BE3# – BE0# (т.е. адреса лежат в пределах одного двойного слова), следует декодировать комбинацию BE3# – BE0# с целью образования сигнала выбора микросхемы, который будет предотвращать ошибочную запись во второе устройство одновременно с записью в первое. Сигнал выбора микросхемы можно образовать с помощью программируемой пользователем логической матрицы или дискретных ТТЛ-схем.

9.2.2. Дескрипторы

1. Поскольку МП 80386 использует содержимое зарезервированного слова (младшего слова) в каждом дескрипторе, программы для МП 80386, помещающие в это слово какое-либо значение, могут выполняться неправильно на МП 80386.

2. Программы, работающие с таблицами дескрипторов, часто с целью идентификации неиспользуемых входов заносят в поля прав доступа элементов таблицы дескрипторов недействительные значения. Значения прав доступа 80H и 00H остаются недействительными и в МП 80386. Другие значения, которые были недействительными для МП 80286, могут оказаться действительными для МП 80386 из-за введения новых типов дескрипторов.

3. Чтобы отличить дескриптор МП 80286 от дескриптора МП 80386, процессор проверяет старшее слово. Если слово содержит 0, дескриптор относится к МП 80286.

4. Исполняемый сегмент, у которого бит согласованности (бит 10) установлен, называется *согласованным сегментом*. Согласованный сегмент предоставляет возможность совместного использования процедур, которые вызываются с различных уровней привилегий, но должны выполняться на уровне привилегий вызывающей процедуры. Когда управление передается согласованному сегменту, текущий уровень привилегий (CPL) не изменяется. Это единственный случай, когда CPL может оказаться не равным DPL текущего исполняемого сегмента.

9.2.3. Команды программы

1. МП 80286 реализует блокирование магистрали не так, как МП 80386. Программы, использующие специфичные для МП 80286 формы блокирования памяти и перенесенные на МП 80386, могут выполняться неправильно.

2. Префикс LOCK может использоваться только с перечисленными ниже командами МП 80386, если они модифицируют память. При использовании префикса LOCK перед любой другой командой возникает исключение неопределенного кода операции.

Одноопернадные арифметические и логические команды: INC, DEC, NOT и NEG.

Двухоперандные и логические команды: ADD, ADC, SBB, SUB, AND, OR и XOR.

Команда обмена XCHG.

Команды проверки и изменения битов: BT, BTS, BTR, BTC.

3. Префикс LOCK и соответствующий ему выходной сигнал должны использоваться только для предотвращения прерывания операций пересылки данных другими ведущими на магистрали.

4. Команда с блокированием гарантированно блокирует только области памяти, определенные операндом-приемником. Типичные конфигурации с МП 8086 и 80286 блокируют все физическое пространство памяти. В МП 80386 гарантированно блокируется доступ к памяти процессора, выполняющего блокируемую команду, если она адресуется к *той же самой* области памяти, т. е. содержит операнд с *теми же* стартовым адресом и длиной.

5. МП 80386 разрешает переход по командам CALL или JMP непосредственно в другой сегмент, только если выполняется одно из следующих условий: 1) бит согласованности дескриптора программного сегмента адресата установлен *и* DPL адресата меньше или равен CPL; или 2) DPL адресата равен CPL. Большинство программных сегментов *не* согласованы. Для таких сегментов управление может быть передано без вентиля только исполняемым сегментам на том же уровне привилегий. Для передачи управления численно меньшим уровням привилегий команда CALL используется с дескрипторами вентиля вызова. Команда JMP не может передать управление несогласованному сегменту, у которого DPL не равен CPL.

6. Только команды CALL могут использовать вентили для перехода на меньшие уровни привилегий. Команда JMP может использовать вентиль только для перехода к исполняемому сегменту с тем же уровнем привилегий, либо к согласованному сегменту.

7. BSF – прямое сканирование битов. Эта команда сканирует слово или двойное слово и записывает номер первого установленного бита в регистр. Сканируемая строка битов может быть как в регистре, так и в памяти. Если все слово равно 0, т. е. в нем нет ни одного установленного в 1 бита, устанавливается флаг ZF. Если найден бит, установленный в 1, ZF сбрасывается.

Примечание: если установленных битов не найдено, содержимое регистра-приемника не определено.

8. DIV – беззнаковое целочисленное деление. Нецелочисленные частные округляются до целых чисел в сторону уменьшения. Остаток всегда меньше делителя. При беззнаковом делении байтов максимальное частное есть 255; при беззнаковом делении слов – 65535. При беззнаковом делении двойных слов – $2^{32} - 1$.

9. SAL – команда сдвига. Флаг переноса (CF) всегда содержит значение последнего бита, выдвинутого из операнда-приемника. Если при одноразрядном сдвиге изменилось значение старшего бита (знака), устанавливается флаг переполнения OF. Если знаковый бит не изменился, OF сбрасывается. После многоразрядного сдвига содержимое OF *всегда* не определено.

10. Различие команд TEST (логическое сравнение) и AND состоит в том, что TEST не изменяет операнд-приемник. Команда TEST отличается от команды BT (проверка бита) тем, что TEST проверяет за одну операцию значения нескольких битов, а BT – одного.

9.2.4. Регистры

1. Некоторые разряды различных регистров описаны, как «зарезервированные» или «неопределенные». Используя регистры с такими разрядами, считайте их действительно неопределенными. Проверяя значения определенных разрядов регистра, не опирайтесь на значения неопределенных разрядов. Исключайте эти разряды из рассмотрения, маскируя их.

Не опирайтесь на значения неопределенных разрядов, записывая их в память или в другой регистр; не надейтесь, что эти разряды будут хранить записанную в них информацию. Загружая из памяти регистры, всегда загружайте в неопределенные или зарезервированные разряды нули, либо в неизменном виде то, что в них содержалось в памяти.

2. Разряд VM регистра EFLAGS можно установить в 1 только двумя способами: командой IRET в защищенном режиме, причем только если текущий уровень привилегий равен 0; и при переключении задачи на любом уровне привилегий.

3. Младшие 16 бит регистра CR0 представляют собой слово состояния машины МП 80386; к ним можно обращаться отдельно, используя обозначение MSW.

9.2.5. Задачи

1. Единственный способ выйти из реального режима заключается в преднамеренном переключении в защищенный режим. Если с помощью команды MOV в регистре CR0 устанавливается разряд включения защиты (PE), МП 80386 переходит в защищенный режим.

2. Сегмент состояния задачи (TSS) может располагаться в любом месте линейного пространства адресов. Надо только соблюдать осторожность, когда TSS пересекает границу страницы. Если МП 80386, читая содержимое TSS при переключении задачи, наталкивается на отсутствующую страницу, возбуждается исключение. Чтобы избежать этого, либо располагайте TSS так, чтобы он не пересекал границ страниц, либо позаботьтесь о том, чтобы обе страницы к моменту переключения задачи присутствовали (или обе отсутствовали). В последнем случае при отсутствии обеих страниц программа обработки отказов страниц перед перезапуском команды, вызвавшей отказ, назначает обе страницы присутствующими.

3. Задачи *не* реентерабельны. B-бит в дескрипторе TSS позволяет процессору зафиксировать попытку переключения на выполняемую уже задачу.

4. В дескрипторе TSS поле LIMIT определяет размер сегмента. Это поле должно содержать значение 103 или выше. Попытка переключения на задачу, у которой LIMIT меньше 103, вызывает исключение.

5. Каждое переключение задачи устанавливает разряд TS в MSW (в CR0). Бит TS сигнализирует о том, что контекст арифметического сопроцессора может *не* соответствовать текущей задаче МП 80386.

9.2.6. Привилегии и защита

1. Уровень привилегий, на котором возобновляется выполнение поступающей задачи, никак не ограничивается уровнем привилегий уходящей задачи.

2. При включенном разбиении на страницы процессор сначала проверяет защиту сегмента, затем оценивает защиту страницы. Если МП 80386 обнаруживает на любом из этих уровней нарушение защиты, затребованная операция отменяется и возбуждается исключение–ошибка защиты.

3. Процессор проверяет информацию о типе (в дескрипторе сегмента) в двух случаях.

- Когда селектор или дескриптор загружается в регистр сегмента. Некоторые сегментные регистры могут содержать только вполне определенные типы дескрипторов, именно:

в регистр SS можно загружать только селекторы сегментов данных с разрешением записи;

в регистр CS можно загрузить только селектор исполняемого сегмента;

селекторы исполняемых сегментов (с запретом чтения) нельзя загружать в регистры сегментов данных.

- Когда команда явно или неявно обращается к регистру сегмента. Некоторые сегменты могут взаимодействовать с командами лишь вполне определенным, установленным заранее образом:

никакая команда не может читать из сегмента данных, если не установлен бит разрешения чтения;

никакая команда не может записывать в сегмент данных, если не установлен бит разрешения записи;

никакая команда не может записывать в исполняемый сегмент.

4. Комбинирование защиты страниц и сегментов: можно определить достаточно большой сегмент данных, в котором имеются некоторые субблоки с разрешением только чтения и другие субблоки с разрешением записи/чтения. В этом случае элементы (входы) каталога страниц или таблиц страниц для субблоков с разрешением только чтения будут содержать биты U/S (пользователь/супервизор) и R/W (чтение/запись), установленные в состояние x0. Это означает, что все страницы, описываемые данным элементом каталога или все конкретные страницы, не будут иметь прав записи.

Такая методика может оказаться полезной в Unix-подобных системах для определения большого сегмента данных, часть которого предназначена только для чтения (разделяемые данные или константы, записанные в ПЗУ). Это позволяет системе определить «сплошное» пространство данных как один большой сегмент, использовать «сплошные» указатели для адресации внутри этого пространства и тем не менее иметь возможность защитить разделяемые данные, области супервизора и разделяемые файлы, отображенные на виртуальное пространство.

5. Сегменты программы могут содержать константы, но в них нельзя записывать. Имеются три способа чтения данных из сегмента программы.

- Используйте префикс замены сегмента в регистре CS для чтения из исполняемого сегмента с разрешением чтения, селектор которого уже загружен в регистр CS.
- Загрузите в регистр сегмента данных селектор несогласованного исполняемого сегмента с разрешением чтения.
- Загрузите в регистр сегмента данных селектор согласованного исполняемого сегмента с разрешением чтения.

Первый вариант всегда правилен, потому что DPL программного сегмента в регистре CS по определению равен CPL. Во втором варианте используются те же правила, что и при доступе к сегменту данных. Третий вариант всегда правилен, потому что уровень привилегий сегмента с установленным битом согласованности равен CPL независимо от его DPL.

9.2.7. Тестирование и отладка

1. Поскольку первый вход таблицы глобальных дескрипторов (GDT) не используется процессором, селектор с нулевым индексом и нулевым индикатором таблицы можно использовать как нуль-селектор. Загрузка регистра сегмента (не CS или SS) нуль-селектором *не* возбуждает исключение. Однако исключение *возбуждается*, когда регистр сегмента используется для обращения к памяти. Это свойство полезно для обнаружения непредусмотренных ссылок, если предварительно инициализировать неиспользуемые сегменты.

2. Буфер ассоциативной трансляции (TLB) представляет кэш-память, используемую для трансляции линейных адресов в физические. Заметьте, что механизм тестирования TLB специфичен для МП 80386 и может быть изменен в будущих

моделях процессоров. Программное обеспечение, использующее этот механизм в настоящем виде, может оказаться несовместимым с будущими моделями процессоров.

3. Инверсии битов D, U и W в регистре тестирования 6 (TR6) введены для принудительной фиксации попаданий и промахов при поиске в TLB. Если и прямой, и инверсный биты сброшены, поиск в TLB фиксирует промах. Если оба бита установлены, фиксируется попадание. Операция записи *всегда* выполняется при противоположных значениях прямого и инверсного битов.

Существенно, чтобы один и тот же линейный адрес не был записан более чем в один элемент буфера ассоциативной трансляции. Если это произойдет, информация о попадании, полученная в ходе просмотра TLB, окажется неопределенной.

4. Операция тестирования TLB. Для поиска или чтения элемента TLB перешлите в TR6 двойное слово, содержащее соответствующий линейный адрес с атрибутами. Установите C = 1 (команда поиска). Запишите содержимое TR7. Если бит HT в TR7 равен 1, тогда остальные поля соответствуют содержимому TLB. Если HT = 0, остальные поля не определены. В процессе тестирования бит V играет роль дополнительного бита адреса. При запросе на поиск бит V следует установить, чтобы неинициализированные биты признаков не давали попадания. Если какие-либо биты признаков не инициализированы, поиск в TLB с V = 0 дает непредсказуемые результаты.

Для записи в элемент TLB перешлите в TR7 двойное слово, содержащее требуемый физический адрес, а также значения HT и REP. HT должен содержать 1. REP должен указывать на ассоциативный блок, в который надо поместить новый элемент. Перешлите в TR6 двойное слово, содержащее соответствующий линейный адрес, а также значения V, D, U и W. Установите C = 0 (команда записи).

Проследите за тем, чтобы биты признаков не повторялись. Результаты будут неопределенными.

Глава 10

Система команд МП 80386

Система команд МП 80386 представляет собой расширение команд предыдущих поколений процессоров вместе с дополнительными командами, специфичными для МП 80386. В настоящей главе все команды перечислены в алфавитном порядке их мнемонических обозначений. Для каждой команды приведены возможные комбинации операндов, образуемые объектные коды и описание команды.

10.1. Формат команд

Команды МП 80386 состоят из отдельных элементов и могут иметь различные форматы. Из всех описанных ниже элементов только один (код операции, Коп) обязательно присутствует в любой команде. Остальные элементы могут отсутствовать, что определяется характером операции, а также местоположением и типом операндов.

Команды состоят из следующих элементов: необязательных префиксов; одного или двух байтов кода операции; возможно – описателя адреса, который включает байт Mod R/M[1)] и байт масштаба, индекса и базы; смещения – если требуется; поля непосредственных данных – если требуется. Коды всех команд оказываются подмножествами общего формата команды, изображенного на рис. 10.1.

Ниже описаны элементы команды в том порядке, как они расположены на рис. 10.1.

Префиксы – один или несколько байтов, предшествующих команде и модифицирующих операцию этой команды. Имеется 4 типа префиксов.

1. Повторение – используется с командами обработки строк; заставляет команду воздействовать на каждый элемент строки.

2. Размер операнда – переключает разрядность операндов, устанавливая их 32-разрядными или 16-разрядными.

3. Размер адреса – переключает разрядность адреса, определяя образование 32-разрядных или 16-разрядных адресов.

4. Замена сегмента – в явной форме указывает, какой сегментный регистр должна использовать команда. Префикс отменяет действующий по умолчанию выбор сегментного регистра, обычно осуществляемый МП 80386 при выполнении этой команды.

Код операции (Коп) – описывает операцию, выполняемую командой. Некоторым командам присущи несколько кодов операций, каждый из которых описывает определенный вариант операции.

Описатель регистра – в команде могут быть описаны один или два регистра в

[1)] Mod R/M – модификатор, регистр-память. – *Прим. перев.*

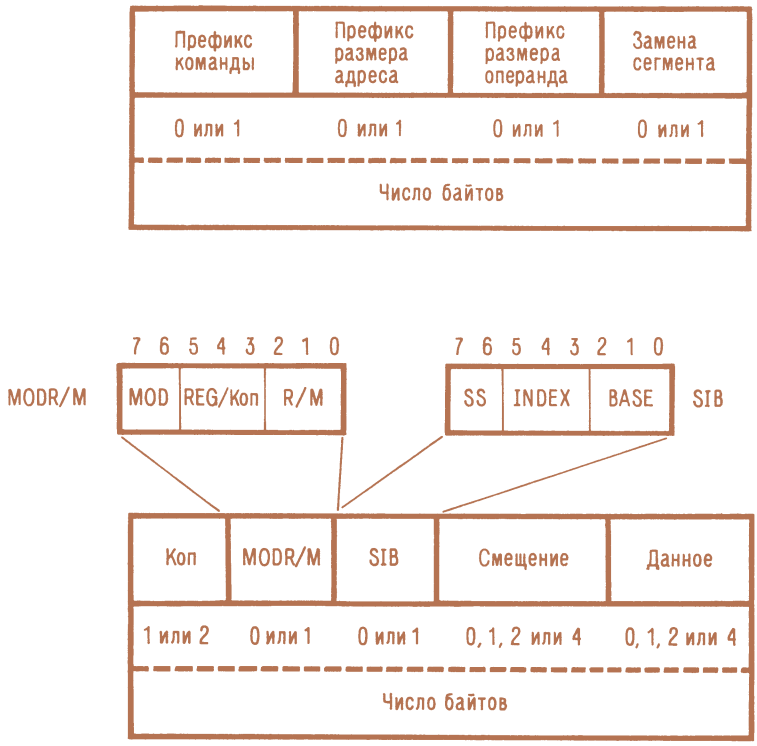


Рис. 10.1. Общий формат команды.

качестве операндов. Описатель регистра может присутствовать как в байте кода операции, так и в байте описателя режима адресации [1].

Описатель режима адресации. Этот элемент, если он присутствует, описывает, является ли операнд содержимым регистра или ячейки памяти. Если операнд находится в памяти, описатель режима указывает, надо ли использовать смещение, индексный регистр, регистр базы и масштабирование [2].

Байты MOD R/M и SIB (Scale–Index–Base, масштаб–индекс–база). Большинство команд, ссылающихся на операнд, находящийся в памяти, содержат после байта (или байтов) основного кода операции еще байт формы адресации. Этот байт описывает используемую форму адреса. Определенные значения кода поля MOD R/M указывают на наличие второго адресного байта SIB.

Байты MOD R/M и SIB содержат следующую информацию:

- тип индексации или номер регистра, используемого в команде;
- используемый регистр или дополнительную информацию о выборе команды;
- информацию о базе, индексе и масштабе.

Байт MOD R/M содержит три поля:

---

[1] В первом случае код операции подразумевает использование определенного регистра в качестве одного из операндов (см., например, варианты команды ADD с кодами 04 и 05); во втором случае используемый регистр указывается в трех битах поля REG.– *Прим. перев.*

[2] Речь идет о поле MOD байта MODR/M. См. приведенные далее таблицы значения поля MOD. *Прим. перев.*

MOD – разряды 7 и 6, составляющие два старших разряда байта. Вместе с полем R/M они образуют 32 возможных значения: восемь регистров и 24 режима индексации.

REG – разряды 5, 4 и 3, которые описывают номер регистра либо являются дополнительными разрядами кода операции. Значение поля REG определяется первым байтом команды (байтом Коп).

R/M – разряды 2, 1 и 0, которые могут описывать регистр в качестве местоположения операнда или образовывать вместе с полем MOD код режима адресации, как упомянуто выше.

Байт SIB включает следующие три поля:

SS – разряды 7 и 6, описывающие масштабный коэффициент.

INDEX – разряды 5, 4 и 3, описывающие номер индексного регистра.

BASE – разряды 2, 1 и 0, описывающие номер регистра базы.

Режимы адресации, приводящие к образованию 32-разрядного адреса, кодируются так, как описано ниже. Все регистровые модификаторы для вычисления адреса представляют собой 32-разрядные регистры (например, EAX). Вычисление адреса при отсутствии байта SIB выполняется следующим образом.

- MOD = 00. Вычисляется адрес вида DS:[r/m]. Поле r/m описывает используемый регистр:

```
000 DS:[EAX]
001 DS:[ECX]
010 DS:[EDX]
011 DS:[EBX]
100 Присутствует SIB
101 DS:32-разрядное смещение
110 DS:[ESI]
111 DS:[EDI]
```

Таким образом, исполнительный адрес определяется как DS:32-разрядный модификатор, находящийся в регистре или в памяти.

- MOD = 01. К адресам, вычисленным по вышеприведенной таблице, прибавляется 8-разрядное смещение, так что при r/m = 000 адрес составит DS:[EAX + смещение].

```
000 DS:[EAX + 8-разрядное смещение]
100 Присутствует SIB
101 SS:[EBP + 8-разрядное смещение]
```

- MOD = 10. Прибавляется 32-разрядное смещение. При r/m = 000 адрес составит DS:[EAX + смещение]

```
000 DS:[EAX + 32-разрядное смещение]
100 Присутствует SIB
101 SS:[EBP + 32-разрядное смещение]
```

- MOD = 11. В поле r/m указан 8- или 32-разрядный регистр. Выбор разрядности регистра зависит от состояния бита W[1)], определяющего разрядность операции (8 или 32 разряда).

[1)] Так называемый старший бит кода команды (см., например, варианты команды ADD с кодами 02 и 03). – *Прим. перев.*

000 AL или EAX
001 CL или ECX
010 DL или EDX
011 BL или EBX
100 AH или ESP
101 CH или EBP
110 DH или ESI
111 BH или EDI

Значение MOD указывает длину поля смещения, прибавляемого к адресу:

00 – смещение отсутствует (кроме базы 101)
01 – 8-разрядное смещение
10 – 32-разрядное смещение

Таким образом, поле SIB модифицирует вычисленный адрес (DS:[база + + масштабированный индекс + смещение]), за исключением тех случаев, когда в качестве базы используются регистры ESP или EBP и когда вместо обозначения сегмента DS: надо подставить SS.

Если байт SIB присутствует, то r/m = 100, а MOD = 00, 01 или 10. При вычислении адреса используется 32-разрядное значение масштабированного индекса, которое находится как содержимое индексного регистра, умноженное на 2^{ss}, где ss – двоичное число, содержащееся в первых двух разрядах поля SIB. Индексный регистр определяется по содержимому поля INDEX следующим образом:

000 EAX
001 ECX
010 EDX
011 EBX
100 индекс отсутствует, масштаб должен быть равен 0
101 EBP
110 ESI
111 EDI

Если байт SIB присутствует в команде, его двоичные разряды можно обозначить SSIIIBBB, где SS – масштаб, III – индекс, а BBB – база. Значение базы определяет базовый регистр:

000 EAX
001 ECX
010 EDX
011 EBX
100 ESP
101 Особый случай. База определяется, как 32-разрядное смещение, если MOD = 0, в противном случае берется из EBP
110 ESI
111 EDI

Смещение. Если описатель режима адресации[1)] указывает, что при вычислении адреса операнда будет использовано смещение, в состав кода команды включается поле смещения. Смещение представляет собой 8-, 16- или 32-разрядное целое число

[1)] То есть поле MOD.– *Прим. перев.*

со знаком. 8-разрядная форма используется в тех случаях, когда значение смещения невелико.

Непосредственный операнд (данное). Если этот элемент присутствует, он представляет значение операнда команды. Непосредственные операнды могут быть 8-, 16- или 32-разрядными. В случаях когда 8-разрядный непосредственный операнд используется в команде вместе с 16- или 32-разрядным операндом, процессор автоматически увеличивает размер 8-разрядного операнда путем расширения его знакового разряда.

10.1.1. Описание обозначений

Ниже объясняются условные обозначения и сокращения, используемые при описании системы команд.

+rb, +rw, +rd Код регистра (от 0 до 7), который складывается с указанным слева от знака плюс шестнадцатеричным числом (байтом) с образованием байта кода операции. Регистрам присвоены следующие коды:

rb	rw	rd
AL = 0	AX = 0	EAX = 0
CL = 1	CX = 1	ECX = 1
DL = 2	DX = 2	EDX = 2
BL = 3	BX = 3	EBX = 3
AH = 4	SP = 4	ESP = 4
CH = 5	BP = 5	EBP = 5
DH = 6	SI = 6	ESI = 6
BH = 7	DI = 7	EDI = 7

/цифра Цифра, стоящая справа от косой черты, имеет, вообще говоря, значение от 0 до 7. Она показывает, что в байте MOD R/M указывается только один операнд r/m (регистр/память). Сама цифра помещается в поле reg и образует расширение кода операции.

/r Обозначение показывает, что байт MOD R/M содержит два операнда (reg *и* r/m).

cb, cw, cd, cp Величина размером 1 байт (cb), 2 байта (cw), 4 байта (cd) или 6 байт (cp) следует за кодом операции и определяет относительный адрес, а также, возможно, новое значение регистра программного сегмента.

ib, iw, id Непосредственный операнд размером 1 байт (ib), 2 байта (iw) или 4 байта (id) следует за байтами кода операции, MOD R/M или SIB. Код операции указывает, является ли операнд знаковой величиной. Обратите внимание на то, что все слова и двойные слова записываются в порядке возрастания старшинства байтов (первым стоит младший байт).

imm8 Непосредственный операнд размером 1 байт. Величина imm8 является знаковой величиной между −128 и +127 включительно. В командах, где размер второго операнда составляет слово или двойное слово, величина imm8 расширяется до слова или двойного слова. Старшие байты расширенной величины заполняются старшим битом непосредственного операнда.

imm16 Непосредственный операнд размером 1 слово. Используется в командах с атрибутом размера операнда 16 разрядов. Может иметь значение от −32768 до +32767 включительно.

imm32 Непосредственный операнд размером 1 двойное слово. Используется в командах с атрибутом размера операнда 32 разряда. Может иметь значение от −2147483648 до +2147483647 включительно.

m8 Байт памяти. Адресуется через регистры DS:SI или ES:DI.
m16 Слово памяти. Адресуется через регистры DS:SI или ES:DI.
m32 Двойное слово памяти. Адресуется через регистры DS:SI или ES:DI.
moffs8, moffs16, moffs32 Относительный адрес. Простая переменная типа BYTE, WORD или DWORD, используемая некоторыми вариантами команды MOV. Фактический адрес записывается в виде простого смещения относительно базы сегмента. Число при аббревиатуре "moffs" указывает разрядность смещения, определяемую атрибутом размера адреса в команде. Байт MOD R/M в команде не используется.
ptr16:16, ptr16:32 Указатель FAR (дальний). Обычно он находится в другом программном сегменте по отношению к команде. Обозначение 16:16 говорит о том, что указатель состоит из двух частей. Величина слева от двоеточия – это смещение в сегменте-приемнике. Величина справа от двоеточия – это 16-разрядный селектор или величина, предназначенная для регистра программного сегмента. Если атрибут размера операнда команды равен 16, используйте 16:16. Для 32-разрядного атрибута используйте 16:32.
r8 Один из байтовых регистров: AL, CL, DL, BL, AH, CH, DH или BH.
r16 Один из однословных регистров: AX, CX, DX, BX, SP, BP, SI или DI.
r32 Один из двухсловных регистров: EAX, ECX, EDX, EBX, ESP, EBP, ESI или EDI.
rel8 Смещение для перехода в диапазоне от 128 байтов перед концом команды до 127 байтов после конца команды.
rel16, rel32 Смещение для перехода в том же программном сегменте, что и ассемблируемая команда. rel16 относится к командам с атрибутом размера операнда 16 разрядов. rel32 относится к командам с атрибутом размера операнда 32 разряда.
r/m8 Однобайтовый операнд. Представляет собой содержимое либо байта памяти, либо байтового регистра – AL, BL, CL, DL, AH, BH, CH или DH.
r/m16 Однословный операнд – содержимое однословного регистра или ячейки памяти. Используется в командах с атрибутом размера операнда 16 разрядов. Содержимое памяти извлекается по найденному путем вычислений исполнительному адресу. Однословные регистры: AX, BX, CX, DX, SP, BP, SI или DI.
r/m32 Двухсловный операнд – содержимое двухсловного регистра или ячейки памяти. Используется в командах с атрибутом размера операнда 32 разряда. Содержимое памяти извлекается по найденному путем вычислений исполнительному адресу. Двухсловные регистры: EAX, EBX, ECX, EDX, ESP, EBP, ESI или EDI.
rrr Когда обозначение rrr появляется в колонке двоичного эквивалента кода операции[1)], оно указывает конкретный адресуемый регистр. Коды rrr обозначают:

000 = AX/EAX	100 = SP/ESP
001 = CX/ECX	101 = BP/EBP
010 = DX/EDX	110 = SI/ESI
011 = BX/EBX	111 = DI/EDI

Sreg Регистр сегмента. Кодирование сегментных регистров в поле reg: ES = 0, CS = 1, SS = 2, DS = 3, FS = 4 и GS = 5.

[1)] Именно кода операции, а не байта MODR/M, где также имеется обозначение rrr. См., например, варианты команды PUSH с кодами 50.– *Прим. перев.*

10.1.2. Описание полей MOD и R/M кода команды

В приведенном ниже перечне команд коды некоторых команд сопровождаются описанием байта MODR/M[1]. Если mm = 00, то поле r/m описывает память, если mm = 11 – регистр (rrr). Ниже указаны правила определения операнда по содержимому полей mm и r/m, причем disp8, disp16 и disp32 обозначают 8-, 16- и 32-разрядное смещение соответственно.

Двоичные обозначения регистров, используемые в поле rrr байта MODR/M

rrr	8-разрядные регистры	16- или 32-разрядные регистры
000	AL	AX, EAX
001	CL	CX, ECX
010	DL	DX, EDX
011	BL	BX, EBX
100	AH	SP, ESP
101	CH	BC, EBP
110	DH	SI, ESI
111	BH	DI, EDI

16-разрядные операции

r/mmod	00	01	10	11
000	[BX + SI]	+ disp8	+ disp16	rrr
001	[BX + DI]			
010	[BP + SI]			
011	[BP + DI]			
100	[SI]			
101	[DI]			
110	disp16	[BP] + disp8	[BP] + disp16	[BP] + disp16
111	[BX]	+ disp8	+ disp16	

32-разрядные операции

r/mmod	00	01	10	11
000	[EAX]	+ disp8	+ disp32	rrr
001	[ECX]	+ disp8		
010	[EDX]	+ disp8		
011	[EBX]	+ disp8		
100	Следует SIB			
101	disp32	[EBP] + disp8	[EBP] + disp32	
110	[ESI]	[EBP] + disp8		
111	[EDI]			

[1] Обозначение вида mm rrr r/m, где mm описывает два бита поля MOD, rrr – три бита поля REG, а r/m – три бита поля R/M. – *Прим. перев.*

Байт SIB

SS	Множитель	Индекс	База
00	1	000 EAX	000 EAX
01	2	001 ECX	001 ECX
10	4	010 EDX	010 EDX
11	8	011 EBX	011 EBX
		100 Нет	100 ESP
		101 EBP	101 Особый случай
		110 ESI	110 ESI
		111 EDI	111 EDI

10.1.3. Примеры выполнения команд

Описания большинства команд, приведенные в этой главе, сопровождаются, в целях демонстрации, реальными программными строками. Команды, используемые только в защищенном режиме, проиллюстрированы в приложении 5. Еще четыре команды (ESC, HLT, LOCK и WAIT) даны без примеров ввиду особенностей их выполнения (например, команда HLT останавливает процессор).

В некоторых случаях команды объединены в небольшие программы с целью демонстрации «нормального» использования комбинаций этих команд. На каждом шаге таких программ показаны состояния регистров и флагов и даны пояснения выполняемых действий. Содержимое указателя команд получает приращение после каждой команды, в результате чего он указывает на следующую команду. Звездочки под обозначением флагов показывают, какие флаги изменили состояние в результате выполнения конкретной команды.

Примеры команд имеют целью показать взаимодействие флагов и т. д. Они не предназначены для обучения составлению эффективных программ.

10.1.4. Используемые флаги и регистры

Ниже приведен перечень флагов и регистров, используемых в описании и примерах команд. При возникновении вопросов обращайтесь к этому списку. Флаги и регистры подробно описаны в гл. 1.

AF	Вспомогательный перенос	Бит 4 регистра EFLAGS
CF	Флаг переноса	Бит 0 регистра EFLAGS
DF	Флаг направления	Бит 10 регистра EFLAGS
EIP	Указатель команд	32-разрядный регистр
EM	Эмуляция	Бит 2 регистра управления CR0
ET	Тип расширения	Бит 4 регистра управления CR0
IF	Разрешение прерываний	Бит 9 регистра EFLAGS
IOPL	Уровень привилегий ввода-вывода	Биты 12–13 регистра EFLAGS
MP	Присутствие арифметического сопроцессора	Бит 1 регистра управления CR0
NT	Флаг вложенности	Бит 14 регистра EFLAGS
OF	Флаг переполнения	Бит 11 регистра EFLAGS
PE	Включение защиты	Бит 0 регистра управления CR0
PF	Флаг четности	Бит 2 регистра EFLAGS
PG	Бит разбиения на страницы	Бит 31 регистра управления CR0
RF	Флаг возобновления	Бит 16 регистра EFLAGS

SF	Флаг знака	Бит 7 регистра EFLAGS
TF	Флаг ловушки	Бит 8 регистра EFLAGS
TS	Переключение задачи	Бит 3 регистра управления CR0
VM	Виртуальный режим МП 8086	Бит 17 регистра EFLAGS
ZF	Флаг нуля	Бит 6 регистра EFLAGS
AH	Регистр общего назначения	Старшие 8 разрядов AX
AL	Регистр общего назначения	Младшие 8 разрядов AX
AX	Регистр общего назначения	Младшие 16 разрядов EAX
BH	Регистр общего назначения	Старшие 8 разрядов BX
BL	Регистр общего назначения	Младшие 8 разрядов BX
BX	Регистр общего назначения	Младшие 16 разрядов EBX
CH	Регистр общего назначения	Старшие 8 разрядов CX
CL	Регистр общего назначения	Младшие 8 разрядов CX
CX	Регистр общего назначения	Младшие 16 разрядов ECX
DH	Регистр общего назначения	Старшие 8 разрядов DX
DL	Регистр общего назначения	Младшие 8 разрядов DX
DX	Регистр общего назначения	Младшие 16 разрядов EDX
EAX	Регистр общего назначения	32-разрядный
EBP	Регистр общего назначения	32-разрядный
EBX	Регистр общего назначения	32-разрядный
ECX	Регистр общего назначения	32-разрядный
EDI	Регистр общего назначения	32-разрядный
EDX	Регистр общего назначения	32-разрядный
EFLAGS	Регистр флагов	32-разрядный регистр флагов
ESI	Регистр общего назначения	32-разрядный
ESP	Регистр общего назначения	32-разрядный указатель стека
FLAGS	Регистр флагов	Младшие 16 разрядов

10.2. Алфавитный список команд

AAA ASCII-коррекция после сложения (ASCII Adjust Register AX After Addition)

Команда	Код операции	Двоичное представление
AAA	37	00110111

Назначение. Команда AAA преобразует содержимое регистра AL в неупакованное десятичное число и сбрасывает его старшие четыре разряда. Она должна всегда следовать за сложением двух неупакованных десятичных операндов в AL. Если возникает перенос, устанавливается флаг CF и происходит инкремент в регистре AH.

Пример выполнения команды. В нем показано, как команда AAA выполняет коррекцию после сложения так, чтобы после операции двоичного сложения распакованное двоично-десятичное число сохранило свой формат.

Последовательность команд:

```
14EB:0100 B80800      MOV AX,0008
14EB:0103 0404        ADD AL,04

14EB:0105 37          AAA
14EB:0106 0404        ADD AL,04
14EB:0108 37          AAA
```

Регистры и флаги перед выполнением[1]:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                    NV UP EI PL NZ NA PE NC
```

Заметьте, что если флаг изменяет свое состояние после выполнения некоторой команды, под ним ставится звездочка.

```
14EB:0100 B80800              MOV AX,0008

AX=0008  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PE NC
```

Чтобы предоставить команде AAA исходные значения, выполняется сложение.

```
14EB:0103 0404                ADD AL,04

AX=000C  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI PL NZ NA PE NC
```

Заметьте, что результат 0C не является десятичной цифрой.

```
14EB:0105 37                  AAA

AX=0102  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL NZ AC PE CY
                                                *     *
```

Команда AAA выполнила вычитание 10(0A) из 12(0C) в регистре AL. К содержимому AH прибавляется десятичный перенос с образованием двухразрядного двоично-десятичного результата. Поскольку произошел перенос при операции с двоично-десятичными числами, устанавливаются флаги коррекции и переноса.

```
14EB:0106 0404                ADD AL,04

AX=0106  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    NV UP EI PL NZ NA PE NC
                                                *     *
```

---

[1] В обозначения флагов вкладывается следующий смысл: NV – non overflow, отсутствие переполнения; UP – вверх; EI – enable interrupt, разрешение прерываний; PL – plus, плюс; NZ – non zero, не нуль; NA – non adjust, отсутствие коррекции; PE – parity even, четность; NC – non carry, отсутствие переноса. Противоположные значения флагов, которые встретятся в дальнейшем тексте: OV – overflow, переполнение; DN – down, вниз; DI – disable interrupt, запрет прерываний; NG – negative, отрицательный; ZR – zero, нуль; AC – adjust carry, перенос при коррекции; PO – parity odd, нечетность; CY – carry yes, перенос. – *Прим. перев.*

Это действие не приводит к переносам. Оба флага сбрасываются. Результат сложения (06) является правильным двоично-десятичным числом.

```
14EB:0108 37              AAA

AX=0106  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0109
                                    NV UP EI PL NZ NA PE NC
```

Выполнение команды AAA не изменило содержимого AX, чего и следовало ожидать, поскольку результат не нуждался в коррекции.

AAD ASCII-коррекция регистра AX перед делением (ASCII Adjust Register AX Before Division)

Команда	Код операции	Двоичное представление
AAD	D5 0A	11010101 00001010

Назначение. Команда AAD преобразует число в регистрах AH и AL, подготавливая деления двух неупакованных десятичных операндов так, чтобы полученное в результате деления частное было бы правильным неупакованным десятичным числом. В регистре AH должна находиться старшая цифра, в AL – младшая. Команда AAD корректирует число и помещает результат в AL. Регистр AH содержит 0.

Пример выполнения команды. Команды AAD и AAM даны в одном примере, так как их действие взаимно противоположно. Результат выполнения каждой команды показан ниже.

Последовательность команд:

```
14EB:0100 90              NOP
14EB:0101 B80508          MOV AX,0805
14EB:0104 D50A            AAD
14EB:0106 90              NOP
14EB:0107 D40A            AAM
14EB:0109 90              NOP
```

Регистры и флаги перед выполнением:

```
AX=0010  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL ZR NA PO NC

14EB:0101 B80508          MOV AX,0805

AX=0805  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL ZR NA PO NC
```

Команда MOV загрузила в регистр AX две двоично-десятичные цифры. Команда MOV не влияет на состояние флагов.

```
14EB:0104 D50A          AAD

AX=0055  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL NZ NA PE NC
                                     *          *
```

В результате выполнения команды AAD двухразрядное двоично-десятичное число преобразуется в его двоичный эквивалент ($55_{16} = 85_{10}$). Состояние флага нуля зависит от результата в регистре AL (не нуль в данном случае), так же, как и состояние флага четности (в данном случае имеет место четность). Если бы флаг знака был отрицательным, любое правильное преобразование сделало бы его положительным.

```
14EB:0106 90            NOP

AX=0055  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL NZ NA PE NC

14EB:0107 D40A          AAM
AX=0805  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0109
                                    NV UP EI PL NZ NA PE NC
```

Действие команды AAM заключается в преобразовании числа в регистре AF в две двоично-десятичные цифры в регистрах AH и AL.

AAM ASCII-коррекция в регистре AX после умножения (ASCII Adjust AX Register After Multiplication)

Команда	Код операции	Двоичное представление
AAM	D4 0A	11010100 00001010

Назначение. Команда AAM корректирует результат умножения двух неупакованных десятичных чисел. Эта команда должна всегда следовать за умножением двух десятичных цифр с целью образования правильного десятичного результата. Старшая цифра помещается в регистр AH, младшая – в AL.

Пример выполнения команды. Команда AAM дана в одном примере с командой AAD, поскольку действие этих команд взаимно противоположно.

AAS ASCII-коррекция регистра AL после вычитания (ASCII Adjust AL Register After Subtraction)

Команда	Код операции	Двоичное представление
AAS	3F	00111111

Назначение. Команда AAS преобразует содержимое регистра AL в неупакованную десятичную цифру и заполняет нулями старшие четыре разряда. Эта команда должна всегда следовать за вычитанием одного неупакованного десятичного операнда из другого в AL. Флаг CF устанавливается, а при наличии займа выполняется декремент в регистре AH.

Пример выполнения команды. Показана роль команды AAS в вычитании двоично-десятичных чисел.

```
14EB:0100 90            NOP
14EB:0101 B80500        MOV   AX,0005
14EB:0104 2C04          SUB   AL,04
14EB:0106 3F            AAS
14EB:0107 2C04          SUB   AL,04
14EB:0109 3F            AAS
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL NZ NA PO NC

14EB:0101 B80500        MOV AX,0005

AX=0005  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL NZ NA PO NC

14EB:0104 2C04          SUB AL,04

AX=0001  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL NZ NA PO NC
```

Результат вычитания все еще представляет собой положительную величину.

```
14EB:0106 3F            AAS

AX=0001  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL NZ NA PO NC
```

Команда AAS не изменяет результат и не действует на флаги.

```
14EB:0107 2C04          SUB AL,04

AX=00FD  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
```

```
SI=0000    DI=0000    DS=14EB    ES=14EB    SS=14EB    CS=14EB
IP=0109
                                                NV UP EI NG NZ AC PO CY
                                                      *     *     *
```

Это вычитание дает отрицательную величину. Флаг знака устанавливается, флаг переноса тоже устанавливается, потому что для установки флага знака потребовался «заем». Флаг коррекции устанавливается из-за займа из старшего полубайта AL.

```
14EB:0109 3F            AAS

AX=FF07   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010A
                                                NV UP EI NG NZ AC PO CY
```

В результате выполнения команды AAS в регистре AL устанавливается правильное десятичное значение, а из содержимого AH вычитается единица.

ADC Сложение с переносом целых чисел (Add Integers with Carry)

Команда	Код операции	Двоичное представление
ADC AL,imm8	14 ib	00010100
ADC AX,imm16	15 iw	00010101
ADC EAX,imm32	15 id	00010101
ADC r/m8,imm8	90 /2 ib	10010000 mm 010 r/m
ADC r/m16,imm16	90 /2 /w	10010000 mm 010 r/m
ADC r/m32,imm32	91 /2 id	10010001 mm 010 r/m
ADC r/m16,imm8	93 /2 ib	10010011 mm rrr r/m
ADC r/m32,imm8	93 /2 ib	10010011 mm rrr r/m
ADC r/m8,r8	10 /r	00010000 mm rrr r/m
ADC r/m16,r16	11 /r	00010001 mm rrr r/m
ADC r/m32,r32	11 /r	00010001 mm rrr r/m
ADC r8,r/m8	12 /r	00010010 mm rrr r/m
ADC r16,r/m16	13 /r	00010011 mm rrr r/m
ADC r32,r/m32	13 /r	00010011 mm rrr r/m

Назначение. Команда ADC суммирует операнды, прибавляет 1, если установлен флаг CF, и помещает результат по адресу назначения. Если флаг CF сброшен, команда ADC выполняет ту же операцию, что и ADD. Комбинация команды ADD и нескольких команд ADC позволяет складывать числа, содержащие более 32 разрядов.

Пример выполнения команды. Показано влияние установленного флага переноса на выполнение команды. Заметьте, что наличие установленного флага переноса приводит к увеличению суммы на 1.

Последовательность команд:

```
14EB:0100 90            NOP
14EB:0101 B80000        MOV AX,0000
14EB:0104 0440          ADD AL,40
14EB:0106 0440          ADD AL,40
```

```
14EB:0108 0440          ADD AL,40
14EB:010A 0440          ADD AL,40
14EB:010C 0440          ADD AL,40
14EB:010E 90            NOP
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL NZ NA PO NC

14EB:0101 B80000        MOV AX,0000

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL NZ NA PO NC
```

Последовательность команд сложения начинается с команды ADD, которая игнорирует состояние флага переноса. Такое обычно происходит при сложении младших цифр многоразрядных чисел. Флаг переноса мог быть также сброшен принудительно командой CLC (сброс флага переноса).

```
14EB:0104 0440          ADD AL,40

AX=0040  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL NZ NA PO NC

14EB:0106 1440          ADC AL,40

AX=0080  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    OV UP EI NG NZ NA PO NC
                                    *        *
```

Заметьте, что в результате установки знакового разряда (старшего разряда регистра) устанавливается флаг знака. Заметьте также, что устанавливается флаг переполнения, поскольку результат не помещается в регистре-приемнике как знаковая величина.

```
14EB:0108 1440          ADC AL,40

AX=00C0  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                    NV UP EI NG NZ NA PE NC
                                    *                    *
```

Заметьте, что поскольку в регистре AL, содержащем младшие 8 бит результата, число установленных бит четно, устанавливается флаг четности. Указатель переполнения сбрасывается, поскольку знаковый результат помещается в регистре.

```
14EB:010A 1440              ADC AL,40

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010C

                                    NV UP EI PL ZR NA PE CY
                                          *  *        *
```

В результате этого сложения возникает перенос из регистра (AL). Флаг переноса теперь устанавливается. Заметьте влияние этого на выполнение следующей команды. Результирующая сумма оказывается равной 0, поэтому устанавливается флаг нуля. Поскольку знаковый бит нуля равен 0, флаг знака делается положительным.

```
14EB:010C 1440              ADC AL,40

AX=0041  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010E

                                    NV UP EI PL NZ NA PE NC
                                          *           *
```

Поскольку флаг переноса был установлен, к результату прибавляется дополнительная 1. Перенос при этом не возникает, поэтому флаг переноса сбрасывается. Флаг нуля также сбрасывается, так как результат отличен от нуля.

ADD Сложение целых чисел (Add Integers)

Команда	Код операции	Двоичное представление
ADD AL,imm8	04 ib	00000100
ADD AX,imm16	05 /w	00000101
ADD EAX,imm32	05 id	00000101
ADD r/m8,imm8	80 /0 ib	10000000 mm 000 r/m
ADD r/m16,imm16	81 /0 iw	10000001 mm 000 r/m
ADD r/m32,imm32	81 /0 id	10000001 mm 000 r/m
ADD r/m16,imm8	83 /0 ib	10000011 mm 000 r/m
ADD r/m32,imm8	83 /0 ib	10000011 mm 000 r/m
ADD r/m8,r8	00 /r	00000000 mm rrr r/m
ADD r/m16,r16	01 /r	00000001 mm rrr r/m
ADD r/m32,r32	01 /r	00000001 mm rrr r/m
ADD r8,r/m8	02 /r	00000010 mm rrr r/m
ADD r16,r/m16	03 /r	00000011 mm rrr r/m
ADD r32,r/m32	03 /r	00000011 mm rrr r/m

Назначение. В результате выполнения команды операнд-приемник заменяется суммой обоих операндов (источника и приемника). При наличии переполнения устанавливается флаг CF.

Пример выполнения команды. Видно, что команда ADD выполняет сложение независимо от состояния флага переноса.

Последовательность команд:

```
14EB:0100 90          NOP
14EB:0101 B80000      MOV AX,0000
14EB:0104 0440        ADD AL,40
14EB:0106 1440        ADC AL,40
14EB:0108 1440        ADC AL,40
14EB:010A 1440        ADC AL,40
14EB:010C 1440        ADC AL,40
14EB:010E 90          NOP
```

Регистры и флаги перед выполнением:

```
AX=0041  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL NZ NA PE NC

14EB:0101 B80000          MOV AX,0000

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL NZ NA PE NC

14EB:0104 0440            ADD AL,40

AX=0040  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL NZ NA PO NC
                                                        *
```

Команда ADD устанавливает нечетность флага четности, потому что число установленных битов в младших 8 разрядах нечетно.

```
14EB:0106 0440            ADD AL,40

AX=0080  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    OV UP EI NG NZ NA PO NC
                                     *        *
```

Поскольку результат сложения как знаковое 8-разрядное число не помещается в регистре AL, устанавливается флаг переполнения. Знаковый бит числа также устанавливается, в результате флаг знака делается отрицательным.

```
14EB:0108 0440                ADD AL,40

AX=00C0  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                    NV UP EI NG NZ NA PE NC
                                     *
```

Основное изменение в состоянии флагов заключается в сбросе флага переполнения: теперь результат как знаковая величина помещается в регистре-приемнике. В младших 8 разрядах установлено 2 бит, поэтому флаг четности устанавливается четным.

```
14EB:010A 0440                ADD AL,40

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010C
                                    NV UP EI PL ZR NA PE CY
                                          *           *
```

В результате выполнения этой команды ADD возникает перенос из регистра-приемника. Устанавливается флаг переноса. Поскольку знаковый бит результата сброшен, флаг знака делается положительным. Результат равен 0, поэтому устанавливается флаг нуля.

```
14EB:010C 0440                ADD AL,40

AX=0040  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010E
                                    NV UP EI PL NZ NA PO NC
                                          *     *  *
```

Эта команда ADD изменяет состояние флага нуля на ненулевое значение, сбрасывает флаг переноса и сбрасывает флаг четности.

AND Логическое И (Logical AND)

Команда	Код операции	Двоичное представление
AND AL,imm8	24 ib	00100100
AND AX,imm16	25 iw	00100101
AND EAX,imm32	25 id	00100101
AND r/m8,imm8	A0 /4 ib	10100000 mm 100 r/m
AND r/m16,imm16	A1 /4 iw	10100001 mm 100 r/m
AND r/m32,imm32	A1 /4 ib	10100001 mm 100 r/m
AND r/m16,imm8	A3 /4 ib	10100011 mm 100 r/m
AND r/m32,imm8	A3 /4 ib	10100011 mm 100 r/m
AND r/m8,r8	20 /r	00100000 mm rrr r/m
AND r/m16,r16	21 /r	00100001 mm rrr r/m
AND r/m32,r32	21 /r	00100001 mm rrr r/m

```
AND r8,r/m8          22 /r      00100010 mm rrr r/m
AND r16,r/m16        23 /r      00100011 mm rrr r/m
AND r32,r/m32        23 /r      00100011 mm rrr r/m
```

Назначение. Команда AND используется для сброса битов, указанных пользователем, например бита четности во входном потоке кодов ASCII от терминала. Будучи использована совместно с командой сравнения, команда AND позволяет убедиться, что указанные биты установлены. В этом случае команда AND выполняется перед командой сравнения. Обе команды используют заданные маски.

Пример выполнения команды. Показано, как с помощью команды AND можно маскировать биты приемника.

Последовательность команд:

```
14EB:0101 B8EE11        MOV AX,11EE
14EB:0104 25FFFF        AND AX,FFFF
14EB:0107 25F1FF        AND AX,FFF1
14EB:010A 2511FE        AND AX,FE11
14EB:010D 25FFE0        AND AX,E0FF
```

Регистры и флаги перед выполнением:

```
AX=0040  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL NZ NA PO CY
14EB:0101 B8EE11                MOV AX,11EE

AX=11EE  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL NZ NA PO CY
14EB:0104 25FFFF                AND AX,FFFF

AX=11EE  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL NZ NA PE NC
                                        *  *
```

Заметьте, что операция И с маской, в которой установлены все биты, не изменяет операнд-приемник. В зависимости от результата операции устанавливаются флаг четности и нуля, а флаг переноса сбрасывается. Флаг переполнения сбрасывается, а флаг знака устанавливается в соответствии с результатом.

```
14EB:0107 25F1FF                AND AX,FFF1

AX=11E0  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                    NV UP EI PL NZ NA PO NC
                                        *
```

Эта команда AND замаскировала все биты младшего полубайта. В результате изменилась четность младших 8 бит, что отразилось на состоянии флага четности.

```
14EB:010A 2511FE                AND AX,FE11

AX=1000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010D
                                    NV UP EI PL NZ NA PE NC
                                                        *
```

Эта команда AND замаскировала все содержимое регистра AL, сделав его равным 0, и в регистре AX остался установленным только бит в AH. Заметьте, что четность, вычисляемая по 8 младшим битам результата, имеет место. Флаг нуля оказался не нулевым, так как он устанавливается в соответствии с полным результатом.

```
14EB:010D 25FFE0                AND AX,E0FF

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0110
                                    NV UP EI PL ZR NA PE NC
                                                        *
```

В результате выполнения этой команды AND получается 0 и устанавливается флаг нуля.

ARPL Коррекция в селекторе уровня привилегий инициатора запроса (Adjust Requestor Privilege Level of Selector)

Команда	Код операции	Двоичное представление
ARPL r/m16,r16	63/r	01100011 mm rrr r/m

Назначение. Команда ARPL используется системными программами для обеспечения правильности передачи подпрограмме селекторов в качестве параметров. Селекторы не должны требовать больше привилегий, чем разрешено инициатору запроса.

В команде ARPL используются два операнда. Первый представляет собой 16-разрядный регистр или слово памяти, содержащее значение селектора. В качестве второго операнда обычно выступает регистр, содержащий значение селектора программного сегмента CS инициатора запроса.

Пример выполнения команды. Команда ARPL используется системными программами для проверки правильности селекторов, передаваемых как указатели параметров. Команда сравнивает селектор с образцом, содержащим максимально допустимый уровень привилегий (обычно используется селектор CS инициатора запроса) и устанавливает проверяемое значение в соответствии с меньшим из двух уровней привилегий (большее число). Если изменение уровня не потребовалось, флаг нуля сбрасывается. Если уровень пришлось изменить (возможно, попытка нарушить защиту), устанавливается флаг нуля.

Последовательность команд:

```
0070:0100 8CCB          MOV   BX,CS
0070:0102 B87300        MOV   AX,0073
0070:0105 63C3          ARPL  BX,AX
```

Регистры и флаги перед выполнением:

```
EAX=00000000    EBX=00000000    CS=0070
IP=0100

                                NV UP EI PL NZ NA PO NC
```

С целью демонстрации действия команды в примере используется процедура, обратная нормальной, а именно, в качестве проверяемого значения выступает содержимое CS (пример выполняется на уровне 0).

```
0070:0100 8CCB          MOV   BX,CS

EAX=00000000    EBX=00000070    CS=0070
IP=0102
                                NV UP EI PL NZ NA PO NC
```

Установим более низкое значение уровня привилегий (3).

```
0070:0102 B87300        MOV   AX,0073

EAX=00000073    EBX=00000070    CS=0070
IP=0105
                                NV UP EI PL NZ NA PO NC
```

Выполним команду ARPL, которая скорректирует проверяемый дескриптор, установив в нем меньшее значение привилегий, и установит флаг нуля, поскольку потребовалось изменение уровня.

```
0070:0105 63C3          ARPL  BX,AX

EAX=00000073    EBX=00000073    CS=0070
IP=0107
                                NV UP EI PL ZR NA PO NC
```

BOUND Проверка индекса массива на выход за границы (Check Array Index Against Bounds)

Команда	Код операции	Двоичное представление
BOUND r16,m16&16	62 /r	01100010 mm rrr r/m
BOUND r32,m32&32	62 /r	01100010 mm rrr r/m

Назначение. Команда BOUND проверяет, лежит ли знаковая величина, находящаяся в заданном регистре, внутри заданных границ. Если значение этой величины меньше нижней границы или больше верхней, возникает прерывание 5.

Каждое из значений верхней и нижней границ может быть словом или двойным словом.

Блок памяти с описанием нижней и верхней границ массива часто располагается перед самим массивом. В этом случае для доступа к границам массива можно использовать постоянное смещение от начала массива. Поскольку адрес массива уже содержится в регистре, отпадает необходимость специальных вычислений для определения исполнительного адреса границ массива.

Команда BOUND включает два операнда. Первый описывает проверяемый регистр. Во втором находится исполнительный адрес двух знаковых значений границ. Команда BOUND выполняется в предположении, что значения верхней и нижней границ расположены в соседних ячейках памяти. Эти значения границ не могут находиться в регистрах. Если такое случится, будет возбуждено исключение неправильного кода операции.

Пример выполнения команды. В примере команда BOUND проверяет значение, попадающее в границы. Ситуация выхода из границ обрабатывается программой, обслуживающей прерывания с идентификатором 5.

Последовательность команд:

```
14EB:0100 B84000          MOV     AX,0040
14EB:0103 62060002        BOUND   AX,[0200]
14EB:0107 B80101          MOV     AX,0101
14EB:010A 62060002        BOUND   AX,[0200]
```

Регистры и флаги перед выполнением команды:

```
AX=0040  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB

IP=0103                                NV UP EI PL NZ NA PO NC

14EB:0103 62060002        BOUND AX,[0200]        DS:0200=0001
                                                 0202=0100
```

Выполнение команды BOUND в этом случае просто приводит к передаче управления на следующую команду. Содержимое AX не меньше левой границы и не больше правой. Заметьте, что ни один флаг не изменяет своего состояния.

```
AX=0040  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                       NV UP EI PL NZ NA PO NC

14EB:0107 B80101          MOV     AX,0101

AX=0101  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                       NV UP EI PL NZ NA PO NC

14EB:010A 62060002        BOUND AX,[0200]        DS:0200=0001
                                                 0202=0100
```

Выполнение команды BOUND вызывает прерывание 5, поскольку содержимое AX выходит за правую границу.

BSF Прямое сканирование битов (Bit Scan Forward)

Команда	Код операции	Двоичное представление
BSF r16,r/m16	0F BC	00001111 10111100 mm rrr r/m
BSF r32,r/m32	0F BC	00001111 10111100 mm rrr r/m

Назначение. Эта команда сканирует слово или двойное слово в поисках бита, равного 1, и заносит в регистр номер первого установленного бита. Сканируемая строка может находиться как в регистре, так и в памяти. Если все слово равно 0, т. е. в нем нет единичных битов, устанавливается флаг ZF. Если единичный бит найден, флаг ZF сбрасывается.

Примечание: если единичных битов не найдено, значение регистра-приемника не определено.

Это новая команда, специфичная для МП 80386. Она может оказаться полезной при сканировании таблицы распределения ресурсов в поисках свободного бита. Возвращаемый командой номер бита может служить относительным номером элемента в сканируемом слове.

Пример выполнения команды. В примере проиллюстрировано действие команды BSF, когда она обнаруживает установленный бит. Следует иметь в виду, что операция выполняется над 16- или 32-разрядным словом. Разряды в слове нумеруются справа налево. Метод хранения (от младшего байта к старшему) игнорируется. Слово извлекается из памяти и затем сканируется.

Последовательность команд:

```
14EB:0100 B84000          MOV  AX,0040
14EB:0103 0FBC060002      BSR  AX,[0200]
14EB:0108 90              NOP
```

Регистры и флаги перед выполнением команды BSF:

```
AX=0040  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL ZR NA PO NC

14EB:0103 0FBC060002      BSF  AX,[200]          DS:0200=0010


Двоичное      0000 0000 0001 0000
Номер бита    1    0 0    0
              5    8 7    0


AX=0004  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    NV UP EI PL NZ NA PO NC
                                                *
```

Номер первого найденного бита помещается в указанный регистр. Нахождение бита фиксируется сбросом флага нуля.

Данные располагаются в памяти следующим образом:

```
14EB:200   10  00  00  01  00  00  00  00     . .
```

BSR Обратное сканирование битов (Bit Scan Reverse)

Команда	Код операции	Двоичное представление
BSR r16,r/m16	0F BD	00001111 10111101 mm rrr r/m
BSR r32,r/m32	0F BD	00001111 10111101 mm rrr r/m

Назначение. Эта команда сканирует слово или двойное слово в поисках бита, равного 1, и заносит в регистр номер первого установленного бита. Сканируемая строка может находиться как в регистре, так и в памяти. Если все слово равно 0, т. е. в нем нет единичных битов, устанавливается флаг ZF. Если единичный бит найден, флаг ZF сбрасывается. Это новая команда, специфичная для МП 80386.

Примечание: если единичных битов не найдено, значение регистра-приемника не определено.

Пример выполнения команды. Команда BSR действует так же, как и BSF, только сканирование выполняется в обратном направлении. Биты в назначенном слове (или двойном слове) сканируются слева направо (от бита 31/15 до 0), и номер первого найденного бита помещается в указанный регистр. Если все слово/двойное слово равно 0, устанавливается флаг нуля.

Последовательность команд:

```
14EB:0100 B84000          MOV   AX,0040
14EB:0103 0FBD060002      BSF   AX,[0200]
14EB:0108 90              NOP
```

Регистры и флаги перед выполнением команды BSR:

```
AX=0040   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0103
                                        NV UP EI PL ZR NA PE NC

14EB:0103 0FBD060002      BSR   AX,[0200]          DS:0200=0201
                                  Двоичное    0000 0010 0000 0001
                                  Номер бита  1      0 0      0
                                              5      8 7      0

AX=0009   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0108
                                        NV UP EI PL NZ NA PE NC
                                                   *
```

Слово сканируется от бита 15 до первого установленного бита. Его номер помещается в указанный регистр, а флаг нуля сбрасывается.

Данные в памяти:

```
14EB:0200  01 02 00 01 00 00 00 00     ...
```

BT Проверка бита (Bit Test)

Команда	Код операции	Двоичное представление
BT r/m16,r16	0F A3	00001111 10100011 mm rrr r/m
BT r/m32,r32	0F A3	00001111 10100011 mm rrr r/m
BT r/m16,imm8	0F BA /r ib	00001111 10111010 mm 100 r/m
BT r/m32,imm8	0F BA /r ib	00001111 10111010 mm 100 r/m

Назначение. Команда BT служит для определения того, установлен или нет определенный бит в битовом массиве. Значение проверяемого бита копируется во флаг CF. Это новая команда, специфичная для МП 80386.

Пример выполнения команды. Ниже показано, как команда BT проверяет значение указанного бита.

Последовательность команд:

```
14EB:0100 B80900          MOV    AX,0009
14EB:0103 0FA3060002      BT     AX,[0200]
14EB:0108 90              NOP
```

Регистры и флаги перед выполнением команды BT:

```
AX=0009  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PE NC

14EB:0103 0FA3060002      BT   AX,[0200]          DS:0200=0200
                                  Двоичное    0000 0010 0000 0000
                                  Номер бита  1      0 0      0
                                              5      8 7      0

AX=0009  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    NV UP EI PL NZ NA PE CY
                                                          *
```

Команда проверяет бит, указанный в регистре (9), и обнаруживает, что он установлен. Этот результат переносится во флаг переноса. Если бы бит оказался сброшенным, и флаг переноса был бы сброшен.

Данные в памяти:

```
14EB:200   00 02  00 01 00 00 00  00      ...
```

BTC Проверка и инверсия бита (Bit Test and Complement)

Команда	Код операции	Двоичное представление
BTC r/m16,r16	0F BB	00001111 10111011 mm rrr r/m
BTC r/m32,r32	0F BB	00001111 10111011 mm rrr r/m
BTC r/m16,imm8	0F BA /7 ib	00001111 10111010 mm 111 r/m
BTC r/m32,imm8	0F BA /7 ib	00001111 10111010 mm 111 r/m

Назначение. Команда BTC проверяет указанный бит, копирует его в CF и инвертирует найденный бит, т. е. если бит был равен 1, он делается равным 0. Эта новая команда, специфичная для МП 80386.

Пример команды. Показано действие команды BTC. Заметьте, что при первом выполнении команда обнаруживает бит установленным, устанавливает перенос и сбрасывает бит. При втором выполнении команда обнаруживает бит сброшенным, сбрасывает флаг переноса и устанавливает бит.

Последовательность команд:

```
14EB:0100 B80900          MOV AX,0009
14EB:0103 0FBB060002      BTC  AX,[0200]
14EB:0108 0FBB060002      BTC  AX,[0200]
14EB:010D 90              NOP
```

Данные перед выполнением:

```
14EB:0200   00 02  00  01  00 00  00  00      ....
```

Регистры и флаги перед выполнением команды BTC:

```
AX=0009   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0103
                                        NV UP EI PL NZ NA PE NC

14EB:0103 0FBB060002          BTCAX,[0200]

AX=0009   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0108
                                        NV UP EI PL NZ NA PE CY
                                                              *

                14EB:0200   00 00 00 01 00 00 00 00       ........
```

Бит перед выполнением команды был установлен. Бит копируется во флаг переноса. Затем бит в памяти инвертируется.

```
14EB:0103 0FBB060002          BTCAX,[0200]

AX=0009   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010D
                                        NV UP EI PL NZ NA PE NC
                                                             *

                14EB:0200  00 02 00 01 00 00 00 00        ........
```

Бит перед выполнением команды был сброшен. Бит копируется во флаг переноса. Затем инвертируется бит в памяти.

BTR Проверка и сброс бита (Bit Test and Reset)

Команда	Код операции	Двоичное представление
BTR r/m16,r16	0F B3	00001111 10110011 mm rrr r/m
BTR r/m32,r32	0F B3	00001111 10110011 mm rrr r/m
BTR r/m16,imm8	0F BA /6 ib	00001111 10111010 mm 110 r/m
BTR r/m32,imm8	0F BA /6 ib	00001111 10111010 mm 110 r/m

Назначение. Команда BTR проверяет указанный бит, копирует его в CF и сбрасывает найденный бит. Это новая команда, специфичная для МП 80386.

Пример выполнения команды. Ниже показано выполнение команды BTR. Заметьте, что команда копирует бит во флаг переноса и сбрасывает его.

Последовательность команд:

```
14EB:0100 B80900              MOV   AX,0009
14EB:0103 0FB3060002          BTR   AX,[0200]
14EB:0108 90                  NOP
```

Данные перед выполнением:

```
14EB:0200  FF  FF 00 01  00  00 00 00      .
```

Регистры и флаги перед выполнением команды BTR:

```
AX=0009   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0103
                                        NV UP EI PL NZ NA PE NC

14EB:01030 FB3060002          BTRAX,[0200]

AX=0009   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0108
                                        NV UP EI PL NZ NA PE CY
                                                             *

                14EB:0200  FF FD 00 01 00 00 00 00        .}......
```

Перед выполнением команды бит был установлен, что видно по состоянию флага переноса. Затем принудительно сбрасывается бит в памяти.

BTS Проверка и установка бита (Bit Test and Set)

Команда	Код операции	Двоичное представление
BTS r/m16,r16	0F AB	00001111 10101011 mm rrr r/m
BTS r/m32,r32	0F AB	00001111 10101011 mm rrr r/m
BTS r/m16,imm8	0F BA /5 ib	00001111 10111010 mm 101 r/m
BTS r/m32,imm8	0F BA /5 ib	00001111 10111010 mm 101 r/m

Назначение. Команда BTS проверяет указанный бит, копирует его в CF и устанавливает в найденном бите значение 1. Это новая команда, специфичная для МП 80386.

Пример выполнения команды. Ниже показано выполнение команды BTS. Заметьте, что команда копирует бит во флаг переноса и затем устанавливает его.

Последовательность команд:

```
14EB:0100 B80900          MOV  AX,0009
14EB:0103 0FAB060002      BTS  AX,[0200]
14EB:0108 90              NOP
```

Данные перед выполнением:

```
14EB:0200  FF  FD 00 01 00 00 00 00    ....
```

Регистры и флаги перед выполнением команды BTS:

```
AX=0009  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PE CY

14EB:0103 0FAB060002        BTSAX,[0200]


AX=0009  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    NV UP EI PL NZ NA PE NC
                                                         *
              14EB:0200  FF FF 00 01 00 00 00 00    ........
```

Перед выполнением команды бит был сброшен, что видно по состоянию флага переноса. Затем принудительно устанавливается бит в памяти.

CALL Вызов процедуры (Call a Procedure)

Команда	Код операции	Двоичное представление
CALL rel16	E8 cw	11101000
CALL r/m16	FF /2	11111111 mm 010 r/m

```
CALL ptr16:16      9A cd          10011010
CALL m16:16        FF /3          11111111  mm 011 r/m
CALL rel32         E8 cd          11101000
CALL r/m32         FF /2          11111111  mm 010 r/m
CALL ptr16:32      9A cp          10011010
CALL ptr32:32      9A cp          10011010
CALL m16:32        FF /3          11111111  mm 011 r/m
```

Назначение. Команда CALL передает управление из одной точки программного сегмента в другую. Эти точки могут располагаться в одном и том же программном сегменте (*ближний* переход) или принадлежать разным сегментам (*дальний* переход). Перед собственно передачей управления команда CALL сохраняет в стеке адрес следующей за CALL команды и текущее содержимое регистра EIP.

Команды CALL могут быть относительными, прямыми и косвенными. В косвенной команде абсолютный адрес перехода указывается одним из двух способов: (1) МП 80386 извлекает адрес приемника из ячейки памяти, определенной в команде; (2) программа переходит в точку, адрес которой указан в одном из регистров общего назначения (EAX, EDX, ECX, EBX, EBP, ESI или EDI).

Пример выполнения команды. В примере совместно используются команды CALL, ENTER, LEAVE и RET, поскольку они логически «согласуются» друг с другом и выполняют взаимно обратные действия.

Команды вызываемой программы:

```
14EB:0110 C80A0001          ENTER   000A,01
14EB:0114 90                NOP
14EB:0115 C9                LEAVE
14EB:0116 C3                RET
```

Команда вызова:

```
14EB:0101  E80C00      CALL   0110
```

Область, на которую указывает SP:

```
14EB:E9D0   00  00  00  00  00   00  00  00-00  00   00   00  00  00  00  00 ...
```

Регистры и флаги перед выполнением команды:

```
AX=0000   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0101
                                        NV UP EI PL NZ NA PO NC

14EB:0101 E80C00              CALL  0110

AX=0000   BX=0000   CX=0000   DX=0000   SP=EA02   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0110
                                        NV UP EI PL NZ NA PO NC
```

Мы запрашиваем ближний переход. Заметьте, что в результате сложения содержимого указателя команд IP с ближним смещением в IP заносится адрес 0110

(0104 + 000C = 0110). Заметьте также, что содержимое SP уменьшилось на 2. Перед вычислением адреса перехода старое содержимое IP (0104) было сохранено в стеке.

```
14EB:0110 C80A0001          ENTER 000A,01

AX=0000  BX=0000  CX=0000  DX=0000  SP=E9F4  BP=EA00
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0114
                                    NV UP EI PL NZ NA PO NC
```

Обратите внимание на действие команды ENTER. Содержимое регистра BP сохранено в стеке, в BP занесено последнее значение указателя стека SP, затем указатель стека смещается на величину первого операнда.

```
14EB:0114 90                NOP

AX=0000  BX=0000  CX=0000  DX=0000  SP=E9F4  BP=EA00
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0115
                                    NV UP EI PL NZ NA PO NC

14EB:0115 C9                LEAVE

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA02  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0116
                                    NV UP EI PL NZ NA PO NC
```

Команда LEAVE выполняет действия, обратные команде ENTER. Содержимое BP копируется в SP, затем BP загружается из стека. В результате указатель стека позиционируется для выполнения команды RET.

```
14EB:0116 C3                RET

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL NZ NA PO NC
```

Команда RET извлекает из стека значение IP, в результате выполняется команда, следующая за командой CALL.

```
14EB:0104 90                NOP

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI PL NZ NA PO NC
```

CBW Преобразование байта в слово (Convert Byte to Word)
CWDE Преобразование слова в двойное слово (Convert Word to Doubleword)

Команда	Код операции	Двоичное представление
CBW	98	10011000
CWDE	99	10011001

Назначение. Эта команда расширяет значение бита знака в старшую часть более длинного регистра так, чтобы арифметические операции над содержимым этого регистра давали правильные результаты.

Пример выполнения команды. Ниже показано действие команды CBW при расширении положительных и отрицательных величин.

Последовательность команд:

```
14EB:0100 B80504          MOV   AX,0405
14EB:0103 98              CBW
14EB:0104 B8F104          MOV   AX,04F1
14EB:0107 98              CBW
```

Регистры и флаги перед выполнением команды CBW:

```
AX=0405   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0103
                                        NV UP EI PL NZ NA PO NC

14EB:0103 98              CBW

AX=0005   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0104
                                        NV UP EI PL NZ NA PO NC
```

Заметьте, что состояние флагов не изменилось. Регистр AX теперь содержит в слове величину, ранее находившуюся в байте AL.

```
14EB:0104 B8F104          MOV   AX,04F1

AX=04F1   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0107
                                        NV UP EI PL NZ NA PO NC
```

В регистр AL помещается отрицательная величина.

```
14EB:0107 98                    CBW

AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    NV UP EI PL NZ NA PO NC
```

В регистре AX теперь отрицательная величина, полученная путем расширения знака. Флаги не изменились.

CLC Сброс флага переноса (Clear Carry Flag)

Команда	Код операции	Двоичное представление
CLC	F8	11111000

Назначение. Команда сбрасывает флаг CF, записывая в него 0. Остальные флаги остаются без изменений.

Пример выполнения команды. Показано действие команды CLC.

Последовательность команд:

```
14EB:0100 F9            STC
14EB:0101 F8            CLC
14EB:0102 90            NOP
```

Регистры и флаги перед выполнением:

```
AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL NZ NA PO CY
                                                        *
```

Заметьте, что перенос установлен.

```
14EB:0101 F8                    CLC

AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102
                                    NV UP EI PL NZ NA PO NC
                                                        *
```

После выполнения команды CLC флаг переноса сбрасывается.

CLD Сброс флага направления (Clear Direction Flag)

Команда	Код операции	Двоичное представление
CLD	FC	11111100

Назначение. Команда CLD сбрасывает флаг DF. Состояния остальных флагов не изменяются. Если флаг DF сброшен, автоматическая индексация будет выполняться с инкрементом (приращением). Автоматическая индексация используется командами обработки строк.

Пример команды. Ниже показано действие команды CLD.

Последовательность команд:

```
14EB:0100 FD            STD
14EB:0101 FC            CLD
```

Регистры и флаги перед выполнением команды CLD:

```
AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV DN EI PL NZ NA PO NC
                                       *
```

Заметьте, что флаг направления установлен в состояние «вниз».

```
14EB:0101 FC                CLD

AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102
                                    NV UP EI PL NZ NA PO NC
                                       *
```

Теперь флаг направления установлен в состояние «вверх», регистр IP получил приращение, а остальные регистры и флаги остались в прежнем состоянии.

CLI Сброс флага прерываний (Clear Interrupt Flag)

Команда	Код операции	Двоичное представление
CLI	FA	11111010

Назначение. Если текущий уровень привилегий по крайней мере столь же высок, как уровень привилегий ввода-вывода, команда CLI сбрасывает флаг прерываний. Состояние остальных флагов не изменяется.

Примечание: после выполнения команды CLI внешние прерывания блокируются до тех пор, пока флаг прерываний не будет установлен.

Пример выполнения команды. Ниже показано действие команды CLI.

Последовательность команд:

```
14EB:0100 FB            STI
14EB:0101 FA            CLI
14EB:0102 FB            STI
```

Регистры и флаги перед выполнением команды CLI:

```
AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL NZ NA PO NC
                                     *
```

Заметьте, что флаг прерываний установлен (прерывания разрешены).

```
14EB:0101 FA            CLI

AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102
                                    NV UP DI PL NZ NA PO NC
                                     *
```

Флаг прерываний сбрасывается (прерывания запрещены), регистр IP смещается на следующую команду, остальные регистры и флаги не изменяют своего состояния.

CLTS Сброс флага переключения задачи в управляющем регистре 0 (Clear Task-Switched Flag in Control Register Zero CR0)

Команда	Код операции	Двоичное представление
CLTS	0F 06	00001111 00000110

Назначение. Команда CLTS сбрасывает флаг TS в CR0. В МП 80386 флаг TS устанавливается каждый раз при переключении задачи.

Примечание: команда CLTS используется в системном программировании. Она представляет собой привилегированную команду, выполняемую только на нулевом уровне привилегий.

Пример выполнения команды. Команда CLTS сбрасывает флаг переключения задачи в CR0. Эта команда используется диспетчером МП 80386 для сообщения сопроцессору о том, что все полученные им данные или результаты относятся к текущей задаче.

Флаг переключения задачи допускает задержку в сохранении состояния сопроцессора (в которой часто нет необходимости, если произошло переключение на системную задачу, такую, как спулер или обработчик прерываний). Состояние флагов в регистре EFLAGS не изменяется.

Команды:

```
0070:0100 0F20C0        MOV   EAX,CR0
0070:0103 0F06          CLTS
0070:0105 0F20C0        MOV   EAX,CR0
```

Регистры и флаги перед выполнением:

```
EAX=00000000
IP=0100
                                    NV UP EI PL NZ NA PO NC

0070:0100 0F20C0        MOV   EAX,CR0
```

Заметьте, что флаг переключения задачи в копии CR0 находящейся в регистре EAX, установлен.

```
EAX=7FFFFFE9
IP=0103
                                        NV UP EI PL NZ NA PO NC

0070:0103 0F06              CLTS
```

Действие команды CLTS внешне незаметно.

```
EAX=7FFFFFE9
IP=0105
                                        NV UP EI PL NZ NA PO NC

0070:0105 0F20C0            MOV     EAX,CR0
```

Теперь флаг переключения задачи сброшен.

```
EAX=7FFFFFE1
IP=0108
                                        NV UP EI PL NZ NA PO NC
```

CMC Инвертирование флага переноса (Complement Carry Flag CF)

Команда	Код операции	Двоичное представление
CMC	F5	11110101

Назначение. Команда CMC инвертирует состояние флага переноса, т. е. если в CF была установлена 1, теперь будет 0. Состояние остальных флагов не изменяется.

Пример выполнения команды. Показано действие команды CMC.

Последовательность команд:

```
14EB:0100 F5                CMC
14EB:0101 F5                CMC
```

Регистры и флаги перед выполнением:

```
AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                        NV UP EI PL NZ NA PO CY
                                                              *
```

Заметьте, что флаг переноса установлен.

```
14EB:0100 F5                CMC

AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                        NV UP EI PL NZ NA PO NC
                                                              *
```

Команда CMC инвертирует флаг с 1 на 0 и наоборот, см. ниже.

```
14EB:0101 F5                CMC

AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102

                                    NV UP EI PL NZ NA PO CY
                                                        *
```

CMP Сравнение (Compare)

Команда	Код операции	Двоичное представление	
CMP AL,imm8	3C ib	00111100	
CMP AX,imm16	3D iw	00111101	
CMP EAX,imm32	3D id	00111101	
CMP r/m8,imm8	80 /7 ib	10000000	mm 111 r/m
CMP r/m16,imm16	81 /7 iw	10000001	mm 111 r/m
CMP r/m32,imm32	81 /7 id	10000001	mm 111 r/m
CMP r/m16,imm8	83 /7 ib	10000011	mm 111 r/m
CMP r/m32,imm8	83 /7 ib	10000011	mm 111 r/m
CMP r/m8,r8	38 /r	00111000	mm rrr r/m
CMP r/m16,r16	39 /r	00111001	mm rrr r/m
CMP r/m32,r32	39 /r	00111001	mm rrr r/m
CMP r8,m8	3A /r	00111010	mm rrr r/m
CMP r16,r/m16	3B /r	00111011	mm rrr r/m
CMP r32,r/m32	3B /r	00111011	mm rrr r/m

Назначение. Команда CMP выполняет вычитание операнда-источника из операнда-приемника. В соответствии с результатом вычитания команда устанавливает состояние флагов OF, SF, ZF, AF, PF и CF, но не изменяет операнд-источник и операнд-приемник.

Пример выполнения команды. Показано действие команды CMP.

Последовательность команд:

```
14EB:0100 B8FEFE          MOV   AX,FEFE
14EB:0103 BBFD0F          MOV   BX,0FFD
14EB:0106 38E0            CMP   AL,AH
14EB:0108 38F8            CMP   AL,BH
14EB:010A 38D8            CMP   AL,BL
14EB:010C 38C7            CMP   BH,AL
14EB:010E 38C3            CMP   BL,AL
14EB:0110 90              NOP
```

Регистры и флаги перед выполнением команды CMP:

```
AX=FEFE  BX=0FFD  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106

                                    NV UP EI NG NZ AC PO CY
```

```
14EB:0106 38E0              CMP             AL,AH

AX=FEFE  BX=0FFD  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    NV UP EI PL ZR NA PE NC
                                     *  *  *  *  *
```

Сравнение −2 и −2 (FE и FE) дает после внутреннего вычитания нуль. Фиксируются отсутствие переноса, четность, отсутствие коррекции, нулевой результат, положительный знак. Заметьте, что, кроме смещения IP, никакие регистры не изменяют своего содержимого.

```
14EB:0108 38F8              CMP             AL,BH

AX=FEFE  BX=0FFD  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                    NV UP EI NG NZ AC PO NC
                                     *  *  *  *
```

Сравнение 15 и −2 (0F и FE) дает после внутреннего вычитания 15 из −2 ненулевой результат (−17). Устанавливается флаг знака, сбрасывается флаг нуля, фиксируется коррекция, нечетное состояние флага четности (результат EF содержит 7 установленных битов), отсутствие переноса.

```
14EB:010A 38D8              CMP             AL,BL

AX=FEFE  BX=0FFD  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010C
                                    NV UP EI PL NZ NA PO NC
                                     *     *
```

Сравнение −3 и −2 (FD и FE) после внутреннего вычитания −3 из −2 дает ненулевой результат (1). Фиксируются положительное значение, не нуль, коррекция, нечетность и отсутствие переноса.

```
14EB:010C 38C7              CMP             BH,AL

AX=FEFE  BX=0FFD  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010E
                                    NV UP EI PL NZ NA PE CY
                                     *     *
```

Сравнение −2 и 15 (FE и 0F) после внутреннего вычитания −2 из 15 дает ненулевой положительный результат (17, или 11_{16}). Изменение состояния флагов отражают четность результата и перенос из-за займа.

```
14EB:010E 38C3          CMP             BL,AL

AX=FEFE  BX=0FFD  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0110

                                    NV UP EI NG NZ AC PE CY
                                       *        *
```

Сравнение −2 и −3 после внутреннего вычитания (−1, или FF_{16}) дает ненулевой, отрицательный результат. Требуется заем, знак результата отрицателен, установлено четное число (8) битов. Требуется также коррекция.

CMPS Сравнение строковых операндов (Compare String Operands)
CMPSB Сравнение строковых байтов (Compare String-Byte)
CMPSW Сравнение строковых слов (Compare String-Word)
CMPSD Сравнение строковых двойных слов (Compare String-Doubleword)

Команда	Код операции	Двоичное представление
CMPS m8,m8	A6	10100110
CMPS m16,m16	A7	10100111
CMPS m32,m32	A7	10100111

Примечание. CMPSB представляет собой мнемонику ассемблера для команды CMPS m8, m8; CMPSW – мнемонику для команды CMPS m16, m16; CMPSD – мнемонику для команды CMPS m32, m32.

Назначение. Эти команды предназначены только для операций над строками. Они сравнивают по одному элементу каждой строки, причем элементами могут быть байт, слово или двойное слово. Элементы строк адресуются через регистры ESI и EDI. После каждой строковой операции ESI и/или EDI автоматически получают положительное или отрицательное приращение и указывают на следующие элементы строк. Если DF = 0, в индексных регистрах осуществляется инкремент, если DF = 1 – декремент. Величина приращения составляет 1, 2 или 4 в зависимости от размера строкового элемента.

Команда CMPS вычитает строковый элемент-приемник (по адресу ES:EDI) из строкового элемента-источника (по адресу DS:ESI). Затем в зависимости от результата устанавливаются флаги AF, SF, PF, CF и OF. Если строковые элементы равны, ZF = 1. В противном случае ZF = 0. Если DF = 0, МП 80386 выполняет инкремент указателей памяти ESI и EDI для обеих строк. Сегментный регистр, используемый для определения адреса источника, может быть переназначен с помощью префикса замены сегмента. Сегментный регистр приемника *не* может быть изменен.

Ассемблер всегда транслирует команду CMPS, как одну из команд CMPSB (сравнение байтов), CMPSW (сравнение слов) или CMPSD (сравнение двойных слов).

Если команда модифицирована с помощью префиксов (REPE или REPNE), МП 80386 выполняет сравнение текущего элемента строки с содержимым регистра AL для байтовых элементов, AX для однословных элементов, EAX для двухсловных элементов. Завершение повторения операции зависит от конечного состояния флага ZF, а также от значения ECX.

Пример выполнения команды. Пример включает иллюстрацию использования префикса команды REP (в данном случае префикса REPZ) в операции сравнения.

Сравниваемые данные:

```
14EB:0200  41 42 43 41 42 43 41 42-44 41 42 42 41 42 43 00
           A  B  C  A  B  C  A  B  D  A  B  B  A  B  C
14EB:0280  41 42 44 41 42 42 41 42-43 41 42 43 41 42 43 00
           A  B  D  A  B  B  A  B  C  A  B  C  A  B  C
```

Последовательность команд:

```
14EB:0100 BE0002          MOV  SI,0200
14EB:0103 BF8002          MOV  DI,0280
14EB:0106 B90F00          MOV  CX,000F
14EB:0109 F3              REPZ
14EB:010A A6              CMPSB
14EB:010B F3              REPZ
14EB:010C A6              CMPSB
14EB:010D F3              REPZ
14EB:010E A6              CMPSB
14EB:010F F3              REPZ
14EB:0110 A6              CMPSB
14EB:0111 F3              REPZ
14EB:0112 A6              CMPSB
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0200  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL ZR NA PE NC

14EB:0103 BF8002          MOV  DI,0280

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0200  DI=0280  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL ZR NA PE NC

14EB:0106 B90F00          MOV  CX,000F

AX=0000  BX=0000  CX=000F  DX=0000  SP=EA04  BP=0000
SI=0200  DI=0280  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0109
                                    NV UP EI PL ZR NA PE NC
```

Регистры DI, SI и CX теперь установлены для выполнения операции сравнения.

```
14EB:0109 F3              REPZ
14EB:010A A6              CMPSB
```

```
AX=0000  BX=0000  CX=000C  DX=0000  SP=EA04  BP=0000
SI=0203  D=0283   DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010B
                                    NV UP EI NG NZ AC PE CY
                                       *  *  *     *
```

Первая группа REP сравнивает первые 3 байт строк по адресам DS:SI и ES:DI до обнаружения неравенства. Заметьте, что после окончания сравнения SI и DI указывают на байты, следующие за неравными, а CX содержит оставшееся число байтов. Это позволяет начать выполнение следующей группы REP. Результат сравнения, обнаружившего неравенство ($C(43_{16}) - D(44_{16}) = -1(FF_{16})$), отражается в состоянии флагов: перенос, отрицательное число, коррекция и не нуль.

Поскольку регистры заполнены, можно продолжать:

```
14EB:010B F3          REPZ
14EB:010C A6          CMPSB
```

```
AX=0000  BX=0000  CX=0009  DX=0000  SP=EA04  BP=0000
SI=0206  DI=0286  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010D
                                    NV UP EI PL NZ NA PO NC
                                       *     *  *  *
```

Операция сравнения повторяется до обнаружения следующего неравенства, причем регистры опять подготовлены для продолжения. Флаги устанавливаются по результату обнаруженного неравенства ($C(43_{16}) - B(42_{16}) = 1(01_{16})$). Фиксируются положительное число, не нуль, отсутствие коррекции, нечетность, отсутствие переноса.

Снова продолжаем:

```
14EB:010D F3          REPZ
14EB:010E A6          CMPSB
```

```
AX=0000  BX=0000  CX=0006  DX=0000  SP=EA04  BP=0000
SI=0209  DI=0289  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010F
                                    NV UP EI PL NZ NA PO NC
```

Дошли до следующего неравенства. Теперь флаги отражают результат вычитания D – C (тот же, что и при операции C – B на предыдущем этапе).

Снова продолжаем:

```
14EB:010F F3          REPZ
14EB:0110 A6          CMPSB
```

```
AX=0000  BX=0000  CX=0003  DX=0000  SP=EA04  BP=0000
SI=020C  DI=028C  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0111
                                    NV UP EI NG NZ AC PE CY
                                       *     *  *  *
```

Дошли до последнего неравенства. Флаги отражают результат вычитания B – C (тот же, что и при операции C – D на первом этапе).

Заканчиваем выполнение:

```
14EB:0111 F3                REPZ
14EB:0112 A6                CMPSB
```

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0113
                                    NV UP EI PL ZR NA PE NC
                                       *  *  *     *
```

Флаги устанавливаются по результату обнаруженного равенства. Операция REP завершилась, так как закончился счет в регистре CX, что обнаруживается по установленному флагу нуля.

CWD Преобразование слова в двойное слово (Convert Word to Doubleword)

CDQ Преобразование двойного слова в четверное (Convert Doubleword to Quad-Word)

CWDE Преобразование слова в двойное слово с расширением (Convert Word to Doubleword Extended)

Команда	Код операции	Двоичное представление
CWD	99	10011001
CDQ	99	10011001

Примечание: команда CDQ используется в режиме 32-разрядных операций.

Назначение. Команда CWD удваивает размер операнда-источника. Команда расширяет знак слова в регистре AX на регистр DX. Эту команду можно использовать для образования двухсловного делимого из однословного перед операцией однословного деления. Команда CWDE расширяет знак слова в регистре AX на регистр EAX. Команда CDQ расширяет знак двойного слова в регистре EAX на регистр EDX. Эту команду можно использовать для образования четырехсловного делимого из двухсловного перед операцией двухсловного деления.

Пример выполнения команды. Показано выполнение команды CWD

Последовательность команд:

```
14EB:0100 B80001            MOV     AX,0100
14EB:0103 99                CWD
14EB:0104 B800FF            MOV     AX,FF00
14EB:0107 99                CWD
```

Регистры и флаги после выполнения команды MOV:

```
AX=0100  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL ZR NA PE NC
```

```
14EB:0103 99                CWD

AX=0100  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL ZR NA PE NC
```

Заметьте, что единственное изменение касается регистра DX, который во всех разрядах содержит бит знака величины в AX. Флаги не изменяются.

```
14EB:0104 B800FF            MOV     AX,FF00

AX=FF00  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL ZR NA PE NC

14EB:0107 99                CWD

AX=FF00  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    NV UP EI PL ZR NA PE NC
```

Опять бит знака расширился на весь регистр.

DAA Десятичная коррекция в регистре AL после сложения (Decimal Adjust AL Register After Addition)

Команда	Код операции	Двоичное представление
DAA	27	00100111

Назначение. Команда DAA корректирует результат сложения двух правильных упакованных десятичных операндов в регистре AL. Эта команда должна всегда следовать за сложением двух пар упакованных десятичных чисел (по одной цифре в каждом полубайте), чтобы получить в результате пару правильных упакованных десятичных цифр. В случае переноса устанавливается флаг CF.

Пример выполнения команды. Ниже показано выполнение команды DAA, которая используется для коррекции после операции ADD упакованных десятичных чисел.

Последовательность команд:

```
14EB:0100 B011              MOV     AL,11
14EB:0102 0409              ADD     AL,09
14EB:0104 27                DAA
14EB:0105 B099              MOV     AL,99
14EB:0107 0409              ADD     AL,09
14EB:0109 27                DAA
```

Начинаем с числа 11 в регистре AL.

```
AX=FF11  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102
                                    NV UP EI PL NZ AC PO CY

14EB:0102 0409            ADD       AL,09

AX=FF1A  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL NZ NA PO NC
                                          *        *
```

Заметьте, что после команды ADD остаток в младшем полубайте регистра AL не является двоично-десятичной цифрой.

```
14EB:0104 27              DAA

AX=FF20  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI PL NZ AC PO NC
                                          *
```

Команда DAA корректирует число, прибавляя к содержимому AL 6, если содержимое младшего полубайта AL больше 9 или если установлен флаг коррекции. Если 6 было прибавлено, устанавливается флаг AC, в противном случае он сбрасывается. Если содержимое старшего полубайта AL больше 9 или если установлен флаг переноса, 6 прибавляется к содержимому старшего полубайта и устанавливается флаг переноса. В противном случае флаг переноса сбрасывается. Заметьте, что в этом примере старшая цифра не требует коррекции, так что флаг переноса сбрасывается.

```
14EB:0105 B099            MOV       AL,99

AX=FF99  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL NZ AC PO NC
```

Для демонстрации выполнения команды в том случае, когда обе цифры требуют коррекции, загружаем в AL 99.

```
14EB:0107 0409            ADD       AL,09

AX=FFA2  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
```

```
SI=020F   DI=028F   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0109
                                        NV UP EI NG NZ AC PO NC
                                           *
```

Добавляем 9 к величине в AL. Устанавливаются флаги знака и коррекции.

```
14EB:0109 27                DAA

AX=FF08   BX=0000   CX=0000   DX=FFFF   SP=EA04   BP=0000
SI=020F   DI=028F   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010A
                                        NV UP EI PL NZ AC PO CY
                                              *              *
```

Обе величины скорректированы; возник двоично-десятичный перенос, что и отражено в состоянии флага переноса.

DAS Десятичная коррекция в регистре AL после вычитания (Decimal Adjust AL Register After Subtraction)

Команда	Код операции	Двоичное представление
DAS	2F	00101111

Назначение. Команда DAS корректирует результат вычитания двух правильных упакованных десятичных операндов в регистре AL. Команда должна всегда следовать за вычитанием одной пары упакованных десятичных чисел (по одной цифре в каждом полубайте) из другой, чтобы получить в результате пару правильных упакованных десятичных цифр. Если требовался заем, устанавливается флаг CF.

Пример выполнения команды. Команда DAS используется для коррекции упакованного двоично-десятичного результата вычитания. Команда DAS действует аналогично команде DAA, за тем исключением, что коррекция выполняется путем вычитания 6 из полубайтов регистра AL (вместо прибавления 6).

Последовательность команд:

```
14EB 0100 B011          MOV   AL,11
14EB:0102 2C19          SUB   AL,19
14EB:0104 2F            DAS
14EB:0105 B099          MOV   AL,99
14EB:0107 2C09          SUB   AL,09
14EB:0109 2F            DAS
```

После загрузки 11 в регистр AL:

```
AX=FF11   BX=0000   CX=0000   DX=FFFF   SP=EA04   BP=0000
SI=020F   DI=028F   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0102
                                        NV UP EI NG NZ NA PE NC
```

```
14EB:0102 2C19              SUB AL,19

AX=FFF8  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI NG NZ AC PO CY
                                       *  *  *
```

Вычитание устанавливает флаги NG, NZ, PO и CY, а также AC. Флаг AC имеет в этом случае большое значение, поскольку он включает коррекцию числа 8 в младшем полубайте.

```
14EB:0104 2F                DAS

AX=FF92  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI NG NZ AC PO CY
```

Команда DAS вычитает 6 из младшего полубайта, потому что флаг AC установлен. Проверяется старший полубайт, и из него вычитается 6, потому что в нем больше 9. Поскольку старший полубайт корректировался, флаг переноса установлен. Если бы в старшем полубайте была правильная двоично-десятичная цифра, коррекция была бы включена флагом переноса.

```
14EB:0105 B099              MOV  AL,99

AX=FF99  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI NG NZ AC PO CY

14EB:0107 2C09              SUB  AL,09

AX=FF90  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0109
                                    NV UP EI NG NZ NA PE NC
                                       *  *  *

14EB:0109 2F                DAS

AX=FF90  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                    NV UP EI NG NZ NA PE NC
```

В этом случае коррекция не требуется и не выполняется.

DEC Декремент на 1 (Decrement by 1)

Команда	Код операции	Двоичное представление
DEC r/m8	FE /1	11111110 mm 001 r/m
DEC r/m16	FF /1	11111111 mm 001 r/m
DEC r/m32	FF /1	11111111 mm 001 r/m
DEC r16	48 + rw	01001rrr
DEC r32	48 + rw	01001rrr

Назначение. Команда DEC вычитает 1 из операнда-приемника. Состояние флага CF не изменяется.

Пример выполнения команды. Показано выполнение команды DEC и его влияние на значение операнда и состояние флагов.

Последовательность команд:

```
14EB:0100 B80200          MOV   AX,0002
14EB:0103 48              DEC   AX
14EB:0104 48              DEC   AX
14EB:0105 48              DEC   AX
```

Регистры и флаги после занесения числа 2 в регистр AX:

```
AX=0002  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI NG NZ AC PE NC

14EB:0103 48              DEC   AX

AX=0001  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL NZ NA PO NC
                                       *     *  *
```

Обратите внимание на вычитание 1 из операнда-приемника. Флаги знака, нуля, коррекции и четности устанавливаются в соответствии с результатом.

```
14EB:0104 48              DEC   AX

AX=0000  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI PL ZR NA PE NC
                                          *     *
```

Обратите внимание на вычитание 1 из операнда-приемника. Флаги знака, нуля, коррекции и четности устанавливаются в соответствии с результатом.

```
14EB:0105 48                    DEC     AX

AX=FFFF  BX=0000  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=020F  DI=028F  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106

                                  NV UP EI NG NZ AC PE NC
                                     *  *  *
```

Обратите внимание на вычитание 1 из операнда-приемника. Флаги знака, нуля, коррекции и четности устанавливаются в соответствии с результатом. Заметьте, что состояние флага переноса не изменяется.

DIV Деление целых чисел без знака (Unsigned Integer Divide)

Команда	Код операции	Двоичное представление
DIV AL,r/m8	F6 /6	11110110 mm 110 r/m
DIV AX,r/m16	F7 /6	11110111 mm 110 r/m
DIV EAX,r/m32	F7 /6	11110111 mm 110 r/m

Назначение. Команда DIV выполняет деление числа без знака в аккумуляторе на операнд-источник. Размер делимого (содержимого аккумулятора) в два раза больше размера делителя (операнда-источника). Частное и остаток имеют те же размеры, что и делитель.

Если делитель равен 0, или если частное не помещается в назначенный регистр, возбуждается прерывание 0.

Примечание. Нецелые частные округляются до целых. Остаток всегда меньше делителя. При беззнаковом байтовом делении частное не может быть больше 255. При беззнаковом однословном делении частное не может быть больше 65535. При беззнаковом двухсловном делении частное не может быть больше $2^{32} - 1$.

Пример выполнения команды. Показано действие команды беззнакового целочисленного деления.

Последовательность команд:

```
14EB:0100 B80502                MOV     AX,0205
14EB:0103 B333                  MOV     BL,33
14EB:0105 F6F3                  DIV     BL
14EB:0107 F7F3                  DIV     BX
```

Делимое помещается в регистр AX.

```
AX=0205  BX=0033  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103

                                  NV UP EI PL NZ NA PE NC
```

Затем помещается делитель в регистр BL.

```
14EB:0103 B333                  MOV     BL,33

AX=0205  BX=0033  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105

                                  NV UP EI PL NZ NA PE NC
```

```
14EB:0105 F6F3              DIV BL

AX=070A  BX=0033  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI NG NZ NA PE CY
```

Заметьте, что короткое деление оставляет частное в регистре AL (517:51 = 10, остаток 7), а остаток в AH. Состояние флагов может изменяться (NG и CY в данном примере), но их значение не определено.

```
14EB:0107 F7F3              DIV BX

AX=0023  BX=0033  CX=0000  DX=0011  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0109
                                    NV UP EI PL NZ NA PE NC
```

Заметьте, что длинное деление оставляет частное в регистре AX, а остаток в DX. Флаги не определены.

ENTER Создание кадра стека для параметров процедуры (Make Stack Frame for Procedure Parameter)

Команда	Код операции	Двоичное представление
ENTER imm16,0	C8 iw 00	11001000
ENTER imm16,1	C8 /2 01	11001000
ENTER imm16,imm8	C8 iw ib	11001000

Назначение. Команда ENTER создает кадр стека, который можно использовать для реализации правил языков высокого уровня с блочной структурой. Команда LEAVE в конце процедуры выполняет обратные действия.

Команда ENTER имеет два параметра. Первый определяет число байтов динамической памяти, выделяемых в стеке для вызванной программы. Второй параметр соответствует лексическому уровню вложенности программы (от 0 до 31). Этот уровень определяет, сколько наборов указателей кадра стека копируются центральным процессором в новой кадр стека из текущего кадра.

Список кадров стека часто называют *таблицей контекстов*. Лексический уровень *не* имеет отношения ни к уровням защиты, ни к уровням привилегий ввода-вывода.

Команда ENTER создает для процедуры новую таблицу контекстов. Затем она выделяет для этой процедуры динамическую память, уменьшая содержимое ESP на число байтов, определенное первым параметром. Это новое значение ESP служит стартовой точкой для выполнения всех операций PUSH и POP в данной процедуре.

Команда ENTER может использоваться с разными уровнями вложенности. Если лексический уровень равен нулю, используется форма без вложенности. Главная процедура выполняется на высшем логическом уровне, уровне 1. Первая вызываемая ею процедура выполняется на следующем по глубине уровне, уровне 2 и т. д. Процедура уровня 2 имеет доступ к главной программе, потому что программа,

выполняемая на более высоком логическом уровне и вызывающая программу нижнего уровня, требует, чтобы вызванная процедура имела доступ к переменным вызывающей программы.

Если процедура вызывает другую процедуру на *том же* уровне, предполагается, что процедуры являются параллельными и *не* должны иметь доступа к переменным вызывающей программы. Новый кадр стека не содержит указателя для адресации кадра стека вызывающей процедуры. Команда ENTER рассматривает реентерабельную процедуру, как процедуру, вызывающую другую процедуру на том же уровне.

Пример выполнения команды. Команды CALL, ENTER, LEAVE и RET рассмотрены в одном примере, поскольку они логически «согласуются» друг с другом и выполняют взаимно обратные действия. См. команду CALL.

ESC Расширение процессора (Escape)

Команда	Код операции	Двоичное представление
ESC	D8 + TTT	11011TTT mod LLL r/m

Здесь TTT и LLL – информация о коде операции и адресах операндов для сопроцессора.

Назначение. Арифметический сопроцессор предоставляет расширение системы команд МП 80386. Сопроцессор поддерживает высокоточные вычисления как целочисленные, так и с плавающей точкой и, кроме того, содержит набор полезных констант, ускоряющих вычисления. Сопроцессор работает параллельно с центральным процессором, обеспечивая таким образом высокую производительность.

Команды сопроцессора включаются в общий поток команд, составляющих программу для МП 80386. Система выполняет команды сопроцессора в том порядке, в котором они появляются в потоке.

Команда ESC начинается с 5-битовой последовательности, которая идентифицирует арифметические команды вычислений с плавающей точкой. Это последовательность «11011B». Этот код является сигналом для МП 80386 посылки кода операции и адресов операндов арифметическому сопроцессору.

Пример выполнения команды. Не приведен, поскольку ESC – это сигнал, который декодируется командой сопроцессора. Действие этой команды, так же как и действие префиксов, нельзя продемонстрировать, если за ними не стоит какая-либо команда.

HLT Останов (Halt)

Команда	Код операции	Двоичное представление
HLT	F4	11110100

Назначение. Команда HLT прекращает выполнение любых команд и переводит МП 80386 в состояние останова. Немаскируемые прерывания, операция запуска и разрешенные прерывания возобновляют работу процессора. Команда HLT обычно является последней командой в последовательности команд останова системы, например для сохранения состояния процесса после обнаружения сбоя питания.

Пример выполнения команды. Не приведен, поскольку команда останавливает выполнение.

IDIV Знаковое деление (Signed Divide)

Команда	Код операции	Двоичное представление
IDIV r/m8	F6/7	11110110 mm 111 r/m
IDIV AX, r/m16	F7/7	11110111 mm 111 r/m
IDIV EAX, r/m32	F7/7	11110111 mm 111 r/m

Назначение. Команда IDIV выполняет знаковое деление. Делимому, частному и остатку неявно назначаются определенные регистры (рис. 10.2), в то время как местонахождение делителя (память или регистр) указывается явным образом. По форме делителя определяется, какие регистры должны использоваться.

Если делитель равен нулю, или частное слишком велико для регистра-приемника, возбуждается прерывание 0.

Размер	Делитель	Частное	Остаток	Делимое
Байт	r/m8	AL	AH	AX
Слово	r/m16	AX	DX	DX : AX
Двойное слово	r/m32	EAX	EDX	EDX : EAX

Рис. 10.2. Регистры, используемые командой IDIV.

Пример выполнения программы. Пример показывает выполнение целочисленного деления. В некоторых случаях в результате выполнения команды IDIV изменяется состояние различных флагов; в этих случаях значение флагов оказывается неопределенным, другими словами, их новое состояние не несет никакой информации. Такие изменения *не* отмечены в строках флагов звездочками (*).

Последовательность команд:

```
14EB:0100 B803F8        MOV    AX,F803
14EB:0103 B370          MOV    BL,70
14EB:0105 F6FB          IDIV   BL
14EB:0107 B803F8·       MOV    AX,F803
14EB:010A B390          MOV    BL,90
14EB:010C F6FB          IDIV   BL
14EB:010E B803F8        MOV    AX,F803
14EB:0111 F7FB          IDIV   BX
```

После загрузки делимого (−2045) в регистр AX:

```
AX=F803  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PE CY
```

Далее загружаем делитель (112) в BX (и BL):

```
14EB:0103  B370      MOV  BX,70

AX=F803  BX=0070  CX=0000  DX=0000  SP=EA04  BP=0000
```

```
SI=0000    DI=0000    DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0105
                                          NV UP EI PL NZ NA PE CY

14EB:0105 F6FB              IDIV    BL

AX=E3EE  BX=0070   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000  DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0107
                                          NV UP EI PL NZ NA PE CY
```

Операция деления дает знаковые (отрицательные) частное и остаток (−18 и −29) в регистрах AL и AH соответственно.

Перезагружаем делимое в AX.

```
14EB:0107 B803F8            MOV     AX,F803

AX=F803  BX=0070   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000  DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010A
                                          NV UP EI PL NZ NA PE CY
```

Изменяем теперь делитель в BL на число −112.

```
14EB:010A B390              MOV     BL,90

AX=F803  BX=0090   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000  DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010C
                                          NV UP EI PL NZ NA PE CY

14EB:010C F6FB              IDIV    BL

AX=E312  BX=0090   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000  DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010E
                                          NV UP EI PL NZ NA PE NC
```

Заметьте, что частное (18) положительно, в то время как остаток остается знаковым (отрицательным), каким было и делимое.

Снова перезагружаем делимое:

```
14EB:010E B803F8            MOV     AX,F803

AX=F803  BX=0090   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000  DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0111
                                          NV UP EI PL NZ NA PE NC
```

Делим пару слов в регистрах DX и AX на слово в регистре BX:

```
14EB:0111 F7FB              IDIV    BX

AX=01B8  BX=0090  CX=0000  DX=0083  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0113
                                    NV UP EI NG NZ NA PE CY
```

Делим 63491 на 144 и получаем частное 440 в AX и остаток 131 в DX. Заметьте, что если бы значение DX не учитывалось, величина в AX была бы отрицательной.

IMUL Знаковое целочисленное умножение (Signed Integer Multiply)

Команда	Код операции	Двоичное представление
IMUL r/m8	F6 /5	11110110 mm 101 r/m
IMUL r/m16	F7 /5	11110111 mm 101 r/m
IMUL r/m32	F7 /5	11110111 mm 101 r/m
IMUL r16,r/m16	0F AF /r	00001111 10101111 mm rrr r/m
IMUL r32,r/m32	0F AF /r	00001111 10101111 mm rrr r/m
IMUL r16,r/m16,imm8	6B /r ib	01101011 mm rrr r/m
IMUL r32,r/m32,imm8	6B /r ib	01101011 mm rrr r/m
IMUL r16,imm8	6B /r ib	01101011 mm rrr r/m
IMUL r32,imm8	6B /r ib	01101011 mm rrr r/m
IMUL r16,r/m16,imm16	69 /r iw	01101001 mm rrr r/m
IMUL r32,r/m32,imm32	69 /r id	01101001 mm rrr r/m
IMUL r16,imm16	69 /r iw	01101001 mm rrr r/m
IMUL r32,imm32	69 /r id	01101001 mm rrr r/m

Назначение. Команда IMUL выполняет операцию знакового умножения. Команда имеет 3 варианта.

1. Одноопérандная форма. Операнд может быть байтом, словом или двойным словом в памяти или регистре общего назначения. Команда IMUL, так же как и команда MUL, неявно использует содержимое регистров EAX и EDX в качестве операндов.

2. Двухоперандная форма. Один из операндов-источников может находиться в любом регистре общего назначения, в то время как другой может быть в регистре общего назначения или в памяти. Произведение размещается на месте операнда в регистре общего назначения.

3. Трехоперандная форма. Два операнда представляют собой источники и один приемник. Один из операндов-источников является непосредственным значением, записанным в команде. Второй может находиться в памяти или в любом регистре общего назначения. Произведение может быть записано в любой регистр общего назначения. Непосредственный операнд считается знаковым. Если он является байтом, процессор перед выполнением умножения автоматически расширяет его знак до размера второго операнда.

Пример выполнения команды. Пример показывает выполнение знакового умножения. В некоторых случаях в результате выполнения команды IMUL изменяется

состояние различных флагов; в этих случаях значение флагов оказывается неопределенным, другими словами, их новое состояние не несет никакой информации. Такие изменения *не* отмечены в строках флагов звездочками (*).

Последовательность команд:

```
14EB:0100 B8EEFF          MOV   AX,FFEE
14EB:0103 B370            MOV   BL,70
14EB:0105 F6EB            IMUL  BL
14EB:0107 B8EEFF          MOV   AX,FFEE
14EB:010A F7EB            IMUL  BX
```

Подготовим регистры к операции умножения:

```
14EB:0100 B8EEFF          MOV   AX,FFEE

AX=FFEE  BX=0070  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI NG NZ AC PO NC
```

Загрузили −18 в регистр AX (и AL).

```
14EB:0103 B370            MOV   BL,70

AX=FFEE  BX=0070  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI NG NZ AC PO NC
```

Положительный множитель (112) загружен в BL.

```
14EB:0105 F6EB            IMUL  BL

AX=F820  BX=0070  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    OV UP EI NG NZ AC PO CY
```

Отрицательное произведение (−2016) получено в регистре AX.
Восстановив содержимое AX

```
14EB:0107 B8EEFF          MOV   AX,FFEE

AX=FFEE  BX=0070  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                    OV UP EI NG NZ AC PO CY
```

и умножив его на однословную величину в BX,

```
14EB:010A F7EB              IMUL   BX

AX=F820  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010C
                                    NV UP EI NG NZ AC PO NC
```

получим произведение в паре регистров DX – AX (в данном случае путем расширения отрицательного бита на весь регистр DX).

IN Ввод из порта (Input from Port)

Команда	Код операции	Двоичное представление
IN AL,imm8	E4 ib	11100100
IN AX,imm8	E5 ib	11100101
IN EAX,imm8	E5 ib	11100101
IN AL,DX	EC	11101100
IN AX,DX	ED	11101101
IN EAX,DX	ED	11101101

Назначение. Команда IN вводит байт или слово из порта и записывает его в регистре (AL, AX или EAX). Порт указывается вторым операндом. Для доступа к порту его номер следует поместить в регистр DX и использовать команду IN с обозначением DX в качестве второго параметра.

Пример выполнения команды. В примере показано действие взаимосвязанных команд IN, INS, OUT и OUTS. Пример иллюстрирует использование группы команд ввода-вывода.

Последовательность команд:

```
14EB:0100 90          NOP
14EB:0101 E440        IN      AL,40
14EB:0103 90          NOP
14EB:0104 E640        OUT     40,AL
14EB:0106 90          NOP
14EB:0107 BA7803      MOV     DX,0378
14EB:010A EC          IN      AL,DX
14EB:010B 90          NOP
14EB:010C EE          OUT     DX,AL
14EB:010D 90          NOP
14EB:010E BF0002      MOV     DI,0200
14EB:0111 6C          INSB
14EB:0112 6D          INSW
14EB:0113 BE0002      MOV     SI,0200
14EB:0116 6E          OUTSB
14EB:0117 6F          OUTSW
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0000  CX=0008  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL NZ NA PO NC

14EB:0101 E440            IN    AL,40

AX=001A  BX=0000  CX=0008  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PO NC
```

Ввели байтовую величину из порта X'40', или 64, в регистр AL.

```
14EB:0103 90              NOP

AX=001A  BX=0000  CX=0008  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL NZ NA PO NC
14EB:0104 E640            OUT   40,AL

AX=001A  BX=0000  CX=0008  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL NZ NA PO NC
```

Вывели байт из регистра AL в порт 40.

```
14EB:0106 90              NOP

AX=001A  BX=0000  CX=0008  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL NZ NA PO NC

14EB:0107 BA7803          MOV   DX,0378

AX=001A  BX=0000  CX=0008  DX=0378  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                    NV UP EI PL NZ NA PO NC

14EB:010A EC              IN    AL,DX
```

```
AX=00AA  BX=0000  CX=0008  DX=0378  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010B
                                    NV UP EI PL NZ NA PO NC
```

Ввели байт из порта, указанного в регистре DX, в регистр AL. Заметьте, что номер адресуемого порта превышает FF (максимальный номер адресуемого порта при использовании 8-разрядного непосредственного значения).

```
14EB:010B 90              NOP

AX=00AA  BX=0000  CX=0008  DX=0378  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010C
                                    NV UP EI PL NZ NA PO NC

14EB:010C EE              OUT     DX,AL

AX=00AA  BX=0000  CX=0008  DX=0378  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010D
                                    NV UP EI PL NZ NA PO NC
```

Вывели байт в порт с адресом, находящимся в регистре DX.

```
14EB:010D 90              NOP

AX=00AA  BX=0000  CX=0008  DX=0378  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010E
                                    NV UP EI PL NZ NA PO NC

14EB:010E BF0002          MOV     DI,0200

AX=00AA  BX=0000  CX=0008  DX=0378  SP=EA04  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0111
                                    NV UP EI PL NZ NA PO NC

14EB:0111 6C              INSB

AX=00AA  BX=0000  CX=0008  DX=0378  SP=EA04  BP=0000
SI=0000  DI=0201  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0112
                                    NV UP EI PL NZ NA PO NC
```

Заметьте, что индекс DI увеличился на 1.

```
14EB:0112 6D                INSW

AX=00AA  BX=0000  CX=0008  DX=0378  SP=EA04  BP=0000
SI=0000  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0113
                                    NV UP EI PL NZ NA PO NC
```

Индекс в регистре DI снова увеличился, на этот раз на размер прочитанного слова.

```
14EB:0113 BE0002            MOV   SI,0200

AX=00AA  BX=0000  CX=0008  DX=0378  SP=EA04  BP=0000
SI=0200  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0116
                                    NV UP EI PL NZ NA PO NC

14EB:0116 6E                OUTSB

AX=00AA  BX=0000  CX=0008  DX=0378  SP=EA04  BP=0000
SI=0201  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0117
                                    NV UP EI PL NZ NA PO NC
```

Заметьте, что содержимое SI увеличилось на размер выведенной величины.

```
14EB:0117 6F                OUTSW

AX=00AA  BX=0000  CX=0008  DX=0378  SP=EA04  BP=0000
SI=0203  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0118
                                    NV UP EI PL NZ NA PO NC
```

Выходной указатель снова получил приращение, на этот раз на 2, поскольку выводилось слово.

Данные, прочитанные командами INS:

```
14EB:0200       AA AA 7F
```

INC Инкремент на 1 (Increment by 1)

Команда	Код операции	Двоичное представление
INC r/m8	FE /0	11111110 mm 000 r/m
INC r/m16	FF /0	11111111 mm 000 r/m
INC r/m32	FF /0	11111111 mm 000 r/m
INC r16	40 + rw	01000rrr
INC r32	40 + rd	01000rrr

Назначение. Команда INC прибавляет 1 к операнду-приемнику, но в отличие от ADD не влияет на флаг CF.

Пример выполнения команды. Показано действие команды INC. Последовательность команд:

```
14EB:0100 B8FE00          MOV   AX,00FE
14EB:0103 FEC0            INC   AL
14EB:0105 FEC0            INC   AL
14EB:0107 FEC0            INC   AL
```

Загружаем −2 в регистр AL:

```
AX=00FE  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI NG NZ AC PO NC

14EB:0103 FEC0            INC AL

AX=00FF  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI NG NZ NA PE NC
                                                 *  *  *
```

Выполнили инкремент в регистре AL, получив −1. Заметьте, что флаги, изменяемые арифметическими командами, изменяются также и командой INC.

```
14EB:0105 FEC0            INC AL

AX=0000  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL ZR AC PE NC
                                                *  *  *
```

Выполнив инкремент от −1 до 0, мы должны были бы сформировать перенос. В этом отношении – отсутствии влияния на флаг переноса – команда INC отличается от обычных арифметических команд.

```
14EB:0107 FEC0            INC AL

AX=0001  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0109
                                    NV UP EI PL NZ NA PO NC
                                                *     *
```

При выполнении инкремента от 0 до 1 результат отражается в состоянии флагов.

INS Ввод строки из порта (Input String from Port)
INSB Ввод байта (Input Byte)
INSW Ввод слова (Input Word)
INSD Ввод двойного слова (Input Doubleword)

Команда	Код операции	Двоичное представление
INS r/m8,DX	6C	01101100
INS r/m16,DX	6D	01101101
INS r/m32,DX	6D	01101101

Примечание: обозначение INSB является мнемоникой ассемблера для команды INS r/m8,DX; INSW – мнемоникой команды INS r/m16,DX; INSD – мнемоникой команды INS r/m32,DX.

Назначение. Команды ввода обеспечивают возможность чтения из устройства в память, при этом устройство ввода указывается в регистре DX. Данные читаются в сегмент, определяемый регистром ES, и замена сегмента невозможна. Команда INS *не* допускает описание номера порта непосредственным значением. Порт следует адресовать через регистр DX.

Команды ввода обычно используются с префиксом REP, чтобы обеспечить чтение заданного (в регистре CX) числа байтов.

Пример выполнения команды. Действие взаимосвязанных команд IN, INS, OUT и OUTS показано в одном примере. См. описание команды IN.

INT Вызов процедуры обработки прерывания (Call to Interrupt Procedure)
INTO Прерывание по переполнению (Interrupt on Overflow)

Команда	Код операции	Двоичное представление
INT 3	CC	11001100
INT imm8	CD ib	11001101
INTO	CE	11001110

Назначение. Команда INT передает управление от одной ячейки программного сегмента на другую. Эти ячейки могут принадлежать одному программному сегменту (*ближний* переход) или разным программным сегментам (*дальний* переход). Команда INT возбуждает программное прерывание, позволяющее пользователю передать управление из своей программы программе обработки прерываний.

Команда INTn активизирует программу обработки прерывания, соответствующую номеру, указанному в команде. Команда может определять прерывание любого типа. Заметьте, что прерывания 0...31 зарезервированы фирмой Intel. Возврат управления из программы обработки прерывания осуществляется командой IRET.

Команда INTO при установленном флаге OF возбуждает прерывание 4, которое специально зарезервировано для этой цели. Флаг OF устанавливается рядом арифметических, логических и строковых команд.

Пример выполнения команды. В примере показано действие команд INT и IRET, имеющих тесную взаимосвязь.

В приведенной ниже последовательности команд мы забираем прерывание 42 из системы MS/DOS

```
14EB:0100 BA0002        MOV    DX,0200
14EB:0103 B042          MOV    AL,42
```

```
14EB:0105 B425          MOV    AH,25
14EB:0107 CD21          INT 21
```

и используем его как наше прерывание.

```
14EB:0109 90            NOP
14EB:010A 90            NOP
14EB:010B CD42          INT 42
14EB:010D 90            NOP
```

Наша программа обработки прерывания:

```
14EB:0200 90            NOP
14EB:0201 CF            IRET
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0000  CX=0008  DX=0000  SP=EA04  BP=0000
SI=0203  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                    NV UP EI PL NZ NA PO NC

14EB:0100 BA0002        MOV    DX,0200

AX=0000  BX=0000  CX=0008  DX=0200  SP=EA04  BP=0000
SI=0203  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PO NC
```

Сообщили системе MS-DOS, где расположена наша программа обработки прерываний (DS:DX),

```
14EB:0103 B042          MOV    AL,42

AX=0042  BX=0000  CX=0008  DX=0200  SP=EA04  BP=0000
SI=0203  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI PL NZ NA PO NC
```

(номер прерывания)

```
14EB:0105 B425          MOV    AH,25

AX=2542  BX=0000  CX=0008  DX=0200  SP=EA04  BP=0000
SI=0203  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL NZ NA PO NC
```

и заказали операцию установки вектора.

```
14EB:0107 CD21                INT  21

AX=2542   BX=0000   CX=0008   DX=0200   SP=EA04   BP=0000
SI=0203   DI=0203   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0109
                                        NV UP EI PL NZ NA PO NC
```

Теперь вызываем систему MS-DOS командой CALL INT.

```
14EB:0109 90                  NOP

AX=2542   BX=0000   CX=0008   DX=0200   SP=EA04   BP=0000
SI=0203   DI=0203   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010A
                                        NV UP EI PL NZ NA PO NC
```

Результат:

```
INT Vector 40        41          42          43
0000:0100 59 EC 00 F0 20 00 3D 08-00 02 EB 14 FC 80 00 F0

14EB:010A 90                  NOP

AX=2542   BX=0000   CX=0008   DX=0200   SP=EA04   BP=0000
SI=0203   DI=0203   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010B
                                        NV UP EI PL NZ NA PO NC

14EB:010B CD42                INT  42

AX=2542   BX=0000   CX=0008   DX=0200   SP=E9FE   BP=0000
SI=0203   DI=0203   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0200
                                        NV UP DI PL NZ NA PO NC
```

Команда INT запоминает в стеке содержимое регистров флагов, CS и IP, для обеспечения возврата.

```
14EB:E9FE  0D 01 EB 14 02 72

14EB:0200 90                  NOP

AX=2542   BX=0000   CX=0008   DX=0200   SP=E9FE   BP=0000
SI=0203   DI=0203   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0201
                                        NV UP DI PL NZ NA PO NC
```

Обрабатываем прерывание и вызываем возврат.

```
14EB:0201 CF                IRET

AX=2542  BX=0000  CX=0008  DX=0200  SP=EA04  BP=0000
SI=0203  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010D
                                    NV UP EI PL NZ NA PO NC
```

Возврат. Обратите внимание на корректировку содержимого регистров IP и SP.

IRET Возврат из прерывания (Return from Interrupt)

IRETD Возврат из прерывания в 32-разрядном режиме (Return from Interrupt – 32-bit Mode)

Команда	Код операции	Двоичное представление
IRET	CF	11001111
IRETD	CF	11001111

Назначение. Команда IRET возвращает управление прерванной процедуре. В отличие от команды RET IRET извлекает из стека значения флагов и помещает их в регистр флагов. Флаги записываются в стек в процессе реализации прерывания.

Примечание. При выполнении команды IRET регистром флагов считается регистр FLAGS. При использовании IRETD – регистр EFLAGS.

Пример выполнения команды. Действие взаимосвязанных команд INT и IRET показано в одном примере. См. команду INT.

JMP Переход (Jump)

Jcc Переход по условию (Jump on some Condition Code)

Команда	Код операции	Двоичное представление
JA rel8	77 cb	01110111
JAE rel8	73 cb	01110011
JB rel8	72 cb	01110010
JBE rel8	76 cb	01110110
JC rel8	72 cb	01110010
JCXZ rel8	E3 cb	11100011
JECXZ rel8	E3 cb	11100011
JE rel8	74 cb	01110100
JZ rel8	74 cb	01110100
JG rel8	7F cb	01111111
JGE rel8	7D cb	01111101
JL rel8	7C cb	01111100
JLE rel8	7E cb	01111110
JNA rel8	76 cb	01110110
JNAE rel8	72 cb	01110010
JNB rel8	73 cb	01110011
JNBE rel8	77 cb	01110111
JNC rel8	73 cb	01110011

JNE rel8	75 cb	01110101	
JNG rel8	7E cb	01111110	
JNGE rel8	7C cb	01111100	
JNL rel8	7D cb	01111101	
JNLE rel8	7F cb	01111111	
JNO rel8	71 cb	01110001	
JNP rel8	7B cb	01111011	
JNS rel8	79 cb	01111001	
JNZ rel8	75 cb	01110101	
JO rel8	70 cb	01110000	
JP rel8	7A cb	01111010	
JPE rel8	7A cb	01111010	
JPO rel8	7B cb	01111011	
JS rel8	78 cb	01111000	
JZ rel8	74 cb	01110100	
JA rel16/32	0F 87 cw/cd	00001111	10000111
JAE rel16/32	0F 83 cw/cd	00001111	10000011
JB rel16/32	0F 82 cw/cd	00001111	10000010
JBE rel16/32	0F 86 cw/cd	00001111	10000110
JC rel16/32	0F 82 cw/cd	00001111	10000010
JE rel16/32	0F 84 cw/cd	00001111	10000100
JZ rel16/32	0F 84 cw/cd	00001111	10000100
JG rel16/32	0F 8F cw/cd	00001111	10001111
JGE rel16/32	0F 8D cw/cd	00001111	10001101
JL rel16/32	0F 8C cw/cd	00001111	10001100
JLE rel16/32	0F 8E cw/cd	00001111	10001110
JNA rel16/32	0F 86 cw/cd	00001111	10000110
JNAE rel16/32	0F 82 cw/cd	00001111	10000010
JNB rel16/32	0F 83 cw/cd	00001111	10000011
JNBE rel16/32	0F 87 cw/cd	00001111	10000111
JNC rel16/32	0F 83 cw/cd	00001111	10000011
JNE rel16/32	0F 85 cw/cd	00001111	10000101
JNG rel16/32	0F 8E cw/cd	00001111	10001110
JNGE rel16/32	0F 8C cw/cd	00001111	10001100
JNL rel16/32	0F 8D cw/cd	00001111	10001101
JNLE rel16/32	0F 8F cw/cd	00001111	10001111
JNO rel16/32	0F 81 cw/cd	00001111	10000001
JNP rel16/32	0F 8B cw/cd	00001111	10001011
JNS rel16/32	0F 89 cw/cd	00001111	10001001
JNZ rel16/32	0F 85 cw/cd	00001111	10000101
JO rel16/32	0F 80 cw/cd	00001111	10000000
JP rel16/32	0F 8A cw/cd	00001111	10001010
JPE rel16/32	0F 8A cw/cd	00001111	10001010
JPO rel16/32	0F 8B cw/cd	00001111	10001011
JS rel16/32	0F 88 cw/cd	00001111	10001000
JZ rel16/32	0F 84 cw/cd	00001111	10000100

Назначение. Команда JMP передает управление из одной точки сегмента в другую. Точка перехода может быть в том же сегменте (*ближний* переход) или в другом сегменте (*дальний* переход). Команда JMP, безусловно, передает управление

в точку перехода и является, таким образом, однонаправленной. Команда *не* сохраняет адрес возврата в стеке.

Команда JMP реализуется по-разному в зависимости от того, указан ли прямой адрес перехода непосредственно в команде, или извлекается косвенным образом из регистра или памяти. В случае прямого перехода адрес приемника является частью команды. В случае косвенного перехода адрес назначения берется из регистра или переменной-указателя, причем абсолютный адрес определяется одним из следующих способов: (1) для выбора адреса приемника регистр модифицирует адрес указателя памяти; (2) в программе осуществляется переход на ячейку, определяемую регистром общего назначения (EAX, EDX, ECS, EBX, EBP, ESI или EDI). МП 80386 загружает это 32-разрядное значение в регистр EIP и возобновляет выполнение; (3) МП 80386 получает адрес приемника из ячейки памяти, адрес которой указан в команде.

Ниже приведены команды условных переходов, которые в зависимости от состояния флагов процессора передают или не передают управление:

JA/JNBE	Выше, не ниже и не равно
JAE/JNB	Выше или равно, не ниже
JB/JNAE	Ниже, не выше и не равно
JBE/JNA	Ниже или равно, не выше
JC	Перенос
JE/JZ	Равно, нуль
JNC	Отсутствие переноса
JNE/JNZ	Не равно, не нуль
JNP/JPO	Отсутствие четности, нечетность
JP/JPE	Четность

Ниже перечислены команды знаковых переходов:

JG/JNLE	Больше, не меньше и не равно
JGE/JNL	Больше или равно, не меньше
JL/JNGE	Меньше, не больше и не равно
JLE/JNG	Меньше или равно, не больше
JNO	Отсутствие переполнения
JNS	Отсутствие знака (положительно, включая нуль)
JO	Переполнение
JS	Знак (отрицательно)

Команда JCXZ (переход, если ECX содержит 0) осуществляет переход на метку, указанную в команде, если содержимое регистра ECX равно 0. Эта команда полезна в тех случаях, когда желательно образовать цикл, выполняемый нуль раз, если переменная цикла в регистре ECX получила значение 0. При использовании с командами повторного сканирования строки или повторения команда JCXZ определяет, закончились ли повторения благодаря тому, что содержимое регистра ECX равно 0, или в силу удовлетворения условий сканирования или сравнения.

Пример 1 выполнения команд. Здесь показано действие команды безусловного перехода JMP.

Последовательность команд:

```
14EB:0100 29C0      SUB   AX,AX
14EB:0102 750C      JNZ   0110
14EB:0104 EB0A      JMP   0110
```

После вычитания устанавливается флаг нуля:

```
AX=0000  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102
                                    NV UP EI PL ZR NA PE NC
                                       *

14EB:0102 750C          JNZ   0110

AX=0000  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL ZR NA PE NC
```

Команда JNZ проверяет условие NZ (не нуль), однако оно не выполняется. Программный счетчик продвигается на следующую по порядку команду:

```
14EB:0104 EB0A          JMP   0110

AX=0000  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0110
                                    NV UP EI PL ZR NA PE NC
```

Команда JMP изменяет последовательность выполнения программных строк безусловно. Ни флаги, ни другие условия не проверяются.

Пример 2 выполнения команд. Ниже показано действие команд условных переходов.

Последовательность команд:

```
14EB:0100 29C0          SUB   AX,AX
14EB:0102 750C          JNZ   0110
14EB:0104 740A          JZ    0110
14EB:0106 90            NOP
```

Обратите внимание на установку флага нуля в результате вычитания.

```
AX=0000  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102
                                    NV UP EI PL ZR NA PE NC
                                       *
```

По ходу программы встретился условный переход JNZ.

```
14EB:0102 750C          JNZ   0110
```

```
AX=0000  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL ZR NA PE NC
```

Заметьте, что указатель команд продвигается на следующую по порядку команду, так как проверяемое условие не удовлетворяется. Однако условие, проверяемое командой JZ, удовлетворяется. Обратите внимание, как это влияет на состояние указателя команд.

```
14EB:0104 740A          JZ   0110
```

```
AX=0000  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0110
                                    NV UP EI PL ZR NA PE NC
```

Последовательность выполнения команд изменилась.

Пример 3 выполнения команд. Команда JCXZ проверяет значение в регистре CX и осуществляет переход, если CX содержит 0. Ценность этой команды проявляется при использовании префикса REP и команды LOOP. В обоих случаях регистр CX служит счетчиком. Команда JCXZ предоставляет простую возможность передачи управления в начале некоторой последовательности команд.

Последовательность команд:

```
14EB:0100 E302          JCXZ  0104
14EB:0102 EBFC          JMP   0100
14EB:0104 B90100        MOV   CX,0001
14EB:0107 E3FB          JCXZ  0104
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                    NV UP EI PL ZR NA PE NC
```

Заметьте, что регистр CX содержит 0

```
14EB:0100 E302          JCXZ  0104
```

```
AX=0000  BX=0070  CX=0000  DX=FFFF  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL ZR NA PE NC
```

Команда JCXZ передает управление команде MOV, пропуская следующую команду (перехода на ячейку 100).

```
14EB:0104 B90100              MOV    CX,0001

AX=0000   BX=0070   CX=0001   DX=FFFF   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0107
                                        NV UP EI PL ZR NA PE NC
```

Теперь значение в регистре CX не равно 0.

```
14EB:0107 E3FB                JCXZ   0104

AX=0000   BX=0070   CX=0001   DX=FFFF   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0109
                                        NV UP EI PL ZR NA PE NC
```

В результате команда JCXZ не препятствует последовательному выполнению стоящих за ней команд.

LAHF Загрузка флагов в регистр AH (Load Flags into AH Register)

Команда	Код операции	Двоичное представление
LAHF	9F	10011111

Назначение. Хотя для изменения флагов CF и DF предусмотрены специальные команды, остальные флаги, используемые в прикладных программах, нельзя изменить непосредственно. Команды пересылки флагов (LAHF и SAHF) дают возможность программного изменения остальных битов флагов с помощью команд побитовых операций после пересылки флагов в стек или в регистр AH.

Команда LAHF копирует биты SF, ZF, AF, PF и CF соответственно в разряды 7, 6, 4, 2 и 0 регистра AH. Значение остальных битов (5, 3 и 1) не определено.

Пример выполнения команды. Команда LAHF копирует младший байт слова флагов в регистр AH.

Последовательность команд:

```
14EB:0100 29C0                SUB    AX,AX
14EB:0102 9F                  LAHF
```

Операция вычитания влияет на установку флагов знака, нуля и четности:

```
AX=0000   BX=0070   CX=0001   DX=FFFF   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0102
                                        NV UP EI PL ZR NA PE NC

14EB:0102 9F                  LAHF

AX=4600   BX=0070   CX=0001   DX=FFFF   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0103
                                        NV UP EI PL ZR NA PE NC
```

Тип	Наименование
1	Доступный TSS МП 80286
2	LDT
3	Занятый вентиль вызова МП 80286
4	Вентиль вызова МП 80286
5	Вентиль задачи МП 80286/80386
6	Вентиль ловушки МП 80286
7	Вентиль прерываний МП 80286
9	Доступный TSS МП 80386
B	Занятый TSS МП 80386
C	Вентиль вызова МП 80386
E	Вентиль ловушки МП 80386
F	Вентиль прерываний МП 80386

Рис. 10.3. Типы дескрипторов специальных сегментов и вентилей (типы 0, 8, A и D недопустимы).

Команда LAHF копирует флаги в регистр AH. Заметьте, что 46 в двоичной форме составляет 01000110 и соответственно установлены флаги: PL, ZR, xx, NA, xx, PE, xx, NC.

LAR Загрузка байта прав доступа (Load Access Rights Byte)

Команда	Код операции	Двоичное представление
LAR r16,r/m16	0F 02/r	00001111 00000010 mm rrr r/m
LAR r32,r/m32	0F 02/r	00001111 00000010 mm rrr r/m

Назначение. Команда LAR читает дескриптор сегмента и заносит биты дробности (23), свободный (20), присутствия (15), DPL (14), типа (9–11) и доступа (8) в 32-разрядный регистр, бит дробности и свободный бит не пересылаются.

Примечание. Допустимы только следующие типы дескрипторов специальных сегментов и вентилей: 1, 2, 3, 4, 5, 6, 7, 9, B, C, E и F.

Пример выполнения команды. Команда LAR загружает замаскированный образ второго двойного слова дескриптора, выбранного вторым операндом, в регистр, указанный первым операндом.

Последовательность команд:

```
0070:0100 6629C0        SUB  EAX,EAX
0070:0103 B84000        MOV  EAX,0040
0070:0106 6689C3        MOV  EBX,EAX
0070:0109 660F02C3      LAR  EAX,EBX
0070:010D 90            NOP
```

Поскольку 32-разрядная форма регистров нарушает принятый нами формат изображения содержимого регистров, ниже указаны только регистры, используемые командами приведенной программы.

```
EAX=00000000        EBX=00000000
IP=0070:0100
                                    NV UP EI PL NZ NA PE NC
```

```
0070:0100 6629C0              SUB    EAX,EAX

EAX=00000000      EBX=00000000
IP=0070:0103
                                        NV UP EI PL ZR NA PE NC
```

Помещаем селектор в регистры EAX

```
0070:0103 B84000              MOV    AX,0040

EAX=00000040      EBX=00000000
IP=0070:0106
                                        NV UP EI PL ZR NA PE NC

u EBX.

0070:0106 6689C3              MOV    EBX,EAX

EAX=00000040      EBX=00000040
IP=0070:0109
                                        NV UP EI PL ZR NA PE NC
```

Проверяем права доступа выбранного дескриптора.

Содержимое дескриптора: 0040 67 00 98 E8 00 8B 00 00 (наш активный сегмент состояния задачи)

```
0070:0109 660F02C3      LAR   EAX, EBX
```

Маской является число 00F × FF00.

```
EAX=00008B00      EBX=00000040
IP=0070:010D
                                    NV UP EI PL ZR NA PE NC
```

Флаг нуля установлен. Если дескриптор недоступен по причине нарушения привилегий или выхода за пределы GDT или LDT, байты доступа не загружаются, а флаг нуля сбрасывается.

```
0070:010D 90      NOP
```

LEA Загрузка исполнительного адреса (Load Effective Address)

Команда	Код операции	Двоичное представление
LEA r16,m16	8D/r	10001101 mm rrr r/m
LEA r32,m32	8D/r	10001101 mm rrr r/m

Назначение. Команда LEA пересылает по адресу приемника относительный адрес операнда-источника (не значение операнда!). Операнд-источник должен находиться в памяти. Операнд-приемник должен быть регистром общего назначения.

Команда LEA особенно полезна для инициализации регистров перед выполнением действий над начальными данными или команды XLAT.

Пример выполнения команды. Команда LEA загружает часть адреса, представляющую собой относительный адрес, в указанный регистр.

Команда:

```
14EB:0100  8D9C0402      LEA   BX,[SI + 0204]
```

Регистры и флаги перед выполнением:

```
AX=4600   BX=0070   CX=0001   DX=FFFF   SP=EA04   BP=0000
SI=0100   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0100
                                        NV UP EI PL ZR NA PE NC

14EB:0100 8D9C0402          LEA BX,[SI+0204]

AX=4600   BX=0304   CX=0001   DX=FFFF   SP=EA04   BP=0000
SI=0100   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0104
                                        NV UP EI PL ZR NA PE NC
```

Команда LEA загружает составной относительный адрес в регистр-приемник.

LEAVE Выход из процедуры высокого уровня (High Level Procedure Exit)

Команда	Код операции	Двоичное представление
LEAVE	C9	11001001

Назначение. Команда LEAVE выполняет действия, противоположные действиям последней команды ENTER. Эта команда не имеет операндов. Она копирует содержимое регистра EBP в регистр ESP, чтобы освободить все место, выделенное процедуре под стек самой последней командой ENTER. Затем LEAVE извлекает из стека старое значение EBP. Последующая команда RET может теперь извлечь из стека любые аргументы, записанные туда вызывающей программой для передачи в вызываемую процедуру.

Пример выполнения команды. Действие команд CALL, ENTER, LEAVE и RET показано в одном примере, поскольку эти команды «согласуются» друг с другом и выполняют взаимно обратные действия. См. команду CALL.

LGDT Загрузка регистра таблицы глобальных дескрипторов (Load Global Descriptor Table Register)

LIDT Загрузка регистра таблицы дескрипторов прерываний (Load Interrupt Descriptor Table Register)

Команда	Код операции	Двоичное представление
LGDT m16&32	0F 01/2	00001111 00000001 mm 010 r/m
LIDT m16&32	0F 01/3	00001111 00000001 mm 011 r/m

Назначение. Команда LIDT оповещает аппаратные средства о точке перехода в случае прерываний. Обе таблицы – GDT и LDT – загружаются при запуске системы

(инициализации операционной системы). Обычно это происходит в начале сеанса работы.

Пример выполнения команды. См. приложение 5.

LGS Загрузка полного указателя (Load Full Pointer)
LSS Загрузка указателя с использованием регистра SS (Load Pointer Using SS)
LDS Загрузка указателя с использованием регистра DS (Load Pointer Using DS)
LES Загрузка указателя с использованием регистра ES (Load Pointer Using ES)
LFS Загрузка указателя с использованием регистра FS (Load Pointer Using FS)

Команда	Код операции	Двоичное представление
LDS r16,m16:16	C5 /r	11000101 mm rrr r/m
LDS r32,m16:32	C5 /r	11000101 mm rrr r/m
LSS r16,m16:16	0F B2 /r	00001111 10110010 mm rrr r/m
LSS r32,m16:32	0F B2 /r	00001111 10110010 mm rrr r/m
LES r16,m16:16	C4 /r	11000100 mm rrr r/m
LES r32,m16:32	C4 /r	11000100 mm rrr r/m
LFS r16,m16:16	0F B4 /r	00001111 10110100 mm rrr r/m
LFS r32,m16:32	0F B4 /r	00001111 10110100 mm rrr r/m
LGS r16,m16:16	0F B5 /r	00001111 10110101 mm rrr r/m
LGS r32,m16:32	0F B5 /r	00001111 10110101 mm rrr r/m

Назначение. Команды указателей данных загружают указатель, состоящий из селектора сегмента и относительного адреса, в регистр сегмента и регистр общего назначения.

Команда LDS пересылает переменную-указатель, являющуюся операндом-источником, в регистр сегмента данных DS и регистр-приемник. Операнд-источник должен быть в памяти. Операнд-приемник должен быть регистром общего назначения. Регистр DS получает из указателя селектор сегмента. Регистр-приемник получает из указателя относительный адрес, указывающий на определенную ячейку в сегменте. Другие команды используют различные регистры, как это следует из их мнемоники.

Особенно большое значение имеет команда LSS, которая позволяет за одну непрерываемую операцию изменить содержимое обоих регистров, идентифицирующих стек (SS:ESP).

Пример выполнения команды. Команды группы LGS/LSS/LDS/LES/LFS загружают полный указатель в соответствующий команде регистр сегмента и указанный регистр-указатель. Загружаемое значение представляет собой 16- или 32-разрядный указатель и 16-разрядный селектор. Команды LGS/LDS/LES/LFS удобно использовать для загрузки полного указателя. Команда LSS обеспечивает единственно абсолютно надежный способ загрузки пары регистров указатель стека/дескриптор сегмента стека. Хотя МП 8086–80286 автоматически запрещают прерывания на время выполнения одной команды, следующей за MOV SS (то же справедливо и по отношению к МП 80386), использование двух команд МП 80386 может привести к тому, что они будут разделены отказом страницы. Команда LSS поддерживает целостность стека. Коды условий не изменяются.

```
14EB:0100 90C506          NOP
14EB:0101 C5065000        LDS     AX,[50]
14EB:0105 0FB2265000      LSS     SP,[54]
14EB:010A 90              NOP
```

Загружаемые данные DW 100H, 1500H, 200H, 1580H

```
14EB:0050 00 01 00 15 00 02 80 15
```

Регистры перед выполнением:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL ZR NA PE NC
```

Обратите внимание на действие команды LDS. Оба регистра AX и DS изменили содержимое в процессе выполнения одной команды.

```
14EB:0101 C5065000        LDS     AX,[0050]

AX=0100  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=1500  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI PL ZR NA PE NC
```

Обратите внимание на действие команды LSS – не только удобство, но и целостность. Оба регистра SP и SS изменили свое содержимое в процессе выполнения одной команды.

```
14EB:0105 0FB2265000      LSS     SP,[54]

AX=0100  BX=0000  CX=0000  DX=0000  SP=0000  BP=0000
SI=0000  DI=0000  DS=1500  ES=14EB  SS=0000  CS=14EB
IP=010A
                                    NV UP EI PL ZR NA PE NC

14EB:010A 90              NOP
```

LLDT Загрузка регистра таблицы локальных дескрипторов (Load Local Descriptor Table Redister)

Команда	Код операции	Двоичное представление
LLDT r/m16	0F 00/2	00001111 00000000 mm 010 r/m

Назначение. Таблица локальных дескрипторов загружается, когда задача или главная подсистема получает или восстанавливает контроль над системой. Команда LLDT загружает регистр таблицы локальных дескрипторов (LDTR). Операнд (находящийся в памяти или в регистре) представляет собой селектор к LDT. Команда не влияет на регистры дескрипторов, и поле LDT в сегменте состояния задачи не изменяется.

Пример выполнения команды. Команда пересылает содержимое 16-разрядного регистра или ячейки памяти в регистр таблицы локальных дескрипторов. Состояние флагов не изменяется. Атрибут размера операнда не влияет на действие команды LLDT.

```
0070:0100 OF00D0        LLDT   AX
0070:0103 90            NOP
```

Регистры и флаги перед выполнением:

```
AX=0010   BX=0000   CX=0464   DX=0000   SP=0200   BP=0000
SI=0000   DI=0000   DS=0078   ES=0000   SS=0080   CS=0070
IP=0100
                                        NV UP EI PL NZ NA PO NC
```

Пересылаем селектор LDT из AX в LDTR

```
0070:0100 0F00D0            LLDT   AX

AX=0010   BX=0000   CX=0464   DX=0000   SP=0200   BP=0000
SI=0000   DI=0000   DS=0078   ES=0000   SS=0080   CS=0070
IP=0103
                                        NV UP EI PL NZ NA PO NC

0070:0103 90                NOP
```

LWSW Загрузка слова состояния машины (Load Machine Status Word)

Команда	Код операции	Двоичное представление
LMSW r/m16	0F 01/6	00001111 00000001 mm 110 r/m

Назначение. Команда LMSW загружает в регистр CR0 слово состояния машины (операнд команды LMSW). Команду можно использовать для переключения в защищенный режим. В этом случае следует очистить очередь команд. Команда LMSW не осуществляет переключение назад в режим реальных адресов.

Пример выполнения команды. Показано, как с помощью команды LMSW младшая часть слова состояния машины загружается в регистр CR0 из указанного в команде регистра-источника. Регистр AX заполняется с помощью команды SMSW.

Последовательность команд:

```
14EB:0101 0F01F0            LMSW AX
14EB:0104 90                NOP
```

Регистры и флаги перед выполнением:

```
AX=FFE0   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0101
                                        NV UP EI PL NZ NA PO NC
```

```
14EB:0101 0F01F0          LMSW   AX

AX=FFE0  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL NZ NA PO NC
```

Младшая часть слова состояния машины загружается в CR0 из указанного регистра. Заметьте, что регистр AX был заполнен с помощью команды SMSW.

LOCK Префикс установки сигнала LOCK # (Assert LOCK # Signal Prefics)

Команда	Код операции	Двоичное представление
LOCK	F0	11110000

Назначение. Сигнал LOCK захватывает общую память, так что МП 80386 получает ее в свое исключительное пользование на время выполнения следующей за LOCK команды. Целостность операции LOCK не нарушается выравниванием полей памяти. Префикс LOCK можно использовать только со следующими командами и способами адресации:

BT, BTS, BTR, BTC память, регистр/непосредственная
ADD, OR, ADC, SBB, AND, SUB, XOR память, регистр/непосредственная
NOT, NEG, INC, DEC память
XCHG регистр–память или память–регистр

Если префикс LOCK используется с какой-либо не указанной выше командой, возникает ловушка неопределенного кода операции. Заметьте, что команда XCHG всегда устанавливает сигнал LOCK# независимо от наличия или отсутствия префикса LOCK.

Примечание. Выполнение операции LOCK не гарантируется, если другой МП 80386 одновременно выполняет команду, отличающуюся какой-либо из перечисленных ниже особенностей:

1. Команда не входит в приведенный выше список.
2. Команда не предваряется префиксом LOCK.
3. В команде определен операнд, который не полностью перекрывает в памяти операнд-приемник. Префикс LOCK не гарантирует блокирования в случае частичного перекрытия, даже если один участок памяти целиком содержится в другом.

Пример выполнения команды. Показать действие команды невозможно.

LODS Загрузка строкового операнда (Load String Operand)
LODSB Загрузка байта (Load Byte)
LODSW Загрузка слова (Load Word)
LODSD Загрузка двойного слова (Load Doubleword)

Команда	Код операции	Двоичное представление
LODS m8	AC	10101100
LODS m16	AD	10101101
LODS m32	AD	10101101

Назначение. Эти команды обрабатывают не логические или арифметические переменные, а строки. Они воздействуют на один элемент строки, который может

быть байтом, словом или двойным словом. Элементы строки адресуются через регистры ESI и EDI. После каждой строковой операции регистры ESI и/или EDI автоматически получают положительное или отрицательное приращение и указывают на следующий элемент строки. Если DF = 0, в индексных регистрах осуществляется инкремент, если DF = 1 – декремент. Величина приращения составляет 1, 2 или 4 в зависимости от размера элемента строки.

Команда LODS помещает элемент строки-источника, на который указывает ESI, в регистр AL для байтовых строк, в AX – для строк, состоящих из слов, и в EAX – для строк из двойных слов. Увеличение или уменьшение содержимого ESI определяется значением DF.

Пример выполнения команды. Ниже показано действие команды LODS в цикле. Такое применение типично для этой команды.

Последовательность команд:

```
14EB:0100 B90500          MOV     CX,0005
14EB:0103 BE0002          MOV     SI,0200
14EB:0106 E307            JCXZ    010F
14EB:0108 AC              LODSB
14EB:0109 3C00            CMP     AL,00
14EB:010B E0FB            LOOPNZ  0108
```

Регистры и флаги после загрузки регистра CX:

```
AX=FFA4  BX=0204  CX=0005  DX=FFB3  SP=EA04  BP=0000
SI=0205  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PE CY
```

```
14EB:0103 BE0002          MOV     SI,0200
```

```
AX=FFA4  BX=0204  CX=0005  DX=FFB3  SP=EA04  BP=0000
SI=0200  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL NZ NA PE CY
```

```
14EB:0106 E307            JCXZ    010F
```

```
AX=FFA4  BX=0204  CX=0005  DX=FFB3  SP=EA04  BP=0000
SI=0200  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    NV UP EI PL NZ NA PE CY
```

Поскольку в CX не 0, переход не выполняется.

```
14EB:0108 AC              LODSB
```

```
AX=FF41  BX=0204  CX=0005  DX=FFB3  SP=EA04  BP=0000
SI=0201  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0109
                                    NV UP EI PL NZ NA PE CY
```

Загружается байт, адрес которого берется из SI, и содержимое SI получает приращение.

```
14EB:0109 3C00                CMP     AL,00

AX=FF41  BX=0204  CX=0005  DX=FFB3  SP=EA04  BP=0000
SI=0201  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010B
                                    NV UP EI PL NZ NA PE NC
```

Сравниваем байт с нулевым значением, часто используемым в качестве индикатора конца строки.

```
14EB:010B E0FB                LOOPNZ  0108

AX=FF00  BX=0204  CX=0001  DX=FFB3  SP=EA04  BP=0000
SI=0204  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010D
                                    NV UP EI PL ZR NA PE NC
                                                *
```

Команда LOOPNZ выполняется трижды с условием NZ (не нуль). Каждый раз она вызывает уменьшение адреса в регистре CX и на четвертом байте не срабатывает из-за кода условия.

```
14EB:010D EBF7                JMP     0106

AX=FF00  BX=0204  CX=0001  DX=FFB3  SP=EA04  BP=0000
SI=0204  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL ZR NA PE NC
```

В данном примере осуществляется переход на извлечение новой строки.

```
14EB:0106 E307                JCXZ    010F

AX=FF43  BX=0204  CX=0000  DX=FFB3  SP=EA04  BP=0000
SI=0205  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010F
                                    NV UP EI PL NZ NA PO NC
```

В CX еще не 0, поэтому мы снова входим в цикл.

```
14EB:0108 AC                  LODSB

AX=FF43  BX=0204  CX=0001  DX=FFB3  SP=EA04  BP=0000
SI=0205  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0109
                                    NV UP EI PL ZR NA PE NC
```

Загружаем ненулевой байт.

```
14EB:0109 3C00                CMP     AL,00

AX=FF43  BX=0204  CX=0001  DX=FFB3  SP=EA04  BP=0000
```

```
SI=0205   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010B
                                          NV UP EI PL NZ NA PO NC
                                           *
```

Выполняем сравнение, чтобы установить код условия NZ.

```
14EB:010B E0FB              LOOPNZ    0108

AX=FF43   BX=0204   CX=0000   DX=FFB3   SP=EA04   BP=0000
SI=0205   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010D
                                          NV UP EI PL NZ NA PO NC
```

Хотя условие NZ еще не выполняется, команда LOOPNZ не срабатывает, так как счетчик CX досчитал до конца. Обычно здесь дополнительно проверяется условие NZ, чтобы определить причину выхода из цикла.

```
14EB:010D EBF7              JMP       0106

AX=FF43   BX=0204   CX=0000   DX=FFB3   SP=EA04   BP=0000
SI=0205   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0106
                                          NV UP EI PL NZ NA PO NC
```

Осуществляем переход на сторожевую команду перед входом в цикл.

```
14EB:0106 E307              JCXZ      010F

AX=FF00   BX=0204   CX=0001   DX=FFB3   SP=EA04   BP=0000
SI=0204   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0108
                                          NV UP EI PL ZR NA PE NC
```

Команда обнаруживает, что в регистре CX нуль и осуществляет обход цикла.

Проверяемые данные.

```
14EB:0200   41 42 43 00 43 42 41 00 ABC·CBA
```

LOOP Циклическое выполнение, пока счетчик ECX не нуль (Loop Control while ECX Counter Not Zero)

LOOP условие

LOOPE Цикл, пока равно (Loop while Equal)

LOOPZ Цикл, пока нуль (Loop while Zero)

LOOPNE Цикл, пока не равно (Loop while Not Equal)

LOOPNZ Цикл, пока не нуль (Loop while Not Zero)

Команда	Код операции	Двоичное представление
LOOP rel8	E2 cb	11100010
LOOPE rel8	E1 cb	11100001
LOOPNE rel8	E1 cb	11100001

Примечание: LOOPZ является альтернативной мнемоникой для LOOPE rel8; LOOPNZ – для LOOPNE rel8.

Назначение. Команды LOOP обеспечивают условный переход для циклического выполнения участка программы; число шагов цикла определяется значением, занесенным в регистр ECX. Все разновидности команды LOOP автоматически выполняют декремент содержимого ECX и останавливают цикл, если ECX = 0.

Перед тем как проверить регистр ECX на условие перехода, команда LOOP выполняет в нем декремент. Если в ECX не 0, выполняется переход на метку, указанную в команде. Если ECX = 0, управление передается на команду, следующую непосредственно за командой LOOP. Если с самого начала в ECX нуль, команда LOOP выполняется 2^{32} раз.

Обозначения LOOPE и LOOPZ представляют синонимы и относятся, в сущности, к одной команде. Эта команда уменьшает содержимое ECX, после чего проверяет ECX и ZF на выполнение условий перехода. Если в ECX не нуль и ZF = 1, осуществляется переход на метку, указанную в команде. Если при выполнении команд LOOPE или LOOPZ обнаруживается, что ECX = 0 или ZF = 0, управление передается на команду, следующую непосредственно за LOOPE или LOOPZ.

Обозначения LOOPNE и LOOPNZ представляют синонимы и относятся, в сущности, к одной команде. Эта команда уменьшает содержимое ECX, после чего проверяет ECX и ZF на выполнение условий перехода. Если ECX не нуль и ZF = 0, осуществляется переход на метку, указанную в команде. Если ECX = 0 или ZF = 1, управление передается команде, следующей непосредственно за LOOPNE или LOOPNZ.

Пример выполнения команд. См. команду LODS.

LSL Загрузка границы сегмента (Load Segment Limit)

Команда	Код операции	Двоичное представление
LSL r16,r/m16	0F 03/r	00001111 00000011 mm rrr r/m
LSL r32,r/m32	0F 03/r	00001111 00000011 mm rrr r/m

Назначение. Команда LSL загружает значение границы сегмента в регистр, указанный пользователем. Если селектор-источник доступен с текущего уровня привилегий и тип дескриптора приемлем для LSL, эта команда устанавливает ZF в 1. В противном случае LSL устанавливает ZF в 0 и не изменяет содержимое регистра-приемника.

Команда LSL в 32-разрядной форме записывает 32-разрядное значение границы в единицах дробности в 16-разрядный регистр-приемник.

Заметьте, что это значение границы представляет собой величину в единицах дробности, выраженную в байтах. Если в дескрипторе используется граница сегмента, выраженная числом страниц, команда LSL переводит ее в число байтов и затем загружает в регистр-приемник.

Пример выполнения команды.

Последовательность команд:

```
0070:0100 6629C0        SUB     EAX,EAX
0070:0103 6689C3        MOV     EBX,EAX
0070:0106 BB7800        MOV     BX,0078H
0070:0109 660F03C3      LSL     EAX,EBX
0070:010D 90            NOP
```

Ниже опять показаны только те регистры, которые участвуют в выполнении команд, поскольку приходится приводить их 32-разрядные значения.

```
EAX=00000000  EBX=00000000
IP=0100
                                        NV UP EI PL NZ NA PO NC

0070:0100 6629C0            SUB     EAX,EAX

EAX=00000000  EBX=00000000
IP=0103
                                        NV UP EI PL ZR NA PO NC
                                                    *

0070:0103 6689C3            MOV     EBX,EAX

EAX=00000000  EBX=00000000
IP=0106
                                        NV UP EI PL ZR NA PO NC

0070:0106 BB7800            MOV     BX,0078H
```

Теперь в EBX находится селектор, а EAX очищен.

```
EAX=00000000  EBX=00000078
IP=0109
                                        NV UP EI PL ZR NA PO NC

0070:0109 660F03C3          LSL     EAX,EBX
```

Команда LSL загружает значение границы из выбранного дескриптора. Если указать в команде не 32-разрядный, а 16-разрядный регистр (например, AX вместо EAX), будут загружены младшие 16 бит границы.

```
EAX=000FFFFF  EBX=00000078
IP=010D
                                        NV UP EI PL ZR NA PO NC

0070:010D 90                NOP
```

Дескриптор 0078 = FF FF 00 AB 01 93 0F 00

LTR Загрузка регистра задачи (Load Task Register)

Команда	Код операции	Двоичное представление
LTR r/m16	0F 00/3	00001111 00000000 mm 011 r/m

Назначение. Первый операнд команды LTR определяет регистр-источник или ячейку памяти, содержащие информацию для регистра задачи. Команда LTR загружает эту информацию в регистр задачи. Загруженный TSS помечается занятым, однако переключения задачи *не* происходит.

Пример выполнения команды. См. листинг программы в приложении 5.

MOV Пересылка данных (Move Data)

Команда	Код операции	Двоичное представление
MOV r/m8,r8	88 /r	10001000 mm rrr r/m
MOV r/m16,r16	89 /r	10001001 mm rrr r/m
MOV r/m32,r32	89 /r	10001001 mm rrr r/m
MOV r8,r/m8	8A /r	10001010 mm rrr r/m
MOV r16,r/m16	8B /r	10001011 mm rrr r/m
MOV r32,r/m 32	8B /r	10001011 mm rrr r/m
MOV r/m16,sreg	8C /r	10001100 mm rrr r/m
MOV Sreg,r/m16	8D /r	10001101 mm rrr r/m
MOV AL,moffs8	A0	10100000
MOV AX,moffs16	A1	10100001
MOV EAX,moffs32	A1	10100001
MOV moffs8,AL	A2	10100010
MOV moffs16,AX	A3	10100011
MOV moffs32,EAX	A3	10100011
MOV reg8,imm8	B0 + rb	10110rrr
MOV reg16,imm16	B8 + rw	10111rrr
MOV reg32,imm32	B8 + rd	10111rrr
MOV r/m8,imm8	C6	11000110 mm rrr r/m
MOV r/m16,imm16	C7	11000111 mm rrr r/m
MOV r/m32,imm32	C7	11000111 mm rrr r/m

Назначение. Команда MOV пересылает операнд-источник (байт, слово или двойное слово) по адресу приемника. Команду MOV можно использовать для пересылок следующего вида:

– непосредственные данные в память;
– непосредственные данные в регистр;
– между регистрами общего назначения;
– из памяти в регистр;
– из регистра в память.

Для работы с регистрами сегментов используются специальные варианты команды MOV.

Примечание. Команда MOV не может пересылать данные из памяти в память или из одного регистра сегмента в другой. Для пересылок из памяти в память можно использовать команду пересылки строк MOVS.

Пример выполнения команды. См. команды пересылки в/из специальных регистров и MOVS.

MOV Пересылка в/из специальных регистров (Move To/From Special Registers)

Команда	Код операции	Двоичное представление
MOV r32,CR0/CR2/CR3	0F 20 /r	00001111 00100000 mm ccc r/m
MOV CR0/CR2/CR3,r32	0F 22 /r	00001111 00100010 mm ccc r/m
MOV r32,TR6/TR7	0F 24 /r	00001111 00100100 mm +++ r/m
MOV TR6/TR7,r32	0F 26 /r	00001111 00100110 mm +++ r/m
MOV r32,DR0-3/6/7	0F 21 /r	00001111 00100001 mm ddd r/m
MOV DR3/6/7,r32	0F 23 /r	00001111 00100011

Назначение. Эти формы команды MOV используются для загрузки и выгрузки специальных регистров и регистров общего назначения. Они предназначены для работы с регистрами управления (CR0, CR2, CR3), тестирования (TR6 и TR7) и отладки (DR0, DR1, DR2, DR3, DR6 и DR7)

Пример 1 выполнения команд. Показана пересылка данных из регистров управления сначала в регистр EAX, затем в память. Значения в регистрах управления соответствуют режиму реальных адресов.

Последовательность команд:

```
14EB:0100 0F20C0          MOV    EAX,CR0
14EB:0103 66A30002        MOV    [0200],EAX
14EB:0107 0F20D0          MOV    EAX,CR2
14EB:010A 66A30402        MOV    [0204],EAX
14EB:010E 0F20D8          MOV    EAX,CR3
14EB:0111 66A30802        MOV    [0208],EAX
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0204  CX=0000  DX=FE13  SP=EA04  BP=0000
SI=0206  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                    NV UP EI PL NZ NA PE CY
   Пересылаем
    CR0 в EAX
14EB:0100 0F20C0          MOV    EAX,CR0
AX=FFE0  BX=0204  CX=0000  DX=FE13  SP=EA04  BP=0000
SI=0206  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PE CY
     Затем в память
14EB:0103 66A30002        MOV    [0200],EAX
AX=FFEO  BX=0204  CX=0000  DX=FE13  SP=EA04  BP=0000
SI=0206  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL NZ NA PE CY
  Теперь CR2 в EAX
14EB:0107 0F20D0          MOV    EAX,CR2
AX=0000  BX=0204  CX=0000  DX=FE13  SP=EA04  BP=0000
SI=0206  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                    NV UP EI PL NZ NA PE CY
14EB:010A 66A30402        MOV    [0204],EAX
AX=0000  BX=0204  CX=0000  DX=FE13  SP=EA04  BP=0000
SI=0206  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010E
                                    NV UP EI PL NZ NA PE CY
```

```
и CR3
14EB:010E 0F20D8              MOV     EAX,CR3
AX=0000  BX=0204  CX=0000  DX=FE13  SP=EA04  BP=0000
SI=0206  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0111
                                          NV UP EI PL NZ NA PE CY
14EB:0111 66A30802            MOV     [0208],EAX
AX=0000  BX=0204  CX=0000  DX=FE13  SP=EA04  BP=0000
SI=0206  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0115
                                          NV UP EI PL NZ NA PE CY
14EB:0115 90                  NOP
```

Результат выполнения:

```
14EB:0200 E0  FF  FF  7F  00  00  00  00  00  00  00  00 .....
              CR0 CR2     CR3
```

Пример 2 выполнения команд. Показана пересылка данных из регистров тестирования в память через регистр EAX.

Последовательность команд:

```
14EB:00FF 90                  NOP
14EB:0100 0F24F0              MOV     EAX,TR6
14EB:0103 66A30002            MOV     [0200],EAX
14EB:0107 0F24F8              MOV     EAX,TR7
14EB:010A 66A30402            MOV     [0204],EAX
```

Регистры и флаги перед выполнением:

```
AX=F01C  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                          NV UP DI PL NZ NA PO NC
```

Пересылаем содержимое TR6 в EAX

```
14EB:0100 0F24F0              MOV     EAX,TR6
AX=FFFF  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                          NV UP DI PL NZ NA PO NC
```

а затем в память.

```
14EB:0103 66A30002            MOV     [0200],EAX
AX=FFFF  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
```

Пересылаем содержимое TR7 в EAX

```
14EB:0107 0F24F8              MOV     EAX,TR7
AX=F01C  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                        NV UP DI PL NZ NA PO NC
```

а затем в память.

```
14EB:010A 66A30402            MOV     [0204],EAX
AX=F01C  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010E
                                        NV UP DI PL NZ NA PO NC
```

Содержимое регистров TR6 и TR7

```
14EB:0200 FF  FF FF  FF 1C  F0 E7 02   ....страница
                TR6      TR7
```

Пример 3 выполнения команд. Показана пересылка содержимого регистров отладки в память через регистр EAX.

Последовательность команд:

```
14EB:00FF 90                  NOP
14EB:0100 0F21C0              MOV     EAX,DR0
14EB:0103 C066A30002          MOV     [0200],EAX
14EB:0107 0F21C8              MOV     EAX,DR1
14EB:010A 66A30402            MOV     [0204],EAX
14EB:010E 0F21D0              MOV     EAX,DR2
14EB:0111 66A30802            MOV     [0208],EAX
14EB:0115 0F21D8              MOV     EAX,DR3
14EB:0118 66A30C02            MOV     [020C],EAX
14EB:011C 0F21F0              MOV     EAX,DR6
14EB:011F 66A31002            MOV     [0210],EAX
14EB:0123 0F21F8              MOV     EAX,DR7
14EB:0126 66A31402            MOV     [0214],EAX
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                        NV UP EI PL NZ NA PO NC
```

Пересылаем DR0 в EAX

```
14EB:0100 0F21C0              MOV     EAX,DR0
```

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PO NC
```

и затем в память.

```
14EB:0103 C066A30002        MOV     [0200],EAX
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL NZ NA PO NC

    То же для  DR1,
14EB:0107 0F21C8            MOV     EAX,DR1
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                    NV UP EI PL NZ NA PO NC

14EB:010A 66A30402          MOV     [0204],EAX
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010E
                                    NV UP EI PL NZ NA PO NC

    DR2,
14EB:010E 0F21D0            MOV     EAX,DR2
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0111
                                    NV UP EI PL NZ NA PO NC

14EB:0111 66A30802          MOV     [0208],EAX
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0115
                                    NV UP EI PL NZ NA PO NC

    DR3,
14EB:0115 0F21D8            MOV     EAX,DR3
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0118
                                    NV UP EI PL NZ NA PO NC

14EB:0118 66A30C02          MOV     [020C],EAX
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=011C
                                    NV UP EI PL NZ NA PO NC
```

```
   DR6,
14EB:011C 0F21F0              MOV    EAX,DR6

AX=4FF0  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=011F
                                    NV UP EI PL NZ NA PO NC
```

и DR7.

```
14EB:011F 66A31002            MOV    [0210],EAX

AX=4FF0  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0123
                                    NV UP EI PL NZ NA PO NC
14EB:0123 0F21F8              MOV    EAX,DR7

AX=0400  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0126
                                    NV UP EI PL NZ NA PO NC
14EB:0126 66A31402            MOV    [0214],EAX

AX=0400  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=012A
                                    NV UP EI PL NZ NA PO NC
```

Записанные данные

```
14EB:0200  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
.............DR0        DR1        DR2        DR3
14EB:0210  F0 4F FF FF 00 04 00 00
p0.....      DR6            DR7
```

MOVS Пересылка данных из строки в строку (Move Data from String to String)
MOVSB Пересылка байта строки (Move String Byte)
MOVSW Пересылка слова строки (Move String Word)
MOVSD Пересылка двойного слова строки (Move String Doubleword)

Команда	Код операции	Двоичное представление
MOVS m8,m8	A4	10100100
MOVS m16,m16	A5	10100101
MOVS m32,m32	A5	10100101

Примечание: обозначение MOVSB является мнемоникой ассемблера для команды MOVS m8,m8; MOVSW – для команды MOVS m16,m16; MOVSD – для команды MOVS m32,m32.

Назначение: Рассматриваемые команды используются для работы со строками, а не логическими или арифметическими величинами. Они воздействуют на один элемент строки, который может быть байтом, словом или двойным словом. Адресация элементов строки осуществляется через регистры ESI и EDI. После

каждой строковой операции регистры ESI и/или EDI автоматически получают положительное или отрицательное приращение и указывают на следующий элемент строки. Если DF = 0, в индексных регистрах выполняется инкремент, если DF = = 1 – декремент. Величина приращения составляет 1, 2 или 4 в зависимости от размера элемента строки.

Команда MOVS пересылает элемент, на который указывает регистр ESI, в ячейку, указанную регистром EDI. При использовании префикса REP команда MOVS выполняет пересылку блоков из памяти в память. Для организации такой пересылки программа должна сначала инициализировать регистры ECX, ESI и EDI. В регистр ECX записывается число байтов, слов или двойных слов в блоке. Если DF = 0, программа должна позиционировать регистр ESI на первый элемент строки-источника, а регистр EDI – на первый элемент строки-приемника. Если DF = 1, программа позиционирует эти регистры на последние элементы строки источника и строки приемника соответственно.

Команда MOVSB работает с байтами, MOVSW – со словами, MOVSD – с двойными словами.

Пример выполнения команды. Показано действие команды пересылки строк. Команда может при каждом выполнении переслать байт, слово или двойное слово. В примере показана пересылка элементов длиной в слово.

Последовательность команд:

```
14EB:0100 90              NOP
14EB:0101 BE0002          MOV   SI,0200
14EB:0104 BF0003          MOV   DI,0300
14EB:0107 B92000          MOV   CX,0020
14EB:010A E302            JCXZ  010E
14EB:010C F3              REPZ
14EB:010D A5              MOVSW
14EB:010E 90              NOP
```

Данные, на которые указывает регистр SI:

```
14EB:0200  01 00 01 00 01 00 01 00-01 00 01 00 01 00 01 00
................
14EB:0210  01 00 01 00 01 00 01 00-01 00 01 00 01 00 01 00
................
14EB:0220  01 00 01 00 01 00 01 00-01 00 01 00 01 00 01 00
................
14EB:0230  01 00 01 00 01 00 01 00-01 00 01 00 01 00 01 00
................
```

Данные, на которые указывает регистр DI:

```
14EB:0300  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
................
14EB:0310  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
................
14EB:0320  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
................
14EB:0330  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
................
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL NZ NA PO NC
```

Обратите внимание на установку флага направления в состояние UP (вверх).

```
14EB:0101 BE0002          MOV   SI,0200

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0200  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL NZ NA PO NC

14EB:0104 BF0003          MOV   DI,0300

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0200  DI=0300  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL NZ NA PO NC

14EB:0107 B92000          MOV   CX,0020

AX=0000  BX=0000  CX=0020  DX=0000  SP=EA04  BP=0000
SI=0200  DI=0300  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                    NV UP EI PL NZ NA PO NC
```

Регистры инициализированы. Команда JCXZ включена в программу, как это обычно и делается в тех случаях, когда в счетчик CX загружается переменная.

```
14EB:010A E302            JCXZ  010E

AX=0000  BX=0000  CX=0020  DX=0000  SP=EA04  BP=0000
SI=0200  DI=0300  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010C
                                    NV UP EI PL NZ NA PO NC

14EB:010C F3              REPZ
14EB:010D A5              MOVSW
```

Пересылка повторяется, пока не будут переданы CX слов.

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010E
                                    NV UP EI PL NZ NA PO NC
```

Данные, на которые указывал регистр SI

```
14EB:0200  01 00 01 00 01 00 01 00-01 00 01 00 01 00 01 00
```

```
14EB:0210  01 00 01 00 01 00 01 00-01 00 01 00 01 00 01 00
................
14EB:0220  01 00 01 00 01 00 01 00-01 00 01 00 01 00 01 00
................
14EB:0230  01 00 01 00 01 00 01 00-01 00 01 00 01 00 01 00
................
```

были переданы по адресу, содержащемуся в DI. Данные источника не изменились.

```
14EB:0300  01 00 01 00 01 00 01 00-01 00 01 00 01 00 01 00
................
14EB:0310  01 00 01 00 01 00 01 00-01 00 01 00 01 00 01 00
................
14EB:0320  01 00 01 00 01 00 01 00-01 00 01 00 01 00 01 00
................
14EB:0330  01 00 01 00 01 00 01 00-01 00 01 00 01 00 01 00
................
```

MOVSX Пересылка с расширением знака (Move with Sign Extension)

Назначение. Команда MOVSX расширяет знак 8-разрядной величины до 16-разрядной и 8- или 16-разрядной величины до 32-разрядной.

Команда	Код операции	Двоичное представление
MOVSX r16,r/m8	0F BE/r	00001111 10111110 mm rrr r/m
MOVSX r32,r/m8	0F BE/r	00001111 10111110 mm rrr r/m
MOVSX r32,r/m16	0F BF/r	00001111 10111111 mm rrr r/m

Пример выполнения команды. Эта команда пересылает байт в слово или двойное слово, а также слово в двойное слово с расширением знака. В примере байт из регистра BL пересылается с расширением знака в регистр AX.

Последовательность команд:

```
14EB:0100 B3F0          MOV     BL,F0
14EB:0102 0FBEC3        MOVSX   AX,BL
14EB:0105 90            NOP
```

Регистры и флаги перед выполнением команды MOVSX:

```
AX=0000  BX=00F0  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102
                                    NV UP EI PL NZ NA PO NC
14EB:0102 0FBEC3        MOVSX   AX,BL
AX=FFF0  BX=00F0  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI PL NZ NA PO NC
```

Знак расширился в регистр AH. Флаги не изменились.

MOVZX Пересылка с расширением нуля (Move with Zero Extension)

Команда	Код операции	Двоичное представление
MOVZX r16,r/m8	0F B6/r	00001111 10110110 mm rrr r/m
MOVZX r32,r/m8	0F B6/r	00001111 10110110 mm rrr r/m
MOVSX r32,r/m16	0F B7/r	00001111 10110111 mm rrr r/m

Назначение. Команда MOVZX расширяет 8-разрядную величину до 16-разрядной и 8- или 16-разрядную величину до 32-разрядной заполнением старших разрядов нулями.

Пример выполнения команды. Эта команда пересылает байт в слово или двойное слово, а также слово в двойное слово с расширением нуля. В примере байт из регистра BL пересылается с расширением нуля в регистр AX.

Последовательность команд:

```
14EB:0100 B3F0          MOV     BL,F0
14EB:0102 0FB6C3        MOVZS   AX,BL
14EB:0105 90            NOP
```

Регистры и флаги перед выполнением команды MOVZX:

```
AX=FFF0  BX=00F0  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102
                                    NV UP EI PL NZ NA PO NC

14EB:0102 0FB6C3          MOVZX   AX,BL

AX=00F0  BX=00F0  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI PL NZ NA PO NC
```

Нуль расширился в регистр AH. Состояние флагов не изменилось.

MUL Целочисленное беззнаковое умножение содержимого регистров AL или AX (Unsigned Integer Multiply of AL Register or AX Register)

Команда	Код операции	Двоичное представление
MUL AL,r/m8	F6/4	11110110 mm 100 r/m
MUL AX,r/m16	F7/4	11110111 mm 100 r/m
MUL EAX,r/m32	F7/4	11110111 mm 100 r/m

Назначение. Команда MUL перемножает операнд-источник и содержимое аккумулятора. Если источник представляет собой байт, МП 80386 умножает его на содержимое AL и возвращает результат удвоенной длины в регистры AH и AL. Если источник представляет собой слово, МП 80386 умножает его на содержимое AX и возвращает результат удвоенной длины в регистры DX и AX. Если источник представляет собой двойное слово, процессор умножает его на содержимое EAX и возвращает 64-разрядный результат в регистры EDX и EAX. Если старшая половина результата не равна нулю, команда MUL устанавливает флаги CF и OF ; в противном случае CF и OF сбрасываются.

Пример выполнения команды. Показано действие команды MUL. Сравните его с действием команды IMUL.

Последовательность команд:

```
14EB:0100 90          NOP
14EB:0101 B033        MOV   AL,33
14EB:0103 B381        MOV   BL,81
14EB:0105 F6E3        MUL   BL
```

Регистры и флаги перед выполнением:

```
AX=0D8C  BX=0044  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    OV UP EI PL NZ NA PE CY
```

Загружаем в регистр AL $33_{16}(51_{10})$.

```
14EB:0101 B033        MOV   AL,33

AX=0D33  BX=0044  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    OV UP EI PL NZ NA PE CY
```

Теперь загружаем в регистр BL $81_{16}(129_{10})$

```
14EB:0103 B381        MOV   BL,81
AX=0D33  BX=0081  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    OV UP EI PL NZ NA PE CY
```

В результате беззнакового умножения получается $19B3_{16}(6579_{10})$, причем число в регистре BL рассматривается, как беззнаковая 8-разрядная величина.

```
14EB:0105 F6E3        MUL   BL

AX=19B3  BX=0081  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    OV UP EI PL NZ NA PE CY
```

NEG Изменение знака, дополнение до 2 (Negate, Two's Complement)

Команда	Код операции	Двоичное представление
NEG r/m8	F6/3	11110110 mm 011 r/m
NEG r/m16	F7/3	11110111 mm 011 r/m
NEG r/m32	F7/3	11110111 mm 011 r/m

Назначение. Команда NEG выполняет вычитание знакового целочисленного операнда из нуля. Она превращает положительное число в отрицательное и наоборот.

Пример выполнения команды. Показано действие команды NEG. Она образует дополнение до 2 величины, над которой выполняется операция.

Последовательность команд:

```
14EB:0100 B80002          MOV   AX,0200
14EB:0103 F7D8            NEG   AX
```

Регистры и флаги после инициализации регистра AX:

```
AX=0200   BX=0044   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0240   DI=0340   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0103
                                        OV UP EI PL NZ NA PE NC
```

Величина в регистре AX заменяется ее дополнением до 2.

```
14EB:0103          F7D8          NEG      AX

AX=FE00   BX=0044   CX=0000   DX=0000   SP=EA04   BP=0000
 SI=0240   DI=0340   DS=14EB   ES=14EB   SS=14EB   CS=14EB
 IP=0105
                                        NV UP EI NG NZ NA PE CY
                                        *          *          *
```

Флаги устанавливаются в соответствии с результатом операции. Если операнд не равен нулю, устанавливается флаг переноса. В противном случае флаг переноса сбрасывается.

NOP Холостая команда (No Operation)

Команда	Код операции	Двоичное представление
NOP	90	10010000

Назначение. Команда NOP занимает в памяти 1 байт. Она действует только на указатель команд EIP. Команда NOP полезна при «выравнивании» адресов переходов. Адрес может требовать 8- или 16-разрядного смещения, и если зарезервированы 16 разрядов, для их заполнения можно использовать 8-разрядное смещение и команду NOP.

Пример выполнения команды. Команда NOP использовалась во многих примерах. См., например, команду MUL.

NOT Инверсия, дополнение до 1 (Negate, One's Complement)

Команда	Код операции	Двоичное представление
NOT r/m8	F6/2	11110110 mm 010 r/m
NOT r/m16	F7/2	11110111 mm 010 r/m
NOT r/m32	F7/2	11110111 mm 010 r/m

Назначение. Команда NOT выполняет инверсию битов указанного в команде операнда с целью образования дополнения операнда до 1. Команда NOT представляет одноместную операцию (термин относится к арифметическому оператору с одним операндом), выполняемую над единственным операндом в памяти или в регистре. Команда NOT не влияет на флаги.

Пример выполнения команды. Показано действие команды NOT. Команда образует дополнение операнда до 1.

Последовательность команд:

```
14EB:0100 B80002          MOV  AX,0200
14EB:0103 F7D0            NOT  AX
```

Регистры и флаги после инициализации регистра AX:

```
AX=0200  BX=0044  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI NG NZ NA PE CY

14EB:0103 F7D0                NOT  AX

AX=FDFF  BX=0044  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI NG NZ NA PE CY
```

Команда NOT выполнила инверсию каждого бита операнда. Флаги не изменились.

OR Логическое ВКЛЮЧАЮЩЕЕ ИЛИ (Logical Inclusive OR)

Команда	Код операции	Двоичное представление
OR AL,imm8	0C ib	00001100
OR AX,imm16	0D iw	00001101
OR EAX,imm32	0D id	00001101
OR r/m8,imm8	80 /1 ib	10000000 mm 001 r/m
OR r/m16,imm16	81 /1 i2	10000001 mm 001 r/m
OR r/m32,imm32	81 /1 id	10000001 mm 001 r/m
OR r/m16,imm8	83 /1 ib	10000011 mm 001 r/m
OR r/m32,imm8	83 /1 ib	10000011 mm 001 r/m
OR r/m8,r8	08 /r	00001000 mm 001 r/m
OR r/m16,r16	09 /r	00001001 mm 001 r/m
OR r/m32,r32	09 /r	00001001 mm 001 r/m
OR r8,r/m8	0A /r	00001010 mm rrr r/m
OR r16,r/m16	0B /r	00001011 mm rrr r/m
OR r32,r/m32	0B /r	00001011 mm rrr r/m

Назначение. Команда OR сравнивает два операнда и вычисляет следующее: если соответствующие биты в операндах равны 0, результат 0; в противном случае результат 1.

Операнд 1	0	0	1	1
Операнд 2	0	1	1	0
Результат	0	1	1	1

Пример выполнения команды. Команда OR комбинирует две величины бит за битом. Если бит установлен в любом или в обоих операндах, он устанавливается и в результате.

Последовательность команд:

```
14EB:0100 B80002          MOV  AX,0200
14EB:0103 0C3F            OR   AL,3F
14EB:0105 0D000A          OR   AX,0A00
```

Регистры и флаги после инициализации регистра AX:

```
AX=0200  BX=0044  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PE NC
```

Команда OR сбрасывает флаги переполнения и переноса. Заметьте, что операция ИЛИ над двумя битами, один из которых установлен, а другой сброшен, устанавливает бит результата, если же оба бита сброшены, результирующий бит тоже сбрасывается.

```
14EB:0103 0C3F                OR  AL,3F

AX=023F  BX=0044  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI PL NZ NA PE NC
```

Операция ИЛИ над двумя установленными битами не изменяет их значения.

```
14EB:0105 0D000A              OR  AX,0A00

AX=0A3F  BX=0044  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0240  DI=0340  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    NV UP EI PL NZ NA PE NC
```

OUT Вывод в порт (Output to Port)

Команда	Код операции	Двоичное представление
OUT imm8,AL	E6 ib	11100110
OUT imm8,AX	E7 ib	11100111
OUT imm8,EAX	E7 ib	11100111
OUT DX,AL	EE	11101110
OUT DX,AX	EF	11101111
OUT DX,EAX	EF	11101111

Назначение. Команда OUT пересылает данные из регистра в порт вывода. Операнд-источник (второй операнд команды) находится в регистре AL, AX или EAX. Номер порта определяется первым операндом. Для вывода данных в любой порт от 0 до 65535 номер порта помещается в регистр DX. В этом случае в качестве первого операнда используется регистр DX. Если в команде указано 8-разрядное значение номера порта, оно расширяется нулями до 16 разрядов.

Пример выполнения команды. Действие взаимосвязанных команд IN, INS, OUT и OUTS показано в одном примере. См. команду IN.

OUTS Вывод строки в порт (Output String to Port)
OUTSB Вывод байта (Output Byte)
OUTSW Вывод слова (Output Word)
OUTSD Вывод двойного слова (Output Doubleword)

Команда	Код операции	Двоичное представление
OUTS DX,r/m8	6E	01101110
OUTS DX,r/m16	6F	01101111
OUTS DX,r/m32	6F	01101111

Примечание: обозначение OUTSB является мнемоникой ассемблера для команды OUTS DX,r/m8; OUTSW – мнемоникой для команды OUTS DX,r/m16; OUTSD – мнемоникой для команды OUTS DX,r/m32.

Назначение. Команда OUTS действует аналогично команде OUT в том отношении, что она пересылает данное (байт, слово или двойное слово памяти), на которое указывает индексный регистр-источник, в порт вывода, адресуемый через регистр DX. После пересылки содержимое индексного регистра-источника (SI или ESI, см. ниже) получает положительное или отрицательное приращение. Если DF = 0 (была выполнена команда CLD), в индексном регистре осуществляется инкремент, если DF = 1 (была выполнена команда STD),– декремент. Величина приращения зависит от размера пересылаемого данного: 1 в случае байта, 2 – слова и 4 – двойного слова.

Адрес данных источника определяется по содержимому индексного регистра-источника. До выполнения команд пересылки в регистры SI или ESI должно быть загружено правильное значение индекса. Если атрибут размера адреса для этих команд составляет 16 разрядов, используется регистр SI. Если атрибут размера адреса составляет 32 разряда, используется ESI.

Порт адресуется с помощью величины в регистре DX. Команда OUTS не допускает указание номера порта как непосредственной величины.

Пример выполнения команды. Действие взаимосвязанных команд IN, INS, OUT и OUTS показано в одном примере. См. команду IN.

POP Извлечение слова из стека (Pop a Word from the Stack)

Команда	Код операции	Двоичное представление
POP m16	8F /0	10001111 mm 000 r/m
POP m32	8F /0	10001111 mm 000 r/m
POP r16	58 + rw	01011rrr
POP r32	58 + rd	01011rrr
POP DS	1F	00011111
POP ES	07	00000111
POP SS	17	00010111
POP FS	0F A1	00001111 10100001
POP GS	0F A9	00001111 10101001

Назначение. Команда POP пересылает слово или двойное слово из текущей вершины стека (на которую указывает регистр ESP) по адресу приемника. Затем в регистре ESP выполняется инкремент и он указывает на новую вершину стека.

Пример выполнения команды. Показано действие команд PUSH и POP. Команды рассмотрены совместно ввиду их логического взаимодействия.

Последовательность команд:

```
14EB:0100 50                PUSH  AX
14EB:0101 5B                POP   BX
```

Регистры и флаги перед выполнением:

```
AX=0101   BX=0202   CX=0303   DX=0404   SP=E9A4   BP=0505
SI=0606   DI=0707   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0100
                                        NV UP EI PL NZ NA PO NC

14EB:0100  50        PUSH    AX

AX=0101   BX=0202   CX=0303   DX=0404   SP=E9A2   BP=0505
SI=0606   DI=0707   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0101
                                        NV UP EI PL NZ NA PO NC
```

Заметьте, что команда PUSH, помещая содержимое регистра в стек, выполняет декремент указателя стека. Изменяется при этом только стек.

```
14EB:0101  5B          POP     BX

AX=0101   BX=0101   CX=0303   DX=0404   SP=E9A4   BP=0505
SI=0606   DI=0707   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0102
                                        NV UP EI PL NZ NA PO NC
```

Команда POP извлекает слово из стека. Выполняется инкремент указателя стека, и регистр-приемник получает данные.

POPA Извлечение из стека содержимого всех регистров (Pop All Registers)

POPAD Извлечение из стека содержимого всех регистров – 32-разрядный режим (Pop All Registers – 32-bit Mode)

Команда	Код операции	Двоичное представление
POPA	61	01100001
POPAD	61	01100001

Назначение. Команда POPA восстанавливает содержимое регистров, сохраненное в стеке командой PUSHA. Сохраненное значение ESP игнорируется.

Пример выполнения команды. Показано действие команды POPA. Содержимое регистров общего назначения восстанавливается из стека. Сначала восстанавливаются двухсловные регистры, затем однословные.

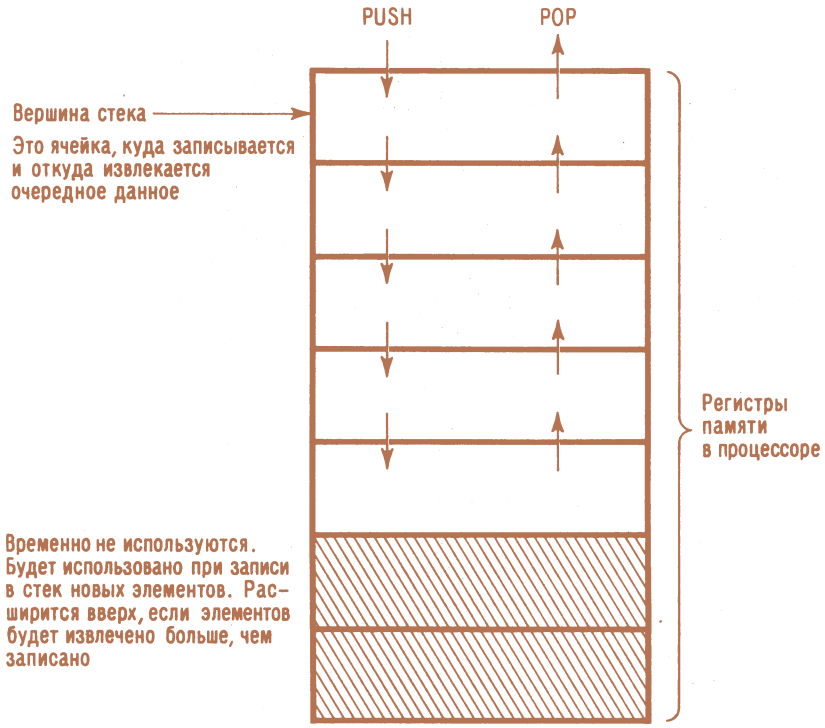


Рис. 10.4. Работа со стеком.

Последовательность команд:

```
14EB:010E 6661          POPAD
14EB:0110 61            POPA
```

Данные в стеке перед выполнением команд:

```
14EB:E9A0                       07 07 00 00-06 06 00 00 05 05 00 00
............
14EB:E9B0       C4 E9 00 00 02 02 00 00-04 04 00 00 03 03 00 00
Di..............
14EB:E9C0       01 01 DC 3D 07 07 06 06-05 05 D4 E9 02 02 04 04
..\=......Ti....
14EB:E9D0       03 03 01 01
....
```

Регистры и флаги перед выполнением команд:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=E9A4  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010E
                                          NV UP EI PL ZR NA PE NC
```

```
14EB:010E 6661        POPAD

AX=0101  BX=0202  CX=0303  DX=0404  SP=E9C4  BP=0505
SI=0606  DI=0707  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0110
                                    NV UP EI PL ZR NA PE NC
```

Обратите внимание, как замена ширины привела к извлечению из стека содержимого двухсловных регистров. Значение SP было увеличено на 32.

```
14EB:0110 61          POPA

AX=0101  BX=0202  CX=0303  DX=0404  SP=E9D4  BP=0505
SI=0606  DI=0707  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0111
                                    NV UP EI PL ZR NA PE NC
```

Из стека восстановлено содержимое однословных регистров. Значение SP увеличено на 16. Флаги не изменились.

POPF Восстановление из стека регистра FLAGS или EFLAGS (Pop Stack into FLAGS or EFLAGS Register)

POPFD Восстановление из стека – 32-разрядный режим (Pop Stack – 32-bit Mode)

Команда	Код операции	Двоичное представление
POPF	9D	10011101
POPFD	9D	10011101

Назначение. Команда POPF пересылает определенные биты слова, находящегося на вершине стека, в младший байт регистра EFLAGS. Затем значение ESP увеличивается на 2. Команда POPFD пересылает биты и увеличивает ESP на 4. Обе команды не влияют на флаги RF и VM.

С помощью команд PUSHF и POPF удобно выполнять пересылку флагов в память с целью их анализа и модификации, а также сохранение состояния регистра флагов на время выполнения процедуры.

Пример выполнения команд. Показано действие команд PUSHF и POPF. Флаги дважды записаны в стек, затем извлечены из стека в регистр AX и инвертированы. Далее содержимое AX записано в стек и перенесено оттуда в регистр флагов. Наконец, из стека восстановлено исходное состояние флагов.

Последовательность команд:

```
14EB:0100 90          NOP
14EB:0101 9C          PUSHF
14EB:0102 9C          PUSHF
14EB:0103 58          POP     AX
14EB:0104 F7D0        NOT     AX
14EB:0106 50          PUSH    AX
14EB:0107 9D          POPF
14EB:0108 9D          POPF
```

Регистры и флаги перед выполнением:

```
AX=8DB9  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL ZR NA PE NC

14EB:0101 9C        PUSHF

AX=8DB9  BX=0000  CX=0000  DX=0000  SP=EA02  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102
                                    NV UP EI PL ZR NA PE NC

14EB:0102 9C        PUSHF

AX=8DB9  BX=0000  CX=0000  DX=0000  SP=EA00  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL ZR NA PE NC
```

Флаги занесены в стек дважды.

```
14EB:0103 58        POP   AX

AX=7246  BX=0000  CX=0000  DX=0000  SP=EA02  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL ZR NA PE NC
```

Одна копия флагов извлекается из стека и пересылается в регистр AX.

```
14EB:0104 F7D0      NOT   AX

AX=8DB9  BX=0000  CX=0000  DX=0000  SP=EA02  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL ZR NA PE NC
```

Значения всех битов изменяются на противоположные.

```
14EB:0106 50        PUSH  AX

AX=8DB9  BX=0000  CX=0000  DX=0000  SP=EA00  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL ZR NA PE NC
```

Инвертированные биты занесены в стек.

```
14EB:0107 9D          POPF

AX=8DB9   BX=0000   CX=0000   DX=0000   SP=EA02   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0108
                                  OV DN DI NG NZ AC PO CY
                                   *  *  *  *  *  *  *  *
```

Пересылаем их в регистр флагов. Заметьте, что все биты изменились.

```
14EB:0108 9D          POPF

AX=8DB9   BX=0000   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0109
                                  NV UP EI PL ZR NA PE NC
```

Наконец, восстановили первоначальные значения флагов.

PUSH Занести операнд в стек (Push Operand onto the Stack)

Команда	Код операции	Двоичное представление
PUSH m16	FF /6	11111111 mm 110 r/m
PUSH m32	FF /6	11111111 mm 110 r/m
PUSH r16	50 + /r	01010rrr
PUSH r32	50 + /r	01010rrr
PUSH imm8	6A	01101010
PUSH imm16	68	01101000
PUSH imm32	68	01101000
PUSH CS	0E	00001110
PUSH SS	16	00010110
PUSH DS	1E	00011110
PUSH ES	06	00000110
PUSH FS	0F A0	00001111 10100000
PUSH GS	0F A8	00001111 10101000

Назначение. Команда PUSH выполняет декремент указателя стека (ESP), затем заносит операнд-источник на вершину стека, куда указывает ESP. Эта команда часто используется для занесения в стек параметров перед вызовом процедуры. Она также удобна для сохранения в стеке значений временных переменных. Команду PUSH можно использовать с операндами, находящимися в памяти, в регистрах (включая регистры сегментов), а также с непосредственными операндами.

Пример выполнения команды. См. команду POP.

PUSHA Занести в стек содержимое всех регистров общего назначения (Push All General Registers)

PUSHAD

Команда	Код операции	Двоичное представление
PUSHA	60	01100000
PUSHAD	60	01100000

Назначение. Команда PUSHA сохраняет в стеке содержимое восьми регистров общего назначения. Команда упрощает вызов процедуры, сокращая число команд, требуемых для сохранения содержимого тех регистров общего назначения, которые будут использоваться в процедуре. Содержимое регистров общего назначения заносится в стек в следующем порядке: EAX, ECX, EDX, EBX, начальное значение ESP (указателя стека) перед тем, как в стек было занесено значение EAX, далее EBP, ESI и EDI.

Пример выполнения команды. Показано действие команды PUSHA. Поскольку программа в примере выполнялась с шириной операнда по умолчанию 16 разрядов, для сохранения в стеке двухсловных регистров использована замена размера операнда. Если бы по умолчанию действовал размер 32 разряда, замена привела бы к сохранению в стеке однословных регистров.

Последовательность команд:

```
14EB:0115 60            PUSHA
14EB:0116 6660          PUSHAD
```

Регистры и флаги перед выполнением:

```
AX=0101   BX=0202   CX=0303   DX=0404   SP=E9D4   BP=0505
SI=0606   DI=0707   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0115
                                        NV UP EI PL NZ NA PE NC

14EB:0115 60            PUSHA

AX=0101   BX=0202   CX=0303   DX=0404   SP=E9C4   BP=0505
SI=0606   DI=0707   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0116
                                        NV UP EI PL NZ NA PE NC
```

Заметьте, что изменение касается только SP.

```
14EB:0116 6660          PUSHAD

AX=0101   BX=0202   CX=0303   DX=0404   SP=E9A4   BP=0505
SI=0606   DI=0707   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0118
                                        NV UP EI PL NZ NA PE NC
```

То же справедливо и для двухсловного варианта команды.

Содержимое стека, показывающее результат выполнения двухсловной команды PUSHA:

```
                          EDI         ESI         EBP
14EB:E9A0                 07 07 00 00-06 06 00 00 05 05 00 00
                    ESP         EBX         EDX         ECX
14EB:E9B0           C4 E9 00 00 02 02 00 00-04 04 00 00 03 03 00 00
```

Результат выполнения однословного варианта команды PUSHA:

```
                    EAX         DI   SI   BP   SP   BX   DX
14EB:E9C0           01 01 DC 3D 07 07 06 06-05 05 D4 E9 02 02 04 04
..\=.....Ti....

                    CX   AX
14EB:E9D0           03 03 01 01
....
```

PUSHF Занесение в стек содержимого регистра флагов (Push Flags Register EFLAGS onto the Stack)
PUSHFD

Команда	Код операции	Двоичное представление
PUSHF	9C	10011100
PUSHFD	9C	10011100

Назначение. Команда PUSHF выполняет декремент на 2 регистра ESP (указателя стека) и затем заносит на вершину стека, куда указывает ESP, младшее слово из регистра EFLAGS. Команда PUSHFD выполняет декремент ESP на 4, затем заносит на вершину стека, куда указывает ESP, оба слова из регистра EFLAGS. Заметьте, что флаги VM и RF не изменяются.

Команды PUSHF и POPF используются для сохранения флагов в памяти, где их можно анализировать и модифицировать. Команды также удобны для сохранения состояния флагов на время выполнения процедуры.

Пример выполнения команды. См. команду POPF.

RCL Циклический сдвиг влево через бит переноса с использованием бита CF для расширения (Rotate Left through Carry – Uses CF for extension)

RCR Циклический сдвиг вправо через бит переноса с использованием бита CF для расширения (Rotate Right through Carry – Uses CF for extension)

ROL Циклический сдвиг влево с циклическим возвратом битов (Rotate Left – Wrap bits around)

ROR Циклический сдвиг вправо с циклическим возвратом битов (Rotate Right – Wrap bits around)

Команда	Код операции	Двоичное представление
RCL r/m8,1	D0 /2	11010000 mm 010 r/m
RCL r/m8,CL	D2 /2	11010010 mm 010 r/m
RCL r/m8,imm8	C0 /2 ib	11000000 mm 010 r/m
RCL r/m16,1	D1 /2	11010001 mm 010 r/m
RCL r/m16,CL	D3 /2	11010011 mm 010 r/m
RCL r/m16,imm8	C1 /2 ib	11000001 mm 010 r/m
RCL r/m32,1	D1 /2	11010001 mm 010 r/m
RCL r/m32,CL	D3 /2	11010011 mm 010 r/m
RCL r/m32,imm8	C1 /2 ib	11000001 mm 010 r/m
RCR r/m8,1	D0 /3	11010000 mm 011 r/m
RCR r/m8,CL	D2 /3	11010010 mm 011 r/m
RCR r/m8,imm8	C0 /3 ib	11000000 mm 011 r/m
RCR r/m16,1	D1 /3	11010001 mm 011 r/m

RCR r/m16,CL	D3 /3	11010011	mm 011 r/m
RCR r/m16,imm8	C1 /3 ib	11000001	mm 011 r/m
RCR r/m32,1	D1 /3	11010001	mm 011 r/m
RCR r/m32,CL	D3 /3	11010011	mm 011 r/m
RCR r/m32,imm8	C1 /3 ib	11000001	mm 011 r/m
ROL r/m8,1	D0 /0	11010000	mm 000 r/m
ROL r/m8,CL	D2 /0	11010010	mm 000 r/m
ROL r/m8,imm8	C0 /0 ib	11000000	mm 000 r/m
ROL r/m16,1	D1 /0	11010001	mm 000 r/m
ROL r/m16,CL	D3 /0	11010011	mm 000 r/m
ROL r/m16,imm8	C1 /0 ib	11000001	mm 000 r/m
ROL r/m32,1	D1 /0	11010001	mm 000 r/m
ROL r/m32,CL	D3 /0	11010011	mm 000 r/m
ROL r/m32,imm8	C1 /0 ib	11000001	mm 000 r/m
ROR r/m8,1	C0 /1	11000000	mm 001 r/m
ROR r/m8,CL	D2 /1	11010010	mm 001 r/m
ROR r/m8,imm8	C0 /1 ib	11000000	mm 001 r/m
ROR r/m16,1	D1 /1	11010001	mm 001 r/m
ROR r/m16,CL	D3 /1	11010011	mm 001 r/m
ROR r/m16,imm8	C1 /1 ib	11000001	mm 001 r/m
ROR r/m32,1	D1 /1	11010001	mm 001 r/m
ROR r/m32,CL	D3 /1	11010011	mm 001 r/m
ROR r/m32,imm8	C1 /1 ib	11000001	mm 001 r/m

Назначение. Команды циклического сдвига дают возможность циклически сдвигать биты в байтах, словах и двойных словах. Биты, сдвинутые до конца операнда, не теряются, как при линейном сдвиге, а «совершают круговорот назад» на другой «конец» операнда. Циклический сдвиг влияет только на флаги переноса и переполнения. Флаг CF в двух командах циклического сдвига используется как расширение операнда. Это позволяет изолировать любой бит и затем анализировать его значение с помощью команд условных переходов (JC или JNC). Флаг CF всегда содержит значение последнего выдвинутого из операнда бита, даже если команда не использует этот флаг для расширения сдвигаемого операнда.

При выполнении одноразрядных сдвигов флаг OF устанавливается, если операция изменяет старший (знаковый) бит операнда-приемника. Если знаковый бит сохраняет свое первоначальное значение, флаг OF сбрасывается. При выполнении многоразрядных сдвигов значение флага OF *всегда* не определено.

Команда RCL сдвигает биты в однобайтовом, однословном или двухсловном операнде-приемнике *влево* на один разряд или на число разрядов, указанное в операнде-счетчике. RCL отличается от ROL тем, что рассматривает флаг CF как однобитовое расширение (со стороны старших битов) операнда-приемника. Каждый старший бит, выдвинутый с левой стороны операнда, помещается в CF перед тем, как вернуться в операнд в качестве самого младшего бита в следующем такте сдвига.

Команда RCR сдвигает биты в однобайтовом, однословном или двухсловном операнде-приемнике *вправо* на один разряд или на число разрядов, указанное в операнде-счетчике. RCR отличается от ROR тем, что рассматривает флаг CF как однобитовое расширение (со стороны младших битов) операнда-приемника. Каждый младший бит, выдвинутый с правой стороны операнда, помещается в CF, перед

тем как вернуться в операнд в качестве самого старшего бита в следующем такте сдвига.

Команда ROL сдвигает биты в однобайтовом, однословном или двухсловном операнде-приемнике *влево* на один разряд или на число разрядов, указанное в операнде-счетчике. В каждом такте сдвига старший бит, выдвинутый с левой стороны операнда, возвращается на его правую сторону, чтобы стать новым младшим битом операнда.

Команда ROR сдвигает биты в однобайтовом, однословном или двухсловном операнде-приемнике *вправо* на один разряд или на число разрядов, указанное в операнде-счетчике. В каждом такте сдвига младший бит, выдвинутый с правой стороны операнда, возвращается на его левую сторону, чтобы стать новым старшим битом операнда.

Пример выполнения команд RCL, ROL, RCR, ROR. Показано действие команд циклических сдвигов: сначала сдвигов через флаг переноса (RCx) с включением флага переноса в качестве одного из сдвигаемых битов, затем простых циклических сдвигов (ROx). Подчеркнуто различие этих групп команд.

Последовательность команд:

```
14EB:0100  F8          CLC
14EB:0101  B081        MOV  AL,81
14EB:0103  B102        MOV  CL,02
14EB:0105  D0D0        RCL  AL,1
14EB:0107  D0D0        RCL  AL,1
14EB:0109  D0D8        RCR  AL,1
14EB:010B  D0D8        RCR  AL,1
14EB:010D  D2D0        RCL  AL,CL
14EB:010F  D2D8        RCR  AL,CL
14EB:0111  90          NOP
14EB:0112  B081        MOV  AL,81
14EB:0114  D0C0        ROL  AL,1
14EB:0116  D0C0        ROL  AL,1
14EB:0118  D0C8        ROR  AL,1
14EB:011A  D0C8        ROR  AL,1
14EB:011C  D2C0        ROL  AL,CL
14EB:011E  D2C8        ROR  AL,CL
```

Регистры и флаги перед выполнением:

```
AX=0081   BX=0000   CX=0002   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0105
                                  OV UP EI PL NZ NA PO NC
```

Мы сбросили флаг переноса, загрузили в счетчик CL число 2, а в регистр AL – число 81_{16}. Биты, подлежащие сдвигу:

```
                          1000 0001 0

14EB:0105  D0D0        RCL  AL,1
AX=0002   BX=0000   CX=0002   DX=0000   SP=EA04   BP=0000
```

```
SI=0000    DI=0000    DS=14EB    ES=14EB    SS=14EB    CS=14EB
IP=0107
                                          OV UP EI PL NZ NA PO CY
                                          *                    *
```

Последовательность битов после первого сдвига: 0000 0010 1. Заметьте, что в результате сдвига установились флаги переноса и переполнения.

```
14EB:0107 D0D0        RCL    AL,1

AX=0005    BX=0000    CX=0002    DX=0000    SP=EA04    BP=0000
SI=0000    DI=0000    DS=14EB    ES=14EB    SS=14EB    CS=14EB
IP=0109
                                          NV UP EI PL NZ NA PO NC
                                          *                    *
```

Последовательность битов после второго сдвига: 0000 0101 0. Теперь и флаг переноса, и флаг переполнения сброшены.

```
14EB:0109 D0D8        RCR    AL,1

AX=0002    BX=0000    CX=0002    DX=0000    SP=EA04    BP=0000
SI=0000    DI=0000    DS=14EB    ES=14EB    SS=14EB    CS=14EB
IP=010B
                                          NV UP EI PL NZ NA PO CY
                                                               *
```

Сдвиг вправо восстанавливает последовательность битов: 0000 0010 1. Флаг переноса установлен, а флаг переполнения сброшен.

```
14EB:010B D0D8        RCR    AL,1

AX=0081    BX=0000    CX=0002    DX=0000    SP=EA04    BP=0000
SI=0000    DI=0000    DS=14EB    ES=14EB    SS=14EB    CS=14EB
IP=010D
                                          OV UP EI PL NZ NA PO NC
                                                               *
```

Сдвиг вернул последовательность битов в исходное состояние.

```
14EB:010D D2D0        RCL    AL,CL

AX=0005    BX=0000    CX=0002    DX=0000    SP=EA04    BP=0000
SI=0000    DI=0000    DS=14EB    ES=14EB    SS=14EB    CS=14EB
IP=010F
                                          NV UP EI PL NZ NA PO NC
```

Сдвиг выполнен через флаг переноса. Флаг переполнения изменил состояние, однако он не определен. Его не следует использовать после многоразрядного сдвига. После двух сдвигов последовательность битов приняла вид 0000 0101 0. Содержимое счетчика (CL) не изменилось.

```
14EB:010F D2D8        RCR   AL,CL

AX=0081  BX=0000  CX=0002  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0111
                                    OV UP EI PL NZ NA PO NC
```

После двухразрядного сдвига вправо последовательность битов приняла вид 1000 0001 0.

```
14EB:0111 90          NOP

AX=0081  BX=0000  CX=0002  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0112
                                    OV UP EI PL NZ NA PO NC

14EB:0112 B081        MOV   AL,81

AX=0081  BX=0000  CX=0002  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0114
                                    OV UP EI PL NZ NA PO NC

14EB:0114 D0C0        ROL   AL,1

AX=0003  BX=0000  CX=0002  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0116
                                    OV UP EI PL NZ NA PO CY
                                    *                     *
```

Простой сдвиг изменил последовательность на 0000 0011 1. Заметьте, что флаги переноса и переполнения установлены, но бит переноса не входит в сдвигаемую последовательность.

```
14EB:0116 D0C0        ROL   AL,1

AX=0006  BX=0000  CX=0002  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0118
                                    NV UP EI PL NZ NA PO NC
                                     *                    *
```

Снова бит переноса не участвует в сдвиге, хотя его значение (0) совпадает со значением бита, выдвинутого из крайней старшей позиции. Результат сдвига: 0000 0110 0.

```
14EB:0118 D0C8        ROR   AL,1

AX=0003   BX=0000   CX=0002   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=011A
                                    NV UP EI PL NZ NA PO NC
```

Сдвиг вправо возвращает нас к значению 0000 0011 0.

```
14EB:011A D0C8        ROR   AL,1

AX=0081   BX=0000   CX=0002   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=011C
                                    OV UP EI PL NZ NA PO CY
                                    *                     *
```

Второй сдвиг вправо возвращает нас к исходному значению 1000 0001 1. Флаг переноса устанавливается при сдвиге вправо, когда в старший разряд вдвигается бит, равный 1. Флаг переполнения устанавливается при сдвиге вправо на 1 разряд, если в результате сдвига изменяется состояние флага переноса.

```
14EB:011C D2C0        ROL   AL,CL

AX=0006   BX=0000   CX=0002   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=011E
                                    NV UP EI PL NZ NA PO NC
                                                          *
```

Сдвиг влево на два разряда привел нас к значению 0000 0110 0. Флаг переноса сброшен, потому что последний бит, выдвинутый из старшей позиции, был 0.

```
14EB:011E D2C8        ROR   AL,CL

AX=0081   BX=0000   CX=0002   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0120
                                    OV UP EI PL NZ NA PO CY
                                                          *
```

Сдвиг вправо на два разряда возвращает исходное состояние и устанавливает флаг переноса, поскольку последний бит, вдвинутый в старшую позицию, был 1.

REP Повторение последующей строковой операции (Repeat Following String Operation)

REPE Повторение, пока равно (Repeat while Equal)

REPZ Повторение, пока нуль (Repeat while Zero)

REPNE Повторение, пока не равно (Repeat while Not Equal)

REPNZ Повторение, пока не нуль (Repeat while Not Zero)

Команда	Код операции	Двоичное представление
REP INS r/m8,DX	F3 6C	11110011 01101100
REP INS r/m16,DX	F3 6D	11110011 01101101
REP INS r/m32,DX	F3 6D	11110011 01101101
REP MOVS m8,m8	F3 A4	11110011 10100100
REP MOVS m16,m16	F3 A5	11110011 10100101
REP MOVS m32,m32	F3 A5	11110011 10100101
REP OUTS DX,r/m8	F3 6E	11110011 01101110
REP OUTS DX,r/m16	F3 6F	11110011 01101111
REP OUTS DX,r/m32	F3 6F	11110011 01101111
REP STOS m8	F3 AA	11110011 10101010
REP STOS m16	F3 AB	11110011 10101011
REP STOS m32	F3 AB	11110011 10101011
REPE CMPS m8,m8	F3 A6	11110011 10100110
REPE CMPS m16,m16	F3 A7	11110011 10100111
REPE CMPS m32,m32	F3 A7	11110011 10100111
REPE SCAS m8	F3 AE	11110011 10101110
REPE SCAS m16	F3 AF	11110011 10101111
REPE SCAS m32	F3 AF	11110011 10101111
REPNE CMPS m8,m8	F2 A6	11110010 10100110
REPNE CMPS m16,m16	F2 A7	11110010 10100111
REPNE CMPS m32,m32	F2 A7	11110010 10100111
REPNE SCAS m8	F2 AE	11110010 10101110
REPNE SCAS m16	F2 AF	11110010 10101111
REPNE SCAS m32	F2 AF	11110010 10101111

Назначение. Префикс REP задает повторение строковой операции, что позволяет МП 80386 обрабатывать строки значительно быстрее, чем с помощью обычного программного цикла. Если строковой операции предшествует один из этих префиксов повторения, операция выполняется многократно. При каждом выполнении операция воздействует на другой элемент строки. Повторение заканчивается, когда выполняется одно из условий, определяемых префиксом.

В процессе повторения команды строковая операция может быть приостановлена на время обработки внешнего прерывания или исключения. После окончания обработки прерывания строковая операция возобновляется.

Пример выполнения команды. См. команду MOVS.

RET Возврат из процедуры (Return from Procedure)

Команда	Код операции	Двоичное представление
RET	C3	11000011
RET imm16	CA/w	11001010

Назначение. Команда RET завершает выполнение вызванной процедуры и передает управление посредством обратной ссылки, хранящейся в стеке. Обратная ссылка указывает на программу, первоначально вызвавшую процедуру. Команда RET восстанавливает значение EIP (указателя команд), которое было сохранено в стеке предыдущей командой CALL. В команду RET может быть включен необязательный непосредственный операнд. Прибавляя эту константу к новому значению

указателя вершины стека, команда RET удаляет аргументы, помещенные вызывающей программой в стек перед выполнением команды CALL.

Пример выполнения команды. Команды CALL, ENTER, LEAVE и RET собраны в одном примере, поскольку они логически «согласуются» друг с другом и выполняют взаимно обратные действия. См. команду CALL.

SAHF Запись содержимого регистра AH в регистр флагов (Store AH into Flags)

Команда	Код операции	Двоичное представление
SAHF	9E	10011110

Назначение. Хотя имеются специальные команды для изменения состояния флагов CF и DF, другие флаги, используемые в прикладной программе, непосредственно изменить нельзя. Команды пересылки флагов (LAHF и SAHF) дают возможность программного изменения других битов флагов с помощью команд обработки битов после пересылки флагов в стек или регистр AH.

Пример выполнения команды. Показана пересылка содержимого регистра AH в регистр флагов.

Последовательность команд:

```
14EB:0100  B80000  MOV   AX,0000
14EB:0103  9E      SAHF
14EB:0104  F6D4    NOT   AH
14EB:0106  9E      SAHF
```

Регистры и флаги перед выполнением команды SAHF:

```
AX=0000  BX=0000  CX=0002  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    OV UP EI PL NZ NA PO CY
```

Сначала сбрасываем флаги (значение в регистре AH).

```
14EB:0103  9E   SAHF

AX=0000  BX=0000  CX=0002  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    OV UP EI PL NZ NA PO NC
                                                        *
```

Инвертируем все биты AH.

```
14EB:0104  F6D4  NOT   AH

AX=FF00  BX=0000  CX=0002  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    OV UP EI PL NZ NA PO NC
```

Теперь устанавливаем флаги в 1.

```
14EB:0106   9E    SAHF

AX=FF00   BX=0000   CX=0002   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0107
                                  OV UP EI NG ZR AC PE CY
                                   *  *  *  *  *  *  *
```

SAL Команды сдвига (Shift Instructions)
SAR
SHL
SHR

Команда	Код операции	Двоичное представление
SAL r/m8,1	D0 /4	11010000 mm 100 r/m
SAL r/m8,CL	D2 /4	11010010 mm 100 r/m
SAL r/m8,imm8	C0 /4 ib	11000000 mm 100 r/m
SAL r/m16,1	D1 /4	11010001 mm 100 r/m
SAL r/m16,CL	D3 /4	11010011 mm 100 r/m
SAL r/m16,imm8	C1 /4 ib	11000001 mm 100 r/m
SAL r/m32,1	D1 /4	11010001 mm 100 r/m
SAL r/m32,CL	D3 /4	11010011 mm 100 r/m
SAL r/m32,imm8	C1 /4 ib	11000001 mm 100 r/m
SAR r/m8,1	D0 /7	11010000 mm 111 r/m
SAR r/m8,CL	D2 /7	11010010 mm 111 r/m
SAR r/m8,imm8	C0 /7 ib	11000000 mm 111 r/m
SAR r/m16,1	C1 /7	11000001 mm 111 r/m
SAR r/m16,CL	D3 /7	11010011 mm 111 r/m
SAR r/m16,imm8	C1 /7 ib	11000001 mm 111 r/m
SAR r/m32,1	D1 /7	11010001 mm 111 r/m
SAR r/m32,CL	D3 /7	11010011 mm 111 r/m
SAR r/m32,imm8	C1 /7 ib	11000001 mm 111 r/m
SHR r/m8,1	D0 /5	11010000 mm 101 r/m
SHR r/m8,CL	C2 /5	11000010 mm 101 r/m
SHR r/m8,imm8	C0 /5 ib	11000000 mm 101 r/m
SHR r/m16,1	D1 /5	11010001 mm 101 r/m
SHR r/m16,CL	D3 /5	11010011 mm 101 r/m
SHR r/m16,imm8	C1 /5 ib	11000001 mm 101 r/m
SHR r/m32,1	D1 /5	11010001 mm 101 r/m
SHR r/m32,CL	D3 /5	11010011 mm 101 r/m
SHR r/m32,imm8	C1 /5 ib	11000001 mm 101 r/m

Примечание: команда SHL совпадает с командой SAL.

Назначение. Биты в байтах, словах и двойных словах могут сдвигаться логически или арифметически. Сдвиг осуществляется на заданное число разрядов вплоть до 31. Число разрядов сдвига указывается в командах сдвига одним из трех способов: 1) в явной форме задается один сдвиг; 2) число сдвигов указывается в виде непосредственной величины; 3) число сдвигов заносится в регистр CL.

Команды сдвига предоставляют удобный способ умножения и деления на целые степени числа 2. Деление знаковых чисел путем сдвига вправо *не* эквивалентно делению, выполняемому командой IDIV.

Флаг CF всегда содержит значение последнего бита, выдвинутого из операнда-приемника. В случае одноразрядного сдвига флаг OF устанавливается, если значение старшего (знакового) бита изменилось в результате операции. Если знаковый бит не изменился, флаг OF сбрасывается. В случае многоразрядного сдвига содержимое OF всегда не определено.

Команда SAL сдвигает однобайтовый, однословный или двухсловный операнд-приемник влево на один разряд или на число разрядов, указанное в операнде-счетчике. Процессор по мере выдвижения битов с левой (старшей) стороны операнда вдвигает нули с правой (младшей) стороны.

Команда SAR сдвигает однобайтовый, однословный или двухсловный операнд-приемник вправо на один разряд или на число разрядов, указанное в операнде-счетчике. Процессор сохраняет знак операнда, вдвигая с левой (старшей) стороны операнда нули, если величина положительна, и единицы, если величина отрицательна.

Команда SHL совпадает с командой SAL.

Команда SHR сдвигает однобайтовый, однословный или двухсловный операнд-приемник вправо на один разряд или на число разрядов, указанное в операнде-счетчике. Процессор по мере выдвижения битов с правой (младшей) стороны операнда вдвигает нули с левой (старшей) стороны.

Пример выполнения команды. Показано действие команд арифметических и логических сдвигов.

```
14EB:0100   B88100   MOV   AX,0081
14EB:0103   F8       CLC
14EB:0104   B103     MOV   CL,03
14EB:0106   D0F8     SAR   AL,1
14EB:0108   D0F0     SAL   AL,1
14EB:010A   D0E8     SHR   AL,1
14EB:010C   D0E0     SHL   AL,1
14EB:010E   B081     MOV   AL,81
14EB:0110   D2F8     SAR   AL,CL
14EB:0112   D2F0     SAL   AL,CL
14EB:0114   D2E8     SHR   AL,CL
14EB:0116   D2E0     SHL   AL,CL
```

Регистры и флаги перед выполнением команды SAR:

```
AX=0081  BX=0000  CX=0003  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL ZR AC PO NC

14EB:0106   D0F8   SAR    AL,1

AX=00C0  BX=0000  CX=0003  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    NV UP EI NG NZ AC PE CY
                                          *  *     *  *
```

Арифметический сдвиг вправо распространяет вправо знаковый бит числа. Циклическая пересылка битов отсутствует. Операцию можно рассматривать как знаковое целочисленное деление на 2 без округления.

```
14EB:0108  D0F0    SAL     AL,1

AX=0080   BX=0000   CX=0003   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010A
                                    NV UP EI NG NZ AC PO CY
                                                        *
```

Арифметический сдвиг влево умножает сдвигаемую величину на 2. Флаг переноса устанавливается, поскольку выдвинутый слева бит равнялся 1.

```
14EB:010A D0E8      SHR   AL,1

AX=0040   BX=0000   CX=0003   DX=0000   SP=EA04   BP=0000

SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010C
                                    OV UP EI PL NZ AC PO NC
                                     *        *         *
```

Логический сдвиг вправо заполняет нулями знаковый разряд. Флаг OV устанавливается, так как старший бит первоначального значения в AL был равен 1. Младший бит равнялся 0 и был сдвинут в флаг переноса.

```
14EB:010C D0E0              SHL     AL,1

AX=0080   BX=0000   CX=0003   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010E
                                    OV UP EI NG NZ AC PO NC
                                              *
```

Логический сдвиг влево выполняет беззнаковое умножение на 2. Флаг знака установлен, потому что установился старший бит операнда. Флаг OV остался установленным, потому что старший бит результата отличается от бита переноса. Флаг OV отражает тот факт, что сдвиг влево изменил бит знака.

```
14EB:010E B081              MOV     AL,81

AX=0081   BX=0000   CX=0003   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0110
                                    OV UP EI NG NZ AC PO NC

14EB:0110 D2F8              SAR     AL,CL

AX=00F0   BX=0000   CX=0003   DX=0000   SP=EA04   BP=0000
```

```
SI=0000    DI=0000    DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0112
                                          NV UP EI NG NZ AC PE NC
                                          *                  *
```

Сдвиг на 3 разряда (над содержимым регистра CL и числом $1F_{16}$ выполняется операция И, чтобы ограничить число сдвигов максимальным значением 31) вызывает расширение знака в регистре AL. Флаг четности и другие арифметические флаги устанавливаются в соответствии с результатом операции. Флаг OV изменил свое состояние, но его значение при многоразрядных сдвигах не определено.

```
14EB:0112 D2F0      SAL   AL,CL

AX=0080   BX=0000   CX=0003   DX=0000   SP=EA04   BP=0000

SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0114
                                       NV UP EI NG NZ AC PO CY
                                                        *  *
```

Сдвиг влево на 3 разряда устанавливает флаг переноса, поскольку при последнем сдвиге из регистра AL был выдвинут бит, равный 1. Справа операнд заполняется нулями.

```
14EB:0114 D2E8              SHR     AL,CL

AX=0010   BX=0000   CX=0003   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0116
                                       NV UP EI PL NZ AC PO NC
                                                  *
```

Логический сдвиг вправо заполняет левые разряды нулями. Заметьте, что бит знака не имеет специального значения и является просто одним из сдвигаемых битов.

```
14EB:0116 D2E0              SHL     AL,CL

AX=0080   BX=0000   CX=0003   DX=0000   SP=EA04   BP=0000
SI=0000   DI=0000   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0118
                                       OV UP EI NG NZ AC PO NC
                                       *           *
```

Логический сдвиг влево снова вдвигает справа нули. Флаг переноса сбрасывается, потому что последний бит, выдвинутый из регистра, равнялся 0. Флаг OV изменился, но он не определен.

SBB Целочисленное вычитание с займом (Subtract Integers with Borrow)

Команда	Код операции	Двоичное представление
SBB AL,imm8	1C ib	00011100
SBB AX,imm16	1D iw	00011101
SBB EAX,imm32	1D id	00011101
SBB r/m8,imm8	80 /3 ib	10000000 mm 011 r/m
SBB r/m16,imm16	81 /3 iw	10000001 mm 011 r/m
SBB r/m32,imm32	81 /3 id	10000001 mm 011 r/m
SBB r/m16,imm8	83 /3 ib	10000011 mm 011 r/m
SBB r/m32,imm8	83 /3 ib	10000011 mm 011 r/m
SBB r/m8,r8	18 /r	00011000 mm rrr r/m
SBB r/m16,r16	19 /r	00011001 mm rrr r/m
SBB r/m32,r32	19 /r	00011001 mm rrr r/m
SBB r8,r/m8	1A /r	00011010 mm rrr r/m
SBB r16,r/m16	1B /r	00011011 mm rrr r/m
SBB r32,r/m32	1B /r	00011011 mm rrr r/m

Назначение. Команда SBB вычитает операнд-источник из операнда-приемника. Если флаг CF установлен, вычитается еще 1. Затем результат помещается по адресу приемника. Если флаг CF сброшен, команда SBB выполняет ту же операцию, что и SUB. Команда SUB, за которой следуют несколько команд SBB, может использоваться для вычитания чисел, содержащих более 32 разрядов.

Пример выполнения команды. Показано действие команды SBB.

Последовательность команд:

```
14EB:0100 F8          CLC
14EB:0101 B80100      MOV  AX,0001
14EB:0104 1D0100      SBB  AX,0001
14EB:0107 1D0100      SBB  AX,0001
14EB:010A 1D0100      SBB  AX,0001
```

Регистры и флаги перед выполнением команды SBB:

```
AX=0001  BX=0000  CX=0003  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI NG NZ NA PO NC

14EB:0104 1D0100          SBB   AX,0001

AX=0000  BX=0000  CX=0003  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL ZR NA PE NC
                                       *  *     *
```

После первого вычитания флаги устанавливаются в соответствии с результатом (1 – 1 = 0).

```
14EB:0107 1D0100              SBB     AX,0001

AX=FFFF  BX=0000  CX=0003  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                    NV UP EI NG NZ AC PE CY
                                       *  *  *     *
```

Это вычитание выполняет заем (как если бы имелся следующий старший разряд за пределами регистра). В результате устанавливается флаг переноса в дополнение к флагам знака и ненулевого значения (0 — 1 = — 1).

```
14EB:010A 1D0100              SBB     AX,0001

AX=FFFD  BX=0000  CX=0003  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010D
                                    NV UP EI NG NZ NA PO NC
                                       *  *  *
```

Последняя команда SBB вычитает из содержимого регистра непосредственное значение, но, кроме того, вычитает еще 1 — признак предыдущего займа, «запомненного» в флаге переноса (—1 — 1 — 1 = — 3).

SCAS Сравнение строковых данных (Compare String Data)
SCASB
SCASW
SCASD

Команда	Код операции	Двоичное представление
SCAS m8	AE	10101110
SCAS m16	AF	10101111
SCAS m32	AF	10101111

Примечание. Обозначение SCASB является обычной мнемоникой ассемблера для команды SCAS m8; обозначение SCASW – мнемоникой команды SCAS m16, а обозначение SCASD – мнемоникой команды SCAS m32.

Назначение. Эти команды используются для работы со строками, а не с логическими или арифметическими величинами. Они воздействуют на один элемент строки, который может быть байтом, словом или двойным словом. Адресация элементов строки осуществляется через регистры ESI и EDI. После каждой строковой операции регистры ESI и/или EDI автоматически получают положительное или отрицательное приращение и указывают на следующий элемент строки. Если DF = 0, в индексных регистрах выполняется инкремент, если DF = 1,– декремент. Величина приращения составляет 1, 2 или 4 в зависимости от размера элемента строки.

Команда SCAS вычитает элемент строки приемника по адресу ES:EDI из содержимого EAX, AX или AL и модифицирует флаги AF, SF, ZF, PF и OF. Если величины равны, ZF = 1. В противном случае ZF = 0. Если DF = 0, МП 80386 выполняет инкремент указателя строки (EDI). Регистр сегмента-приемника (ES) не может быть заменен.

Команда SCASB сканирует байты, SCASW – слова, SCASD – двойные слова.

Если перед этими командами используются префиксы REPE или REPNE, МП 80386 сравнивает значение текущего элемента строки с содержимым AL (для однобайтовых элементов), AX (для однословных) и EAX (для двухсловных). Завершение повторения операции зависит от результирующего состояния флага ZF, а также от значения в EAX.

Пример выполнения команды. Показан поиск с помощью команды SCAS конкретного байта в строке.

Последовательность команд:

```
14EB:0100 B90400          MOV    CX,0004
14EB:0103 BF0002          MOV    DI,0200
14EB:0106 B84300          MOV    AX,0043
14EB:0109 F2              REPNZ
14EB:010A AE              SCASB
```

Сканируемые данные:

```
14EB:0200 41 42 43 44
```

Регистры и флаги после выполнения первой команды:

```
AX=0043  BX=0000  CX=0004  DX=0000  SP=EA04  BP=0000
SI=0300  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PE NC
```

В качестве счетчика используется регистр CX.

```
14EB:0103 BF0002          MOV    DI,0200

AX=0043  BX=0000  CX=0004  DX=0000  SP=EA04  BP=0000
SI=0300  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL NZ NA PE NC
```

Регистр DI указывает на строку, подлежащую сканированию.

```
14EB:0106 B84300          MOV    AX,0043

AX=0043  BX=0000  CX=0004  DX=0000  SP=EA04  BP=0000
SI=0300  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0109
                                    NV UP EI PL NZ NA PE NC
```

Искомое значение находится в регистре AL. Для пропуска несовпадающих элементов использован префикс REPNZ. Если бы мы хотели пропускать совпадающие символы, например пробелы, следовало использовать префикс REPZ.

```
14EB:0109 F2              REPNZ
14EB:010A AE              SCASB
```

```
AX=0043  BX=0000  CX=0001  DX=0000  SP=EA04  BP=0000
SI=0300  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010B
                                    NV UP EI PL ZR NA PE NC
                                     *
```

Заметьте, что флаг нуля установлен. Это указывает на то, что в строке был обнаружен искомый элемент. Если бы совпадение обнаружилось в последнем байте, содержимое CX было бы 0. Регистр DI указывает на позицию, следующую за той, в которой обнаружено совпадение.

SETcc Установка байта по условию (Set Byte on Condition)

Команда	Код операции	Двоичное представление
SETA r/m8	0F 97	00001111 10010111 mm rrr r/m
SETAE r/m8	0F 93	00001111 10010011 mm rrr r/m
SETB r/m8	0F 92	00001111 10010010 mm rrr r/m
SETBE r/m8	0F 96	00001111 10010110 mm rrr r/m
SETC r/m8	0F 92	00001111 10010010 mm rrr r/m
SETE r/m8	0F 94	00001111 10010100 mm rrr r/m
SETG r/m8	0F 9F	00001111 10011111 mm rrr r/m
SETGE r/m8	0F 9D	00001111 10011101 mm rrr r/m
SETL r/m8	0F 9C	00001111 10011100 mm rrr r/m
SETLE r/m8	0F 9E	00001111 10011110 mm rrr r/m
SETNA r/m8	0F 96	00001111 10010110 mm rrr r/m
SETNAE r/m8	0F 92	00001111 10010010 mm rrr r/m
SETNB r/m8	0F 93	00001111 10010011 mm rrr r/m
SETNBE r/m8	0F 97	00001111 10010111 mm rrr r/m
SETNC r/m8	0F 93	00001111 10010011 mm rrr r/m
SETNE r/m8	0F 95	00001111 10010101 mm rrr r/m
SETNG r/m8	0F 9E	00001111 10011110 mm rrr r/m
SETNGE r/m8	0F 9C	00001111 10011100 mm rrr r/m
SETNL r/m8	0F 9D	00001111 10011101 mm rrr r/m
SETNLE r/m8	0F 9F	00001111 10011111 mm rrr r/m
SETNO r/m8	0F 91	00001111 10010001 mm rrr r/m
SETNP r/m8	0F 9B	00001111 10011011 mm rrr r/m
SETNS r/m8	0F 99	00001111 10011001 mm rrr r/m
SETNZ r/m8	0F 95	00001111 10010101 mm rrr r/m
SETO r/m8	0F 90	00001111 10010000 mm rrr r/m
SETP r/m8	0F 9A	00001111 10011010 mm rrr r/m
SETPE r/m8	0F 9A	00001111 10011010 mm rrr r/m
SETPO r/m8	0F 9B	00001111 10011011 mm rrr r/m
SETS r/m8	0F 98	00001111 10011000 mm rrr r/m
SETZ r/m8	0F 94	00001111 10010100 mm rrr r/m

Назначение. Команда SETcc записывает в байт 0 или 1 в зависимости от любого из 16 условий, определяемых флагами состояния. Байт может быть в памяти или в однобайтовом регистре общего назначения. Если условие cc истинно, команда SETcc записывает в байт 1; в противном случае – 0.

Пример выполнения команды. Показано действие команды SETcc.

Последовательность команды:

```
14EB:0100 0F95C0          SETNZ  AL
14EB:0103 0F94C0          SETZ   AL
14EB:0106 90              NOP
```

Регистры и флаги перед выполнением:

```
AX=0001  BX=0000  CX=0001  DX=0000  SP=EA04  BP=0000
SI=0300  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                    NV UP EI PL ZR NA PE NC

14EB:0100 0F95C0          SETNZ  AL

AX=0000  BX=0000  CX=0001  DX=0000  SP=EA04  BP=0000
SI=0300  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL ZR NA PE NC
```

Поскольку флаг нуля был установлен, команда SETcc помещает в AL 0, указывая тем самым, что проверяемое условие ложно.

```
14EB:0103 0F94C0          SETZ   AL

AX=0001  BX=0000  CX=0001  DX=0000  SP=EA04  BP=0000
SI=0300  DI=0203  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL ZR NA PE NC
```

В этом случае в регистр AL записывается 1, так как проверяемое условие истинно.

SGDT Запись в память содержимого регистра таблицы глобальных дескрипторов (Store Global Descriptor Table Register)

SIDT Запись в память содержимого регистра таблицы дескрипторов прерываний (Store Interrupt Descriptor Table Register)

Команда	Код операции	Двоичное представление
SGDT m	0F 01/0	00001111 00000001 mm 000 r/m
SIDT m	0F 01/1	00001111 00000001 mm 001 r/m

Назначение. Эти команды копируют содержимое регистра таблицы дескрипторов в поле из 6 байт, на которое указывает операнд. Команды SGDT/SIDT в 16-разрядной форме совместимы с МП 80286, но только если к старшим восьми битам не происходит обращения. МП 80286 записывает в эти биты 1, а МП 80386 – 0.

Пример 1 выполнения команды. Показана запись содержимого регистра таблицы глобальных дескрипторов.

```
14EB:0100 660F0105        SGDT   [DI]
14EB:0104 90              NOP
```

Данные по адресу ES:DI

```
14EB:0200  01 01 01 01 01 01 01 01...
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                    NV UP EI NG NZ NA PO NC
```

Обратите внимание на модификатор ширины операнда.

```
14EB:0100 660F0105          SGDT   [DI]

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI NG NZ NA PO NC
```

Результат выполнения (содержимое 48-разрядного регистра):

```
14EB:0200  FF FF 00 00 00 00 01 01 .
```

Пример 2 выполнения команды. Показано действие команды SIDT. Последовательность команд:

```
14EB:0100 660F010D          SIDT   [DI]
14EB:0104 90                NOP
```

Данные по адресу ES:DI

```
14EB:0200  01 01 01 01 01 01 01 01..
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                    NV UP EI NG NZ NA PO NC
```

Обратите внимание на префикс замены ширины операнда.

```
14EB:0100 660F010D          SIDT   [DI]

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI NG NZ NA PO NC
```

Результирующее 48-разрядное значение:

```
14EB:0200  FF FF 00 00 00 00 01 01 . . .
```

SHLD Сдвиг влево с двойной точностью (Double Precision Shift Left)
SHRD Сдвиг вправо с двойной точностью (Double Precision Shift Right)

Команда	Код операции	Двоичное представление
SHLD r/m16,r16,imm8	0F A4	00001111 10100100 mm rrr r/m
SHLD r/m32,r32,imm8	0F A4	00001111 10100100 mm rrr r/m
SHLD r/m16,r16,CL	0F A5	00001111 10100101 mm rrr r/m
SHLD r/m32,r32,CL	0F A5	00001111 10100101 mm rrr r/m
SHRD r/m16,r16,imm8	0F AC	00001111 10101100 mm rrr r/m
SHRD r/m32,r32,imm8	0F AC	00001111 10101100 mm rrr r/m
SHRD r/m16,r16,CL	0F AD	00001111 10101101 mm rrr r/m
SHRD r/m32,r32,CL	0F AD	00001111 10101101 mm rrr r/m

Назначение. Команды **SHLD** и **SHRD** предоставляют возможность реализации операций на длинных невыровненных строках битов. Двойной сдвиг либо 1) использует два однословных входных операнда и образует на выходе одно слово, либо 2) использует два двухсловных входных операнда и образует на выходе двойное слово.

Один из двух входных операндов может находиться как в регистре общего назначения, так и в памяти; второй операнд должен быть в регистре общего назначения. Результат помещается на место операнда в памяти или в регистре. Число разрядов сдвига указывается либо в регистре CL, либо в байте непосредственных данных команды. Флаг CF устанавливается в соответствии с последним битом, выдвинутым из операнда-приемника. Флаги SF, ZF и PF устанавливаются соответственно значению результата. Флаги OF и AF остаются неопределенными.

Пример 1 выполнения команды. Показано действие команды SHLD.

Последовательность команд:

```
14EB:0100 B80000          MOV   AX,0000
14EB:0103 BB3412          MOV   BX,1234
14EB:0106 0FA4D808        SHLD  AX,BX,8
14EB:010A 6650            PUSH  EAX
14EB:010C B10C            MOV   CL,0C
14EB:010E 0FA5D8          SHLD  AX,BX,CL
```

Регистры и флаги перед выполнением:

```
AX=2123  BX=1234  CX=000C  DX=5555  SP=EA00  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                    OV UP EI PL NZ AC PO CY

14EB:0100 B80000          MOV   AX,0000

AX=0000  BX=1234  CX=000C  DX=5555  SP=EA00  BP=0000
```

```
SI=0000    DI=0200    DS=14EB    ES=14EB    SS=14EB    CS=14EB
IP=0103
                                                OV UP EI PL NZ AC PO CY

14EB:0103 BB3412            MOV    BX,1234

AX=0000    BX=1234    CX=000C    DX=5555    SP=EA00    BP=0000
SI=0000    DI=0200    DS=14EB    ES=14EB    SS=14EB    CS=14EB
IP=0106
                                                OV UP EI PL NZ AC PO CY

14EB:0106 0FA4D808          SHLD   AX,BX,8

AX=0012    BX=1234    CX=000C    DX=5555    SP=EA00    BP=0000
SI=0000    DI=0200    DS=14EB    ES=14EB    SS=14EB    CS=14EB
IP=010A
                                                NV UP EI PL NZ AC PE NC
                                                    *     *
```

Команды двойного сдвига используют третий операнд в качестве счетчика сдвига. Первый операнд принимает биты, выдвинутые из копии второго операнда. Число сдвигов может быть непосредственным значением, как в данном примере, или содержаться в регистре CL. Заметьте, что второй операнд не изменился. Флаги знака, нуля и четности устанавливаются в соответствии с результатом операции, находящимся в первом операнде. Флаг переноса принимает значение последнего бита, выдвинутого из первого операнда. Флаги коррекции и переполнения не определены.

```
14EB:010A 6650              PUSH   EAX

AX=0012    BX=1234    CX=000C    DX=5555    SP=E9FC    BP=0000
SI=0000    DI=0200    DS=14EB    ES=14EB    SS=14EB    CS=14EB
IP=010C
                                                NV UP EI PL NZ AC PE NC
```

Мы сохранили содержимое регистра EAX в стеке, чтобы показать, что при ширине операнда 16 разрядов его верхняя половина не затрагивается.

```
14EB:010C B10C              MOV    CL,0C

AX=0012    BX=1234    CX=000C    DX=5555    SP=E9FC    BP=0000
SI=0000    DI=0200    DS=14EB    ES=14EB    SS=14EB    CS=14EB
IP=010E
                                                NV UP EI PL NZ AC PE NC

14EB:010E 0FA5D8            SHLD   AX,BX,CL

AX=2123    BX=1234    CX=000C    DX=5555    SP=E9FC    BP=0000
SI=0000    DI=0200    DS=14EB    ES=14EB    SS=14EB    CS=14EB
IP=0111
                                                OV UP EI PL NZ AC PO CY
                                                    *     *
```

В результате этого 12-разрядного сдвига возник перенос, потому что последний бит, выдвинутый из регистра AX, равен 1.

Содержимое EAX, сохраненное в стеке:

```
14EB:E9FC  12 00 DC 3D
```

Пример 2 выполнения команды. Показано действие команды SHRD. Последовательность команд:

```
14EB:0100 B80000          MOV   AX,0000
14EB:0103 BB3412          MOV   BX,1234
14EB:0106 0FACD808        SHRD  AX,BX,8
14EB:010A 6650            PUSH  EAX
14EB:010C B10C            MOV   CL,0C
14EB:010E 0FADD8          SHRD  AX,BX,CL
14EB:0111 90              NOP
```

Регистры и флаги перед выполнением:

```
AX=2123  BX=1234  CX=000C  DX=5555  SP=E9FC  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                    OV UP EI PL NZ AC PO CY

14EB:0100 B80000          MOV   AX,0000

AX=0000  BX=1234  CX=000C  DX=5555  SP=E9FC  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    OV UP EI PL NZ AC PO CY

14EB:0103 BB3412          MOV   BX,1234

AX=0000  BX=1234  CX=000C  DX=5555  SP=E9FC  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    OV UP EI PL NZ AC PO CY

14EB:0106 0FACD808        SHRD  AX,BX,8

AX=3400  BX=1234  CX=000C  DX=5555  SP=E9FC  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010A
                                    NV UP EI PL NZ AC PE NC
                                       *     *
```

Двойной сдвиг вправо на 8 разрядов вдвинул правые 8 бит копии второго операнда в первый операнд. Флаг переноса сбрасывается, отражая значение последнего бита, выдвинутого из первого операнда. Флаги знака, нуля и четности

устанавливаются соответственно результату операции в первом операнде. Флаги переполнения и коррекции не определены.

```
14EB:010A 6650              PUSH    EAX

AX=3400   BX=1234   CX=000C   DX=5555   SP=E9F8   BP=0000
SI=0000   DI=0200   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010C
                                          NV UP EI PL NZ AC PE NC

14EB:010C B10C              MOV     CL,0C

AX=3400   BX=1234   CX=000C   DX=5555   SP=E9F8   BP=0000
SI=0000   DI=0200   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=010E
                                          NV UP EI PL NZ AC PE NC

14EB:010E 0FADD8            SHRD    AX,BX,CL

AX=2343   BX=1234   CX=000C   DX=5555   SP=E9F8   BP=0000
SI=0000   DI=0200   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0111
                                          NV UP EI PL NZ AC PO NC
                                                              *
```

Завершен 12-разрядный сдвиг вправо копии BX в регистр AX. Состояние флагов указано выше.

Сохраненное в стеке содержимое EAX:

```
14EB:E9F8 00 34 DC 3D
```

SLDT Запись содержимого регистра таблицы локальных дескрипторов (Store Local Descriptor Table Register)

Команда	Код операции	Двоичное представление
SLDT r/m16	0F 00/0	00001111 00000000 mm 000 r/m

Назначение. В регистре таблицы локальных дескрипторов содержится селектор, указывающий на LDT. Команда SLDT записывает содержимое LDTR в регистр или ячейку памяти с исполнительным адресом, указанным в операнде команды.

Примечание: атрибут размера операнда не отражается на выполнении команды SLDT.

Пример выполнения команды. Команда пересылает содержимое регистра-селектора таблицы локальных дескрипторов в память или 16-разрядный регистр. Флаги остаются без изменения.

Последовательность команд:

```
0070:0100 0F00C0        SLDT   AX
0070:0103 90            NOP
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0000  CX=0464  DX=0000  SP=0200  BP=0000
SI=0000  DI=0000  DS=0078  ES=0000  SS=0080  CS=0070
IP=0100
                                    NV UP EI PL NZ NA PO NC
```

Пересылаем селектор LDT в регистр AX.

```
0070:0100 0F00C0          SLDT   AX

AX=0010  BX=0000  CX=0464  DX=0000  SP=0200  BP=0000
SI=0000  DI=0000  DS=0078  ES=0000  SS=0080  CS=0070
IP=0103
                                    NV UP EI PL NZ NA PO NC
```

Теперь в регистре AX значение селектора.

```
0070:0103 90      NOP
```

SMSW Запись слова состояния машины (Store Machine Status Word)

Команда	Код операции	Двоичное представление
SMSW r/m16	0F 01/4	00001111 00000001 mm 100 r/m

Назначение. Слово состояния машины является частью регистра управления 0 (CR0). Команда SMSW записывает это слово в двухбайтовый регистр или ячейку памяти с исполнительным адресом, указанным в операнде команды. Команда SMSW обеспечивает совместимость с МП 80286. В программах для МП 80386 следует использовать команду MOV...CR0.

Пример выполнения команды. Показана запись слова состояния машины в регистр.

Последовательность команд:

```
14EB:0101 0F01E0        SMSW   AX
14EB:0104 90            NOP
```

Регистры и флаги перед выполнением:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL NZ NA PO NC

14EB:0101 0F01E0          SMSW   AX

AX=FFE0  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0104
                                    NV UP EI PL NZ NA PO NC
```

Часть содержимого CR0, представляющая слово состояния машины, записана по адресу назначения.

STC Установка флага переноса (Set Carry Flag CF)

Команда	Код операции	Двоичное представление
STC	F9	11111001

Назначение. Команда STC устанавливает флаг CF в 1.

Пример выполнения команды. Показано действие команды STC.

Последовательность команд:

```
14EB:0100 F8          CLC
14EB:0101 F9          STC
14EB:0102 90          NOP
```

Регистры и флаги перед выполнением команды STC:

```
AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL NZ NA PO NC
                                                        *
```

Заметьте, что флаг переноса сбросился

```
14EB:0101 F9          STC

AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102
                                    NV UP EI PL NZ NA PO CY
                                                        *
```

В результате выполнения команды STC флаг переноса установлен в 1.

STD Установка флага направления (Set Direction Flag DF)

Команда	Код операции	Двоичное представление
STD	FD	11111101

Назначение. Команда STD устанавливает флаг DF в 1.

В результате во всех последующих строковых операциях выполняется декремент индексных регистров SI (или ESI), DI (или EDI).

Пример выполнения команды. Показано действие команды STD.

Последовательность команд:

```
14EB:0100 FC          CLD
14EB:0101 FD          STD
```

Регистры и флаги перед выполнением команды STD:

```
AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL NZ NA PO NC
                                     *
```

Заметьте, что флаг направления установлен «вверх» (нуль).

```
14EB:0101 FD        STD

AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102
                                    NV DN EI PL NZ NA PO NC
                                       *
```

Теперь флаг направления установлен «вниз» (единица), регистр IP получил приращение, а состояние остальных флагов не изменилось.

STI Установка флага прерываний (Set Interrupt Flag)

Команда	Код операции	Двоичное представление
STI	FB	11111011

Назначение. Команда STI устанавливает флаг IF в 1. После выполнения следующей команды МП 80386 отзывается на внешние прерывания, если следующая команда оставляет установленным флаг разрешения прерываний. Если внешние прерывания запрещены и в программе встретились строки STI, RET (например, в конце подпрограммы), то сначала выполнится команда RET и лишь затем будут распознаваться внешние прерывания. Аналогично, если внешние прерывания запрещены и в программе встретились строки STI, CLI, прерывания *не* распознаются, потому что команда CLI сбрасывает в процессе своего выполнения флаг прерываний.

Пример выполнения команды. Показано действие команды STI.

Последовательность команд:

```
14EB:0100 FA        CLI
14EB:0101 FB        STI
14EB:0102 FA        CLI
```

Регистры и флаги перед выполнением команды STI:

```
AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP DI PL NZ NA PO NC
                                     *
```

```
Заметьте, что флаг прерываний сбросился (прерывания запрещены)

14EB:0101 FB                STI

AX=FFF1  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0102
                                    NV UP EI PL NZ NA PO NC
                                    *
```

Флаг прерываний установлен (прерывания разрешены), и регистр IP получил приращение. Содержимое остальных регистров и флагов не изменилось.

STOS Запись строки данных (Store String Byte)
STOSB Запись байта (Store Byte)
STOSW Запись слова (Store Word)
STOSD Запись двойного слова (Store Doubleword)

Команда	Код операции	Двоичное представление
STOS m8	AA	10101010
STOS m16	AB	10101011
STOS m32	AB	10101011

Назначение. Эти команды используются для работы со строками, а не с логическими или арифметическими величинами. Они воздействуют на один элемент строки, который может быть байтом, словом или двойным словом. Адресация элементов строки осуществляется через регистры ESI и EDI. После каждой строковой операции регистры ESI и/или EDI автоматически получают положительное или отрицательное приращение и указывают на следующий элемент строки. Если DF = 0, в индексных регистрах выполняется инкремент, если DF = 1, – декремент. Величина приращения составляет 1, 2 или 4 в зависимости от размера элемента строки[1)].

Пример выполнения команды. Показано, как команда STOS используется для инициализации области памяти (занесения заданных значений).

Последовательность команд:

```
14EB:0100 BF0002        MOV   DI,0200
14EB:0103 B8FF00        MOV   AX,00FF
14EB:0106 B90800        MOV   CX,0008
14EB:0109 F3            REP
14EB:010A AB            STOSW
```

Начальное содержимое памяти:

```
14EB:0200  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

[1)] Команда STOS записывает в память по адресу, определяемому с помощью регистра EDI, содержимое регистра AL (для байтовых строк), AX (для однословных строк) или EAX (для двухсловных строк). – *Прим. перев.*

Регистры и флаги перед выполнением:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0000  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                    NV UP EI NG NZ NA PO NC

14EB:0100 BF0002          MOV   DI,0200

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI NG NZ NA PO NC

14EB:0103 B8FF00          MOV   AX,00FF

AX=00FF  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI NG NZ NA PO NC

14EB:0106 B90800          MOV   CX,0008

AX=00FF  BX=0000  CX=0008  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0200  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0109
                                    NV UP EI NG NZ NA PO NC
```

Префикс REP приводит к многократному повторению процесса записи в память до тех пор, пока не будет записано число элементов, определенное в регистре CX. В данном случае ширина операнда по умолчанию составляет 16 разрядов, поэтому содержимое регистра AX копируется в 8 последовательно расположенных однословных ячеек памяти.

```
14EB:0109 F3              REP
14EB:010A AB              STOSW

AX=00FF  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010B
                                    NV UP EI NG NZ NA PO NC
```

Содержимое памяти после выполнения команды:

```
14EB:0200  FF 00 FF 00 FF 00 FF 00-FF 00 FF 00 FF 00 FF 00
................
```

STR Запись регистра задачи (Store Task Register)

Команда	Код операции	Двоичное представление
STR r/m16	0F 00/1	00001111 00000000 mm 001 r/m

Назначение. Команда STR копирует содержимое регистра задачи в двухбайтовый регистр или ячейку памяти с исполнительным адресом, указанным в операнде команды. Атрибут размера операнда не оказывает влияния на выполнение команды STR.

Пример выполнения команды. Команда пересылает содержимое регистра задачи в память или 16-разрядный регистр. Состояние флагов не изменяется. Атрибут размера операнда не оказывает влияния на выполнение команды STR.

```
0070:0100 0F00C8        STR   AX
0070:0103 90            NOP
```

Регистры и флаги перед выполнением:

```
AX=0000   BX=0000   CX=0464   DX=0000   SP=0200   BP=0000
SI=0000   DI=0000   DS=0078   ES=0000   SS=0080   CS=0070
IP=0100
                                        NV UP EI PL NZ NA PO NC
```

Пересылаем селектор TSS в регистр AX.

```
0070:0100 0F00C8              STR     AX

AX=0040   BX=0000   CX=0464   DX=0000   SP=0200   BP=0000
SI=0000   DI=0000   DS=0078   ES=0000   SS=0080   CS=0070
IP=0103
                                        NV UP EI PL NZ NA PO NC
```

Заметьте, что значение селектора загружено в регистр AX.

```
0070:0103 90        NOP
```

SUB Вычитание целых чисел (Subtract Integers)

Команда	Код операции	Двоичное представление
SUB AL,imm8	2C ib	00101100
SUB AX,imm16	2D iw	00101101
SUB EAX,imm32	2D id	00101101
SUB r/m8,imm8	80 /5 ib	10000000 mm 101 r/m
SUB r/m16,imm16	81 /5 iw	10000001 mm 101 r/m
SUB r/m32,imm32	81 /5 id	10000001 mm 101 r/m
SUB r/m16,imm8	83 /5 ib	10000011 mm 101 r/m
SUB r/m32,imm8	83 /5 ib	10000011 mm 101 r/m
SUB r/m8,r8	28 /r	00101000 mm rrr r/m
SUB r/m16,r16	29 /r	00101001 mm rrr r/m
SUB r/m32,r32	29 /r	00101001 mm rrr r/m
SUB r8,r/m8	2A /r	00101010 mm rrr r/m
SUB r16,r/m16	2B /r	00101011 mm rrr r/m
SUB r32,r/m32	2B /r	00101011 mm rrr r/m

Назначение. Команда SUB вычитает операнд-источник из операнда-приемника и помещает результат на место операнда-приемника. Если потребовался заем, устанавливается флаг CF. Операнды могут располагаться в байтах, словах или двойных словах и быть как знаковыми, так и беззнаковыми.

Пример выполнения команды. Показано действие команды целочисленного вычитания. Команда SUB так же действует по отношению к команде SBB, как ADD по отношению к ADC. Она игнорирует значение займа.

Последовательность команд:

```
14EB:0100 B80100          MOV    AX,0001
14EB:0103 89C3            MOV    BX,AX
14EB:0105 29D8            SUB    AX,BX
14EB:0107 29D8            SUB    AX,BX
14EB:0109 29D8            SUB    AX,BX
14EB:010B 29D8            SUB    AX,BX
```

Регистры и флаги перед выполнением:

```
AX=FFFF  BX=0001  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                    NV UP EI NG NZ AC PE CY

14EB:0100 B80100          MOV    AX,0001

AX=0001  BX=0001  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI NG NZ AC PE CY

14EB:0103 89C3            MOV    BX,AX

AX=0001  BX=0001  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0105
                                    NV UP EI NG NZ AC PE CY

    Начинаем с 1 в AX и BX

14EB:0105 29D8            SUB    AX,BX

AX=0000  BX=0001  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL ZR NA PE NC
                                     *  *  *     *
```

После первого вычитания в регистре AX 0. Флаги устанавливаются соответственно.

```
14EB:0107 29D8                SUB     AX,BX

AX=FFFF  BX=0001  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0109
                                  NV UP EI NG NZ AC PE CY
                                        *  *  *     *
```

Результат —1. Флаг переноса установлен, потому что потребовался заем. Остальные флаги установлены в соответствии с результатом. Флаг коррекции AC установлен, потому что при вычитании произведен заем из старших четырех бит AL в младшие 4 бита.

```
14EB:0109 29D8                SUB     AX,BX

AX=FFFE  BX=0001  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010B
                                  NV UP EI NG NZ NA PO NC
                                        *  *  *
```

Результат —2. Флаг переноса в операции не участвовал. Заем не потребовался, и флаг переноса был сброшен.

TEST Логическое сравнение (Logical Compare)

Команда	Код операции	Двоичное представление
TEST AL,imm8	A8 ib	10101000
TEST AX,imm16	A9 iw	10101001
TEST EAX,imm32	A9 id	10101001
TEST r/m8,imm8	F6 /0 ib	11110110 mm 000 r/m
TEST r/m16,imm16	F7 /0 iw	11110111 mm 000 r/m
TEST r/m32,imm32	F7 /0 id	11110111 mm 000 r/m
TEST r/m8,r8	84 /r	10000100 mm rrr r/m
TEST r/m16,r16	85 /r	10000101 mm rrr r/m
TEST r/m32,r32	85 /r	10000101 mm rrr r/m

Назначение. Команда TEST выполняет логическую операцию И над двумя операндами. Затем команда сбрасывает флаги OF и CF, оставляет AF неопределенным и модифицирует SF, ZF и PF. Состояние флагов можно затем проанализировать с помощью команд условного перехода или командами установки байта по условию.

Отличие команды TEST от команды AND заключается в том, что TEST *не* изменяет операнд-приемник. Отличие от команды BT (проверка бита) заключается в том, что команда TEST за одно действие анализирует значение нескольких битов, а BT – только одного.

Пример выполнения команды. Показано действие команды TEST.

Последовательность команд:

```
14EB:0100 B80F0F          MOV    AX,0F0F
14EB:0103 BE0002          MOV    SI,0200
14EB:0106 29DB            SUB    BX,BX
14EB:0108 A90F00          TEST   AX,000F
14EB:010B 8504            TEST   [SI],AX
14EB:010D F7C71000        TEST   DI,0010
```

Данные по адресу в SI:

```
14EB:0200  F0 F0
```

Регистры и флаги перед выполнением:

```
AX=0F0F  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0200  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PO NC

14EB:0103 BE0002          MOV    SI,0200

AX=0F0F  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0200  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106
                                    NV UP EI PL NZ NA PO NC

14EB:0106 29DB            SUB    BX,BX

AX=0F0F  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0200  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0108
                                    NV UP EI PL ZR NA PE NC
                                          *        *

14EB:0108 A90F00          TEST   AX,000F

AX=0F0F  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0200  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010B
                                    NV UP EI PL NZ NA PE NC
                                                *
```

Команда TEST выполняет операцию И над непосредственным значением 000F и величиной в регистре AX (0F0F), в результате чего получается число 000F, отличное от нуля. Флаги устанавливаются соответственно этому временному результату, который затем отбрасывается. Результат операции положителен, не нуль и характеризуется четностью в младшем байте. В любом случае флаги переноса и переполнения сбрасываются.

```
14EB:010B 8504              TEST    [SI],AX
DS:0200=F0F0

AX=0F0F  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0200  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010D
                                    NV UP EI PL ZR NA PE NC
                                       *
```

Теперь команда TEST выполняет операцию И над величиной по адресу [SI] (F0F0) и значением в регистре AX (0F0F) с получением нулевого результата. Флаги устанавливаются соответственно.

```
14EB:010D F7C71000          TEST    DI,0010

AX=0F0F  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0200  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0111
                                    NV UP EI PL NZ NA PO NC
                                          *     *
```

Эта команда TEST выполняет операцию И над величиной в регистре DI (0210) и непосредственным значением (0010) с получением числа 0010. Флаги устанавливаются соответственно временному результату.

VERR Проверка сегмента на чтение (Verify a Segment for Reading)
VERW Проверка сегмента на запись (Verify a Segment for Writing)

Команда	Код операции	Двоичное представление
VERR r/m16	0F 00/4	00001111 00000000 mm 100 r/m
VERW r/m16	0F 00/5	00001111 00000000 mm 101 r/m

Назначение. Эти команды проверяют, доступен ли выбранный селектором сегмент при текущем уровне привилегий и разрешены ли в нем чтение или запись. Если сегмент доступен, флаг ZF устанавливается в 1. Если нет, ZF сбрасывается в 0. Проверка дает тот же результат, что и загрузка сегмента в DS, ES, FS или GS и выполнение затребованной операции.

Поскольку результат проверки направляется в ZF, значение селектора не возбуждает исключение – ошибку защиты. Это дает возможность программе заранее обнаруживать ошибки доступа к сегментам.

Пример выполнения команд. Эти команды проверяют, может ли данная программа использовать дескриптор, соответствующий указанному селектору, для чтения или записи данных. Если проверяемый доступ разрешен, устанавливается флаг нуля, в противном случае флаг нуля сбрасывается.

```
0070:0100 B87000            MOV     AX,0070
0070:0103 09C0              OR      AX,AX
0070:0105 0F00E0            VERR    AX
0070:0108 0F00E8            VERW    AX
```

```
AX=0000  BX=0000  CX=0464  DX=0000  SP=0200  BP=0000
SI=0000  DI=0000  DS=0078  ES=0000  SS=0080  CS=0070
IP=0100
                                    NV UP EI PL NZ NA PO NC
```

Загружаем селектор в регистр AX с целью проверки (селектор мог также находиться в памяти).

```
0070:0100 B87000          MOV   AX,0070

AX=0070  BX=0000  CX=0464  DX=0000  SP=0200  BP=0000
SI=0000  DI=0000  DS=0078  ES=0000  SS=0080  CS=0070
IP=0103
                                    NV UP EI PL NZ NA PO NC

0070:0103 09C0            OR    AX,AX
```

Команда OR обеспечивает сброс флага знака.

```
AX=0070  BX=0000  CX=0464  DX=0000  SP=0200  BP=0000
SI=0000  DI=0000  DS=0078  ES=0000  SS=0080  CS=0070
IP=0105
                                    NV UP EI PL NZ NA PO NC
```

Используем команду VERR для проверки прав доступа к сегменту с целью чтения.

```
0070:0105 0F00E0          VERR  AX

AX=0070  BX=0000  CX=0464  DX=0000  SP=0200  BP=0000
SI=0000  DI=0000  DS=0078  ES=0000  SS=0080  CS=0070
IP=0108
                                    NV UP EI PL ZR NA PO NC
                                                *
```

Установка флага нуля в результате выполнения команды говорит о том, что нам разрешено чтение сегмента (сегмента нашей программы). Теперь используем команду VERW для проверки наших прав записи в сегмент.

```
0070:0108 0F00E8 VERW  AX

AX=0070  BX=0000  CX=0464  DX=0000  SP=0200  BP=0000
SI=0000  DI=0000  DS=0078  ES=0000  SS=0080  CS=0070
IP=0111
                                    NV UP EI PL NZ NA PO NC
                                                *
```

Сброс флага нуля в результате выполнения команды показывает, что нам не разрешена запись в сегмент (сегмент нашей программы).
xx (флаг доступа)
Дескриптор: 0070 = FF FF 00 AB 01 9B 0F 00

Флаг доступа указывает, что сегмент присутствует, является сегментом программы, доступен для чтения, несогласованный, с разрешением доступа.

WAIT Ожидание пассивного (высокого) состояния вывода BUSY# (Wait until BUSY # Pin is Inactive (High))

Команда	Код операции	Двоичное представление
WAIT	9B	10011011

Назначение. Арифметический сопроцессор предоставляет расширение системы команд МП 80386. Сопроцессор поддерживает вычисления повышенной точности над целыми числами и числами с плавающей точкой и к тому же содержит набор полезных констант, ускоряющих арифметические вычисления. Сопроцессор выполняет операции параллельно с ЦП, обеспечивая максимальную производительность системы.

Команды сопроцессора включаются в единый поток команд программы вместе с командами МП 80386. Система выполняет команды сопроцессора в том порядке, в каком они появляются в потоке команд.

Команда WAIT приостанавливает выполнение программы МП 80386 до тех пор, пока ЦП 80386 не обнаружит пассивное состояние вывода BUSY. Это указывает на то, что сопроцессор завершил выполнение задачи и что ЦП может получить результат.

Пример выполнения команды. Действие команды показать нельзя.

XCHG Обмен между регистрами или между памятью и регистром (Exchange Register/Memory with Register)

Команда	Код операции	Двоичное представление
XCHG AX,r16	90 + r	10010rrr
XCHG r16,AX	90 + r	10010rrr
XCHG EAX,r32	90 + r	10010rrr
XCHG r32,EAX	90 + r	10010rrr
XCHG r/m8,r8	86 /r	10000110 mm rrr r/m
XCHG r8,r/m8	86 /r	10000110 mm rrr r/m
XCHG r/m16,r16	87 /r	10000111 mm rrr r/m
XCHG r16,r/m16	87 /r	10000111 mm rrr r/m
XCHG r/m32,r32	87 /r	10000111 mm rrr r/m
XCHG r32,r/m32	87 /r	10000111 mm rrr r/m

Заметьте, что команда «XCHG AX, AX» эквивалентна команде NOP.

Назначение. Команда XCHG выполняет обмен содержимым двух операндов и заменяет три команды MOV. Команда не нуждается во вспомогательной ячейке для сохранения одного из операндов при загрузке другого. Команда XCHG полезна при реализации семафоров или подобных структур данных, используемых для синхронизации процессов.

Пример выполнения команды. Показано действие команды обмена.

Последовательность команд:

```
14EB:0100 93          XCHG   AX,BX
14EB:0101 87F0        XCHG   SI,AX
14EB:0103 871E0002    XCHG   BX,[0200]
14EB:0107 871E0002    XCHG   BX,[0200]
```

Регистры и флаги перед выполнением:

```
AX=0200  BX=0F0F  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100
                                    NV UP EI PL NZ NA PO NC

14EB:0100 93                XCHG   AX,BX

AX=0F0F  BX=0200  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0000  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0101
                                    NV UP EI PL NZ NA PO NC
```

Значения в регистрах AX и BX поменялись местами.

```
14EB:0101 87F0              XCHG   SI,AX

AX=0000  BX=0200  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0F0F  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103
                                    NV UP EI PL NZ NA PO NC
```

Значения в регистрах SI и AX поменялись местами.

```
14EB:0103 871E0002          XCHG   BX,[0200]
DS:0200=0000

AX=0000  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0F0F  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0107
                                    NV UP EI PL NZ NA PO NC
```

Значение в регистре BX поменялось местами со значением по адресу DS:0200.

```
14EB:0107 871E0002          XCHG   BX,[0200]
DS:0200=0200
AX=0000  BX=0200  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0F0F  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=010B
                                    NV UP EI PL NZ NA PO NC
```

Снова выполняется обмен и значение в BX восстановлено. Заметьте, что команда обмена не влияет на состояние флагов.

XLAT Табличное перекодирование (Table Look-up Translatation)

Команда	Код операции	Двоичное представление
XLAT m8	D7	11010111

Назначение. Команду XLAT удобно использовать при преобразовании из одной системы кодов в другую, например из EBCDIC в ASCII. Таблица перекодирования может иметь длину до 256 байт. Значение, помещенное в регистр AL, служит индексом для нахождения соответствующего перекодированного значения.

Команда XLAT замещает байт AL байтом из таблицы перекодирования, созданной пользователем. При выполнении команды XLAT в регистре AL должно находиться беззнаковое значение индекса таблицы, адресуемой через регистр EBX. Команда XLAT заменяет в регистре AL индекс таблицы на элемент таблицы. Содержимое регистра EBX не изменяется.

Пример выполнения команды. Показано действие команды XLAT.

Последовательность команд:

```
14EB:0100 BB0002          MOV   BX,0200
14EB:0103 B80500          MOV   AX,0005
14EB:0106 D7              XLAT
14EB:0107 D7              XLAT
```

Данные по адресу DS:0200

```
14EB:0200  FF FF FF 0F 04 03 02 01-00 00 FF 00 FF 00 FF 00  ................
```

Регистры и флаги перед выполнением:

```
AX=00FF  BX=0000  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0F0F  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0100                                  NV UP EI PL NZ NA PO NC
```

В регистр BX следует загрузить смещение к адресу нашей таблицы перекодирования.

```
14EB:0100 BB0002          MOV   BX,0200

AX=00FF  BX=0200  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0F0F  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0103                                  NV UP EI PL NZ NA PO NC
```

Перекодируемый байт должен быть в регистре AL.

```
14EB:0103 B80500          MOV   AX,0005

AX=0005  BX=0200  CX=0000  DX=0000  SP=EA04  BP=0000
SI=0F0F  DI=0210  DS=14EB  ES=14EB  SS=14EB  CS=14EB
IP=0106                                  NV UP EI PL NZ NA PO NC

14EB:0106 D7              XLAT
```

```
AX=0003   BX=0200   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0F0F   DI=0210   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0107
                                        NV UP EI PL NZ NA PO NC
```

Команда XLAT заменяет байт в регистре AL байтом, извлеченным из таблицы по адресу DS:BX + AL. В данном случае адрес составляет 205, а само значение —03.

```
14EB:0107 D7              XLAT

AX=000F   BX=0200   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0F0F   DI=0210   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0108
                                        NV UP EI PL NZ NA PO NC
```

Теперь команда XLAT определяет, что адрес извлекаемого байта составляет 203. Состояние флагов не изменяется.

XOR Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ (Logical, Exclusive OR)

Команда	Код операции	Двоичное представление
XOR AL,imm8	34 ib	00110100
XOR AX,imm16	35 iw	00110101
XOR EAX,imm32	35 id	00110101
XOR r/m8,imm8	80 /6 ib	10000000 mm 110 r/m
XOR r/m16,imm16	81 /6 iw	10000001 mm 110 r/m
XOR r/m32,imm32	81 /6 id	10000001 mm 110 r/m
XOR r/m16,imm8	83 /6 ib	10000011 mm 110 r/m
XOR r/m32,imm8	83 /6 ib	10000011 mm 110 r/m
XOR r/m8,r8	30 /r	00110000 mm rrr r/m
XOR r/m16,r16	31 /r	00110001 mm rrr r/m
XOR r/m32,r32	31 /r	00110001 mm rrr r/m
XOR r8,r/m8	32 /r	00110010 mm rrr r/m
XOR r16,r/m16	33 /r	00110011 mm rrr r/m
XOR r32,r/m32	33 /r	00110011 mm rrr r/m

Назначение. Команда XOR сравнивает биты в двух своих операндах. Каждый из битов результата устанавливается в 1, если соответствующие биты операндов *различны*. Каждый бит сбрасывается в 0, если соответствующие биты одинаковы. Результат помещается на место первого операнда.

```
Операнд 1      110001
Операнд 2      001101
Результат      111100
```

Пример выполнения команды. Показано действие команды XOR. Эта команда вычисляет ИСКЛЮЧАЮЩЕЕ ИЛИ двух операндов. Результат находится путем побитовой операции, выполняемой в соответствии с формулой R = (A ИЛИ B) И НЕ (A И B).

Последовательность команд:

```
14EB:0100 B8FF0F          MOV   AX,0FFF
14EB:0103 350FF0          XOR   AX,F00F
```

Регистры и флаги перед выполнением:

```
AX=000F   BX=0200   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0F0F   DI=0210   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0100
                                        NV UP EI PL NZ NA PO NC

14EB:0100 B8FF0F          MOV   AX,0FFF

AX=0FFF   BX=0200   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0F0F   DI=0210   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0103
                                        NV UP EI PL NZ NA PO NC

14EB:0103 350FF0          XOR   AX,F00F

AX=FFF0   BX=0200   CX=0000   DX=0000   SP=EA04   BP=0000
SI=0F0F   DI=0210   DS=14EB   ES=14EB   SS=14EB   CS=14EB
IP=0106
                                        NV UP EI NG NZ NA PE NC
                                         .      *  *      *
```

Команда XOR помещает результат вычисления ИСКЛЮЧАЮЩЕГО ИЛИ по адресу приемника. Флаги знака, нуля и четности устанавливаются соответственно полученному результату. Флаги переноса и переполнения сбрасываются. Флаг коррекции не определен.

Глава 11

Введение в интегральные схемы

Очередным результатом успешного развития электронной промышленности явился выпуск 32-разрядных микросхем, позволивших ввести персональные компьютеры в мир виртуальных систем. Следует отметить, что краеугольным камнем индустрии электронных средств обработки данных, как никакой другой области промышленности, является принцип «будущее покоится на прошлом». Чтобы понять, в каком направлении движется эта техника, посмотрим сначала, как развивались электроника и технология интегральных схем.

Микропроцессор становится законченным изделием – микрокомпьютером – лишь после соединения с целым рядом дополнительных компонентов. При выполнении этих соединений следует учитывать природу и временны́е характеристики сигналов, возбуждаемых и обрабатываемых каждым компонентом. Сигналы можно разбить на три класса: данных, адресов и управления. Очевидно, что эти сигналы должны передаваться от одного компонента другому. Этот процесс носит название *сопряжения*. Обзор процессов сопряжения дан в гл. 12.

Информацию можно хранить с помощью разных средств; представим себе память в виде канала, где информация хранится для дальнейшего использования. В одной интегральной схеме (ИС) могут содержаться тысячи одноразрядных регистров с общими системами входов и выходов и единой синхронизующей цепью. Такая схема носит название микросхемы *памяти*. Гл. 13 посвящена организации памяти и общим вопросам ее управления, а гл. 14 – особенностям работы с памятью процессора 80386.

Скорость работы компьютера определяется тремя основными ограничениями. Одно связано с аппаратной организацией: насколько быстро переключаются вентили и какие расстояния по проводам или на плате должен пройти сигнал. Второе ограничение определяется логикой взаимодействия узлов машины, заложенной в ее организацию. Третье является программным и зависит от эффективности использования аппаратных средств операционной системой. В гл. 14 рассмотрены две последние проблемы.

Чтобы лучше понять отличия последнего поколения микросхем Intel, просмотрите приложение 1, где обсуждаются предыдущие микросхемы Intel того же семейства. Это поможет вам разобраться в том, какие проектные направления и решения повлияли на принципы разработки нового микропроцессора 80386.

11.1. Начало

Бурное развитие электроники стало возможным после первых исследований явления, открытого Томасом Эдисоном. Расположив в вакуумной электрической лампе изолированную металлическую пластинку, Эдисон обнаружил, что при прикосновении к проводу, идущему от этой пластинки, ощущается электрический

удар. Удар этот объясняется тем, что металлическая пластинка собирала электроны, испускаемые нитью накала лампы. С открытия Эдисона началось развитие электровакуумной техники.

В 1904 г. Дж. А. Флемминг развил идеи Эдисона и разработал двухэлектродную вакуумную лампу – диод. Эти лампы имели длину около 6 дюйм (1 дюйм = 2,54 см) и диаметр 1,5 дюйм. Несмотря на высокую стоимость и низкую эффективность, стеклянные вакуумные лампы использовались в первых радиоприемниках, с которых начался век радио.

В 1948 г. Дж. Бардин и У. Х. Бретейн из Bell Telephone Laboratories изобрели транзистор. Само слово *транзистор* произошло от **tran**sfer re**sistor** (передаваемое сопротивление). Первый транзистор получил жизнь, когда две острые металлические иглы были приведены в контакт с поверхностью куска германия. Если на одну из этих игл подавалось прямое смещение относительно основания (*базы*) – пластины германия *n*-типа, в пластину инжектировались дырки. Соответствующий вывод транзистора стал именоваться *базой*. На вторую иглу, расположенную рядом с первой, подавалось смещение, обратное относительно смещения базы. Она собирала значительную часть инжектированных дырок. В этом и заключается действие транзистора.

Описанная конструкция с иглами была слишком хрупкой для массового промышленного производства и использования. В 1949 г. У. Шокли также из Bell Laboratories предложил идею полностью твердой структуры транзисторного перехода. Этим эпохальным открытием и началась в 50-х годах эра транзисторов.

Транзистор отличается малыми размерами и экономным потреблением энергии. Источник носителей тока не требует нагрева. К тому же некогда требовавшееся для управления вакуумными лампами высокое напряжение оказалось теперь ненужным. Из-за исключительно малых расстояний между отдельными областями транзистора, т. е. малых длин перемещения носителей, можно использовать относительно небольшие напряжения. Все более плотное размещение элементов должно привести к дальнейшему снижению существующего сейчас стандарта 5 В на питание логических микросхем. Даже 5 В представляется слишком большим напряжением для миниатюрных транзисторов в разрабатываемых сверхбольших интегральных схемах (СБИС).

В качестве исторической справки: для демонстрации низкого потребления энергии первыми транзисторами простейшая транзисторная схема питалась с помощью пары проводов, воткнутых в лимон. Электрической энергии, развиваемой в результате электролиза, было достаточно для работы схемы.

В первых, точечных транзисторах, для образования двух переходов в полупроводниковую пластину вваривались два тонких провода. Такая конструкция имела хорошие характеристики на высоких частотах, но могла рассеивать лишь незначительную мощность.

Затем наступила пора практического внедрения транзисторов. Первый электронный калькулятор был создан в 1963 г. английской фирмой Bell Punch Company. Он состоял из дискретных транзисторов и занимал объем кассового аппарата. Четырьмя годами позже фирма Texas Instruments вышла на рынок со своим первым калькулятором на интегральных схемах, который положил начало гонке по пути миниатюризации.

11.2. Из чего сделаны ИС?

Все известные нам материалы содержат элементарные частицы электрического заряда двух типов: *электроны* и *протоны*. Электрон является мельчайшей частицей

электрического заряда, обладающей характеристикой, которую мы называем *отрицательной полярностью*. Протон имеет *положительную полярность*. Именно расположение в том или ином порядке электронов и протонов как элементарных частиц определяет электрические характеристики вещества.

Если электроны могут свободно перемещаться внутри вещества от атома к атому, такое вещество называется *проводником*. Металлы в большинстве своем являются хорошими проводниками (наилучший проводник – серебро, следующий за ним – медь), потому что атомная структура металлов не затрудняет свободного перемещения электронов.

Интегральная схема представляет собой маленький кусочек кремния, который с помощью процесса, напоминающего контактную фотопечать, приобрел возможность управления положительными и отрицательными электрическими сигналами. ИС обычно выглядит, как тусклая крупинка алюминия размером с ноготь новорожденного.

Основной физической составляющей ИС является кремний, второй по распространенности элемент на Земле. Другая историческая справка: элемент Si (кремний) был открыт в 1823 г. Кремний является *полупроводником*. Другими словами, он представляет собой такой тип твердого тела, у которого электрическая проводимость имеет среднюю величину, где-то между проводимостью меди (высокая) и резины (низкая). Способность полупроводника проводить электричество может быть увеличена добавлением примесных элементов с помощью процесса, называемого *легированием*.

Целью легирования является введение свободных зарядов, которые могут легко перемещаться под действием приложенного напряжения. В зависимости от типа легирования эти дополнительные заряды могут быть как отрицательными, так и положительными. Если добавляются электроны, легирование делает полупроводник отрицательным *n*-типа. При недостатке электронов образуется положительный материал *p*-типа. Легирование увеличивает проводимость материала от 10 до 50 раз. Чистый, нелегированный полупроводник называется *собственным* полупроводником. Легирование полупроводника делает его *примесным*.

При изготовлении ИС известный технологический процесс образования транзисторов видоизменяется так, чтобы получить на одной подложке большое количество транзисторов, диодов и резисторов. Отдельные элементы схемы формируются на единой кремниевой подложке с помощью процесса, включающего фотолитографию, диффузию, нагрев и отжиг, испарение и осаждение, а также ионную имплантацию. На каждой кремниевой пластине одна и та же схема может быть повторена много раз, что существенно сокращает стоимость производства. Так, микросхема на кремниевой подложке размером менее 1/4 дюйма по длине и ширине может содержать более 250 000 полупроводниковых компонентов (рис. 11.1).

С целью уменьшения стоимости изготовления каждой ИС кремниевые пластины, которые сначала имели диаметр 3/4 дюйма (19 мм), были увеличены до 1,5 дюйма (38 мм). В 1971 г. технологические линии по производству ИС перевели на пластины диаметром 3 дюйма (76 мм). Такие пластины использовались в массовом производстве до начала 1980 г., когда их сменили пластины размером 4 дюйма (100 мм). Затем в конце 1982 г. в производство ИС были внедрены пластины диаметром 5 дюймов (125 мм).

К концу 1983 г. с технологических линий по производству ИС сходили пластины размером 6 дюймов (150 мм), и разрабатывалась технология обработки 8-дюймовых пластин. Эти пластины столь велики, что для уменьшения поверхностных напряжений в процессе обработки их приходится просверливать в центре. Потребление кремниевых пластин только в США превышает 70 млн. в год. Будучи положены

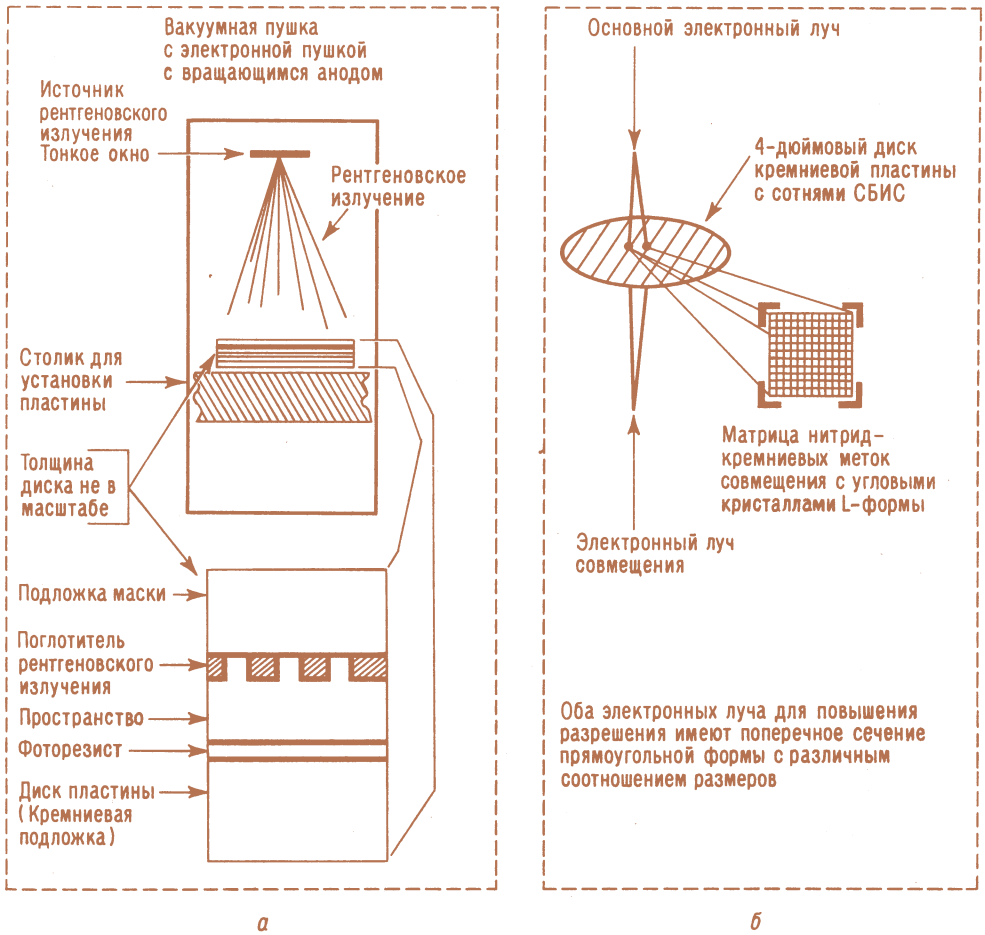


Рис. 11.1. Формирование кремниевой микросхемы с помощью: *а* – рентгеновской литографии; *б* – электронно-лучевой литографии.

друг на друга, эти пластины образовали бы столб длиной около 15 миль (1 миля = 1609 м).

Интегральная схема состоит из транзисторов. В принципе для транзисторов характерны две области применения. Первая – это аналоговые устройства для радио, телевидения и т. д. В этой области транзисторы служат главным образом в качестве усилителей. Вторая область – это цифровые схемы для компьютеров, калькуляторов и подобных им устройств. Транзисторы здесь выполняют функции ключей, которые включаются и выключаются миллионы раз в секунду.

Полевой транзистор – это полупроводниковый прибор, в котором сопротивление между двумя выводами управляется напряжением на третьем выводе. В одной из разновидностей полевого транзистора, содержащей структуру *металл-окисел-полупроводник* (*МОП-транзистор*), на поверхности слаболегированной полупроводниковой подложки формируются две области материала противоположного типа. Одна из этих областей, из которой вытекают носители заряда, называется *истоком*. Вторая область, в которой носители заряда прекращают свое движение, получила название *стока*.

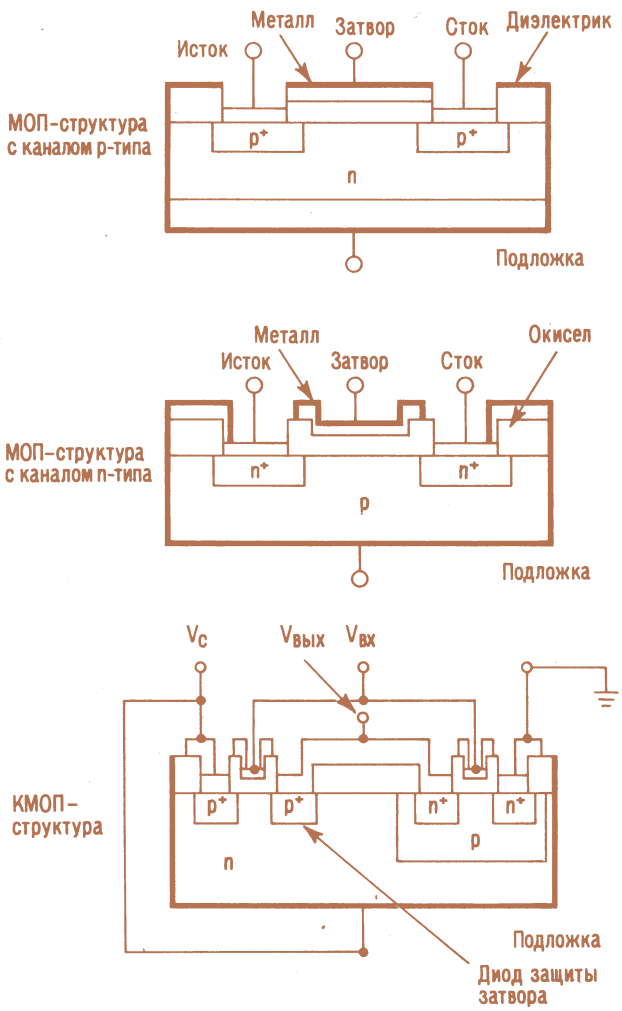


Рис. 11.2. Сравнение МОП-структур с каналом *p*-типа, *n*-типа и КМОП.

Область полупроводникового материала, лежащая между истоком и стоком, называется *каналом*. Над каналом располагается металлический электрод, называемый *затвором*, который имеет ту же длину, что и канал, и отделен от него слоем изоляции. Название полевого транзистора такого типа (МОП-транзистор) происходит от составляющих его элементов: металлического затвора, изолирующего слоя из окисла и полупроводниковой подложки (рис. 11.2).

11.2.1. МОП-транзисторы с каналами *p*- и *n*-типов

На базе полевых транзисторов разработано несколько семейств логических полупроводниковых приборов. Наиболее известны МОП-приборы с каналами *n*- и *p*-типов. Для их изготовления материал легируется так, чтобы получить полупроводник либо *n*-, либо *p*-типа. Для получения материала *n*-типа в качестве легирующих

примесей берутся мышьяк, сурьма или фосфор. Для получения *p*-типа используются алюминий, галлий, бор и иридий.

Биполярный транзистор – это управляемый током полупроводниковый прибор, в котором имеются как неосновные, так и основные носители. Изготовление биполярных транзисторов обычно требует 12 этапов маскирования и 4 диффузионных этапа. Пластины МОП-транзисторов с каналами *p*-типа можно изготавливать при меньшем числе этапов обработки. Эта простота обусловила изготовление более сложных логических микросхем. Именно технология МОП-транзисторов с каналами *p*-типа позволила в начале 70-х годов организовать коммерческий выпуск больших интегральных схем (БИС). Сегодня транзисторы этого типа в основном устарели и используются лишь в небольшом числе выпускаемых ИС.

За последнее время технологи получили новые сведения о физике кремниевой поверхности. Возникли серьезные основания к разработке процесса изготовления *n*-канальных МОП-транзисторов, обладающих в три раза лучшими характеристиками, чем приборы *p*-типа.

Схемы на МОП-транзисторах с каналом *n*-типа отличаются большей скоростью работы и к тому же содержат больше транзисторов. И те и другие приборы могут работать в режиме обогащения. Режим обогащения попросту означает, что приложенное напряжение увеличивает число носителей заряда в канале между истоком и стоком.

11.2.2. КМОП-транзисторы

Третий широко известный тип полевых МОП-транзисторов носит название *комплементарных МОП-транзисторов* (КМОП-транзисторы). Эти приборы имеют в своем составе МОП-транзисторы с каналами обоих типов. Логический вентиль на КМОП-транзисторе состоит из двух элементов, один из которых отпирает выход вентиля при действии на входе правильной комбинации сигналов, а другой запирает выход для других комбинаций входных сигналов.

Логические схемы с использованием КМОП-транзисторов более сложны, чем схемы с МОП-приборами *n*- или *p*-типа, и не обеспечивают столь высоких плотностей размещения вентилей. КМОП-структуры, однако, имеют заметное преимущество перед всеми другими логическими микросхемами – в устойчивом состоянии между нулевой линией и источником питания не существует проводящих каналов. Поэтому все время, пока состояния входов постоянны, схема практически не потребляет энергии.

Такой подход к потреблению энергии имеет огромное значение для устройств с батарейным питанием. Во время переключения энергия потребляется по двум основным причинам. Во-первых, в схеме существуют емкости, прежде всего между электродами МОП-транзисторов. Емкость – это свойство сохранения противоположных зарядов, пока между проводниками поддерживается разность потенциалов. Каждый раз при переключении вентиля емкость должна перезарядиться. Во-вторых, в течение времени переключения под действием входных сигналов оба элемента логического вентиля на КМОП-транзисторе оказываются частично открытыми. В результате от источника питания через схему течет кратковременный ток.

Дополнительные этапы обработки, необходимые для изготовления КМОП ИС, увеличивают их стоимость. Эти затраты были переложены на покупателей.

11.2.3. Сверхбольшие интегральные схемы (СБИС)

С точки зрения количества составляющих вентилей микросхемы можно разделить на четыре категории: схемы с низким уровнем интеграции (МИС), схемы среднего

уровня интеграции (СИС), большие интегральные схемы (БИС) и сверхбольшие интегральные схемы (СБИС). Схемы с низким уровнем интеграции содержат от одного до 4 вентилей. В схемах среднего уровня интеграции может быть от 4 до 100 вентилей. 100 вентилей на микросхему – это порог, за которым начинаются БИС. Если, наконец, схема содержит более 20 000 вентилей, ее относят к СБИС. Число вентилей в некоторых СБИС доходит до 250 000.

Технология современных *n*-канальных МОП ИС повышенной плотности позволяет постоянно увеличивать число вентилей, размещаемых на одной микросхеме. Уплотнение вентилей и сокращение их размеров дает возможность надеяться на увеличение скорости переключений в микросхеме и снижение потребления энергии.

Внедрение технологии СБИС стало возможным благодаря умению изготовителей ИС выполнять точное экспонирование фотошаблонов схем с высоким разрешением на кремниевых пластинах, покрытых резистом. Разрешение этих шаблонов позволяет уменьшить минимальную ширину линий до 1–2 микрон. Для получения такого разрешения используется широкий спектр литографических методов: контактный, с совмещением, проекционный 1:1, с последовательным шаговым экспонированием, рентгеновский и электронно-лучевой.

По оценкам промышленных экспертов технология СБИС скоро достигнет такого уровня, что на микросхеме можно будет разместить до миллиона вентилей. Серьезной проблемой в разработке СБИС становится затрата значительных ресурсов, которые требуются для проектирования внутренней логической структуры микросхемы. Возрастает вероятность потери рынка сбыта, потому что последний может не согласиться с базовыми идеями, заложенными в проект на различных этапах его разработки.

Чтобы избежать этих неприятностей, компании, выпускающие полупроводниковую продукцию, вынуждены переходить на новые методы разработки. Теперь инженеры и проектировщики уже не могут «вручную» продумывать, разрабатывать, а затем тестировать новую микросхему – это заняло бы слишком много времени. Весь комплексный проект разбивается на серию задач, каждую из которых можно запрограммировать на ЭВМ.

Сначала выполняется функциональное проектирование – синтез, верификация, моделирование и тестирование на архитектурном, системном, логическом, схемном, приборном и процессном уровнях. Затем идет физическая разработка – разбиение, разработка топологии и топологический анализ. На обеих фазах параллельно с проектированием должны рассматриваться такие факторы, как функциональность, тестируемость и технологичность.

Системы автоматизированного проектирования (САПР) требуют значительных ресурсов памяти и многих часов процессорного времени для решения задач, возникающих при проектировании топологии сложной микросхемы. К тому же компьютерные программы не достигли пока такого уровня, чтобы они могли предлагать лучшие решения, чем опытные разработчики, так что на каждом шаге, перед тем как двинуться дальше, должны выполняться перекрестные проверки и контроль. Тем не менее применение САПР ускорило разработку микросхем.

11.3. Упаковка ИС

Одним из существенных ограничений, накладываемых на проект ИС, является необходимость помещения ее в корпус. При этом надо иметь в виду следующее:

- процессор должен быть оснащен шиной данных. Эта шина требует стольких выводов, какова ее ширина (16 для МП 8086 и 80286, 32 для 80386);

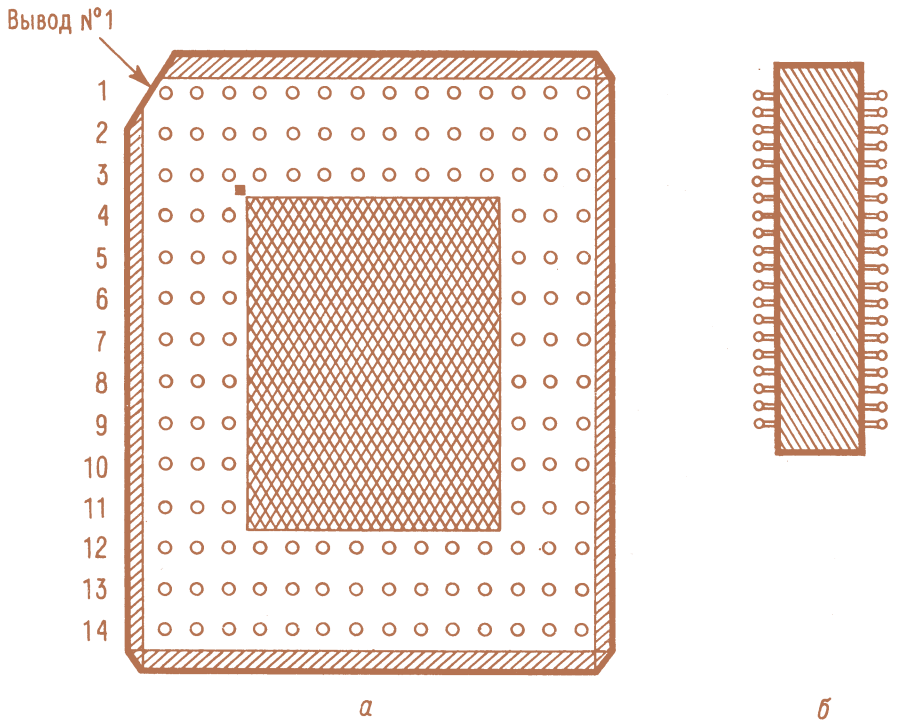


Рис. 11.3. *а* – корпус с матричным расположением выводов МП 80386; *б* – корпус с двухрядным расположением выводов.

- дополнительные выводы требуются для адресной шины, причем их количество опять совпадает с полной шириной адреса;
- два вывода нужны для подачи напряжения питания и заземления;
- уменьшение числа выводов увеличивает сложность устройства (пример – выводы RAS/CAS в микросхемах памяти);
- наконец, по крайней мере два вывода (иногда четыре) требуются для подключения внешнего тактового генератора.

В дополнение к этим пяти пунктам требуется еще по крайней мере несколько выводов для синхронизации внешними событиями. Практически это означает, что 32-разрядный микропроцессор нельзя поместить в корпусе с 40 выводами. Обратите внимание на расположение выводов МП 80386 (рис. 11.3).

Другое ограничение носит технологический характер и связано с конечными размерами полезной площади микросхемы, определяемыми сегодняшним уровнем технологии. Разработчики стремятся реализовать в микросхеме максимально возможное число функций.

Появление матриц логических элементов побудило промышленность ИС искать пути размещения на поверхности корпуса большого числа выводов, 68 и более. Появляются комбинированные корпуса с планарными и штыревыми выводами, причем как керамические (для военных применений), так и пластмассовые (для остальных потребителей).

Высокую плотность обеспечивают *корпуса с матричным расположением выводов*. По форме такой корпус напоминает прямоугольную коробку, из которой торчат ряды гвоздей. Корпуса с матричным расположением выводов имеют

керамическое основание, малое расстояние между выводами (0,1 дюйма), превосходные тепловые характеристики и высокую плотность выводов. На рис. 11.3 показаны для сравнения в одном масштабе современный корпус ИС с матрицей выводов и плоский корпус с традиционным двухрядным расположением выводов.

Совершенствование кремниевых микросхем приближается к своему физическому пределу. Требует рассмотрения, например, снижение надежности, являющееся результатом резкого уменьшения размеров изделий. Сказывается также ограниченность экономических ресурсов при использовании необходимых для проектирования машинных средств; лишь очень немногие программисты имеют достаточно глубокое понимание требований САПР, обеспечивающее поиск новых решений. Кроме того, выпуск нового оборудования для производства СБИС сопровождается ростом его стоимости и снижением производительности.

Отход от 5-вольтового стандарта питания может встретить активную оппозицию, хотя такое решение представляется неизбежным, если будут использоваться транзисторы с меньшей длиной канала. Слишком много разработок выполнено в нынешнем стандарте, чтобы можно было легко и быстро заменить его новым.

Мы подходим к супербольшим интегральным схемам. Мы также входим в век распределенной обработки. Внутри самого микропроцессора каждая из микросхем ввода-вывода будет содержать в себе микропроцессор.

С точки зрения человеческих и социальных запросов можно предсказать, что многофункциональные микропроцессоры со встроенным интеллектом станут столь же обычными приборами, какими сегодня являются электродвигатели.

Глава 12

Принципы сопряжения

Как уже отмечалось, микропроцессор становится законченным изделием – микрокомпьютером – лишь после соединения с целым рядом дополнительных компонентов. При выполнении этих соединений следует учитывать природу и временные характеристики сигналов, возбуждаемых и воспринимаемых каждым компонентом.

Сигналы можно разбить на три класса: данных, адресов и управления. По отношении к центральному процессору (ЦП) сигналы данных и управления являются двунаправленными (т. е. и входными, и выходными), а сигналы адресов – однонаправленными (выходными). Очевидно, что сигналы должны передаваться от одного компонента к другому и переносить с собой информацию и данные. Этот процесс носит название *сопряжения*.

Для того чтобы передать сигналы от одного компонента другому, следует разработать вспомогательные схемы и правильно выбрать сами компоненты.

12.1. Сопряжение

Для компьютеров характерны два основных метода сопряжения: аппаратный и программный. Большинство проектировщиков используют оба метода. Программное сопряжение характеризуется двумя основными параметрами: временем выполнения программы и памятью, необходимой для ее хранения. Создавая программное обеспечение, избегайте узких мест. Если, например, в процессе выполнения программного сопряжения большая часть времени расходуется на выполнение сложной подпрограммы ввода-вывода, для других важных вычислений просто может не остаться времени.

На аппаратном уровне удобно организовывать взаимодействие сигналов, посылаемых и принимаемых для управления такими жестко заданными и критичными по времени операциями, как получение строк битов от устройств последовательной передачи данных. Вместо того чтобы в цикле получать и сдвигать бит за битом с целью формирования байта, системная программа может обрабатывать запросы на прерывания, поступающие от аппаратных сигналов, и читать сразу целый байт. Учтите, однако, что аппаратуру сложнее и разрабатывать, и тестировать, чем программы. На рис. 12.1 показаны типичные средства сопряжения процессора с контроллером периферийного устройства.

12.1.1. Уровни

Сопряжение можно описывать на различных уровнях: механическом, электрическом, функциональном (от процессора к периферийным устройствам) и пользовательском (от периферийных устройств к процессору).

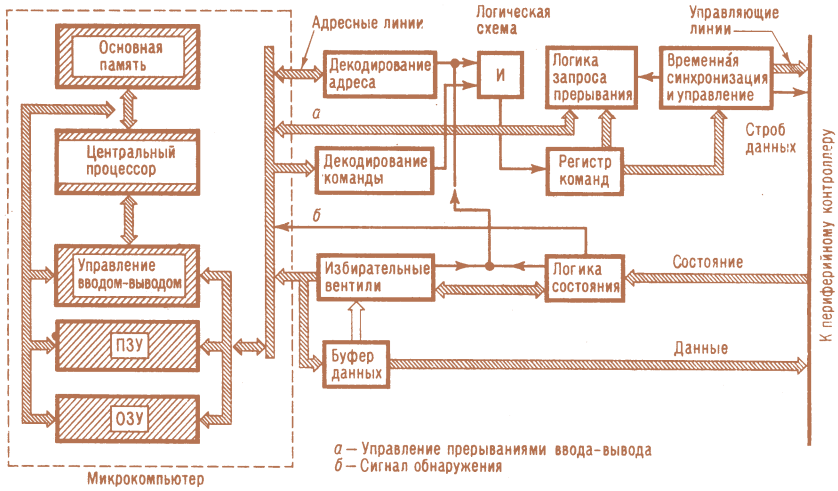


Рис. 12.1. Типичное логическое сопряжение микрокомпьютера с периферийным контроллером.

Механическое сопряжение учитывает физические характеристики соединения двух устройств или подсистем. Сопряжение может быть воплощено в виде соединительного разъема, который характеризуется определенными размерами, формой и количеством контактов. Электрическое сопряжение – это описание электрических сигналов на каждом контакте, включающее в себя уровни напряжения и специфические функции электрических сигналов.

Под функциональным сопряжением понимаются правила, или протокол передачи сигналов через соединение, включая описание определяющей цель взаимодействия последовательности сигналов, возбуждаемых обоими устройствами.

Сопряжение на уровне пользователя учитывает содержание информации или данных, передаваемых от одного устройства другому. Это содержание может быть определено в контексте конкретной операционной системы или прикладной программы. Пример – последовательность битов, интерпретируемая как сообщение «задание завершилось». Смысл сообщения может задаваться устройством; например, конкретная последовательность битов может указывать на тип ошибки ввода-вывода.

12.1.2. Регистр

Важнейшим элементом любого сопряжения процессор – устройство является регистр, в котором хранится информация, передаваемая в процессор или из него. Регистр используется в качестве временной или буферной памяти и служит для обмена информацией между процессором и устройствами, у которых скорости передачи информации могут сильно отличаться. Без этой временной памяти более быстрое устройство (обычно процессор) большую часть времени будет тратить на ожидание завершения операции более медленным устройством.

Информация может храниться в виде байта, слова, двойного слова или некоторого количества слов. Большие объемы временно хранимой информации характерны для некоторых видеотерминалов. Поскольку изображение должно «регенерироваться», или обновляться на экране терминала от 10 до 20 раз в секунду, весь объем выводимой на экран информации должен считываться с той же

скоростью. Для того чтобы не монополизировать процессор, видеотерминалы часто содержат встроенную память объемом несколько тысяч байтов, хранящую экранный кадр.

Довольно часто при проектировании устройств сопряжения возникает необходимость в регистре с такими внутренними соединениями, чтобы внешний импульс увеличивал или уменьшал на единицу содержимое регистра. Такой регистр носит название *счетчика*. Типичное применение счетчика в средствах сопряжения, обеспечивающих передачу данных блоками, – контроль объема передачи. В счетчик загружается число передаваемых слов. При передаче каждого слова в счетчик посылается импульс, уменьшающий его содержимое на единицу. Как только число в счетчике уменьшится до нуля, передача завершается.

Другой пример использования счетчика – интервальный таймер. Каждый сигнал системного таймера уменьшает содержимое счетчика на 1. Это дает возможность программно загружать в счетчик значение отсчитываемого «назад» интервала.

12.1.3. Буфер

При использовании в устройстве сопряжения буферов важно оптимизировать буферное пространство, что позволяет добиться координированности процессов в системе. Другими словами, центральный процессор и остальные схемы должны быть так сбалансированы, чтобы достигалась максимальная производительность системы. Целесообразно, например, принимать и сортировать по приоритетам поступающие сигналы прерываний и лишь после этого давать разрешение на прерывание вычислений, выполняемых центральным процессором; это дает процессору возможность обратиться к наиболее срочной программе обработки прерываний.

С целью упрощения задачи сопряжения узлов системы многие изготовители выпускают специальные интерфейсные микросхемы, предназначенные для конкретных устройств. Эти интерфейсные микросхемы содержат все логические цепи, необходимые для обеспечения типичного сопряжения. Процессор обращается к интерфейсной микросхеме по адресу, присвоенному ей путем распайки выведенных наружу линий выбора адреса.

12.1.4. Периферийные устройства

Для проверки состояния периферийного устройства и для передачи данных в устройство или из него предусматриваются специальные команды и управляющие сигналы. Проверка состояния и передача данных осуществляются в рамках трех основных способов коммуникации между процессором и внешним миром: программная передача данных, прямой доступ к памяти (ПДП) и прерывания.

Периферийные устройства могут подключаться к шинам компьютера с помощью тех же управляющих сигналов, которые используются для обращения к памяти. В этом случае сопряжение периферийного устройства с процессором выполняется *с отображением на память*. Обмениваясь данными с периферийным устройством, процессор выполняет команды обращения к памяти, в результате чего возбуждаются сигналы управления памятью[1)].

Если процессор запрашивает данные, периферийное устройство помещает информацию на шину данных, чтобы процессор мог ее прочитать. Если процессор выводит данные, периферийное устройство забирает информацию с шины данных.

[1)] Которые в данном случае взаимодействуют с интерфейсом периферийного устройства. *Прим. перев.*

Периферийные устройства могут также адресоваться с использованием отдельного адресного пространства. Адреса, расположенные в этом пространстве ввода-вывода, носят названия портов ввода-вывода. Для обмена данными с портами ввода-вывода предусматриваются специальные команды типа IN и OUT.

12.1.5. Память

Память микрокомпьютера представляет собой массив ячеек для хранения данных и команд. Существует большое количество различных устройств памяти. Оперативные запоминающие устройства могут применяться в сочетании с портами ввода-вывода, цифро-аналоговыми и аналого-цифровыми преобразователями и т. д.[1)] Управляющие сигналы, используемые в операциях обращения к памяти, обеспечивают те или иные формы запроса памяти, записи в память и чтения из памяти.

Простейшим приспособлением является одиночный цифровой выход, служащий для включения и выключения некоторого устройства. Это приспособление включает в себя одноразрядную ячейку памяти, устанавливаемую или сбрасываемую программой. Поскольку такого рода ячейки обычно устанавливаются после включения питания в случайное состояние, компьютер перед началом вычислений устанавливает все эти схемы, как внутренние, так и внешние, в некоторое исходное состояние.

Несколько более сложное сопряжение используется при работе с цифро-аналоговым преобразователем (ЦАП). В простейшей форме ЦАП вырабатывает напряжение, значение которого определяется числом, поступившим в ЦАП. Это напряжение сохраняется неизменным, пока ЦАП не получит другое число. В устройстве сопряжения данные снимаются с линий данных в соответствующий регистр под управлением программы. Регистр хранит полученное число до тех пор, пока оно не будет заменено следующим. Если требуется иметь несколько аналоговых сигналов, к выходу преобразователя можно подключить мультиплексор, и тогда преобразователь сможет подавать напряжение на несколько различных устройств.

Часто компьютер участвует в аналого-цифровой обработке. Другими словами, компьютер дискретизирует аналоговое напряжение, подаваемое извне, и преобразует его значение в цифровую форму. Выполняя такое преобразование, следует соблюдать осторожность. Время, требуемое для выполнения преобразования с точностью 10–12 двоичных разрядов, может занимать много машинных тактов, скажем от 30 до 50 мкс, что для МП 80386, имеющего тактовую частоту 16 МГц, составит от 480 до 800 тактов. Таким образом, начав процесс преобразования, важно выждать определенное время перед тем, как пытаться пересылать содержимое буферного регистра преобразователя в аккумулятор.

В некоторых аналого-цифровых преобразователях предусматривается возможность переключения длины выходного слова, причем слова меньшего размера дают менее точные, но более быстрые результаты. В этом случае возникает возможность вручную или программно выбрать для конкретного приложения наиболее удобное сочетание скорости и точности. Естественно, выигрыш в скорости получится лишь при автоматической подстройке программной задержки, которая должна немного превышать время преобразования. Обычно завершение преобразования вызывает установку некоторого одноразрядного флага.

[1)] Запоминающее устройство (обычно небольшого объема), входящее в состав периферийного оборудования, принято относить к буферным устройствам, а не к устройствам памяти.– *Прим. перев.*

12.2. Шины

Шина – это канал пересылки данных, используемый совместно различными блоками системы. Шина может представлять собой набор проводящих линий, вытравленных на печатной плате, провода, припаянные к выводам разъемов, в которые вставляются печатные платы, либо плоский кабель. Компоненты компьютерной системы физически расположены на одной или нескольких печатных платах, причем их число и функции зависят от конфигурации системы, ее изготовителя, а часто и от поколения микропроцессора.

Информация передается по шине в виде групп битов. В составе шины для каждого бита слова может быть предусмотрена отдельная линия (параллельная шина), или все биты слова могут последовательно во времени использовать одну линию (последовательная шина). На рис. 12.2 изображено типичное подключение устройств к шине данных.

12.2.1. Шина с тремя состояниями

Шина с тремя состояниями напоминает телефонную линию общего пользования, к которой подключено много абонентов. Три состояния на шине – это состояния высокого уровня, низкого уровня и высокого импеданса. Состояние высокого импеданса позволяет устройству или процессору отключиться от шины и не влиять на уровни, устанавливаемые на шине другими устройствами или процессорами[1]. Таким образом, только одно устройство является ведущим на шине. Управляющая логика активизирует в каждый конкретный момент только одно устройство, которое становится ведущим. Когда устройство активизировано, оно помещает свои данные на шину, все же остальные потенциальные ведущие переводятся в пассивное состояние.

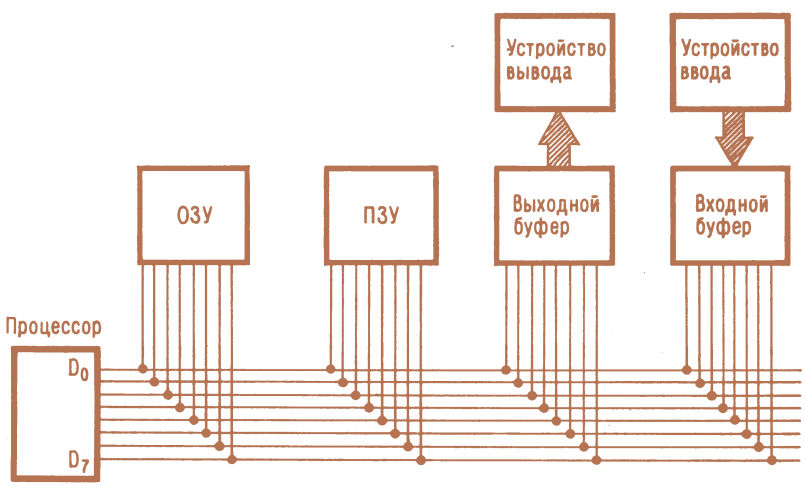


Рис. 12.2. Типичное подключение устройств к шине данных.

[1] Строго говоря, здесь идет речь не о шине (магистрали) в целом, а об отдельных линиях шины. – *Прим. перев.*

Рис. 12.3. Двунаправленные отправители/получатели, подключенные к линиям шины.

К шине может быть подключено много приемных устройств – получателей. Обычно данные на шине предназначаются только для одного из них. Сочетание управляющих и адресных сигналов определяет, для какого именно. Управляющая логика возбуждает специальные стробирующие сигналы, чтобы указать получателю, когда ему следует принимать данные. Получатели и отправители могут быть однонаправленными (т. е. осуществлять только *либо* передачу, *либо* прием) и двунаправленными (осуществлять и то и другое). На рис. 12.3 показаны двунаправленные отправители/получатели, подключенные к шине.

Шинная (магистральная) организация получила широкое распространение, поскольку в этом случае все устройства используют единый протокол сопряжения. Как правило, стандартная архитектура микропроцессоров требует соединения модулей центральных процессоров и устройств ввода-вывода с помощью трех шин.

12.2.2. Типы шин

Сопряжение с ЦП осуществляется посредством трех шин: *шины данных*, *шины адресов* и *шины управления*. Шина данных служит для пересылки данных между ЦП и памятью или ЦП и устройствами ввода-вывода. Эти данные могут представлять собой как команды ЦП, так и информацию, которую ЦП посылает в порты ввода-вывода или принимает оттуда. В МП 8088 шина данных имеет ширину 8 разрядов. В МП 8086, 80186 и 80286 ширина шины данных 16 разрядов; в МП 80386 – 32 разряда.

Шина адресов используется ЦП для выбора требуемой ячейки памяти или устройства ввода-вывода путем установки на шине конкретного адреса, соответствующего одной из ячеек памяти или одного из элементов ввода-вывода, входящих в систему. Наконец, по шине управления передаются управляющие сигналы, предназначенные памяти и устройствам ввода-вывода. Эти сигналы

указывают направление передачи данных (в ЦП или из ЦП), а также моменты передачи.

Магистральная организация предполагает, как правило, наличие управляющего модуля, который выступает в роли директора-распорядителя при обмене данными. Основное назначение этого модуля – организация передачи слова между двумя другими модулями.

12.2.3. Операции на магистрали

Операция на системной магистрали начинается с того, что управляющий модуль устанавливает на шине кодовое слово модуля-отправителя и активизирует линию строба отправителя. Это позволяет модулю, кодовое слово которого установлено на шине, понять, что он является отправителем. Затем управляющий модуль устанавливает на шине кодовое слово модуля-получателя и активизирует линию строба получателя. Это позволяет модулю, кодовое слово которого теперь установлено на шине, понять, что он является получателем.

После этого управляющий модуль возбуждает линию строба данных, в результате чего содержимое регистра отправителя пересылается в регистр получателя. Этот шаг может быть повторен любое число раз, если требуется передать много слов.

Данные пересылаются от отправителя получателю в ответ на импульс, возбуждаемый управляющим модулем на соответствующей линии строба. При этом предполагается, что к моменту появления импульса строба в модуле-отправителе данные подготовлены к передаче, а модуль-получатель готов принять данные. Такая передача данных носит название *синхронной* (синхронизированной).

Что произойдет, если модули, участвующие в обмене (один или оба), могут передавать или принимать данные только при определенных условиях? Процессы на магистрали могут носить асинхронный (несинхронизированный) характер. Передачу данных от отправителя получателю можно координировать с помощью линий состояния, сигналы на которых отражают условия работы обоих модулей. Как только модуль назначается отправителем, он принимает контроль над линией готовности отправителя, сигнализируя с ее помощью о своей готовности передавать данные. Модуль, назначенный получателем, контролирует линию готовности получателя, сигнализируя с ее помощью о своей готовности принимать данные.

При передаче данных должны соблюдаться два условия. Во-первых, передача осуществляется лишь в том случае, если и получатель, и отправитель сигнализируют о своей готовности. Во-вторых, каждое слово должно передаваться один раз. Для обеспечения этих условий предусматривается определенная последовательность действий при передаче данных. Эта последовательность носит название *протокола*.

В соответствии с протоколом отправитель, подготовив новое слово, информирует об этом получателя. Получатель, приняв очередное слово, информирует об этом отправителя. Состояние линий готовности в любой момент времени определяет действия, которые должны выполнять оба модуля.

Каждый шаг в передаче данных от одной части системы к другой называется *циклом магистрали* (или часто *машинным циклом*). Частота этих циклов определяется тактовыми сигналами ЦП. Длительность цикла магистрали связана с частотой тактовых сигналов. Типичными являются тактовые частоты 5, 8, 10 и 16 МГц. Наиболее современные схемы работают на частоте до 24 МГц.

12.3. Порты ввода-вывода

Адресное пространство ввода-вывода организовано в виде портов. Порт представляет собой группу линий ввода-вывода, по которым происходит параллельная передача информации между ЦП и устройством ввода-вывода, обычно по одному биту на линию. Число линий в порте чаще всего совпадает с размером слова, характерным для данного процессора. Входной порт обычно реализуется в виде совокупности логических вентилей, через которые входные сигналы поступают на линии системной шины данных. Выходной порт реализуется в виде совокупности триггеров, в которых хранятся сигналы, снятые с шины данных.

Если в передаче информации участвует процессор, то направление потока входной или выходной информации принято рассматривать относительно самого процессора. *Входной порт* – это любой источник данных (например, регистр), который избирательным образом подключается к шине данных процессора и посылает слово данных *в* процессор. Наоборот, *выходной порт* представляет собой приемник данных (например, регистр), который избирательным образом подключается к шине данных процессора. Будучи выбран, выходной порт принимает слово данных *из* микропроцессора.

Процессор должен иметь возможность координировать скорость своей работы со скоростью работы внешнего устройства, с которым он обменивается информацией. В противном случае может получиться, что входной порт начнет пересылать данные еще до того, как процессор их затребует, и процесс пересылки данных наложится на какой-то другой процесс в ЦП[1]. Как уже отмечалось, эта координация работы двух устройств носит название «рукопожатия», или квитирования.

12.3.1. Режимы работы портов ввода-вывода

Существуют три вида взаимодействия процессора с портами ввода-вывода: программное управление, режим прерываний и прямой доступ к памяти (ПДП).

Программно-управляемый ввод-вывод инициируется процессором, который выполняет программу, управляющую работой внешнего устройства. Режим прерываний отличается тем, что инициатором ввода-вывода является внешнее устройство. Устройство, подключенное к выводу прерываний процессора, повышает уровень сигнала на этом выводе (или в зависимости от типа процессора понижает его). В ответ процессор, закончив выполнение текущей команды, сохраняет содержимое программного счетчика в соответствующем стеке и переходит на выполнение программы, называемой программой обработки прерываний, чтобы завершить передачу данных.

ПДП тоже инициируется устройством. Передача данных между памятью и устройством ввода-вывода осуществляется без вмешательства процессора. Как правило, для организации ПДП используются контроллеры ПДП, выполненные в виде интегральных микросхем.

---

[1] На практике, конечно, возможно и обратное: процессор, работая быстрее любого внешнего устройства, начнет извлекать из входного порта данные еще до того, как они там появились. В частности, это может привести к многократному считыванию одних и тех же данных. – *Прим. перев.*

12.4. Универсальный синхронно-асинхронный приемопередатчик

Микропроцессор взаимодействует с периферийными устройствами, принимающими и передающими данные в последовательной форме. В процессе этого взаимодействия процессор должен выполнять преобразование параллельного кода в последовательный, а также последовательного в параллельный.

Чаще всего пересылка данных между процессором и периферийными устройствами выполняется асинхронно. Другими словами, устройство может передавать данные в любой момент времени. Если данные не передаются, устройство посылает просто биты маркера, обычно высокий уровень сигнала, что дает возможность немедленно обнаружить любой разрыв цепи передачи. Если устройство готово передавать данные, передатчик посылает нулевой бит, обозначающий начало посылки. За этим нулевым битом следуют данные, затем бит четности и, наконец, один или два стоп-бита. Закончив передачу, отправитель продолжает посылать высокий уровень сигнала в знак того, что данные отсутствуют.

Для удобства проектирования интерфейса процессора с устройствами последовательного ввода-вывода (как синхронными, так и асинхронными) разрабо-

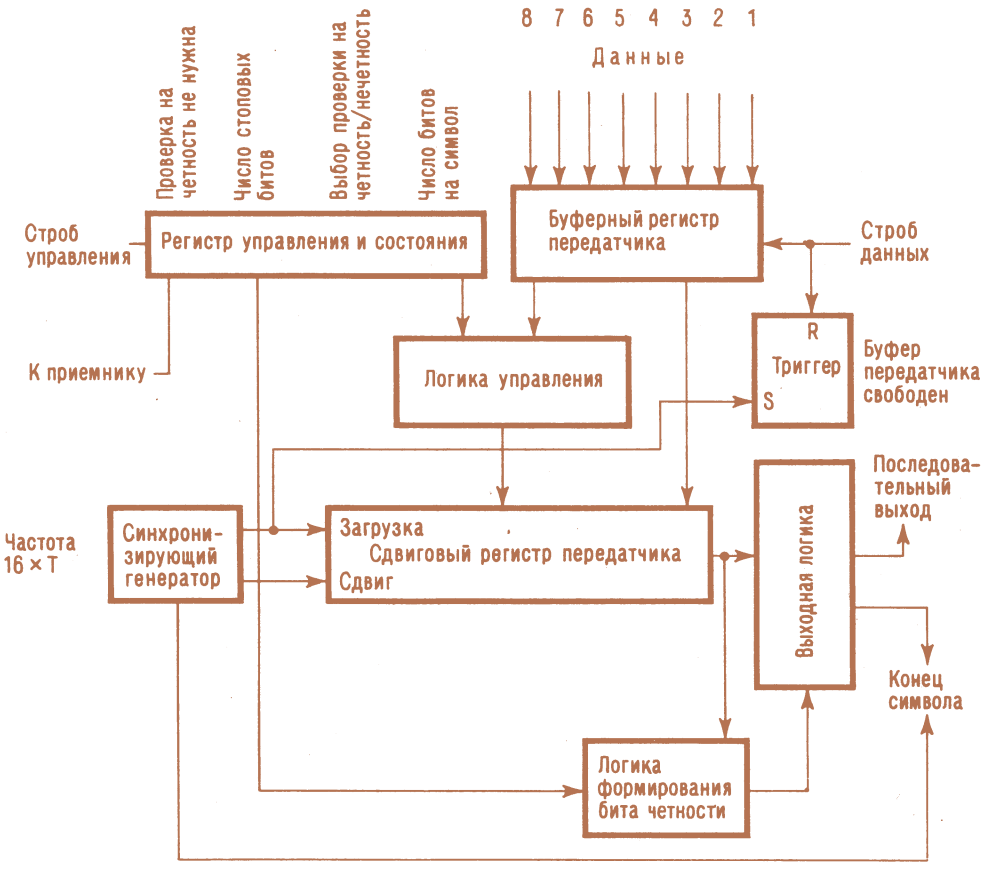


Рис. 12.4,а. Типичная структура секции передачи УСАПП.

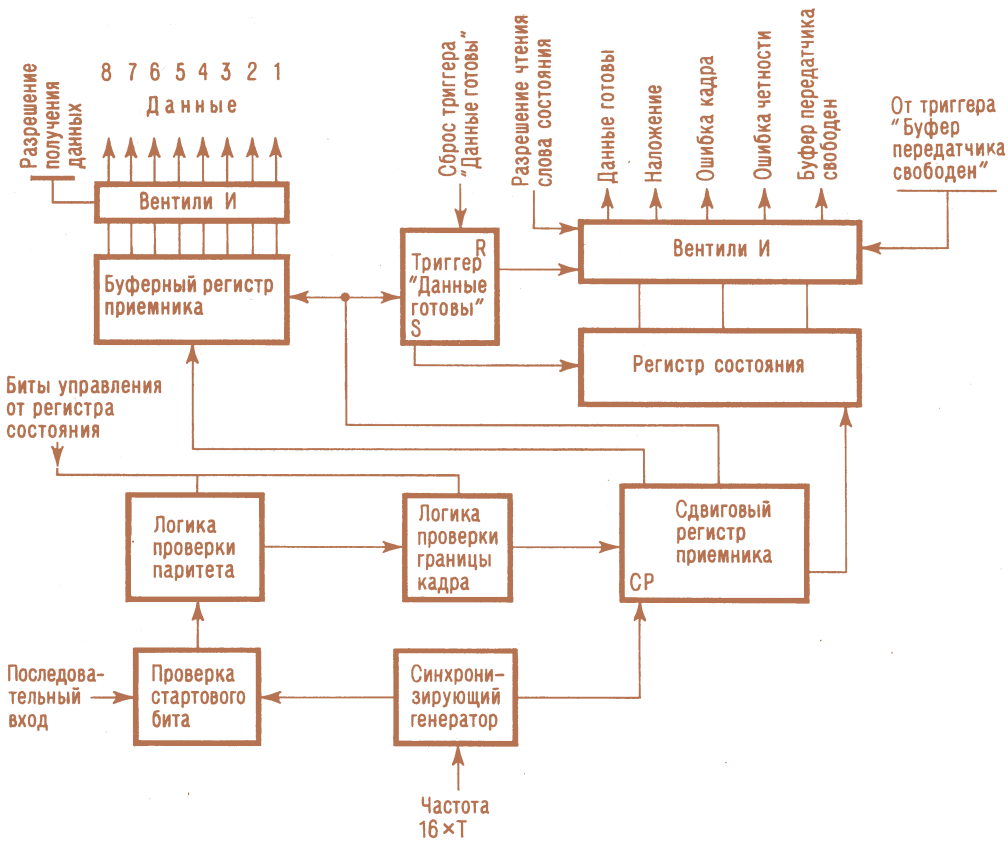


Рис. 12.4,б. Типичная структура секции приема УСАПП.

таны микросхемы универсальных синхронно-асинхронных приемопередатчиков (УСАПП). В состав УСАПП входят функционирующие независимо секции приемника-передатчика. На рис. 12.4 изображен типичный УСАПП.

УСАПП заключен в корпус с 40 выводами и является дуплексным устройством (т. е. может передавать и принимать одновременно). Он выполняет логическое форматирование посылок. Для подключения УСАПП могут потребоваться дополнительные схемы, однако нет необходимости в общем тактовом генераторе, синхронизирующем УСАПП и то устройство, с которым установлена связь. В передатчике УСАПП предусмотрена двойная буферизация, поэтому следующий байт данных может приниматься из процессора, как только текущий байт подготовлен для передачи.

Выпускаются микросхемы УСАПП со скоростями передачи до 200 Кбод. Скорость работы передатчика и приемника (не обязательно одинаковые) устанавливаются с помощью внешних генераторов, частота которых должна в 16 раз превышать требуемую скорость передачи. Сигналы от внешних генераторов поступают на раздельные тактовые входы приемника и передатчика.

Обычно и микропроцессор, и устройства ввода-вывода подключаются к своим УСАПП параллельно. Между УСАПП действует последовательная связь (например, по стандарту RS-232C).

12.5. MULTIBUS

Структура магистрали, обеспечивающей сопряжение всех аппаратных средств, является важнейшим элементом вычислительной системы. Магистраль позволяет многочисленным компонентам системы взаимодействовать друг с другом. Кроме того, в структуру магистрали заложены возможности возбуждения прерываний, ПДП, обмена данными с памятью и устройствами ввода-вывода и т. д.

Магистраль общего назначения MULTIBUS фирмы Intel представляет собой коммуникационный канал, позволяющий координировать работу самых разнообразных вычислительных модулей (дополнительные сведения о МП 80386 и магистралях MULTIBUS I и II приведены в приложении 4). Основой координации служит назначение модулям системы MULTIBUS атрибутов *ведущего* и *ведомого* [1].

12.5.1. Ведущие

Ведущим является любой модуль, который обладает возможностью захвата магистрали. Модуль захватывает магистраль с помощью логических схем обмена и инициирует передачу данных по магистрали, используя для этого либо встроенные процессоры, либо специальные логические схемы. Ведущие генерируют сигналы управления, адресные сигналы, а также адреса памяти или устройств ввода-вывода.

Ведущий может работать в одном из двух режимов: режиме 1 или режиме 2. В режиме 1 ведущий ограничен одной передачей по магистрали через каждое подключение к шине. Если все ведущие в системе используют режим 1, скорость работы системы ограничивается максимальной величиной цикла занятости магистрали. Это позволяет разработчикам прогнозировать общую производительность конкретной системы.

В режиме 2 у ведущих больше возможностей захвата магистрали, они могут инициировать обмен с наложением на текущую операцию. В этом режиме разрешены тайм-ауты магистрали, и операции ведущих не ограничены максимальной величиной цикла занятости магистрали. Режим 2 обеспечивает широкий класс операций, что придает системе гибкость при удовлетворении запросов пользователей.

12.5.2. Ведомые

Ведомые выполняют лишь функции получателей в процессе реализации запросов на передачу данных. Ведомые декодируют состояние адресных линий и действуют в соответствии с сигналами управления от ведущих. На рис. 12.5 изображены примеры ведущих и ведомых на магистрали MULTIBUS.

12.5.3. Операции на магистрали

Система MULTIBUS допускает наличие нескольких ведущих на магистрали, каждый из которых захватывает магистраль по мере возникновения необходимости в передаче данных. Ведущие осуществляют захват магистрали с помощью специальной последовательности обмена. В эту последовательность входят шесть сигналов, позволяющих ведущему определять, свободна ли магистраль и нет ли запросов на ее захват от других ведущих с более высоким приоритетом, а также захватывать и освобождать магистраль.

---

[1] Иногда употребляются термины «задатчик и исполнитель». *Прим. перев.*

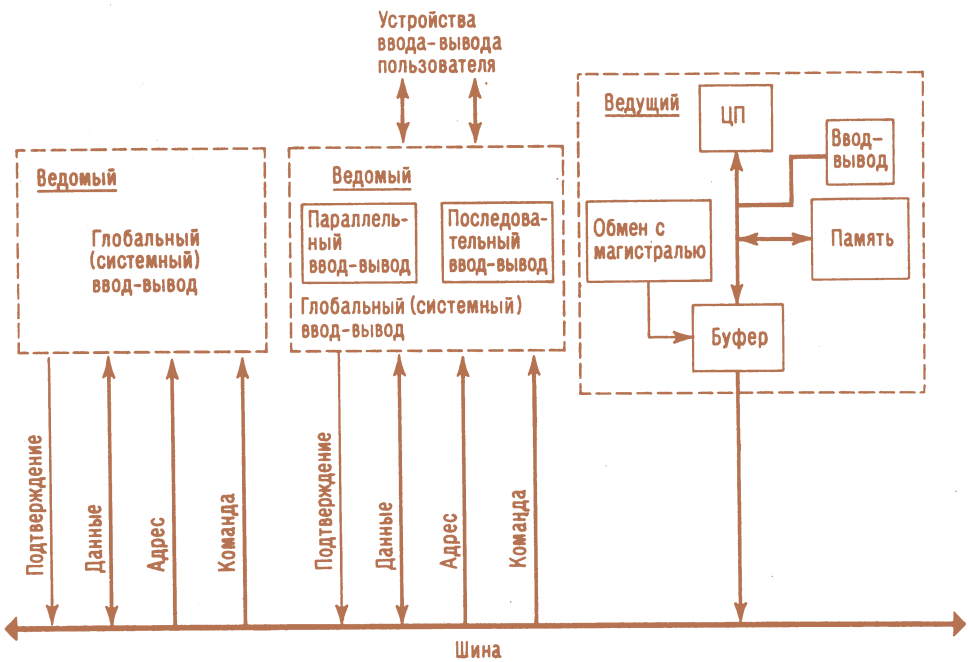


Рис. 12.5. Примеры ведущих и ведомых модулей системы MULTIBUS.

Арбитраж приоритетов. Система предусматривает две схемы арбитража приоритетов: последовательную и параллельную. В последовательной схеме приоритет ведущего определяется с помощью последовательной цепочки, в которой выход разрешения от каждого модуля соединяется с входом разрешения модуля с более низким приоритетом. На одном конце цепочки оказывается модуль с наивысшим приоритетом, на другом конце – с наинизшим.

Приоритет в последовательной схеме определяется при каждом запросе магистрали. Если магистраль не захвачена ведущим с более высоким или равным приоритетом, запрос данного ведущего удовлетворяется. Число ведущих, объединенных последовательной цепочкой, ограничено временем прохождения по цепочке сигнала приоритета, которое не должно превышать длительности цикла магистрали. Если используется частота 10 МГц, в цепочке может быть не более трех ведущих. На рис. 12.6 показана схема последовательного арбитража.

В параллельной схеме доступом к магистрали ведает специальный арбитр. При этом определение очередного ведущего на магистрали производится на основе списка фиксированных приоритетов или каким-то другим способом, заданным в системе. На рис. 12.7 показана одна из схем параллельного арбитража.

12.5.4. Архитектура магистрали

В магистраль MULTIBUS входят 16 линий данных, 20 адресных линий, 8 линий многоуровневых прерываний, а также линии управления и арбитража. Такое большое количество линий позволяет одновременно использовать в системе и 8- и 16-разрядные ведущие модули.

Система MULTIBUS использует собственный тактовый генератор, независимый от тактовых генераторов объединяемых модулей. Наличие независимого

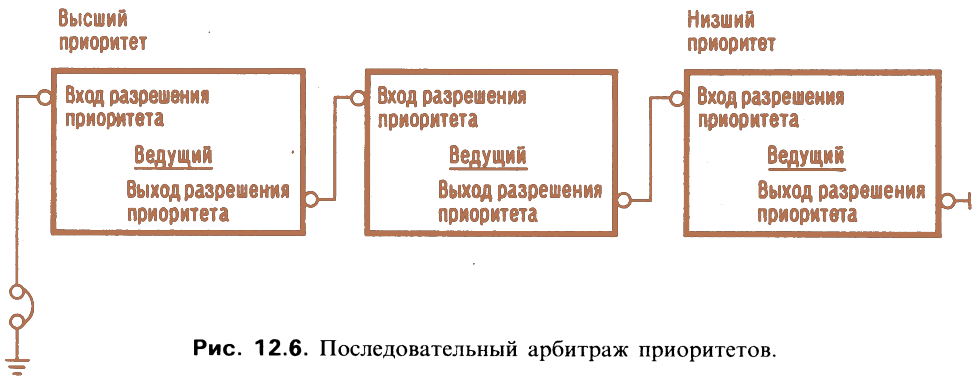


Рис. 12.6. Последовательный арбитраж приоритетов.

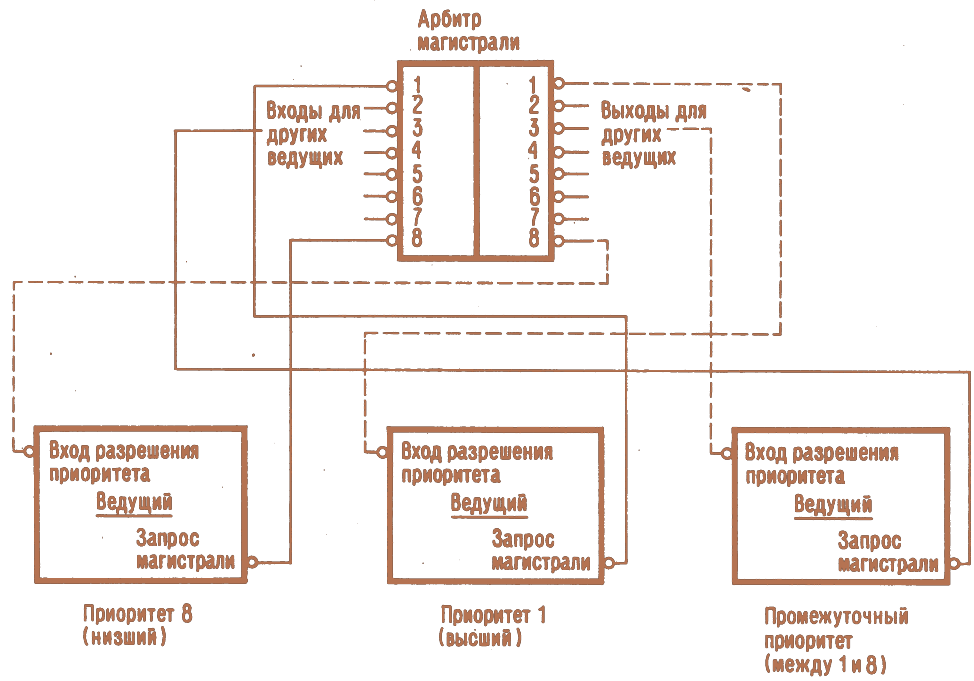


Рис. 12.7. Параллельный арбитраж приоритетов.

генератора позволяет использовать магистраль ведущими с различными тактовыми частотами, причем они могут выходить на магистраль асинхронно по отношению друг к другу. Принципы арбитража в системе MULTIBUS позволяют медленным ведущим равноправно конкурировать за захват магистрали. Однако после того, как модуль захватил магистраль, скорость передачи определяется возможностями передающего и принимающего модулей.

Основное назначение магистрали MULTIBUS состоит в обеспечении канала для передачи данных между модулями, подключенными к шине. Система позволяет использовать платы с различными возможностями, изменять ширину шин данных и адресов ввода-вывода, устанавливать атрибуты прерываний.

Для реализации мультипроцессорных возможностей системы, построенной на основе МП 80386, и для увеличения ее производительности разработана магистраль MULTIBUS II. В новую архитектуру включена передача сообщений, способствующая повышению производительности мультипроцессорной системы. При использовании передачи сообщений все пересылки по магистрали выполняются с максимально возможной скоростью пакетами 32-разрядных данных.

В дополнение к передаче сообщений модули платы MULTIBUS II обеспечивают виртуальные прерывания, географическую адресацию и распределенный арбитраж. При наличии виртуальных прерываний один процессор может выполнять запись в специальные ячейки памяти другого процессора, что почти неограниченно увеличивает гибкость механизма прерываний.

Географическая адресация, реализуемая с помощью смонтированных на плате регистров межкомпонентных соединений, обеспечивает пространство межкомпонентных соединений для программных конфигураций законченных комплексных систем. Распределенный арбитраж предоставляет модулям MULTIBUS II столько отдельных уровней арбитража, сколько в системе имеется плат (или гнезд). В этом случае все платы в системе имеют одинаковый приоритет относительно времени доступа к магистрали, что предотвращает блокирование плат с низким приоритетом ведущими платами высокой производительности.

Ключевым вопросом при построении систем на основе магистрали MULTIBUS является нахождение оптимального соотношения между *требуемыми* и *фактическими* характеристиками. Для каждого элемента характерно индивидуальное множество присущих ему характеристик. Взаимодействие двух таких элементов ограничивается множеством характеристик, которое определяется как пересечение множеств характеристик обоих элементов. В некоторых случаях пересечение может быть пустым, что приводит к принципиальной неработоспособности системы.

12.6. Электрическое питание

Современные устройства требуют хорошо стабилизированного постоянного напряжения. Стабилизация питания может осуществляться в источнике питания или, как это имеет место в системах S-100, на каждой плате.

Большинство компьютерных систем питается от источника постоянного напряжения 5 В с выходным током от 1 до 5 А. Типичными значениями являются 1, 3, 4 и 5 А в зависимости от системы. Некоторые системы потребляют ток 10 А. Часто источники питания имеют также выход 12 В, 2 А для подключения дисководов.

12.7. Резюме

Под сопряжением понимается связь микропроцессора с внешним миром и с его собственными внутренними узлами. Чем большим уровнем интеллекта обладает устройство, тем большее внимание надо обращать на характеристики сопряжения.

Сопряжение относится к числу наиболее сложных аспектов разработки аппаратного обеспечения. Именно здесь приходится принимать большинство компромиссов и именно здесь можно получить максимальную экономию средств. В большинстве приложений сигналы, обрабатываемые процессором, несовместимы с его характеристиками. В простейшем случае может потребоваться преобразование напряжения сигнала в стандарт транзисторно-транзисторной логики (ТТЛ) 5 В. Несовместимое напряжение может легко исказить процесс обработки сигналов. В других случаях могут потребоваться аналого-цифровые преобразователи.

Глава 13

Организация запоминающих устройств

Память переносит информацию сквозь время. Процесс этот идет всегда в одном направлении – вперед. В компьютерной системе информацию можно хранить различными способами; отождествим память с каналом, в котором данные сохраняются для использования в будущем.

Мельчайшей единицей информации, сохраняемой в цифровой системе, является двоичный разряд, или бит: двоичные единица (1) или нуль (0). Бит хранится в 1-разрядном регистре – *триггере*. Триггером называется ячейка памяти, имеющая только два стабильных *состояния*, в которых она может находиться до тех пор, пока не будет выключено напряжение питания. Входные сигналы могут изменять состояние этой ячейки памяти. На рис. 13.1 изображен статический МОП-триггер – элемент регистра.

Тысячи таких 1-разрядных регистров можно объединить в одну интегральную схему с общей тактовой линией и общими наборами входов и выходов. Эта микросхема носит название запоминающего устройства или микросхемы *памяти*. Емкость памяти обычно определяется как максимальное число хранимых в ней байтов.

Микропроцессор представляет собой систему модулей, включающую процессор, память и узлы ввода-вывода. Для достижения большей гибкости процессор можно также разбить на подузлы. Один из возможных подходов заключается в разделении частей процессора, работающих с данными и командами. Часть про-

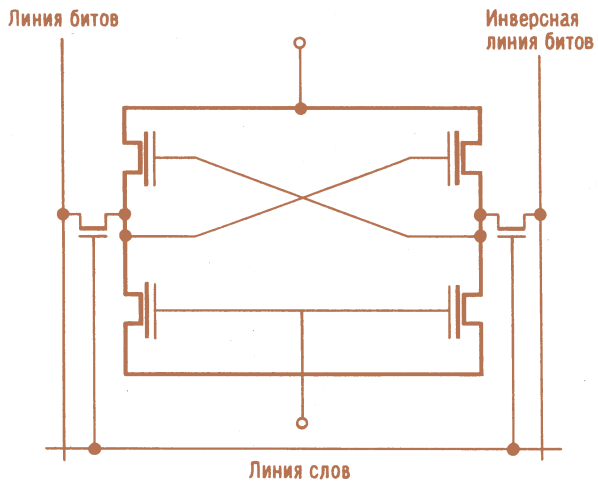


Рис. 13.1. Электрическая схема статического МОП-триггера.

цессора, работающая с данными, включает арифметико-логическое устройство (АЛУ) и различные регистры данных, причем она образуется путем каскадирования нескольких идентичных узлов. Каждый из этих узлов называется *микропроцессорной секцией* и состоит из АЛУ и регистров.

Для того чтобы из микропроцессорных секций образовать законченный микропроцессор, требуется добавить некоторые дополнительные компоненты, такие как программный счетчик, регистр команд, дешифратор команд, узлы синхронизации и управления. Все эти компоненты нужны для обработки последовательности команд и их дешифрации. С их помощью команды извлекаются из основной памяти в соответствии с адресом в программном счетчике и пересылаются в регистр команд.

13.1. Реальная память

В настоящее время память реализуется на КМОП-транзисторах по технологии СБИС (в гл. 11 даны сведения о физическом составе КМОП-структур). Некоторые изготовители включают память ограниченного объема в состав микросхемы процессора. Чаще, однако, «земельный участок» ЦП предпочитают не расходовать понапрасну и память (за иключением кэш-памяти) размещают на отдельных микросхемах.

13.2. ПЗУ и ОЗУ

Полупроводниковые запоминающие устройства разделяются на два принципиально различающихся класса: программно изменяемые (запоминающие устройства с произвольной выборкой, ЗУПВ)[1] и неизменяемые (постоянные запоминающие устройства, ПЗУ).

Оперативные запоминающие устройства (ОЗУ). ОЗУ состоит из ячеек памяти, содержимое которых может изменяться программным образом. Используются два типа ОЗУ: *статические* и *динамические*. В статических ОЗУ каждый элемент памяти представляет собой триггер, который можно устанавливать, сбрасывать и хранить, таким образом, двоичные 1 или 0. Состояние триггера не изменяется до тех пор, пока в него не будет записана новая информация или не выключится питание ОЗУ.

В динамических ОЗУ в качестве запоминающих элементов используются конденсаторы. Наличие на конденсаторе заряда соответствует логической 1; отсутствие – 0. Такой принцип хранения информации упрощает ячейку памяти и позволяет располагать их более плотно, однако приходится бороться с утечкой заряда конденсатора, которая происходит за время порядка нескольких миллисекунд. Для коррекции утечки динамические ОЗУ должны периодически *регенерироваться*.

Регенерация динамического ОЗУ заключается в чтении содержимого последовательных ячеек памяти в течение некоторого времени. Считываемые данные автоматически записываются назад в те же ячейки. В результате все ячейки, в которых записана 1, полностью восстанавливают свой заряд, те же, где записан 0, остаются разряженными. Обычно такие ОЗУ регенерируются по крайней мере каждые 2 мс.

Постоянные запоминающие устройства (ПЗУ). ПЗУ предоставляют возможность постоянного (долговременного) хранения программ и данных, поскольку информация в ПЗУ сохраняется даже при выключении питания. Для изготовления ПЗУ

---

[1] Поскольку ЗУПВ используются прежде всего в качестве оперативных запоминающих устройств (ОЗУ), именно это обозначение используется в настоящем переводе. – *Прим. перев.*

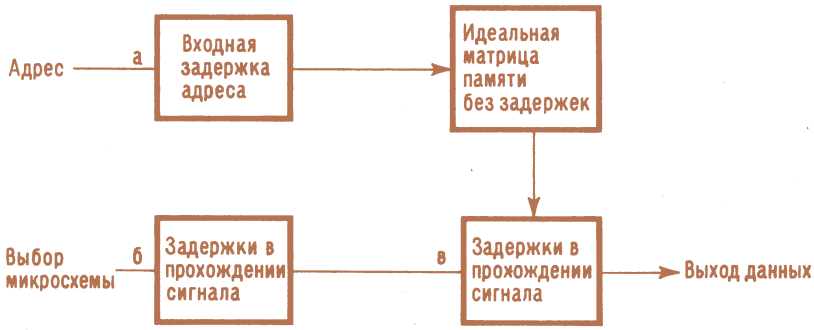


Рис. 13.2. Упрощенная модель ПЗУ. *а* – включает все задержки, связанные с входными адресными буферами, дешифраторами столбцов и строк, матрицей памяти; *б* – включает все задержки, связанные с логикой выбора микросхемы; *в* – включает все задержки, связанные с выходными буферами.

наиболее удачной оказывается МОП-технология, обеспечивающая чрезвычайно плотное размещение элементов ПЗУ. На рис. 13.2 показана упрощенная модель ПЗУ.

Существуют четыре типа ПЗУ: с масочным программированием, программируемые ПЗУ (ППЗУ), репрограммируемые ПЗУ (РПЗУ) и электрически репрограммируемые ПЗУ (ЭРПЗУ).

В ПЗУ с масочным программированием программа заносится изготовителем микросхемы на основе спецификации заказчика (или своей собственной). Эти ПЗУ обычно используются в изделиях, выпускаемых крупными сериями, поскольку они дешевле других ПЗУ [1)] и отличаются максимальной плотностью битов.

ППЗУ программируются пользователем электрически с помощью специального устройства, называемого программатором. После того как программа записана в ППЗУ, изменить ее уже нельзя. Такие ПЗУ удобны в тех специальных случаях, когда пользователю требуется в принципе изменяемая программная среда, которая, однако, после установки должна сохраняться определенное время в неизменном виде.

РПЗУ схожи с ППЗУ, за тем исключением, что их можно стирать и репрограммировать. Биты программы храняться в виде заряда на конденсаторе с ничтожно малой утечкой. Стирание выполняется путем облучения ультрафиолетовым светом через прозрачное окошко в корпусе микросхемы. РПЗУ удобно использовать в опытных образцах или мелкосерийных изделиях.

ПЗУ четвертого типа, ЭРПЗУ, можно стирать, не вынимая из схемы. Большим преимуществом ЭРПЗУ перед РПЗУ является возможность стирания небольших областей ПЗУ, в то время как РПЗУ можно стереть только целиком. Типичными применениями ЭРПЗУ являются калиброванные преобразователи, автоматические телефонные номеронабиратели и цифровые телевизионные тюнеры.

13.3. Системы памяти

Хранение информации в вычислительной системе не может быть обеспечено одним ЗУ; для получения системы памяти следует объединить несколько устройств. Так,

---

[1)] Только в условиях массового выпуска. – *Прим. перев.*

емкость системы можно увеличить, увеличивая число слов и/или длину слова по сравнению с тем, что дает одно устройство.

Длина слова увеличивается, если информация параллельно снимается с выходов двух или более ЗУ. Число слов в системе памяти увеличивается при мультиплексировании выходов двух или более ЗУ. Большинство ЗУ содержат средства для такого мультиплексирования.

В системе памяти должны быть предусмотрены возможности расширения адреса. Другими словами, должен быть какой-то способ увеличения числа битов в адресах, на которые отзывается система памяти. Число адресных битов, обеспечиваемое процессором, определяет физическое *пространство адресов памяти.*

13.3.1. Организация памяти

ЗУ разрабатываются таким образом, что каждое отдельное устройство содержит все основные компоненты: 1) матрицу ячеек памяти, каждая из которых может хранить один бит; 2) логические схемы адресации ячеек памяти; 3) схемы чтения содержимого любой ячейки памяти; 4) схемы записи в ячейки памяти, если используемое запоминающее устройство допускает запись в память.

Для облегчения подключения запоминающих устройств к другим логическим или запоминающим устройствам они содержат входные формирователи, выходные буфера и схемы расширения адреса. С целью достижения высокой скорости работы, большой емкости и низкой стоимости самой матрицы памяти и периферийных устройств используются различные способы внутренней организации памяти.

Пословная организация с линейной выборкой. В простейшем случае матрица памяти имеет *пословную организацию с линейной выборкой.* Другими словами, длина столбцов матрицы памяти совпадает с числом слов, а длина строки – с числом битов в слове. Для выборки слов используется дешифратор с взаимоисключающими выходными сигналами для каждого слова памяти. Для каждого входного адреса такой дешифратор выбирает одно и только одно слово в матрице памяти. Метод линейной выборки, отличаясь в принципе простотой, требует для большого числа слов большого дешифратора.

Двухступенчатая дешифрация. Такая организация матрицы памяти и логики выборки, при которой возможно использование *двухступенчатой* дешифрации, позволяет уменьшить размер дешифратора. В этом случае одна ступень матрицы памяти соответствует физическому слову, а другая – логическому слову. Физическое слово образуется битами, содержащимися в строке матрицы. Логическое слово включает те биты физического слова, которые выбираются и направляются на выход за одно обращение к памяти.

При двухступенчатой дешифрации дешифратор строк выбирает физическое слово, а дешифратор столбцов – одно логическое слово из выбранного физического слова.

Одноразрядная организация. Часто используется способ организации памяти, при котором число сегментов физического слова совпадает с числом битов. Другими словами, длина логического слова составляет 1 бит. Для одноразрядной памяти характерны один выход, квадратная матрица памяти и равные по сложности дешифраторы строк и столбцов.

13.3.2. Защита памяти

Основной проблемой в любой системе с перекрывающимися запросами на использование памяти является защита содержимого памяти от несанкционированного доступа или модификации. Обычно наблюдение за тем, чтобы отдельные процессы

не мешали друг другу и в то же время защищали себя, ведется операционной системой в соответствии с тем или иным заложенным в нее алгоритмом.

В простейшей системе прикладные программы используют одни области памяти, а операционная система – другие и оба типа программ имеют доступ к своим областям. В таких системах обычно предполагается, что любой адрес, сформированный операционной системой, является «хорошим» и не нуждается в проверке. Любой адрес, сформированный прикладной программой, является «подозрительным» и проходит проверку, чтобы удостовериться в том, что он попадает в разрешенную область. Часто эта проверка выполняется аппаратными средствами.

Для того чтобы аппаратные средства могли выполнить эту проверку, им должно быть известно, где начинается и кончается операционная система, а также какая программа выполняется в настоящий момент – прикладная или операционной системы. Наиболее распространенным способом удовлетворения этих требований является введение двух режимов выполнения программ: *режима пользователя* и *привилегированного режима*.

В режиме пользователя разрешен доступ лишь к ограниченной области адресов, отведенных под программы пользователя. В привилегированном режиме доступны все адреса памяти. Переход из одного режима в другой происходит по прерыванию, либо при выполнении специальной команды «Вход в операционную систему».

13.3.3. Адресация памяти

В системе памяти с произвольным доступом можно прямо адресоваться к любой ячейке памяти. Каждой ячейке памяти, состоящей из фиксированного числа запоминающих элементов, присвоен идентификатор, являющийся *адресом* этой ячейки. Адрес позволяет отличить одну ячейку от другой в процессе выполнения операций чтения или записи.

Прямая адресация объединяет в одном слове код команды и исполнительный адрес, что позволяет команде с обращением к памяти выполняться за две машинные фазы: выборки и выполнения. Местоположение операнда команды определяет вид операнда: регистр-источник или регистр-приемник. Обычно прямой адресации доступны только две страницы (нулевая и первая) статического или динамического ОЗУ.

Часто и прикладные программы, и программы операционной системы используют одну и ту же реальную память и при этом должны быть защищены друг от друга. Одним из способов защиты является использование регистра границы. Этот способ, однако, имеет ограниченную пригодность, если несколько прикладных программ совместно и многократно используют одно и то же пространство адресов. Наиболее удовлетворительным решением вопроса является применение какого-то механизма перемещения в самой аппаратуре доступа к памяти. В этом случае любой прикладной или внутренний процесс работает с *виртуальной памятью*, которая простирается от базового адреса до некоторой границы.

Преобразование виртуальных адресов в реальные выполняется при каждом обращении к памяти в режиме пользователя. Для этого в микропроцессоре должны иметься соответствующие аппаратные средства, связывающие ЦП и память. Наиболее эффективные аппаратные средства имеют в своем составе регистры базы и границы, позволяющие выполнять преобразование и проверку на границу. На рис. 13.3 показана характерная структура такой аппаратуры.

Перед обращением к памяти в регистр базы заносится начальный адрес области памяти, выделенной данному процессу, а в регистр границы – размер этой области. Заполнение регистров базы и границы выполняется вместе с инициализацией остальных регистров при запуске процесса. В режиме пользователя преобразование

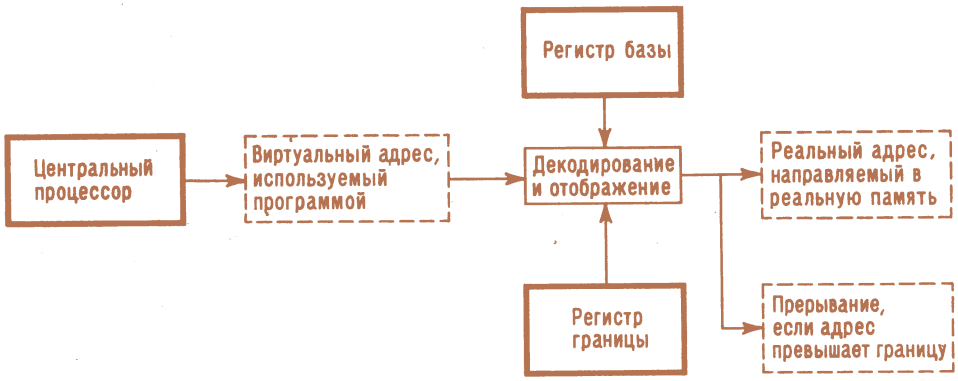


Рис. 13.3. Регистры базы и границы.

всех адресов осуществляется путем прибавления к виртуальному адресу, генерируемому программой, содержимого регистра базы. Образующийся в результате реальный адрес используется для обращения к памяти.

13.3.4. Диспетчер памяти

Многие вычислительные системы имеют в своем составе тот или иной вариант диспетчера памяти [1], с помощью которого многозадачная операционная система выделяет память каждой задаче и обеспечивает защиту от программ пользователей. Например, типичная проблема возникает в том случае, когда прикладная программа делает ошибку при вычислении адреса, возможно используя слишком большое или слишком маленькое значение индекса. Если в системе нет защиты, такого рода ошибка может привести к изменению кодов, входящих в программы операционной системы, или к модификации таблиц устройств и даже вызвать незапланированный запуск устройства с фатальными последствиями, такими как несанкционированная запись в файл с данными.

Помимо защиты операционной системы от неумышленного разрушения диспетчер памяти обеспечивает автоматическое перемещение программы. С помощью диспетчера памяти логические или программные адреса программ пользователей преобразуются в физические, или аппаратные адреса. Эти адреса могут располагаться в памяти совершенно не там, куда указывают логические адреса. Преобразование адресов приводит к полному выводу операционной системы и защищенных устройств ввода-вывода из адресного пространства пользователя. Любая попытка чтения или записи в память вне пространства адресов, отведенных пользователю, заставляет процессор аварийно завершить программу пользователя.

Информация обычно передается порциями, состоящими из фиксированного числа битов; МП 80386 использует порции размером 32 бит, если только он не получил указаний об уменьшении размеров порции до 16 бит. Эти порции носят названия слов. Процесс записи слова в запоминающую систему называется *записью в память*; процесс получения слова из памяти – *чтением из памяти*.

Существуют два метода доступа к памяти: произвольный и последовательный. Последовательный доступ используется в тех запоминающих системах, где обращение к словам осуществляется в некотором определенном заранее порядке. Произвольный доступ, с другой стороны, предполагает возможность обращения к

[1] Другое название – устройство управления памятью (УУП). – *Прим. перев.*

словам запоминающей системы в любом порядке и приблизительно за то же время.

Каким бы совершенным ни было ЗУ, при прохождении сигнала между устройствами возникают задержки в его распространении. Задержка распространения определяется как время, требуемое для прохождения логического сигнала через устройство или последовательность устройств, образующих логическую цепочку. Эта задержка также учитывает прохождение сигнала по всем соединительным линиям между микросхемами.

13.3.5. Прямой доступ к памяти (ПДП)

Прямой доступ к памяти предоставляет возможность внешним устройствам и памяти непосредственно обмениваться данными без вмешательства программы. ПДП обеспечивает максимальную скорость ввода-вывода и максимальный параллелизм процессов. В то время как ввод-вывод по прерываниям и программно-управляемый ввод-вывод пересылают данные через процессор, в случае ПДП данные передаются непосредственно между устройством ввода-вывода и памятью.

Чтобы свести к минимуму число шин данных, принимаются специальные меры, позволяющие использовать обычную магистраль для ПДП. Меры эти заключаются в том, что процессор освобождает магистраль, а внешнее устройство захватывает ее и использует для передачи данных.

На время ПДП выполнение программы обычно приостанавливается. Магистраль освобождается, как только возбуждается управляющая линия запроса ПДП. Процессор завершает текущую операцию, освобождает линии адреса и данных и устанавливает сигнал на одной из линий управления, чтобы предотвратить непреднамеренную расшифровку неопределенных сигналов управления.

Интерфейс ввода-вывода пересылает данные непосредственно в память с помощью специального регистра. Когда возникает необходимость в передаче данных, интерфейс запрашивает у процессора цикл памяти. Получив подтверждение, интерфейс пересылает данные непосредственно в память, в то время как процессор останавливается на один цикл памяти. Логика, выполняющая эту пересылку, называется *каналом*.

Канал содержит регистр для хранения адреса ячейки памяти, в которую или из которой осуществляется пересылка данных. В большинстве случаев в канал входит также счетчик слов для отсчета числа выполненных прямых пересылок. Кроме того, канал должен содержать схему, обеспечивающую обмен управляющими сигналами, синхронизацию и другие вспомогательные операции. На рис. 13.4 показаны логические связи между ЦП и контроллером ПДП.

Компоненты ПДП. Основными компонентами ПДП являются триггер запроса, регистр адреса, счетчик и регистр данных, который используется периферийным устройством. Пересылка данных по каналу ПДП включает несколько этапов: 1) инициализация логики ПДП для выполнения ПДП в течение повторяющихся циклов занятия магистрали; 2) асинхронная по отношению к операциям программы активизация ПДП; 3) оповещение об окончании пересылок (по сигналу от счетчика или в результате изменении состояния устройства); 4) вызов программы «очистки» при окончании пересылок; 5) завершение ПДП с помощью программы обработки прерываний, которая возвращает управление в основную программу.

Блочная пересылка данных. Процедура ПДП для высокоскоростных устройств использует пересылку данных блоками. Выполняя текущую программу, процессор инициирует пересылку блока данных и определяет число слов, составляющих блок. Однако фактическая передача слов выполняется под управлением отдельного устройства – контроллера ПДП. Максимальная скорость блочной передачи по

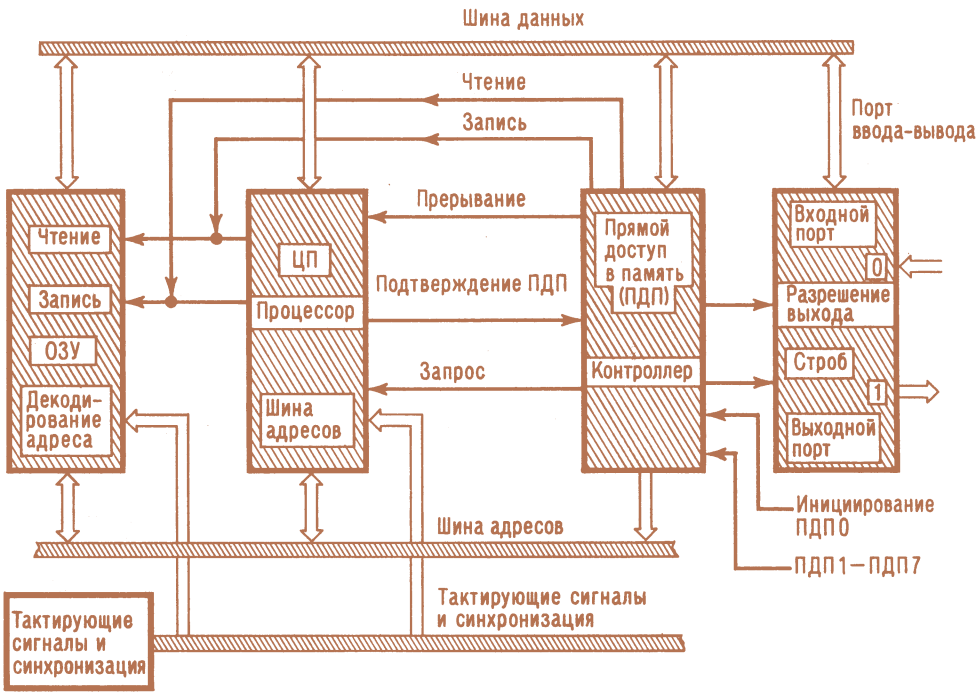


Рис. 13.4. Логические взаимосвязи между контроллером ПДП и ЦП.

каналу ПДП ограничивается только длительностью цикла памяти (чтения или записи) и скоростью работы контроллера ПДП.

ПДП с занятием цикла памяти. Программа инициирует пересылку блока, помещая стартовый адрес в адресный счетчик, а число слов – в счетчик слов, и выдавая команду запуска. Этот тип ПДП часто называют ПДП с *занятием цикла памяти*, поскольку он приостанавливает выполнение программы каждый раз приблизительно на один машинный цикл.

При использовании ПДП с занятием цикла памяти пересылка данных осуществляется параллельно с другими процессами, выполняемыми ЦП. Последовательность действий здесь такая же, как и при блочной передаче, за тем исключением, что контроллер ПДП занимает циклы памяти у процессора и, следовательно, замедляет его работу [1].

13.3.6. Сегментация памяти

Сегмент – это маленькая часть большого целого. Сегментация – это деление памяти на сегменты. С точки зрения разработчиков программного обеспечения, сегментация дает удобный способ совместного использования информации несколькими процессами. Конкретный сегмент может использоваться совместно с другими без нарушения требований его защиты.

Сегментация также предполагает естественное разделение программных строк и данных и отделение модуля от модуля.

---

[1] Блочная передача данных по каналу ПДП также занимает циклы памяти, если только не используется ПДП по отдельной шине. – *Прим. перев.*

С точки зрения разработчиков аппаратного обеспечения, сегментация обозначает также разделение памяти на физические блоки, называемые ПЗУ и ОЗУ. Если, например, каждое ПЗУ имеет объем 64К, тогда и размер каждого сегмента составляет 64К. Дополнительные цепи, физические регистры, триггеры-фиксаторы и прочее оборудование компонуется вокруг этих сегментов.

При разработке МП 8080 была реализована возможность адресации вплоть до 65536 байтов (64К), и это число определило максимальный размер памяти. В течение сравнительно длительного периода времени величина 64К служила стандартом при разработке микропроцессоров. МП 8086 изменил ситуацию, введя программную сегментацию, с помощью которой сегменты размером 64К отображались на адресное пространство объемом более 1 Мбайт.

При использовании МП 80386 сегменты физических адресов могут иметь размеры от 1 байт до 4 Гбайт. В виртуальной памяти МП 80386 допустима адресация до 64 Тбайт, причем каждому сегменту придается индивидуальный уровень защиты или приоритета.

13.3.7. Виртуальная память

В простой системе памяти логическое и физическое пространства адресов совпадают. В этом случае индивидуальные программы практически не требуют управления памятью. Система виртуальной памяти использует отображение логического пространства адресов (суммарный объем памяти, требуемый для всех программ и процессов) на первичную физическую память (память, физически существующую в системе).

Если, например, используются 4096 байт памяти, то в любой момент времени все 4096 байт могут быть физически и непосредственно адресованы. Однако эти 4096 байт не обязательно один к одному соотносятся с байтами физической памяти. Операционная система может дать указание аппаратуре, что, начиная с некоторого момента, при обращении по адресу 4096 должна использоваться ячейка памяти с адресом 0 и т. д. Другими словами, операционная система определяет отображение адресного пространства на реальные адреса памяти. На рис. 13.5 показано отображение части адресного пространства (4К) на адреса основной памяти.

В каждый момент времени логическое пространство адресов может находиться в первичной физической памяти целиком или частично. Части адресного пространства, не отображенные на основную память, обычно хранятся в иерархически

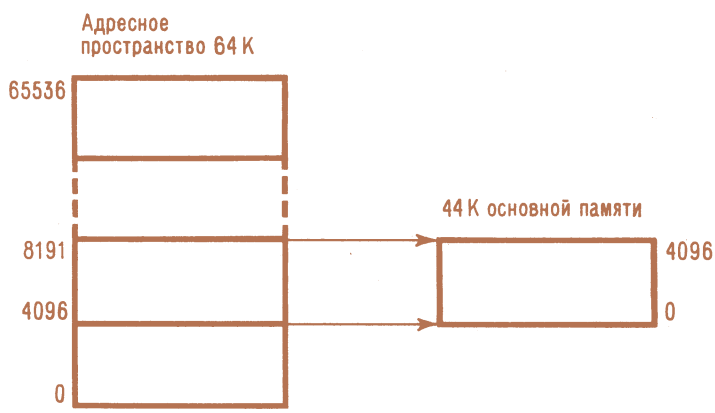


Рис. 13.5. Отображение адресов на память.

организованной вспомогательной памяти, такой как дисковое пространство. Когда возникает потребность в логических адресах, не находящихся в первичной области, они извлекаются из места хранения и переносятся в первичную память, при этом выполняется отображение логических адесов на физические.

Управление виртуальной памятью. Управление памятью представляет собой применение стратегии и алгоритмов распределения вычислительных ресурсов к распределению ресурса памяти между параллельными процессами. Управление виртуальной памятью включает настройку адресов, т. е. отображение адресного пространства, используемого процессом, на физическое адресное пространство. Основа распределения памяти, *динамическая настройка*, и есть это отображение: помещение логической информации программы в физическую память. Отображение должно быть выполнено перед тем, как начнут выполняться какие-либо команды программы.

В *сегментированной виртуальной системе памяти* полное логическое пространство адресов делится на меньшие логические сегменты, которые могут иметь любой размер. Для того, чтобы обратиться к конкретному элементу внутри сегмента, должны быть определены как имя сегмента, так и символическое имя элемента внутри сегмента или его адрес.

Когда какой-либо сегмент переносится в первичную память, он помещается в область памяти, равную или большую по размеру, чем сам сегмент. Поскольку сегмент заранее никак не связан с его физическим расположением, создается таблица сегментов, в которой содержатся базовые адреса всех сегментов, находящихся в памяти. Затем ссылка на сегмент отображается системой в номер сегмента, представляющий собой смещение в таблице сегментов. Дополнительное отображение дает индекс в сегменте, позволяющий обратиться в конкретному элементу.

Виртуальная память с разбиением на страницы. В *виртуальной системе памяти, разбитой на страницы*, пространство первичных адресов разделено на блоки равного размера, называемые страницами. Страница представляет собой блок программных строк и/или данных, расположенных по последовательным адресам. Поскольку пространство логических адресов также разделено на страницы того же размера, при пересылке информации в физическую память единицей пересылки является страница и заполняется при этом кадр страницы.

Если память распределена неудачно или при распределении использовались блоки переменного размера, может возникнуть фрагментация памяти, т. е. образование свободных участков слишком малого размера и неудобных для использования. Фрагментация снижает производительность системы, поскольку ЦП вынужден откладывать диспетчеризацию и реорганизовывать физическую память, перемещая или перекачивая через диск другие задачи, пока в памяти не образуется достаточно большого слитного пространства, чтобы можно было продолжить вычисления.

Для решения проблемы фрагментации предложено проводить распределение памяти блоками фиксированного размера с автоматическим наложением блоков на адреса реальной памяти. Этот процесс получил название разбиения на страницы. Участки программы, считываемые из вторичной памяти, называются страницами точно так же, как и в виртуальной системе с разбиением на страницы, описанной выше. Участки основной памяти, куда помещаются страницы, носят название *кадров страниц*.

Разбиение на страницы виртуального пространства системы предназначено прежде всего для борьбы с многочисленными проблемами, возникающими из-за фрагментации памяти. Основной недостаток разбиения на страницы – неэффективное использование пространства памяти – проявляется, когда требуются участки

очень маленького размера, поскольку наименьшей единицей распределения памяти является страница.

Программы составляются в предположении, что в реальной памяти имеется достаточно места для всего требуемого пространства. Внутренняя логика аппаратного обеспечения так взаимодействует с операционной системой, что программист может игнорировать факт загрузки страниц в память поверх ранее использовавшихся страниц. В отличие от этого при сегментировании программист должен помнить, как используется каждый сегмент и как он адресуется.

Поиск страницы, которая требуется для выполнения очередной команды, осуществляется обычно с помощью таблицы, называемой *таблицей страниц*. В таблице содержатся начальные адреса каждой страницы, а также биты, показывающие, находится ли в настоящий момент конкретная страница в памяти. Если страница уже загружена, аппаратура настраивает указатель на начало страницы и продолжает выполнение программы. Если страница не загружена, аппаратура возбуждает прерывание, чтобы операционная система могла найти страницу во вспомогательной памяти, загрузить ее в память реальную и продолжить вычисления.

Если программа обращается к странице, отсутствующей в основной памяти, возникает ошибка, называемая *отказом страницы*. Зафиксировав отказ страницы, операционная система считывает требуемую страницу из вторичной памяти, заносит ее новый физический адрес в таблицу страниц, затем повторяет команду, вызвавшую отказ.

13.3.8. Кэш-память

Кэш-память (рис. 13.6) представляет собой быстродействующую буферную память, в которую из основной памяти с умеренной скоростью переносятся команды и данные. В вычислительной системе, использующей кэш-память, команды и данные программы хранятся в основной памяти, а некоторые выбранные участки программы копируются в кэш-память. На рис. 13.6 изображена структурная схема кэш-памяти МП 80386.

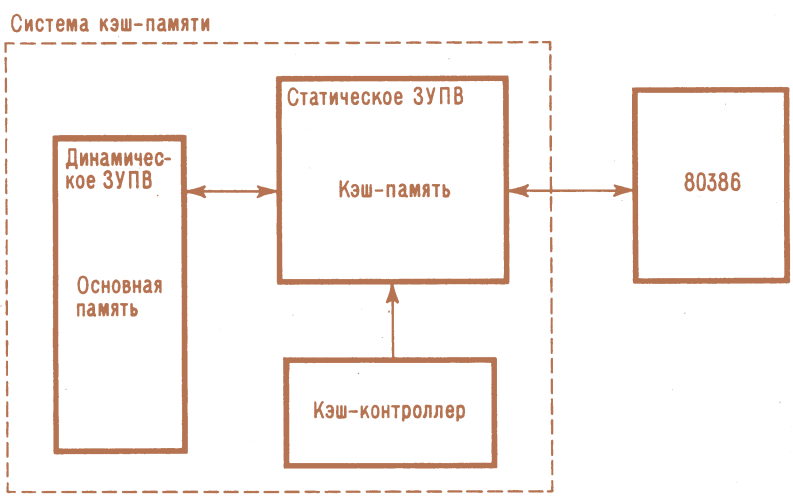


Рис. 13.6. Структурная схема системы кэш-памяти.

Кэш-память была предложена, когда появились процессоры, работающие значительно быстрее ОЗУ и вынужденные ожидать, пока из памяти будут введены требуемые элементы данных. В системах с кэш-памятью информация пересылается по цепочке: вторичная память – основная память – кэш-память. Чтобы информация попала в кэш-память, нужно выполнить следующие действия.

Программа, обращаясь к своему адресному пространству, передает аппаратуре или микропрограмме виртуальный адрес. Если страница или сегмент, содержащие этот адрес, находятся в основной памяти, виртуальный адрес преобразуется в адрес основной памяти. В противном случае возникает отказ страницы, и страница или сегмент загружаются в основную память.

После этого аппаратура кэш-памяти проверяет, не находится ли требуемый блок уже в кэш-памяти. Если так и оказалось, содержимое адресуемой ячейки извлекается из кэш-памяти, либо, наоборот, результат вычислений заносится в кэш-память. В противном случае возбуждается отказ кэш-блока (аналогичный отказу страницы), и информация копируется из основной памяти в кэш-память. После загрузки блока в кэш-память команда выполняется повторно. Поскольку все эти процессы должны происходить очень быстро, система загрузки из основной в кэш-память управляется внешней по отношению к ЦП логикой без участия операционной системы.

Выбор информации для кэш-памяти. Процесс выбора является ключевым для эффективного использования процессора. Если бы обращение к памяти осуществлялось случайным образом, предсказание адреса следующей адресуемой ячейки памяти было бы невозможным. Однако обращение к памяти чаще всего не носит случайного характера. Большинство программ обращается к памяти вблизи той ячейки, куда было обращение перед этим. Это явление носит название *локальности программы*. Именно локальность делает кэш-систему эффективной.

Например. переменные программы обычно организуются в виде некоторой логической последовательности (часто восходящей). В результате следующая считываемая переменная располагается непосредственно за предыдущей. Символьные строки чаще всего просматриваются последовательно. Стек изменяется с одного конца, так что несколько соседних обращений делаются в окрестностях вершины стека.

Когда процессору требуется следующая команда или следующий элемент данных, он сначала обращается к кэш-памяти, перед тем, как затребовать данные из более далекой и медленной памяти. Доступ, при котором данные обнаруживаются в кэш-памяти, называют *кэш-попаданием*. В противном случае фиксируется *кэш-промах*. Коэффициент кэш-попаданий – это доля обращений, давших попадания. На величину коэффициента кэш-попаданий влияют размеры и организация кэш-памяти, алгоритмы поиска и характеристики выполняемой программы.

Выборка блока. Кэш-память снабжается контроллером, в функции которого входит достижение максимальной эффективности кэш-памяти и управление ее ресурсами. Контроллер разбивает основную память на блоки с типичным размером 2, 4, 8 или 16 байт. 32-разрядный процессор обычно использует блоки размером 2 или 4 слова. Когда фиксируется кэш-промах, кэш-контроллер перемещает в кэш-память из основной памяти требуемое слово вместе с целым блоком, содержащим это слово.

Выборка блока с данными может осуществляться одним из трех способов: 1) с упреждением – когда извлекаются данные, расположенные за требуемым байтом; 2) с отставанием – когда извлекаются данные, расположенные перед требуемым байтом; 3) и с тем и с другим. Обычно блоки выравниваются на границу двойного слова. Обращение к любому байту блока приводит к копированию в кэш-память всего блока.

Размер блока является важнейшим параметром системы кэш-памяти. Если величины упреждения и отставания невелики, снижается коэффициент кэш-попаданий. Выбор блоков слишком большого размера уменьшает число блоков, хранящихся в кэш-памяти. К тому же по мере увеличения размера блока каждое новое слово становится все дальше от требуемого и вероятность его реального использования падает (из-за локальности программы). Наконец, передача блоков большего размера требует более широкой шины между основной и кэш-памятью, что повышает стоимость системы.

Организация кэш-памяти. Кэш – это быстродействующая память, используемая примерно так же, как мы используем записную книжку-еженедельник. Если нам вскоре понадобится телефонный номер, мы записываем его в еженедельник. Когда необходимость помнить этот номер отпадает, он стирается, а освободившееся место используется для чего-то другого. От способа организации такой «записной книжки» зависит, как система будет ее использовать. Существуют три основных способа организации кэш-памяти: полностью ассоциативный, с прямым отображением и частично ассоциативный.

Полностью ассоциативная кэш-память. В этом случае в кэш-памяти содержатся блоки, наиболее часто используемые процессором. Однако между адресами всех этих блоков нет однозначной зависимости, поэтому в кэш-памяти кроме самих блоков надо хранить еще и их полные адреса.

Когда процессор запрашивает данные, кэш-контроллер сравнивает адрес требуемых данных с *каждым* адресом в кэш-памяти. Если обнаружено соответствие, данные, расположенные по этому адресу, направляются в процессор. Каждое сравнение адресов дает либо попадание, либо промах. Поскольку адреса не имеют взаимозависимости, в случае промаха можно поступить двояко: либо продолжать медленный последовательный поиск, либо использовать более дорогую аппаратуру для параллельного поиска, содержащую компараторы и контроллеры.

Кэш-память с прямым отображением. В этом случае каждый блок из основной памяти может занимать в кэш-памяти одно-единственное место. Это уменьшает число шагов поиска и сравнений, необходимых для фиксации попадания или промаха, до одного.

Каждый адрес состоит из двух частей. Первая часть представляет собой поле индекса кэш-памяти. Эта часть должна иметь достаточный размер, чтобы вместить адрес блока в кэш-памяти. Вторая часть, поле признака (тега), содержит биты, позволяющие различать блоки, которые могут быть записаны в конкретную ячейку кэш-памяти.

Основной недостаток способа прямого отображения заключается в том, что контроллер выделяет для конкретного участка основной памяти единственное место в кэш-памяти. Если программа попеременно обращается к двум конфликтующим в кэш-памяти ячейкам, в кэш-памяти каждый раз будет осуществляться подкачка. Однако такая закономерность при выполнении программы встречается редко, поэтому кэш-система с прямым отображением обладает приемлемой производительностью при более низкой по сравнению с полностью ассоциативной системой стоимостью.

Частично ассоциативная кэш-память. Такой способ организации является промежуточным между кэш-памятью с прямым отображением и полностью ассоциативной кэш-памятью. Частично ассоциативная кэш-память характеризуется наличием нескольких наборов блоков с прямым отображением, которые функционируют параллельно. Каждый индекс кэш-памяти указывает на несколько блоков, по одному в каждом наборе. Когда из основной памяти поступает блок данных, он может быть записан на соответствующее этому блоку место любого набора.

Используя память того же объема, частично ассоциативная кэш-память со-

держит вдвое меньше отображаемых блоков, но каждый блок может соответствовать двум адресам, которые в одновходовой кэш-памяти были бы конфликтующими. Наиболее эффективными в плане достижения компромисса между быстродействием и стоимостью оказываются 2- и 4-входовые кэш-системы.

При выборке нового блока кэш-контроллер должен решить, на место какого блока в кэш-памяти его записать. Здесь есть три возможности, адекватность которых определяется поведением программы: 1) случайный выбор; 2) запись в последовательном порядке; 3) запись на место блока, который не адресовался дольше других. Последний алгоритм требует наличия в контроллере «счетчика возраста», который указывает, на место какого блока можно записывать новый. Этот счетчик должен модифицироваться каждым запросом к кэш-памяти.

Обновление кэш-памяти. Важной особенностью, которую следует иметь в виду при использовании кэш-памяти, является наличие двух копий данных – одной в кэш-памяти, а другой – в основной памяти. Если одна из этих копий модифицирована, а другая – нет, возникают два различных набора данных с одним адресом памяти, что может привести к сохранению ошибочных данных, если сохраняется не тот набор.

Наличие двух копий требует механизма обновления, который предотвращал бы использование устаревших данных. Предложено несколько способов записи только «хороших» данных – со сквозной записью, с буферизованной сквозной записью, с обратной записью и организация согласованной кэш-памяти.

В системе со *сквозной записью* кэш-контроллер сразу же после записи данных в кэш-память переносит их в основную память, выполняя, таким образом, две операции записи на каждую операцию модификации данных. В результате в основной памяти всегда содержатся правильные данные и любой блок в кэш-памяти может быть стерт (в процессе записи на его место нового блока) без потери данных. Основной недостаток такого способа заключается в потере производительности из-за затрат времени на запись в основную память. Кроме того, возрастает интенсивность использования магистрали, что для мультипроцессорных систем является серьезным недостатком.

В системе с *буферизованной сквозной записью* запросы на запись в основную память буферизуются. Это позволяет процессору, не дожидаясь завершения цикла записи в основную память, приступить к новому циклу. Увеличение скорости достигается в том случае, когда следующая операция чтения дает кэш-попадание, что позволяет процессору продолжить выполнение операции. В результате запись и чтение выполняются одновременно. В такой системе буферизуется только один запрос на запись в основную память. Поэтому, если возникает необходимость во второй операции записи, процессор переходит в состояние ожидания до завершения первой записи. К тому же, если за записью следует кэш-промах, процессор также должен ожидать поступления из основной памяти новых данных.

Следующий способ носит название *обратной записи*. Поле признака каждого блока в кэш-памяти включает бит, показывающий, был ли изменен блок. Перед наложением записи на любой блок в кэш-памяти контроллер проверяет состояние бита изменения. Если бит установлен, контроллер перед чтением новых данных в кэш-память записывает измененный блок в основную память. Способ обратной записи быстрее двух предыдущих, потому что число модифицированных блоков, требующих записи в основную память, обычно значительно меньше числа считываний в кэш-память.

Системе обратной записи присущи некоторые недостатки. Если большинство блоков в кэш-памяти оказались модифицированы, перед продолжением вычислений *все* блоки должны быть записаны в основную память. Далее, логика обратной записи в кэш-контроллере сложнее той, что требуется для реализации первых двух

способов. Наконец, в случае неисправимой ошибки, например, отказа питания, содержимое кэш-памяти теряется, и тогда невозможно определить, содержит ли основная память правильные данные.

Описанные выше способы минимизируют объем ошибочных данных в основной памяти, возникающих в результате операций записи в кэш-памяти. Однако они не решают проблему согласованности кэш-памяти с основной памятью, характерной для систем, в которых содержимое основной памяти обновляется более чем одним устройством. Если обновление памяти осуществляется несколькими устройствами, трудно определить, где находятся неправильные данные – в кэш-памяти или в основной памяти. Для решения этой проблемы были предложены три способа.

Первый способ заключается в *очистке кэш-памяти*. В случае изменения данных они записываются в основную память (чаще всего способом переноса записи), после чего кэш-память очищается. Если все блоки кэш-памяти успевают очищаться перед тем, как устройство выполняет запись в общую область памяти, кэш-память не будет содержать неправильных данных.

Второй подход к обеспечению связности кэш-памяти использует принцип *прозрачности аппаратуры*. В этом случае аппаратура выполняется так, что все обращения к областям памяти, скопированным в кэш-память, обнаруживаются кэш-памятью. Это достигается одним из двух способов. Первый способ заключается в том, что все записи в кэш-память копируются как в основную память, так и во все области кэш-памяти, отображенные на ту же память; этот способ носит название трансляции. Второй способ обеспечивает правильность данных путем направления запросов к основной памяти от всех устройств через одну и ту же область кэш-памяти.

Наконец, согласованность кэш-памяти может обеспечиваться объявлением обобщенной памяти некэшируемой. Все запросы к обобщенной памяти вызывают кэш-промахи, поскольку обобщенная память не копируется в кэш-память. Программное обеспечение может компенсировать снижение коэффициента кэш-попаданий путем копирования данных из некэшируемой памяти в кэшируемую и отображения запросов к обобщенной памяти на кэшируемые области памяти.

13.4. Буфера

Буферизацией называется временное запоминание результатов (выходов) процесса перед передачей их следующему процессу. Буфера позволяют делать более равномерным поток событий в системе в тех случаях, когда временные характеристики модулей, участвующих в вычислениях, не постоянны, а также когда скорости их работы сильно отличаются (даже в масштабе миллисекунд). Например, в составе компьютера может быть исключительно быстрый процессор и значительно более медленная память.

В дополнение к матрице памяти и дешифрирующей логике, которые образуют запоминающее устройство, выходные буфера используются для буферизации данных, извлекаемых из памяти, перед направлением их на выводы микросхемы памяти. Эти буфера полезны тем, что обеспечивают требуемые уровни выходного напряжения и выходные токи, а также позволяют выполнять мультиплексирование выходов нескольких микросхем. Буфера управляются одним или несколькими входными сигналами – «выбор микросхемы», «разрешение записи» и «разрешение выхода».

13.5 Синхронизация

Программируемый счеткик/таймер является полезным устройством, управляющим внешними событиями. Счетчик/таймер может адресоваться либо как ячейка памяти, либо как порт ввода-вывода. Начальное значение может быть занесено в счетчик под управлением программы. С каждым сигналом тактового генератора содержимое счетчика может как увеличиваться, так и уменьшаться.

Команды выполняются за разное время, в основном от 1 до 6 тактов. Полное время определяется количеством тактов, требуемых для выполнения каждой фазы команды. В некоторых случаях аппаратура автоматически добавляет один или два такта, чтобы внешняя логика завершила затребованную операцию. На это время ЦП может перейти в состояние ожидания.

13.5.1. Временны́е ограничения

В последовательности сигналов данных, адресов и управления ОЗУ имеются определенные временны́е ограничения. Эти ограничения обеспечивают правильное выполнение операций. Процессор, направив в устройство памяти сигналы адреса и выбора микросхемы, должен подождать некоторое время, равное времени доступа, перед съемом сигналов с выхода микросхемы памяти.

13.6. Резюме

В профессиональных и деловых приложениях программы могут выполняться в течение длительного времени, и одна программа может заполнить всю доступную память. Отсюда возникла необходимость в использовании быстрой памяти и многозадачного режима.

Прямой доступ к памяти обеспечивает пересылку данных параллельно с выполнением процессором задач обработки. ПДП представляет собой специальный вид пересылки, когда ЦП временно изолируется от шины, а пересылку данных между памятью и устройством ввода-вывода выполняет аппаратура ПДП.

Глава 14

Повышение производительности системы

Концепции разработки ранних микрокомпьютеров основывались на предположении, что в каждый момент компьютер используется только одним человеком, который выполняет на нем одну программу. Типичный пользователь терпеливо (возможно) ожидал, пока программа завершится, и лишь тогда запускал следующее задание. Сегодня пользователи склонны выполнять одновременно несколько заданий (например, печать, редактирование и сортировку данных) и ожидают от компьютера эффективного выполнения всей этой работы. Пользователям нужны «быстрые» компьютеры.

Быстродействие компьютера ограничивается тремя основными факторами. Первое ограничение связано с работой аппаратуры – насколько быстро переключаются вентили и какие расстояния по проводам или по печатной плате проходит сигнал. Второе ограничение определяется логикой взаимодействия узлов микрокомпьютера и его организацией. Третье зависит от эффективности использования операционной системой аппаратных средств компьютера. Настоящая глава посвящена двум последним факторам.

Первые микрокомпьютеры выполняли операции последовательно, одну за другой. Они либо занимались вычислениями, либо управляли вводом-выводом, либо обрабатывали прерывания. Однако проектировщики всегда понимали, что совмещение операций во времени ускорит работу машины. Еще в октябре 1842 г. Чарльз Бэббидж говорил в своей лекции: «... если нужно выполнить длинную последовательность... вычислений, например таких, которые требуются для формирования числовых таблиц, можно привлечь к работе машины так, чтобы получить несколько результатов одновременно, что значительно сократит объем работы».

14.1. Узлы управления

В 50-х гг. в машинах начал использоваться принцип одновременного выполнения нескольких операций для решения дифференциальных уравнений. Сегодня многие изготовители микрокомпьютеров предлагают двухпроцессорные системы (например, с арифметическим сопроцессором), а некоторые доводят комплект до четырех процессоров специального назначения. Ускорение вычислений объясняется параллелизмом операций, так же как и параллелизмом в работе памяти и процессора. Для обеспечения высокого быстродействия во многих системах используются узлы управления разного рода.

Узлы управления быстродействующих компьютеров выполняют такие традиционные функции, как выборка команды, управление вводом-выводом и прерываниями, установление последовательности операций при выполнении команд, отображение адресов и индексирование памяти. Правильно разработанный узел управ-

ления не замедляет работу процессора (процессоров) или памяти. Он функционирует достаточно быстро, чтобы извлекать и подготавливать машинные команды с той скоростью, с какой они требуются процессору.

Важной частью любой компьютерной системы является механизм распределения ресурсов процессора (процессоров) между несколькими пользователями. В современных системах это распределение выполняется автоматически с помощью прерываний и интервального таймера.

14.2. Независимость процессов

В основе повышения производительности любой системы лежит независимость отдельных процессов. Эта независимость проистекает из того, что определенные последовательности действий, выстраиваясь в естественные цепочки, оказываются более или менее независимыми от других последовательностей. Например, вывод содержимого диска на печать и пересылка данных из одной области памяти в другую представляют собой два раздельных процесса, которые логически могут выполняться независимо друг от друга. В действительности, правда, оба этих процесса могут обращаться к одной шине, но концептуально они вполне независимы.

Одна из основных задач операционной системы состоит в максимально эффективном разделении ресурсов компьютера между многими программами, которые непредсказуемо (асинхронно) выставляют требования на эти ресурсы. Эффективная операционная система, которая оптимальным образом использует аппаратные средства, существенно увеличивает производительность системы.

Обсудим теперь некоторые принципы, эффективное использование которых повышает общую производительность компьютера: 1) выборка данных до того, как они потребуются (предвыборка); 2) конвейерная обработка команд с целью увеличения быстродействия; 3) использование сопроцессоров для параллельного выполнения работ; 4) параллелизм операций; 5) многозадачность и мультипрограммирование.

14.3. Предвыборка

Увеличение эффективного быстродействия процессоров потребовало такого же улучшения характеристик запоминающих устройств.

Три основных фактора помешали этому: задержки распространения сигнала, плотность вентилей и меньшие успехи в области технологии. Пока ищутся пути преодоления этих трудностей, используются другие подходы.

Чтобы не заставлять процессор слишком долго ожидать, пока из памяти будут получены требуемые данные, можно использовать параллельный или конвейерный принцип. Эта методика, ставшая сейчас практически обязательной, заключается в следующем: когда процессор обращается к памяти за первой порцией информации, из памяти извлекаются и последующие данные. Выборка информации еще до того, как она требуется, носит название *предвыборки*. Основная трудность этой методики заключается в определении, какие именно данные понадобятся в следующий момент.

14.3.1. Эффективность предвыборки

Эффективность предвыборки тесно связана с явлением локальности. Было подмечено, что область обращений программы (относительно) медленно изменяется во времени и пространстве. На протяжении определенного периода выполнения программа обращается к множеству ячеек памяти, лежащих близко друг к другу. Это множество изменяется с тѐчением времени. Простые алгоритмы предвыборки осуществляют выборку «следующих по ходу» данных относительно предыдущей выборки.

14.3.2. Алгоритмы предвыборки

В целом алгоритмы предвыборки можно разделить на два класса: требования и предупреждения.

Алгоритмы *требования* основаны на том, что процессор работает с информацией, уже находящейся в системе, а затем возникает отказ страницы. Другими словами, выдается требование на ввод следующей порции информации.

Алгоритмы *предупреждения*, с другой стороны, осуществляют предвыборку не обязательно в момент отказа; выборка может быть выполнена при первой ссылке на новую страницу. Обычно в этом случае получается выигрыш во времени благодаря промежутку времени между моментом, когда страница затребована, и моментом, когда она действительно нужна.

14.3.3. Выборка следующей страницы

Методика выборки следующей страницы используется в предположении, что в выполнении программы имеются определенные закономерности, а именно, что для продолжения выполнения потребуется следующая по порядку страница.

14.4. Конвейерная обработка

Конвейерная обработка становится возможной, если несколько отдельных задач выполняются одновременно в некотором временном интервале. Конвейерная обработка является одной из форм организации параллельных процессов в вычислительной системе и реализуется разделением машинных действий на несколько фаз, выполняемых предназначенными для этого автономными узлами. Последовательные действия могут при этом выполняться с наложением во времени.

Например, ЦП может быть занят дешифрацией и выполнением одной команды, в то время как с помощью дополнительных регистров, имеющих выход на шину данных, выполняется предвыборка следующей команды. Рис. 14.1 иллюстрирует разницу в выполнении команд с использованием конвейерной обработки и без нее.

14.4.1. Методы

Выполнение команды разбивается на три фазы: выборка команды, дешифрация кода операции и выборка операндов [1]. На рис. 14.1 показано, как четыре последовательные и независимые команды могут выполняться параллельно.

Пока модуль ИК исполняет первую команду, модуль ВО выбирает операнды,

---

[1] В этот перечень следует включить четвертую фазу – собственно исполнение команды (операции). – *Прим. перев.*

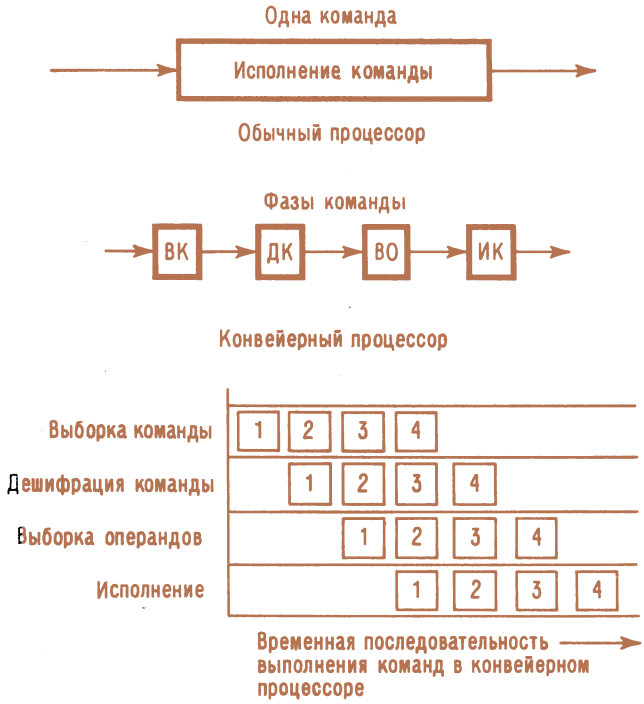


Рис. 14.1. Обычное и конвейерное выполнение команд. ВК – выборка команды; ДК – дешифрация команды; ВО – выборка операндов; ИК – исполнение команды.

необходимые для выполнения второй команды, модуль ДК подготавливает различные операции для третьей команды, а модуль ВК осуществляет выборку четвертой команды. На диаграмме пространство – время показано совмещение во времени действий четырех модулей.

Конвейерную обработку можно организовать на нескольких уровнях. Первый уровень обычно реализуется в процессоре, где выполняется конвейерная обработка команд, как это показано на рис. 14.1. Следующим является уровень подсистемы. Типичный пример – конвейерные арифметические узлы. Выполнение в конвейерном режиме сложения, умножения, деления и извлечения квадратного корня типично для многих современных компьютеров.

14.4.2. Проблемы

На работу конвейера разрушительное действие оказывают программные ветвления (условные переходы). Когда в программе встречается команда ветвления по условию, система не может определить, какую последовательность команд надо будет выполнять, пока на выходе не появится результат анализа условия. В результате условное ветвление не только задерживает дальнейшее выполнение, но влияет также на весь конвейер, начиная от фазы выборки команды. Перед тем как загрузить в конвейер ветвь программы, соответствующую условию ветвления, неправильную ветвь следует удалить из конвейера.

Один из способов снижения влияния ветвлений заключается в использовании буфера ветвей, или механизма предвидения. После команды ветвления фактически

осуществляется выборка *обеих* ветвей и дешифрация операндов. В этом случае какое бы направление ветвления ни оказалось правильным, наборы последующих команд уже загружены в конвейер.

Непрерывное выполнение потока команд может быть нарушено прерыванием. Для компенсации этого нарушения можно использовать механизм восстановления, который сохраняет содержимое конвейера и восстанавливает его после завершения обработки прерывания. Это довольно дорого и применяется редко.

14.4.3. Алгоритмы управления

В системах конвейерной обработки используются два основных типа алгоритмов управления. Более простой предполагает плавный (синхронизированный) поток команд через систему. Одна команда или задача следует за другой так, что они начинаются и заканчиваются в одном и том же порядке. В системе предусматриваются простые блокировки, обеспечивающие одинаковые скорости потоков через модули с различным быстродействием. Если в каком-либо сегменте возникает узкое место, входной поток останавливается до освобождения сегмента.

Второй алгоритм управления сложнее, но обладает большей гибкостью и мощностью. Поток, проходящий через систему, является полностью асинхронным, другими словами, каждая фаза может двигаться со своей собственной скоростью. Если возникает узкое место, следующая команда может вырваться вперед. Такой метод полезен в тех случаях, когда в системе имеется много исполняющих блоков (физических или виртуальных) или каких-либо параллельно работающих средств.

14.5. Сопроцессор

Побудительным мотивом для дальнейшей разработки параллельных вычислительных подсистем служат четыре системные характеристики: производительность, гибкость, доступность и надежность. Улучшение этих характеристик повышает качество системы. Поэтому многие изготовители стали выпускать сопроцессоры, представителем которых является арифметический сопроцессор 80387. *Сопроцессором* называется процессор, работающий совместно с центральным процессорным устройством под объединенным управлением.

Одним из наиболее плодотворных применений параллельных вычислений является выполнение арифметических операций. Чаще всего арифметический сопроцессор используется в тот момент, когда главный процессор «откладывает арифметические операции в сторону» и продолжает вычисления. Результат выполненных сопроцессором арифметических операций передается назад и ждет в готовом виде, когда он понадобится процессору. Весь этот процесс прозрачен для программиста системы.

В основе использования сопроцессоров лежат понятия взаимодействия и прозрачности. Работа сопроцессора должна быть прозрачна для программиста и отчетливо ясна для разработчика апаратуры.

14.6. Многозадачность и мультипрограммирование

Выполнение одним ЦП двух или более вычислительных задач называется *мультипрограммированием* или *многозадачностью*. Центральный процессор быстро переключается между двумя или несколькими программами, по очереди выполняя команды каждой. Такое переключение стало целесообразным, когда скорость ЦП превысила скорость процессов ввода-вывода.

В большинстве операционных систем планирование выполнения прикладных программ осуществляется на базе алгоритма квантования времени для программных задач, не ожидающих в настоящий момент завершения ввода-вывода. С помощью системного таймера операционная система определяет, сколько времени каждая прикладная программа может использовать основные системные ресурсы. При этом всем программам последовательно в циклическом порядке выделяются для выполнения равные кванты времени.

Алгоритм квантования времени относится к числу наиболее простых алгоритмов планирования. Обычно, пока одна программа ожидает завершения выполнения запроса на ввода-вывод, может выполняться другая. Поскольку операции ввода-вывода иногда связаны с передачей больших массивов информации и могут потреблять много процессорного времени, некоторые операционные системы разделяют большие запросы на ввод-вывод на серию более мелких, обеспечивая тем самым более частое переключение задач.

К числу наиболее сложных средств обеспечения параллелизма относится управление памятью. Некоторые операционные системы требуют, чтобы в памяти были выделены разделы фиксированной длины, каждому из которых присвоены определенные операционные характеристики. Другие операционные системы распределяют память динамически, используя для описания характеристик и потребностей прикладных пакетов специальные информационные файлы. В информационном файле программы записано, сколько требуется памяти, к каким периферийным устройствам нужен прямой доступ, какие прерывания будут обрабатываться и т. д.

Для управления доступом к дисплеям и клавиатурам можно определить оперативные (foreground) и фоновые (background) задачи. Оперативной задаче отдается контроль над клавиатурой и всем пространством экрана; все остальные задачи являются фоновыми.

Экранные окна дают возможность нескольким задачам работать с одним экраном. В этом случае на экране выделяются перекрывающиеся области, называемые *окнами*, в каждое из которых выводится часть выходных сообщений прикладных задач. При необходимости одна прикладная задача может использовать несколько окон; в ряде случаев пользователю дается право перемещения, скрытия или изменения размера окон, а также прокрутки изображения внутри окна.

В многозадачных системах серьезной проблемой является правильное выполнение прикладных программ, в которых осуществляется прямое обращение к периферийным видеосредствам, когда программа выводит изображение непосредственно на экран, минуя операционную систему. Поскольку эти действия могут вступить в противоречие с алгоритмами операционной системы, программы такого рода называют *некорректными* (ill-behaved).

В системе, поддерживающей мультипрограммирование, должны быть предусмотрены средства синхронизации пересылки информации в обобщенные структуры, например в буфера видеоустройств.

14.6.1. Упорядочение событий

При любом упорядочении системных процессов или событий внутри этих процессов важнейшим является понятие временной синхронизации. Любая компьютерная система представляет собой объединение процессов. Любой процесс можно разделить на события, возникающие в некоторой последовательности. На упорядочение событий в процессе влияет, что именно мы включим в понятие события.

Синхронизацию событий можно рассматривать просто как способ назначения событиям номеров, если под номером понимать момент времени, когда данное событие должно произойти.

Многозадачность представляет собой методику организации параллельных процессов путем разделения одной или нескольких программ на две или несколько взаимосвязанных задач, совместно использующих программные блоки, буфера и файлы. Многозадачный режим предполагает наличие средств, позволяющих программе устанавливать взаимосвязь вызываемых процедур и данных независимо от их физического расположения в памяти. Кроме того, в этом режиме программы могут использовать обобщенные процедуры и форматирование данных. Многозадачный режим ускоряет переключение вычислительных аппаратных средств с одной программы (задачи) на другую.

В мультипрограммных системах несколько пользователей или программ могут одновременно работать с системой. Эта возможность возникает благодаря тому, что пока одна программа ожидает завершения некоторого внешнего процесса (например, запроса на получение данных), другая может использовать процессор. Процессор в этом случае не переходит в состояние *ожидания*, а занят вычислениями. Состояние ожидания – это внутренняя пауза, которая наступает, если отсутствует синхронизирующий сигнал, или процессор остановлен программой с целью ожидания ввода от устройства. Как только второй пользователь приступает к ожиданию внешнего процесса или завершает свою задачу, процессор переключается на продолжение первой программы. Такое совместное пользование увеличивает производительность системы, поскольку процессы теперь обрабатываются не друг за другом, а одновременно.

Все большая доля вычислительной работы выполняется мультипрограммными системами. Такие системы характеризуются использованием вычислительных ресурсов (процессора, основной памяти, периферийных устройств) во многих независимых, но взаимосвязанных вычислениях. *Вычисление* – это множество процессов, гармонично сочетающихся в одной программе или задаче.

14.6.2. Свойства мультипрограммных систем

Мультипрограммные системы обладают четырьмя важными свойствами: 1) вычислительные процессы могут идти, как параллельные операции для нескольких пользователей; 2) многие вычисления совместно и гибко используют ресурсы, например память; 3) требования конкретных вычислений к вычислительным ресурсам в каждый момент времени могут различаться в широких пределах; 4) мультипрограммные системы развиваются с целью удовлетворения изменяющихся запросов пользователей. Параллелизм в мире микрокомпьютеров обычно ограничивается аппаратно-контролируемыми параллельными вычислениями, выполняемыми главным процессором и сопроцессором. МП 80386 с командой LOCK/XCHG, предоставляет возможность мультиобработки на уровне большой ЭВМ.

14.7. Резюме

Важнейшей характеристикой системы является ее производительность, определяемая числом выходных данных или выполненных команд за единицу времени. Производительность прямо отражает вычислительную мощность системы: чем больше производительность системы, тем выше ее мощность.

Концепция параллелизма в организации машины была предложена еще во времена вычислительных устройств Чарльза Бэббиджа. Приложение этой концепции к современным машинам вместе с повышением их быстродействия позволяет строить все более быстрые компьютеры. Ожидается, что стоимость аппаратного обеспечения будет падать быстрее, чем растет его быстродействие. Это приведет к широкому использованию параллелизма и других изложенных выше идей.

Приложение 1

Интегральные схемы и фирма Intel

Полупроводниковая промышленность в сущности зародилась в шт. Нью-Джерси в лабораториях фирмы Bell Telephone Laboratories (Bell Labs.). Вскоре она пустила корни в интеллектуально плодородном районе шт. Калифорния к югу от Сан-Франциско. Долина Санта-Клара, получившая название «Кремниевой долины», дает сегодня пристанище сотням компаний, продукция которых основана на достижениях тех ранних дней.

В середине 1950-х гг. небольшая группа ученых покинула Bell Labs. и при поддержке фирм Fairchild Camera и Istrument Company рискнула создать собственное дело. Ученые образовали компанию Fairchild Semiconductors и осели в долине Санта-Клара. Двое из них – Роберт Нойс и Гордон Мур – ушли в 1968 г. из компании Fairchid Semiconductors и обосновали новую компанию в г. Саннивейл, шт. Калифорния; они назвали ее Intel (от Integrated Electronics – интегральная электроника). Через три года фирма Intel выпустила первый микропроцессор, принесший ей успех и заслуживший широкое признание в промышленности.

Первые калькуляторы на интегральных микросхемах имели ограниченные функции и стоили сотни долларов. Далеко не сразу инженеры и изготовители осознали потенциальные возможности этих первых неуклюжих машин. Но, по мере того как вырисовывались все новые области применения микропроцессоров, инженеры начали разрабатывать специальные микросхемы под конкретные применения. Когда возникла массовая потребность в дешевых карманных калькуляторах, технологические вопросы их выпуска были уже решены.

В середине 1969 г. японская фирма по выпуску средств вычислительной техники «Бусиком» предложила корпорации Intel разработать серию микросхем для программируемых калькуляторов нового образца. Инженеры, работающие в фирме «Бусиком», создали проект, в котором предполагалось использовать двенадцать логических и запоминающих микросхем с числом транзисторов в каждой от 3 до 5 тысяч. Перед этим фирма Intel освоила технологию производства микросхем, содержащих 2000 транзисторов. Фирма «Бусиком» полагала, что эту технологию нетрудно использовать в интересах рынка калькуляторов.

Концепции новых разработок

Вплоть до начала 1970-х гг. промышленная стратегия в области микропроцессоров заключалась в том, что каждая микросхема разрабатывалась применительно к требованиям конкретного заказчика. Процессоры были маломощными и абсолютно не отвечали требованиям, предъявляемым к вычислительным средствам общего назначения.

Работая над проектом для «Бусикома», инженеры фирмы Intel предложили революционную концепцию проектирования микропроцессоров: почему бы не разработать микропроцессор общего назначения, который мог бы выполнять

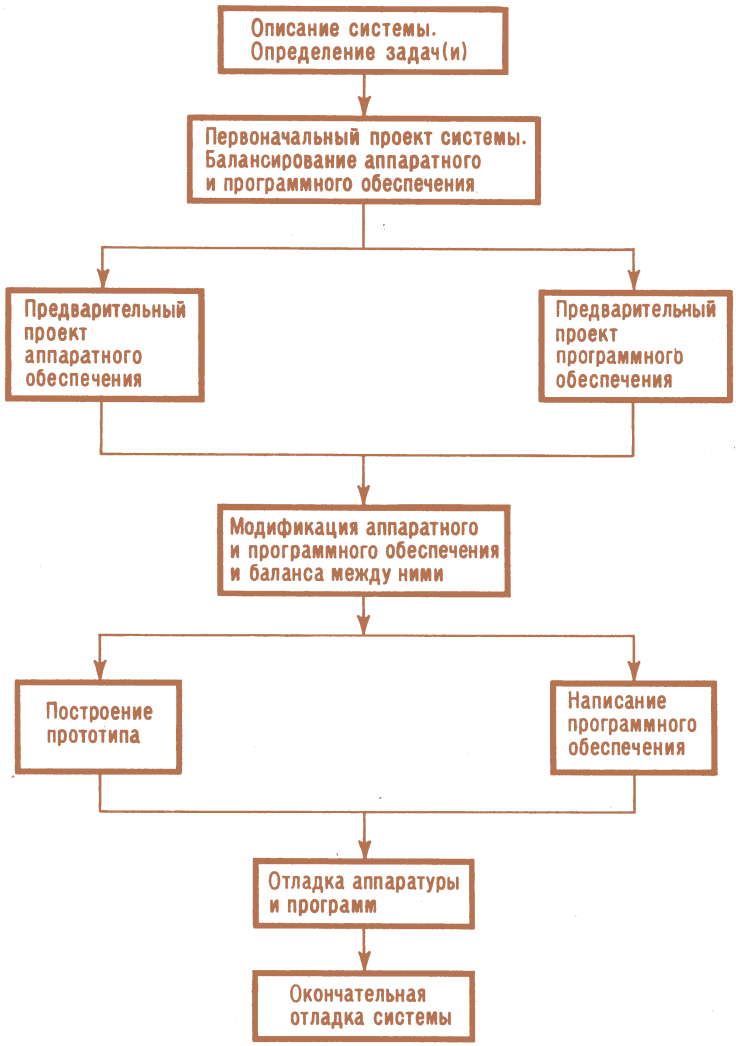


Рис. П1.1. Этапы проектирования микропроцессорной системы.

любую логическую задачу? Такой процессор должен быть программируемым, а команды для него можно хранить в микросхемах ЗУПВ или ПЗУ. Если заказчикам требуется изделие с конкретными свойствами, они могут его запрограммировать, а фирма Intel запишет эту программу в ПЗУ (рис. П1.1). В этом случае фирме уже не придется разрабатывать для каждого заказчика новый набор логических схем. Бремя разработки перемещалось от изготовителей микросхем к фирмам, собирающим законченные изделия из комплектующих деталей. Указанная концепция легла в основу популярного теперь семейства микросхем Intel.

Первый многоцелевой процессор 4004, разработанный тремя сотрудниками менее чем за год, начал выпускаться в конце 1970 г. Мнения промышленных экспертов разделились. Одни считали, что процессор общего назначения слишком универсален, чтобы найти широкое применение. Другие удивлялись, почему эта концепция не была реализована раньше.

Микросхема 4004

Микросхема 4004 представляет собой 4-разрядный (полубайтовый) микропроцессор. Вместо того чтобы распределять арифметические и логические функции калькулятора по нескольким ИС, соединенным проводами, разработчики включили эти функции в состав самой микросхемы 4004.

МП 4004 содержит 2250 МОП-транзисторов и может выполнять около 60 000 операций в секунду. В нем предусмотрены возможности адресации 1280 полубайтов данных и 4 кбайт запрограммированных команд. МП 4004 слишком маломощен, чтобы выполнять функции процессора, однако вполне может служить основой калькулятора.

Микросхема 4004 складывает 4-разрядные числа за время около 11 мкс, однако умножает только путем многократных сложений. Она включает 4-разрядный сумматор, аккумулятор и 16 регистров для временного хранения результатов. ПЗУ, содержащее внутренние программы, имеет объем 2 кбит, в то время как ОЗУ – всего 320 бит. Размер микросхемы составляет 0,11 × 0,15 дюйма.

МП 4004 содержит значительное количество логических схем, присущих центральному процессорному узлу. Одна БИС заменила сотни схем, составлявших в то время элементную базу промышленных мини-комьютеров. Система из 46 команд кажется скромной по сегодняшним меркам, однако она вполне адекватна для решения задач управления с принятием решений, которые не помещаются в программируемые логические матрицы.

Микросхема 8008

В апреле 1972 г. фирма Intel выпустила свой первый 8-разрядный микропроцессор 8008. Он включает около 3300 МОП-транзисторов, на тысячу больше, чем МП 4004, выполняет свыше 30 000 операций в секунду и адресует 16 кбайт памяти. Одна операция выполняется в этом микропроцессоре за 12,5 мкс. Умножения, как и в МП 4004, реализуются путем многократных сложений, что требует много времени. Размер микросхемы – 0,125 × 0,17 дюйма.

МП 8008 выполнен, как и МП 4004, на базе *p*-канальной МОП-технологии, но имеет шину данных (электронный канал для передачи данных) шириной 8 разрядов и более обширную систему из 48 команд. Однако системы команд МП 8008 и 4004 несовместимы.

Поскольку в МП 8008 данные, необходимые для выполнения и дешифрации команд, а также для представления операндов, размещаются в 8-разрядных секциях, время выполнения команд у него меньше, чем у МП 4004. К тому же МП 8008 адресует 16 384 ячеек памяти длиной 8 разрядов каждая, включает семь 8-разрядных регистров, средства работы со стеком и одноуровневую систему прерываний.

Микросхема 8080

В 1974 г. фирма Intel выпустила МП 8080, предназначенный для самых разнообразных применений. Уже МП 8008 имел достаточную мощность, чтобы служить центральным процессором микроЭВМ, однако ему были присущи некоторые недостатки, которые фирма Intel хотела бы преодолеть, прежде всего невысокая скорость вычислений. МП 8080 содержит более 4500 транзисторов и выполняет около 200 000 операций в секунду.

Система команд МП 8080 кроме 30 собственных команд включает набор команд МП 8008, что обеспечивает их совместимость. Наличие 79 команд решительно изменило область приложений МП 8080: из устройства, обеспечивающего

	4004	8008	8080
Класс микросхемы	4 разряда	8 разрядов	8 разрядов
Технология	*p*-МОП	*p*-МОП	*n*-МОП
Размер слова:			
данные	4 разряда	8 разрядов	8 разрядов
команды	8 разрядов	8 разрядов	8 разрядов
Объем адресного пространства	4K	16K	64K
Тактовая частота, кГц	740	800	2083
Время сложения, мкс	10,8	20	2
Размер корпуса	16 выводов	18 выводов	40 выводов
Объем системы команд	45	48	72 (включает команды МП 8008)

Рис. П1.2. Сравнительные характеристики первых микропроцессоров.

главным образом задачи управления, он стал процессором общего назначения. Впервые микросхемы фирмы Intel получили совместимость снизу вверх. Владельцы МП 8008 могли теперь использовать более мощный и гибкий микропроцессор, не изменяя своего программного обеспечения.

МП 8080 был изготовлен на основе *n*-канальной МОП-технологии и работал на более высокой, чем его предшественники, тактовой частоте. Сложение двух 8-разрядных операндов выполнялись им со скоростью 500 000 операций в секунду. Все остальные команды также выполнялись быстрее, так как при размещении микропроцессора в корпусе с 40 выводами ЦП тратит гораздо меньше времени на разделение шины данных между операциями передачи данных и выполнение команд.

МП 8080, как и его предшественники МП 4004 и 8008, не содержит команды умножения и выполняет умножение с помощью операций сложения. Однако его аппаратные средства более развиты, чем у МП 8008. Адресуемое пространство увеличилось с 16 до 64K. Вместо небольшого 7-уровневого стека в самом процессоре МП 8080 предоставляет стек во внешней памяти. Размер микросхемы 8080 составляет 0,165 × 0,191 дюйма.

Разработка МП 8080 поставила фирму Intel с числом служащих 21 500 человек и годовым оборотом свыше 1 млрд. долл. в ряд крупнейших компаний по производству ИС в США.

На рис. П1.2 приведены сравнительные характеристики первых микропроцессоров.

Микросхема 8080A

В 1976 г. фирма Intel выпустила МП 8080A (рис. П1.3 и П1.4), являющийся модификацией МП 8080. Это 8-разрядный однокристальный микропроцессор, изготавливаемый на основе *n*-канальной МОП-технологии. Микросхема размером 0,165 × 0,191 дюйма в 40-выводном DIP-корпусе содержит около 400 транзисторов. 16 выводов (A0–A15) используются как выходы с тремя состояниями для адресации памяти и устройств ввода-вывода. 8 выводов (D0–D7) также с тремя состояниями обеспечивают двунаправленную передачу данных (информационных и о внутреннем состоянии). Наконец, 10 выводов используются для тактирующих и управляющих сигналов.

Система команд МП 8080A содержит 72 команды, причем они поддерживаются также следующей микросхемой семейства, МП 8085A.

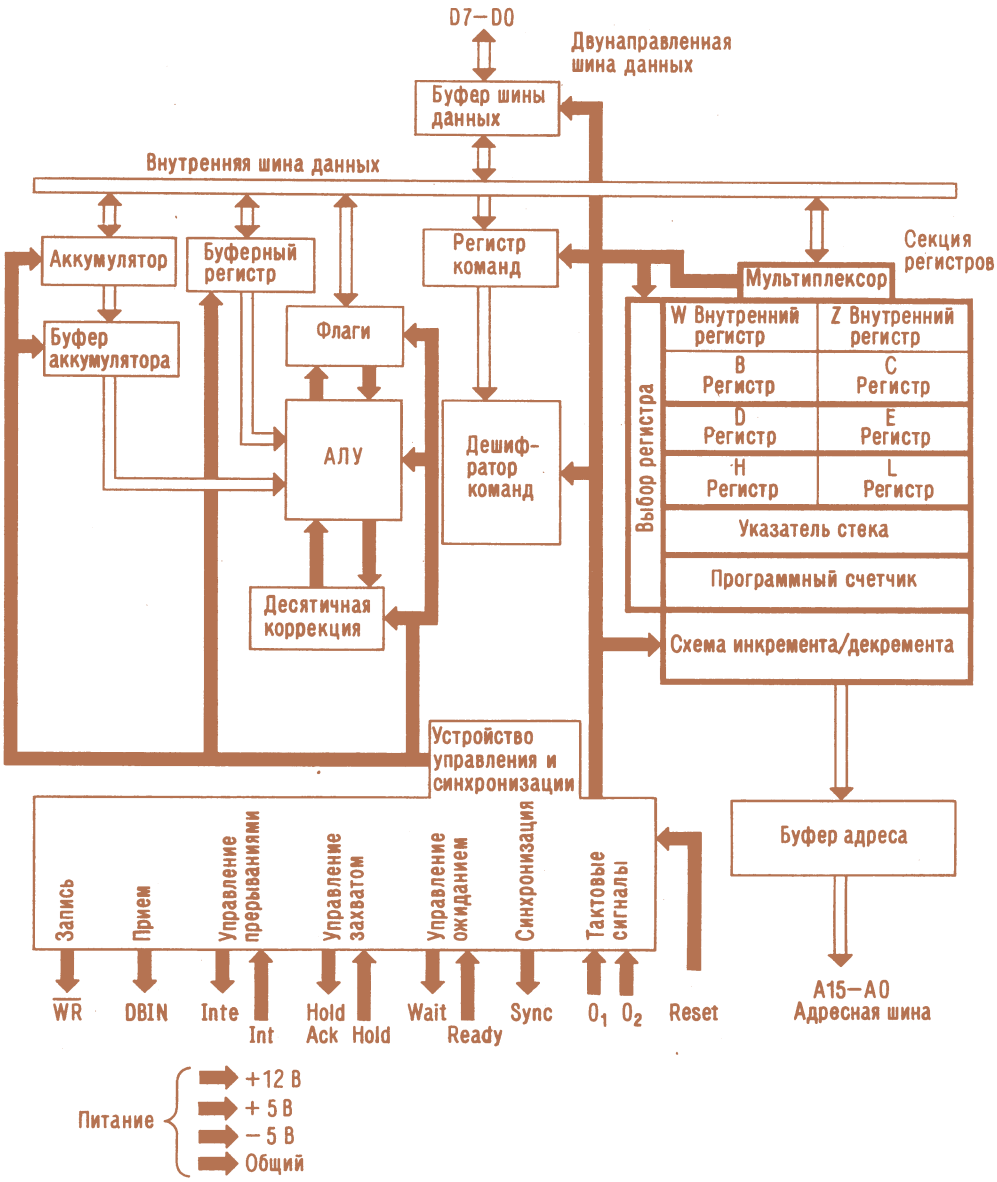


Рис. П1.3. Функциональная схема МП 8080А.

Микросхема 8085А

МП 8085А представляет собой улучшенную модификацию МП 8080А, почти полностью функционально эквивалентную последнему и обеспечивающую совместимость снизу вверх. В основном МП 8085А является однокристальной комбинацией трех микросхем: микропроцессора 8080А, формирователя тактовых сигналов 8224 и системного контроллера 8228.

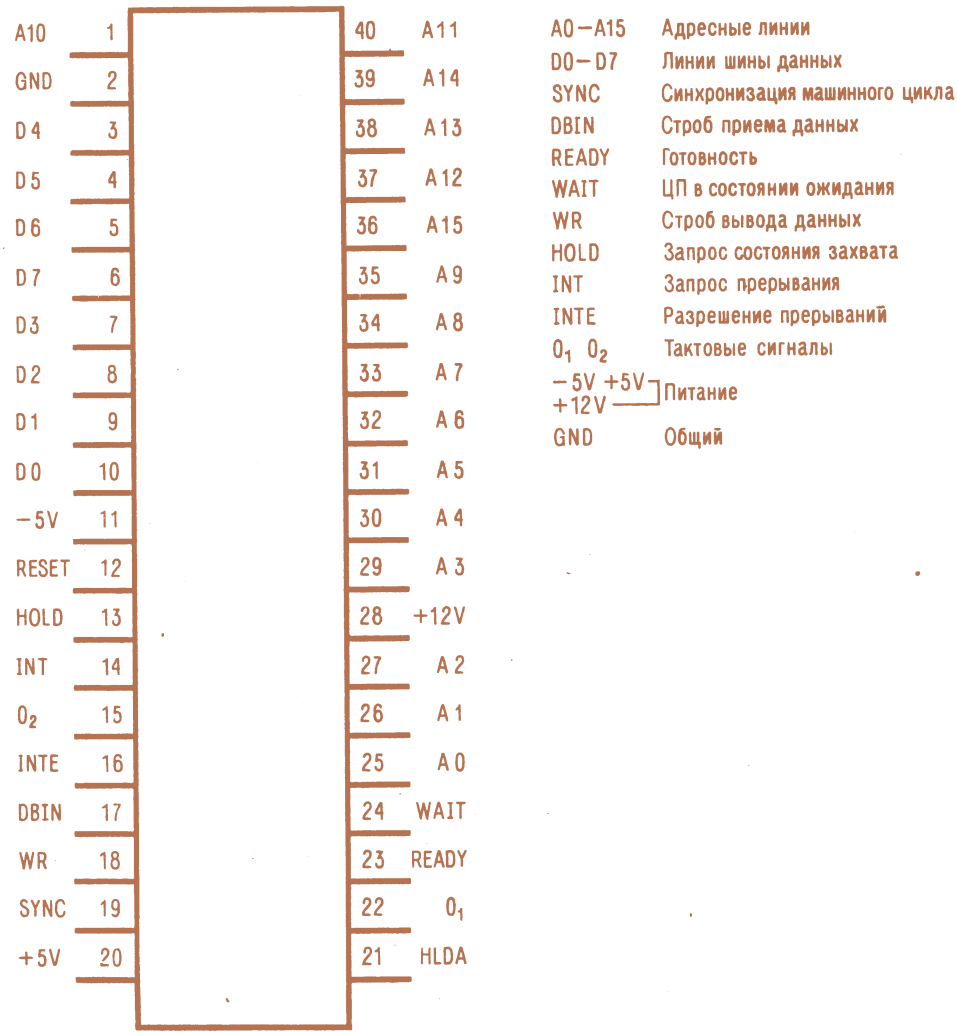


Рис. П1.4. Назначение выводов МП 8080А.

8085А – это 8-разрядный микропроцессор, он изготавливается по *n*-канальной МОП-технологии и содержит около 6200 транзисторов. Схема заключена в 40-выводной DIP-корпус размером 0,164 × 0,222 дюйма. Назначение выводов показано на рис. П1.5 .

Микросхема требует лишь одной шины питания 5 В. Используя 16-разрядные адреса, микропроцессор может адресовать до 64К ячеек памяти. Порты ввода-вывода рассматриваются либо как внешние регистры, либо как ячейки памяти. В последнем случае для чтения и записи вниз могут использоваться любые команды обращения к памяти.

Кроме того, в МП 8085А предусмотрена непосредственная адресация 256 входных и 256 выходных портов с помощью специальных команд ввода-вывода с 8-разрядными адресами. Внутренняя шина данных МП 8085А, имеющая ширину 8 разрядов, служит для пересылки команд и данных между различными внутренними

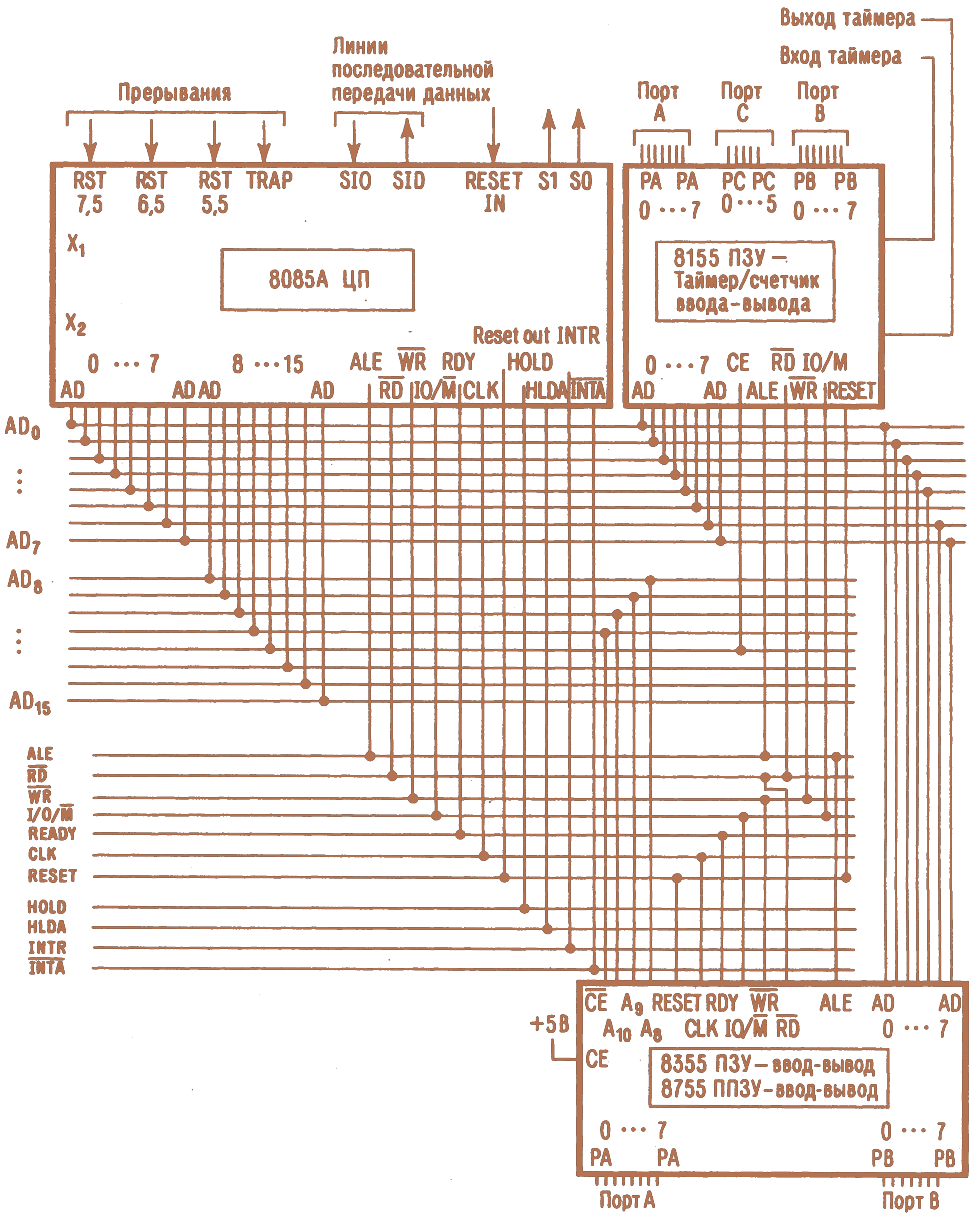


Рис. П1.5. Назначение выводов МП 8085А. ПЗУ объемом 2К байт; ОЗУ объемом 256 байт; 1 интервальный таймер/счетчик событий; четыре 8-разрядных порта ввода-вывода; один 6-разрядный порт состояния ввода-вывода; 4 уровня прерываний; 2 линии последовательного ввода-вывода.

регистрами или во внешние устройства с помощью буферного регистра мультиплексированной шины адресов/данных.

К системе команд МП 8080 были добавлены всего две команды для чтения и записи последовательных данных и обработки прерываний. Поскольку 74 команды МП 8085А включают весь набор команд МП 8080А, программы, написанные для МП 8080А, могут выполняться на МП 8085А. Однако время выполнения этих программ будет различным, даже если частоты тактовых сигналов выбраны так, чтобы обеспечить одинаковые длительности тактов в обоих микропроцессорах. Разница эта связана с тем, что одни и те же команды на двух машинах требуют для выполнения разного числа тактов.

Микросхема 8086

Выпущенный в 1978 г. МП 8086 был первым микропроцессором, способным работать одновременно с 16 разрядами данных. Начиная с МП 8086, в микропроцессорах стали использоваться две определяющие архитектурные концепции: сегментация памяти и новая схема кодирования команд. Процессор 8086 разделен на два независимых блока. Первый – это *блок сопряжения с магистралью* BIU, второй – *исполнительный блок* EU (рис. П1.6).

Блок сопряжения с магистралью выполняет две основные функции: (1) выборка команд из памяти и (2) пересылка данных между исполнительной аппаратурой и

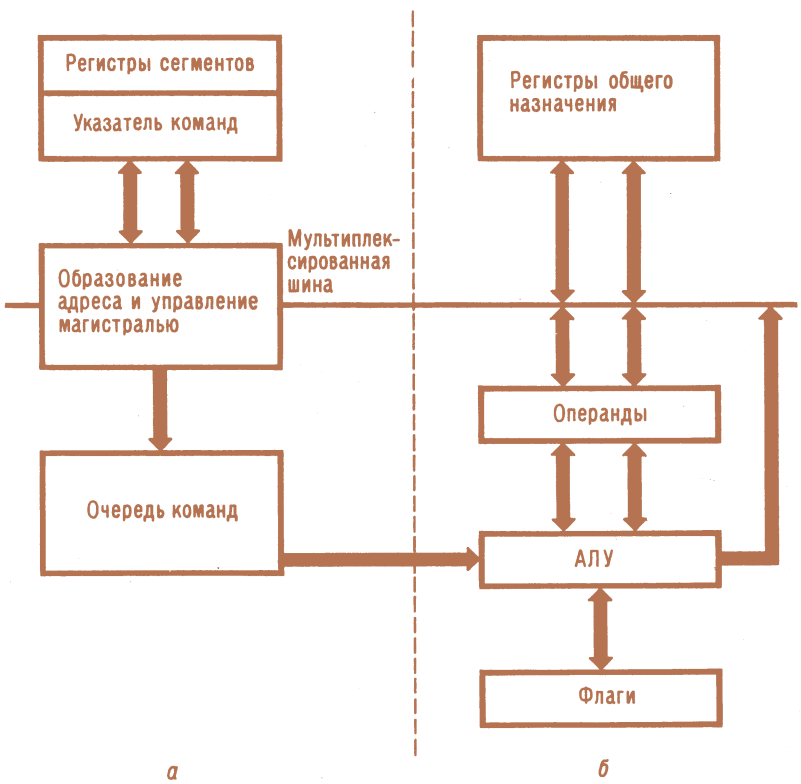


Рис. П1.6. Блок сопряжения с магистралью BIU (*а*) и исполнительный блок EU (*б*).

внешним миром (внешним с точки здения процессора). BIU содержит *указатель команд* (IP), в котором хранится адрес очередной выбираемой команды. В BIU также находятся байтовая очередь потока команд и *регистры сегментов*.

Регистры сегментов МП 8086 содержат 16 разрядов и носят названия CS (сегмент программы), DS (сегмент данных), SS (сегмент стека) и ES (дополнительный сегмент).

Блоки BIU и EU работают независимо. Пока BIU осуществляет выборку дополнительных команд, EU выполняет уже выбранные команды. Такое разделение функций стало возможным благодаря использованию конвейера (очереди) команд. BIU заполняет этот конвейер командами, ожидающими выполнения. В тот момент, когда EU заканчивает выполнение данной команды, следующая команда, как правило, уже подготовлена для немедленного выполнения; это устраняет задержки, связанные с выборкой команд. Очередь BIU в МП 8086 хранит до 6 байтов потока команд.

Исполнительный блок не имеет связи с системной магистралью, которая для него является внешним миром. EU получает команды из очереди, поддерживаемой BIU. Если команда требует обращения к памяти или к внешней магистрали, EU запрашивает BIU на выборку или запись данных. Все адреса, с которыми оперирует EU, 16-разрядные, но BIU выполняет преобразование адресов так, чтобы EU имел доступ ко всему возможному адресному пространству (1 Мбайт) памяти.

Исполнительный блок содержит 16-разрядное *арифметико-логическое устройство* (АЛУ), регистры операндов и флагов, а также набор регистров общего назначения AH, AL, BH, BL, CH, CL, DH, DL, BP, SP, SI и DI. Все регистры и связи между ними – 16-разрядные. 8-разрядные регистры объединяются с образованием 16-разрядных регистров [1]. Например, регистры AH и AL объединяются в регистр-аккумулятор AX; BH и BL образуют регистр базы BX; CH и CL образуют регистр-счетчик CX; DH и DL образуют регистр данных DX.

К регистрам общего назначения можно адресоваться, как к 8- или 16-разрядным. Это позволяет программам для МП 8080 выполняться на МП 8086 при условии их повторной трансляции.

В зависимости от числа и сложности периферийных устройств разработчик системы может выбрать максимальный или минимальный режим работы системы. Режимы назначаются с помощью вывода (MN/MX), который либо заземляется (для получения максимального режима), либо подключается к источнику питания +5В (для получения минимального режима).

В максимальном режиме для возбуждения сигналов управления памятью и устройствами ввода-вывода используется контроллер магистрали. Это освобождает выводы управляющих сигналов минимального режима для выполнения других функций, таких, как блокирование магистрали и управление дополнительным прямым доступом к памяти. В минимальном режиме сигналы управления памятью и устройствами ввода-вывода генерирует процессор 8086.

Команды МП 8086 делятся на шесть категорий: передачи данных, арифметические, логические, обработки строк, передачи управления и управления процессором.

Память МП 8086 (рис. П1.7) состоит из 8-разрядных байтов, причем два любых соседних байта образуют 16-разрядное слово. Впервые была реализована возможность адресации более 64К памяти. Сама память образуется объединением произвольного числа сегментов, каждый из которых содержит 64К последовательных байтов. Важность концепции *сегментов* определяется тем, что именно с помощью сегментов происходит адресация памяти.

Любой сегмент памяти начинается с адреса, кратного 16. Доступ к требуемому

[1] Или регистровых пар, как их иногда называют.– *Прим. перев.*

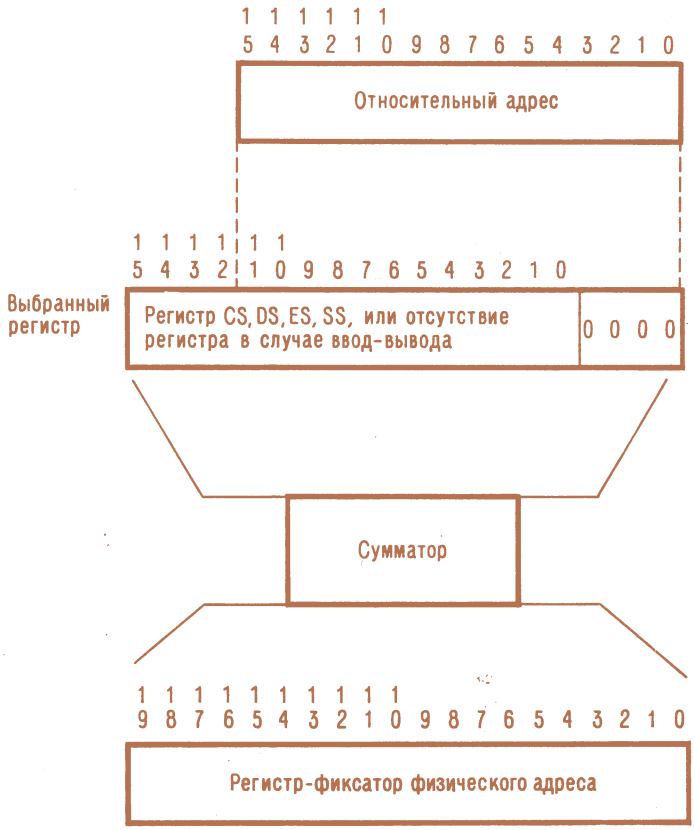


Рис. П1.7. Адресация памяти в МП 8086.

сегменту осуществляется с помощью четырех сегментных регистров. В любой конкретный момент МП 8086 может непосредственно адресовать содержимое четырех сегментов: сегмента программы, сегмента данных, сегмента стека и дополнительного сегмента данных. Сегменты могут перекрывать друг друга.

Для адресации байта или слова в сегменте 16-разрядный относительный адрес, часто называемый *исполнительным* (*эффективным*) *адресом* (EA), используется вместе с содержимым регистра сегмента. Программист указывает в программе, какой сегментный регистр он хочет использовать, и обозначение этого регистра включается в саму команду.

МП 8086 может адресовать до 1 Мбайт памяти (1 048 576 байт). Физический адрес образуется из относительного адреса и содержимого указанного регистра сегмента. Для этого адрес сегмента сдвигается влево на 4 разряда и освободившиеся 4 младших разряда заполняются нулями. Затем к этому 20-разрядному числу прибавляется исполнительный адрес, в результате чего формируется действительный физический адрес.

Основной формат команд МП 8086 показан на рис. П1.8. Первый байт включает код операции (коп), одноразрядное поле направления D и одноразрядное поле ширины W. Если D = 0, предполагается направление «внутрь» (регистра). Если D = 1, предполагается направление «из» (регистра). Поле ширины определяет, работает ли команда с байтом (W = 0) или со словом (W = 1).

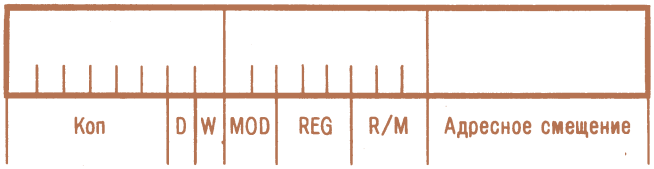


Рис. П1.8. Формат команд МП 8086.

Содержимое разрядов	W = 0	W = 1	Содержимое разрядов	W = 0	W = 1
000	AL	AX	100	AH	SP
001	CL	CX	101	CH	BP
010	DL	DX	110	DH	SI
011	BL	BX	111	BH	DI

Рис. П1.9. Соответствие кода в поле R/M используемому регистру.

Поле режима MOD образует следующие два разряда формата команды. В этом поле указывается режим адресации. Он определяет, как используется содержимое поля R/M (register/memory, регистр/память) при нахождении операнда. Если MOD = 11, поле R/M рассматривается как поле указания регистра.

Поле регистра REG определяет 8- или 16-разрядный регистр, в котором находится операнд. Расшифровка кодов поля REG приведена на рис. П1.9.

Поле R/M определяет либо способ вычисления адреса операнда, находящегося в памяти, либо регистр, в котором находится операнд. Затем могут следовать один или два байта смещения, если код в поле R/M указывает, что для вычисления адреса требуется смещение. Наконец, один или два байта могут содержать сам операнд, если используется непосредственная адресация.

Некоторые ячейки памяти, входящие в адресуемое процессором 8086 (а также и процессорами 8088 и 80186) пространство, зарезервированы фирмой Intel для последующего использования аппаратными и программными средствами. Эти ячейки показаны на рис. П1.10.

Сопроцессоры 8087. Одновременно с совершенствованием процессоров фирмы Intel шло параллельное совершенствование другого вида микросхем – сопроцессоров. *Сопроцессор* – это подчиненный процессор, выполняющий определенные операции для процессора общего назначения. Первый популярный сопроцессор 8087 выполнял вычисления с плавающей точкой для МП 8086 и 8088. Когда был разработан МП 80186, сопроцессор 8087 стал использоваться и с этой микросхемой.

Микросхема 8088

МП 8088, появившийся в 1979 г., является 8-разрядным вариантом МП 8086. В нем, как и в МП 8086, предусмотрена адресация памяти до 1 Мбайт. Однако доступ к памяти осуществляется байтами. Операнды-слова выбираются из памяти за два цикла магистрали независимо от их выравнивания. Команды также выбираются по одному байту за цикл.

Как и в МП 8086, память состоит из произвольного числа сегментов, каждый из которых содержит максимум 64К байт. Вычисление адреса выполняется так же: содержимое 16-разрядного регистра сегмента, расширенное справа четырьмя нуля-

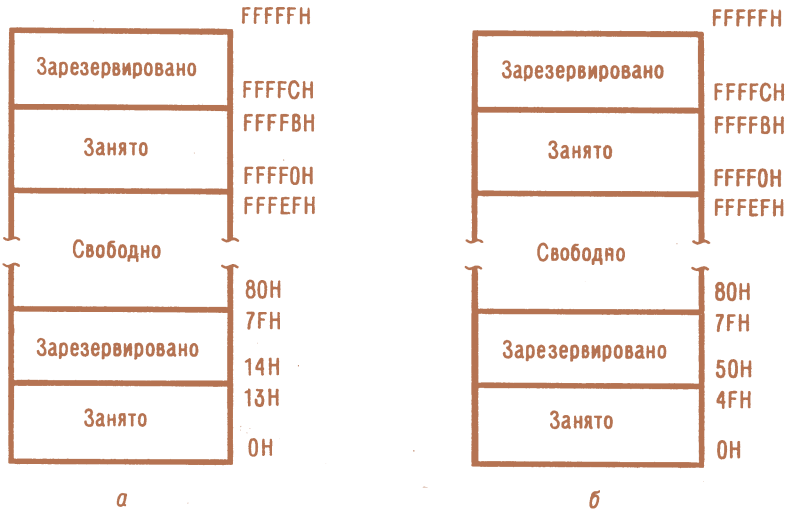


Рис. П1.10. Зарезервированная и занятая память, а также пространство ввода-вывода. *а* – память МП 8086; *б* – память МП 80186.

ми, прибавляется к 16-разрядному относительному адресу (называемому также логическим адресом) с образованием 20-разрядного адреса слова. Более полное описание можно найти в документации МП 8088. Фирма IBM использовала МП 8088 в качестве процессора для своей популярной машины IBM PC.

Микросхема 80186

В дополнение к средствам, содержащимся в МП 8086 и 8088, в состав МП 80186 включены логический блок выбора микросхемы, два независимых скоростных канала ПДП, три программируемых таймера, программируемый контроллер прерываний и генератор тактовых импульсов (рис. П1.11).

Наборы регистров МП 80186, 8086 и 8088 идентичны, за тем небольшим исключением, что регистр, называемый в МП 8086 и 8088 регистром флагов, в МП 80186 носит название регистра слова состояния. Содержимое этих регистров одинаково. МП 80186 совместим с МП 8086 и 8088 на уровне машинных команд, но имеет 10 дополнительных команд по сравнению с базовым набором МП 8086/88.

Логика выбора микросхемы обеспечивает генерацию программно-управляемых сигналов выбора для микросхем памяти и периферийных устройств. Шесть выходов выбора микросхемы памяти дают возможность выбирать три области адресов: 1) верхнюю память, 2) среднюю память, 3) нижнюю память. Область выбора каждого выхода программно задается пользователем. МП 80186 может также генерировать сигналы выбора для семи периферийных устройств.

Контроллер ПДП МП 80186 представляет собой два независимых скоростных канала ПДП. Контроллер пересылает данные между памятью и устройством ввода-вывода, из памяти в память или между двумя устройствами ввода-вывода. Данные могут пересылаться словами или байтами и находиться (как на стороне источника, так и на стороне приемника) и по четным, и по нечетным адресам. Каналы содержат 20-разрядные указатели адресов источника и приемника, которые могут получать положительное или отрицательное приращение после каждой пересылки данных.

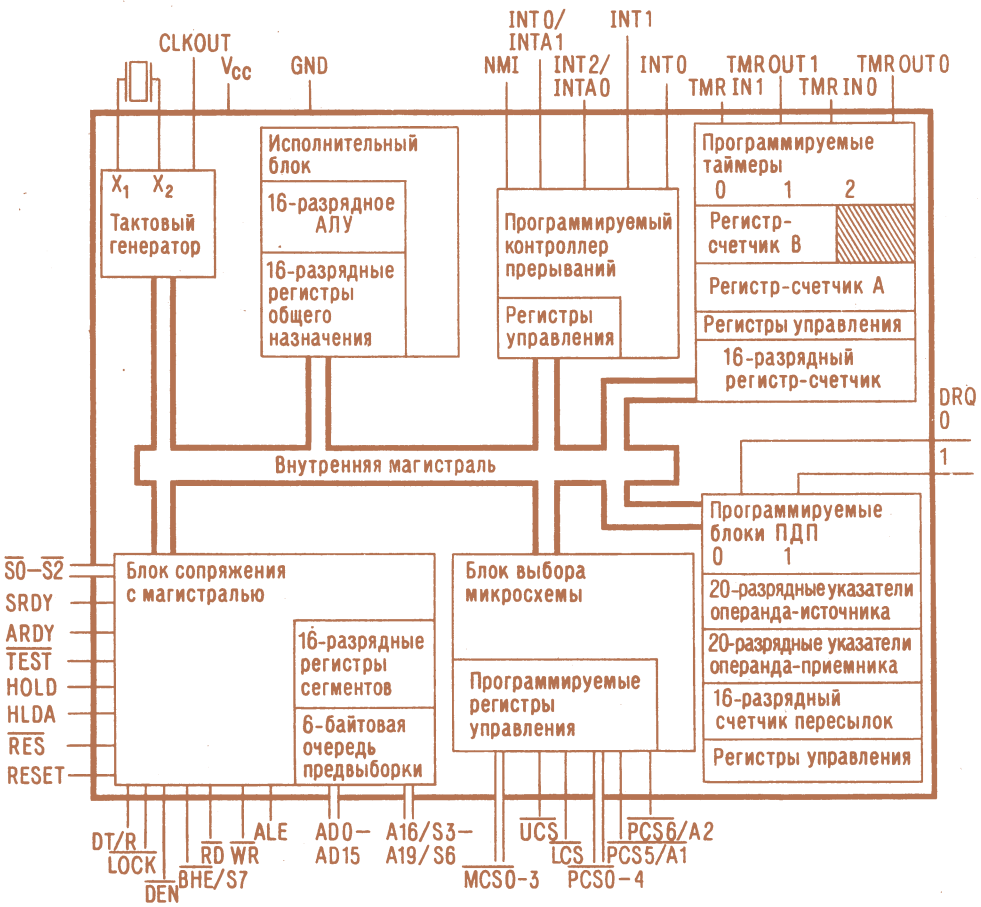


Рис. П1.11. Структурная схема МП 80186.

В микропроцессоре имеется 16 внутренних программируемых таймеров. Два из них, подключенные к внешним выводам, могут перенастраиваться. Их можно использовать для счета и временной синхронизации внешних событий, генерации неповторяющихся сигналов и т. д. Третий таймер, не подключенный к внешним выводам, удобен для создания временных задержек и в программах реального времени.

Включенный в состав микросхемы тактовый генератор обеспечивает внутреннюю генерацию тактовых сигналов, а также подключение внешнего генератора. Он состоит из собственно кварцевого генератора, двоичного счетчика, синхронного и асинхронного входов готовности и схемы сброса.

Контроллер прерываний МП 80186 воспринимает прерывания от ряда источников, как внутренних, так и внешних. Он смешивает запросы обоих видов с учетом их приоритетов и направляет в ЦП для индивидуального обслуживания. В контроллере имеются собственные регистры управления, с помощью которых устанавливается его режим работы.

Блок сопряжения с магистралью в МП 80186 функционально не отличается от BIU МП 8088, но реализован по-другому, поскольку должен оперировать данными,

поступающими по 16-разрядной шине. Если в МП 8088 очередь команд рассчитана на четыре байта, то МП 80186 – на шесть байтов. Длина очереди команд выбрана так, чтобы исполнительный блок в большинстве реализуемых случаев имел достаточное количество предвыбранных команд и не замедлял бы процессы на магистрали.

МП 8086 и 80186 могут за один цикл магистрали выбрать из памяти 8 или 16 разрядов данных. Если выполняемая команда оперирует над переменной длиной в слово и эта переменная расположена по четному адресу, оба микропроцессора выбирают полное слово за один цикл магистрали. Если слово расположено по нечетному адресу, для его выборки (байт за байтом) требуются два последовательных цикла магистрали.

Система команд МП 80186 включает все команды ранних моделей микропроцессоров, начиная с МП 8080. Дополнительные 10 типов команд позволяют ускорить выполнение существующих программ и оптимизировать программы для МП 80186.

Микросхема 80286

В 1983 г. фирма Intel сделала еще один рывок вперед, выпустив МП 80286. Годом позже фирма IBM объявила, что МП 80286 послужит процессором для их разработки IBM PC/AT (AT от Advanced Technology, т. е. перспективная технология).

Для МП 80286 характерны два режима работы: *реальный* и *виртуальный*. В *реальном режиме* МП 80286 функционирует практически так же, как МП 8086 с повышенным быстродействием. В этом режиме память можно рассматривать, как некоторое число сегментов, каждый из которых содержит максимум 2^{16} байт. Сегменты начинаются с адресов, кратных 16. В любой момент времени процессору доступно содержимое четырех сегментов: текущего сегмента программы (CS), текущего сегмента данных (DS), текущего сегмента стека (SS) и текущего дополнительного сегмента (ES). Сегменты могут задаваться в программах произвольно и могут перекрываться в памяти. Адреса сегментов хранятся в сегментных регистрах.

Полный адрес вычисляется путем помещения в регистр 16 старших битов адреса. Со стороны младших разрядов адрес расширяется четырьмя нулями. Эта процедура выполняется так же, как и в МП 8086.

Для обращения к байтам в сегменте используется 16-разрядный относительный адрес – смещение в сегменте. На рис. П1.12 приведен пример расположения сегментов. Обратите внимание на перекрытие в примере сегментов стека и данных.

В *виртуальном режиме* по-прежнему используются сегменты и относительные адреса в них. Виртуальный режим отличается от реального тем, что начальный адрес сегмента не вычисляется путем расширения содержимого регистра сегмента четырьмя нулями, а извлекается из таблиц, индексируемых с помощью сегментных регистров. Более подробно процедуры вычисления адреса были описаны в гл. 8.

При включении в МП 80286 устанавливается реальный режим и инициализируются различные регистры и флаги. Затем процессор с помощью команды LMSW (загрузка слова состояния машины) переключается из реального в виртуальный режим. Эта команда загружает слово состояния, в котором установлен бит разрешения защиты. После того как МП 80286 перешел в виртуальный режим, вернуть его в реальный режим можно только аппаратным сигналом установки в исходное состояние.

Процессор адресует до 2 Мбайт реальной памяти или 4 Гбайт виртуальной памяти. Однако, поскольку этот процессор разрабатывался для выполнения 16-разрядной арифметики, он оперирует адресами с длиной не более 16 разрядов.

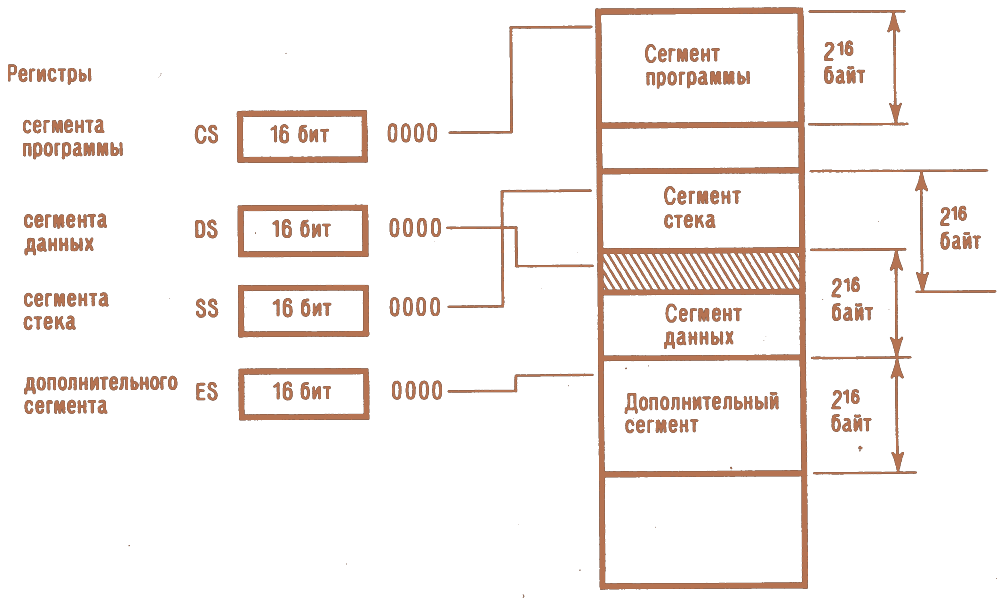


Рис. П1.12. Пример сегментов памяти.

Расширение адресного пространства было получено с помощью виртуального режима.

При разработке МП 80286 контроллер ПДП был вынесен из микросхемы процессора. Канал ПДП служит для обеспечения непрерывности потока обращений к памяти без использования ресурсов ЦП. Применение отдельной микросхемы контроллера позволяет совместить выполнение команд с вводом или выводом из памяти. Контроллер ПДП содержит четыре канала ПДП, что дает возможность подключать до четырех устройств с прямым доступом к памяти.

Система команд МП 80286 включает наборы команд всех ранних моделей, начиная с МП 8086. Кроме того, добавлены несколько новых команд для реального и виртуального режимов. В реальном режиме отличительные характеристики МП 80286 подавляются. Реализуются они в виртуальном режиме.

Сопроцессор 80287. Популярный процессор 8087 для вычислений с плавающей точкой, который использовался с МП 8086 и 8088, был модифицирован специально для МП 80286 с целью реализации нового сопряжения с сопроцессором, разработанного фирмой Intel. Новый сопроцессор получил наименование 80287.

Микросхема 80287 содержит дополнительные регистры для вычислений с плавающей точкой. Операнды с плавающей точкой, предназначенные для команд сопроцессора 80287, размещаются либо в памяти, либо в одном из 8 арифметических регистров. В этих регистрах числа могут храниться только в формате с повышенной точностью.

Микросхемы 80286 и 80287 могут выполнять команды одновременно. Когда МП 80286 обнаруживает команду сопроцессора 80287 (первые пять битов составляют «11011»), он ожидает освобождения сопроцессора, а затем инициирует его для выполнения этой команды. При этом МП 80286 не пребывает в состоянии ожидания. Он переходит к выполнению следующей команды из очереди команд. Если, однако, МП 80286 обнаруживает вторую команду сопроцессора *до того*, как сопроцессор закончил выполнение предыдущей, процессор переходит в состояние ожидания. В противном случае он выполняет следующую команду.

Уровни привилегий. В МП 80286 предусмотрены средства защиты различных областей памяти при работе в виртуальном режиме. Чем меньше номер уровня, тем выше его привилегии. Уровни привилегий представляются тремя различными описателями. Первый – это *дескриптор уровня привилегий* (DPL), который входит в классификацию сегмента. Второй – *затребованный уровень привилегий* (RPL). Третий – *текущий уровень привилегий* (CPL.) Все эти описатели автоматически поддерживаются процессором 80286.

Для того чтобы получить доступ к сегменту с целью чтения или записи, программа должна загрузить селектор этого сегмента в сегментный регистр. Чтобы исключить в программе возможность произвольного доступа к сегменту путем образования селектора и загрузки его в регистр сегмента, МП 80286 возбуждает исключение защиты, если уровень выполняемой программы ниже уровня, присвоенного сегменту.

Многозадачность. Другая особенность МП 80286 состоит в поддержке многозадачного режима путем переключения с одной задачи на другую. МП 80286 создает для каждой задачи сегмент памяти, содержащий всю информацию, необходимую для запуска и останова задачи. Этот специальный сегмент носит название *сегмента состояния задачи* (TSS). Основное назначение TSS заключается в сохранении содержимого регистров задачи на то время, пока эта задача не выполняется.

Микросхема 80386

В этом разделе дан только краткий обзор МП 80386 для сравнения с ранними моделями. Более подробное описание архитектуры и особенностей МП 80386 содержится в гл. 1–7.

Процессоры 8086 и 8088 положили начало промышленному выпуску персональных компьютеров. С них началось также развитие семейства процессоров Intel в направлении расширения функций и повышения производительности. Однако аппаратные усовершенствования переросли архитектуру МП 8086/8088. В табл. П1.1 показан рост функциональных возможностей микропроцессоров фирмы Intel.

МП 80386 представляет высокопроизводительный 32-разрядный микропроцессор, выполняющий до 3–4 млн. операций в секунду. Он отличается полной 32-разрядной архитектурой с физическим адресным пространством 4 Гбайт и встроенными средствами страничной виртуальной памяти, МП 80386 может адресовать до 2^{46} байт = 64 Тбайт виртуальной памяти. На рис. П1.13 показаны функциональные блоки МП 80386.

Таблица П1.1. Сравнение микропроцессоров фирмы Intel

Свойства	8086/8088	80286	80386
Максимальный объем физической памяти	1 Мбайт	16 Мбайт	4 Гбайт
Максимальный объем виртуальной памяти	1 Мбайт	1 Гбайт	64 Тбайт
Максимальный размер сегмента	64 Кбайт	64 Кбайт	64 Кбайт/ /4 Гбайт
Размер очереди предвыборки, байт	4/6	6	12
Размер операндов, бит	8, 16	8, 16	8, 16, 32
Размер регистров, бит	8, 16	8, 16	8, 16, 32
Аппаратура разбиения на страницы	Нет	Нет	Есть
Защита памяти и ввода-вывода	Нет	Есть	Есть
Сопроцессор	8087	80287	80287/80387

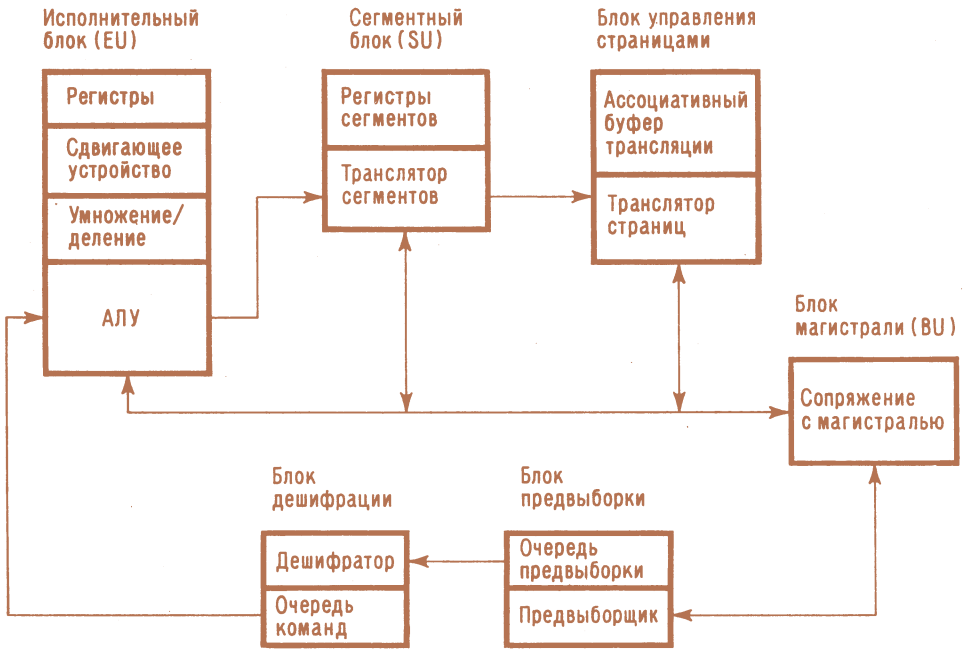


Рис. П1.13. Функциональные блоки МП 80386.

МП 80386 впервые предоставил возможность реализации истинно виртуальной машины. Виртуальная память позволяет очень большим программам или группам программ выполняться в значительно меньшей физической памяти в режиме без перекрытий (оверлеев). Операционная система виртуальной памяти хранит все сегменты или страницы в большом дисковом пространстве, часто называемом областью подкачки [1]. Реальная память значительно меньшего объема содержит только наиболее часто используемые сегменты или страницы.

МП 80386 может осуществлять переключение программ, выполняемых под управлением различных операционных систем, например MS-DOS или Unix. Это позволяет разработчикам программного обеспечения переносить обычные прикладные программы для 16-разрядных машин непосредственно на новые 32-разрядные системы.

Система команд МП 80386 содержит 9 категорий команд: пересылки данных, арифметических, сдвига, обработки строк, обработки битов, передачи управления, поддержки языков высокого уровня, поддержки операционной системы и управления процессором. Длина команды в среднем составляет 3,2 байт. Поскольку очередь команд МП 80386 имеет длину 16 байт, предварительной выборке подвергаются в среднем 5 команд.

Операнды в командах МП 80386 могут иметь длину 8, 16 или 32 разряда. При выполнении программ, предназначенных для 16-разрядных МП 80286 или 8086, длина операндов составляет 8 или 16 разрядов. Два поколения процессоров (8086 и 80286) совместимы с МП 80386 на уровне двоичных кодов. Наиболее отличительной чертой МП 80386 является режим виртуального МП 8086. В этом режиме

---

[1] Или областью свопинга.– *Прим. перев.*

в рамках многозадачной среды МП 80386 устанавливается защищенная среда МП 8086.

Сегменты. Логическое пространство адресов МП 80386 состоит из одного или большего числа сегментов любого размера от 1 байт до 4 Гбайт. Сегменты независимо друг до друга могут быть защищены уровнем привилегий и таким образом обобщенно, но избирательно использоваться различными задачами.

Программа для МП 80386 потенциально может оперировать со многими сегментами. Поэтому в логическом адресе должно содержаться определение сегмента. Логический адрес состоит из 16-разрядного *селектора сегмента* и 32-разрядного *относительного адреса* в выбранном сегменте. Адрес сегмента находится в *таблице дескрипторов*, индексом которой в процессе извлечения адреса требуемого сегмента служит селектор. Процессор прибавляет относительную часть логического адреса к базовому адресу, извлеченному из дескриптора сегмента, в результате чего образуется адрес операнда.

Сопроцессор. МП 80386 может работать совместно как с арифметическим сопроцессором 80287, так и с недавно разработанным сопроцессором 80387. Оба арифметических сопроцессора содержат регистры, повышающие производительность МП 80386. При подключении того или иного сопроцессора к основному процессору регистры сопроцессора используются как эффективное расширение МП 80386.

Уровни привилегий. МП 80386 предоставляет набор механизмов защиты, которые могут избирательно использоваться операционными системами по мере надобности. Многие средства защиты МП 80386 основываются на понятии *иерархии привилегий*. Другими словами, различным задачам или программам могут быть назначены те или иные уровни, используемые другими задачами в процессе их взаимодействия в системе.

Незащищенную систему можно реализовать, попросту поместив все процедуры в сегмент (или сегменты) с уровнем привилегий 0. В слабо защищенной среде операционную систему можно поместить на уровень 0, в то время как прикладные программы будут выполняться на уровне 1 и т. д.

В защищенной системе уровень привилегий задачи определяет, какие команды она может выполнять и к какому подмножеству сегментов в своем адресном пространстве может обращаться. Попытка задачи использовать сегмент с более высоким уровнем защиты приводит к исключению общей защиты.

Кроме определения сегментов, доступных задаче, уровень привилегий задачи определяет и команды, разрешенные для выполнения. При разработке МП 80386 в его систему команд были включены команды, выполнение которых должно строго контролироваться, чтобы избежать разрушения системы. Эти команды называются *привилегированными*.

Многозадачность. Многозадачность – это методика организации работы системы, когда эта работа включает множество действий. Три примера таких действий: редактирование одного файла, компилирование другого, перенос третьего на другую вычислительную систему. В многозадачной системе каждое действие, протекающее параллельно с другими действиями, называется задачей или процессом.

Операционная система имитирует многопроцессорный комплекс путем предоставления каждой задаче *виртуального процессора*. В каждый момент времени операционная система назначает реальный процессор одному из виртуальных процессоров, выполняющих связанную с ним задачу. Операционная система достаточно часто переключает реальный процессор от одного виртуального процессора другому, поддерживая таким образом иллюзию, что каждая задача работает со своим собственным процессором.

Операционная система сменяет выполняемые задачи в соответствии с алгоритмом планирования задач, задающим порядок, в котором должны выполняться задачи. МП 80386 хранит селектор и дескриптор сегмента состояния (TSS) выполняемой задачи в *регистре задачи*.

Каждая задача может использовать пространство логических адресов, доступных всей системе, описание которого хранится в *таблице глобальных дескрипторов* (GDT), а также пространство локальных логических адресов, описанное в *таблице локальных дескрипторов* (LDT). Каждая из этих таблиц может содержать до 8192 дескрипторов. Обе они вместе определяют логическое пространство адресов задачи.

Приложение 2

Перечень регистров, битов и флагов

Ниже приведен алфавитный перечень всех регистров, используемых в процессоре 80386.

AH	Старший байт регистра AX
AL	Младший байт регистра AX
AX	16-разрядный регистр, часть EAX
BH	Старший байт регистра BX
BL	Младший байт регистра BX
BP	16-разрядный регистр, часть EBP
BX	16-разрядный регистр, часть EBX
CH	Старший байт регистра CX
CL	Младший байт регистра CX
CR0	Регистр управления, младшие 16 разрядов MSW
CR1	Регистр управления, зарезервирован
CR2	Регистр линейного адреса отказа страницы
CR3	Базовый адрес каталога страниц
CX	16-разрядный регистр, часть ECX
DH	Старший байт регистра DX
DI	16-разрядный регистр, часть EDI
DL	Младший байт регистра DX
DR0	Регистр отладки, линейный адрес точки останова 0
DR1	Регистр отладки, линейный адрес точки останова 1
DR2	Регистр отладки, линейный адрес точки останова 2
DR3	Регистр отладки, линейный адрес точки останова 3
DR4	Регистр отладки, зарезервирован фирмой Intel
DR5	Регистр отладки, зарезервирован фирмой Intel
DR6	Регистр отладки, состояние точки останова
DR7	Регистр отладки, управление точкой останова
DS	Регистр сегмента данных
DX	16-разрядный регистр, часть EDX
EAX	32-разрядный регистр общего назначения
EBP	32-разрядный регистр общего назначения, указатель базы кадра стека
EBX	32-разрядный регистр общего назначения
ECX	32-разрядный регистр общего назначения
EDI	32-разрядный регистр общего назначения
EDX	32-разрядный регистр общего назначения
EFLAGS	Регистр флагов
EIP	Указатель команд
ES	Регистр сегмента данных

ESI	32-разрядный регистр общего назначения
ESP	Регистр указателя стека
FLAGS	Младшие 16 разрядов EFLAGS
FS	Регистр сегмента данных
GDT	Таблица глобальных дескрипторов
GS	Регистр сегмента данных
IDT	Таблица дескрипторов прерываний
IP	Указатель команд, младшие 16 разрядов
LDT	Таблица локальных дескрипторов
SI	16-разрядный регистр, часть ESI
SP	16-разрядный регистр, часть ESP
SS	Регистр сегмента стека
TR6	Тестовый регистр кэш-памяти страниц, управление тестированием
TR7	Тестовый регистр кэш-памяти страниц, состояние тестирования
TSS	Сегмент состояния задачи

Ниже приведен перечень флагов, используемых в МП 80386 и системах на его основе, с указанием содержащих их регистров. Заметьте, что биты содержимого регистра EFLAGS обычно называются «флагами», в остальных случаях они просто «биты».

Бит/флаг	**Местоположение**	**Описание**
A	Бит 5 входа таблицы страниц	Обращение
AF	Бит 4 регистра EFLAGS	Флаг вспомогательного переноса
CF	Бит 0 регистра EFLAGS	Флаг переноса
D	Бит 6 входа таблицы страниц	Бит «мусора»
DF	Бит 10 регистра EFLAGS	Флаг направления
EM	Бит 2 регистра CR0	Эмуляция сопроцессора
ET	Бит 4 регистра CR0	Тип процессора расширения
IF	Бит 9 регистра EFLAGS	Флаг разрешения прерываний
IOPL	Бит 13/12 регистра EFLAGS	Флаг уровня привилегий ввода-вывода
MP	Бит 1 регистра CR0	Управление сопроцессором
NT	Бит 14 регистра EFLAGS	Флаг вложенной задачи
OF	Бит 11 регистра EFLAGS	Флаг переполнения
P	Бит 0 входа таблицы страниц	Бит присутствия
PE	Бит 0 регистра CR0	Защита включена
PF	Бит 2 регистра EFLAGS	Флаг четности
PG	Бит 31 регистра CR0	Разбиение на страницы включено
R/W	Бит 1 входа таблицы страниц	Чтение/запись
RF	Бит 16 регистра EFLAGS	Флаг возобновления
SF	Бит 7 регистра EFLAGS	Флаг знака
TF	Бит 8 регистра EFLAGS	Флаг ловушки
TS	Бит 3 регистра CR0	Переключение задачи
U/S	Бит 2 входа таблицы страниц	Пользователь/супервизор
VM	Бит 17 регистра EFLAGS	Флаг виртуального режима 8086
ZF	Бит 6 регистра EFLAGS	Флаг нуля

Приложение 3

Мнемонические обозначения и условные сокращения

AC	Alternating Current – переменный ток
ACC	Accumulator – аккумулятор
ACK	Acknowledge – подтверждение
ADMA	Advanced Direct Memory Accsess – прямой доступ к памяти с упреждением
ADS#	Address Status Pin on 80386 – вывод состояния адреса МП 80386
ALU	Arithmetic/Logic Unit – арифметико-логическое устройство, АЛУ
ASCII	American Standard Code for Information Interchange – американский стандартный код обмена информацией
ASR	Automatic Send and Receive - автоматическая приемопередача
A0-Ann	Address Pins from A0 to a number, usually 32 – адресные выводы от A0 до Ann, обычно nn = 32
BCD	Binary Coded Decimal – число в двоично-десятичном коде
BCR	Byte Count Register – регистр счета байтов
BEn#	Byte Enable Pins on 80386, n = 0 to 3 – выводы разрешения байта МП 80386, n = 0...3
BIU	Bus Interface Unit – блок сопряжения с магистралью
BPS	Bits Per Second – бит в секунду
BS16#	Bus Size Pin on 80386 – вывод размера шины МП 80386
BSC	Binary Synchronous Communication – двоичная синхронная передача
CAD	Computer-Aided Design – система автоматизированного проектирования, САПР
CAM	Computer-Aided Manufacturing, also Content-Addressable Memory – автоматизированная система управления производством, АСУП, также ассоциативная память
CAS	Column Address Select – выбор адреса столбца
CCD	Charge-Coupled Device – прибор с зарядовой связью, ПЗС
CE	Chip Enable – разрешение микросхемы
CLK	Clock – такт, тактовый генератор
CML	Current Mode Logic – логика текущего режима
CMOS	Complementary Metal Oxide Semiconductor – комплементарная МОП-структура, КМОП
CPG	Clock Pulse Generator – генератор тактовых импульсов
CPL	Current Privilege Level – текущий уровень привилегий
CPS	Characters Per Second – символов в секунду
CPU	Central Processing Unit – центральный процессор, ЦП
CRn	Control Registers, 0 to 3 – регистры управления, 0...3
CRT	Cathode Ray Tube – электронно-лучевая трубка, ЭЛТ
CRTC	CRT Controller – контроллер ЭЛТ

CS	Chip Select, also Code Segment Register – выбор микросхемы, также регистр сегмента программы
CU	Control Unit – устройство управления
DC	Direct Current – постоянный ток
DCD	Data Carrier Detect – детектор принимаемого линейного сигнала данных
DIP	Dual In-Line Pins or Package – корпус с двухрядным расположением выводов, DIP-корпус
DMA	Direct Memory Access – прямой доступ к памяти, ПДП
DMAC	DMA Controller – контроллер ПДП
DOS	Disk Operating System – дисковая операционная система
DPL	Descriptor Privilege Level – уровень привилегий дескриптора
DRAM	Dynamic Random Access Memory – динамическое ЗУПВ, динамическое ОЗУ
DS	Data Segment Register – регистр сегмента данных
DTL	Diod Transistor Logic – диодно-транзисторная логика, ДТЛ
DO-Dnn	Data Lines zero through number – линии данных, 0...nn
E	Enable – разрешение, включение
EA	Effective Address – исполнительный адрес
EAROM	Electrically Alterable Read Only Memory – электрически программируемое ПЗУ, ЭППЗУ
EBCDIC	Extended Binary-Coded-Decimal Information Code – расширенный двоично-десятичный код обмена информацией
ECL	Emitter Coupled Logic – эмиттерно-связанная логика, ЭСЛ
EIP	Instruction Pointer, 32-bit implementation – указатель команд, 32-разрядная реализация
EMI	Electomagnetic Interface – электромагнитный интерфейс
EOF	End of File – конец файла
EOR	Exclusive OR – ИСКЛЮЧАЮЩЕЕ ИЛИ
EOT	End of Tape, or End of Text – конец ленты, также конец текста
EPL	Effective Privilege Level – эффективный уровень привилегий
EPROM	Erasable Programmable Read Only Memory – стираемое программируемое ПЗУ, СППЗУ
ES	Extra Segment Register – регистр дополнительного сегмента (экстрасегмента)
ESI	Electrostatic Interface – электростатический интерфейс
ESP	Extended Stack Pointer – расширенный указатель стека
EU	Execution Unit – исполнительный блок
FDC	Floppy Disk Controller – контроллер гибкого диска
FDM	Frequency-Division Multiplexing – частотное мультиплексирование (с разделением частот)
FET	Field Effect Transistor – полевой транзистор
FF	Flip-Flop – триггер
FIFO	First-In-First-Out – первым вошел, первым вышел
FPLA	Field Programmable Logic Array – логическая матрица, программируемая пользователем
FSK	Frequency Shift Keying – частотная манипуляция
GDT	Global Descriptor Table – таблица глобальных дескрипторов
GDTR	Global Descriptor Table Register – регистр таблицы глобальных дескрипторов
GP	General Purpose – общего назначения

GPIB	General Purpose Interface Bus – магистраль интерфейса общего назначения
GR	General Registers – регистры общего назначения, РОН
HDLC	High-Level Data Link Control – высокоуровневый протокол управления каналом
HEX	Hexadecimal – шестнадцатеричный
HMOS	High-Density Metal Oxide Semiconductor – МОП ИС с высокой плотностью упаковки
I	Interrupt, also Interrupt Mask – прерывание, также маска прерывания
IC	Integrated Circuit – интегральная схема, ИС
IDT	Interrupt Descriptor Table – таблица дескрипторов прерываний
IDTR	Interrupt Descriptor Table Register – регистр таблицы дескрипторов прерываний
INT	Interrupt – прерывание
INTR	Maskable Interrupt – маскируемое прерывание
I/O	Input/Output – ввод-вывод
IOCS	Input/Output Control System – система управления вводом-выводом
IOPL	Input/Output Privilege Level – уровень привилегий ввода-вывода
IP	Instruction Pointer, 16-bit implementation – указатель команд, 16-разрядная реализация
IRET	Interrupt Return – возврат из прерывания
IPQ	Interrupt Request – запрос прерывания
JP	Jump – переход
K	Kilo, also 1024 – K (= 1024)
KSR	Keyboard Send Receive – телетайп
LAN	Local Area Network – локальная вычислительная сеть, ЛВС
LCD	Liquid Cristal Display – жидкокристаллический дисплей
LED	Light-Emitting Diod – светодиод
LDT	Local Descriptor Table – таблица локальных дескрипторов
LDTR	Local Descriptor Table Register – регистр таблицы локальных дескрипторов
LIFO	Last-In-First-Out – последним вошел, первым вышел
LP	Line Printer – устройство построчной печати
LPM	Lines Per Minute – строк в минуту
LSB	Least Significant Bit – младший бит (разряд)
LSI	Large Scale Integration – большая интегральная схема (БИС)
M/IO	Memory and Input/Output – память и ввод-вывод
MMU	Memory Management Unit – диспетчер памяти
MOFFS	Memory Offset – смещение в памяти
MOS	Metal Oxide Semiconductor – структура металл – окисел – полупроводник, МОП-транзистор
MOSFET	Metal Oxide Semiconductor Field Effect Transistor – полевой МОП-транзистор
MPSC	Multi-Protocol Serial Controller – мультипротокольный последовательный контроллер
MPU	Microprocessor Unit – микропроцессор, МП
MSB	Most Significant Bit – старший бит (разряд)
MSI	Medium Scale Integration – ИС средней степени интеграции, СИС
MSW	Machine Status Word (used on 80286) – слово состояния машины (используется в МП 80286)

MUX	Multiplexer – мультиплексор
N	Negative – отрицательный
NDRO	Non-Destructive Read Out – считывание без разрушения информации
NMI	Non-Maskable Interrupt – немаскируемое прерывание
NMOS	N-Channel Metal Oxide Semiconductor – *n*-канальный МОП-транзистор
OEM	Original Equipment Manufacturer – изготовитель комплексного оборудования из комплектующих изделий
OP	Operation – операция
OPLA	One-Page Lookahead – с предвыборкой страницы
OS	Operating System – операционная система, ОС
P	Positive – положительный
PAL	Programmable Array Logic – программируемая логическая матрица, ПЛМ
PC	Printed Circuit, also Personal Computer – печатная плата, также персональный компьютер
PCI/O	Program Controlled Input/Output – программно-управляемый ввод-вывод
PCM	Pulse Code Modulation – кодово-импульсная модуляция, КИМ
PDBR	Page Directory Base Register – регистр базы каталога страниц
PEREQ	Coprocessor Request – запрос сопроцессора
PFR	Power Fail Restart – рестарт по сбою питания
PGA	Pin Grid Array – корпус с матричным расположением выводов
PIC	Priority Interrupt Control – управление приоритетом прерываний
PIO	Programmable Input/Output Chip (Interface) – программируемый интерфейс ввода-вывода
PIT	Programmable Interval-Timer – программируемый интервальный таймер
PL	Privilege Level – уровень привилегий
PLA	Programmable Logic-Array – программируемый логический массив
PLL	Phase-Locked Loop – фазовая автоподстройка частоты, ФАПЧ
PMOS	P-Channel Metal Oxide Semiconductor – *p*-канальный МОП-транзистор
PROM	Programmable Read Only Memory – программируемое ПЗУ, ППЗУ
PSW	Program Status Word – слово состояния программы
R	Read – чтение
R/W	Read/Write – чтение/запись
RALU	Register Arithmetic Logic Unit – регистровое АЛУ, РАЛУ
RAM	Random Access Memory – запоминающее устройство с произвольной выборкой, ЗУПВ; оперативное запоминающее устройство, ОЗУ
RAS	Row Address Select – выбор адреса строки
RDSR	Receiver Data Service Request – запрос приемника на обслуживание данных
RDY	Ready – готовность
RES	Reset – установка в исходное состояние, сброс
ROM	Read Only Memory – постоянное запоминающее устройство, ПЗУ
RPL	Requestor's Privilege Level – уровень привилегий запрашивающего устройства

RPT	Repeat – повторение
RS	Register Select – выбор регистра
RST	Restart – рестарт, перезапуск
RTC	Real-Time Clock – часы истинного времени
RTS	Request To Send – запрос передачи
SDLC	Synchronous Data Link Control – синхронное управление передачей данных
Si	Silicon (element) – кремний
SIB	Scale Index Base – масштаб – индекс – база
SIP	Single In-Line Package – корпус с однорядным расположением выводов
SLSI	Super Large Scale Integration – сверхбольшая интегральная схема
SOS	Silicon-On-Saphire – структура типа «кремний на сапфире», КНС-структура
SP	Stack Pointer – указатель стека
SR	Service Request – запрос обслуживания
SRAM	Static Random Access Memory – статическое ЗУПВ
SS	Stack Segment Register – регистр сегмента стека
SSI	Small Scale Integration – ИС малой степени интеграции, МИС
STB	Strobe – строб
STEN	Status Enable – разрешение состояния
SUB	Subroutine – подпрограмма
TDM	Time Division Multiplexing – мультиплексирование с разделением времени
TSS	Task State Segment – сегмент состояния задачи
TLB	Translation Lookaside Buffer – буфер ассоциативной трансляции
TOS	Top Of Stack – вершина стека
TTL	Transistor-Transistor Logic – транзисторно-транзисторная логика, ТТЛ
U/S	User/Supervisor – пользователь/супервизор
USART	Universal Synchronous/Asynchronous Receiver Transmitter – универсальный синхронно-асинхронный приемопередатчик, УСАПП
V_{ss}	Ground – заземление, общая линия
VLSI	Very Large Scale Integration – сверхбольшие интегральные схемы, СБИС
VM	Virtual Mode, also Virtual Machine – виртуальный режим, также виртуальная машина
W	Write – запись
WPM	Words Per Minute – слов в минуту
X	Index – индекс
XOR	Exclusive OR – ИСКЛЮЧАЮЩЕЕ ИЛИ
Z	Zero Bit – нулевой бит

Приложение 4

Магистрали MULTIBUS I/II и микропроцессор 80386

Одним из наиболее важных элементов вычислительной системы является структура системной магистрали, осуществляющей сопряжение всех аппаратных средств. Системная магистраль обеспечивает взаимодействие друг с другом различных компонентов системы и совместное использование системных ресурсов. Последнее обстоятельство играет важную роль в существенном увеличении производительности всей системы. Кроме того, системная магистраль обеспечивает передачу данных с участием памяти и устройств ввода-вывода, прямой доступ к памяти и возбуждение прерываний.

Системные магистрали обычно выполняются таким образом, что сбои, происходящие в других частях системы, не влияют на их функционирование. Это увеличивает общую надежность системы. Примерами магистралей общего назначения являются предложенные фирмой Intel архитектуры MULTIBUS I и II, обеспечивающие коммуникационный канал для координации работы самых разнообразных вычислительных модулей.

Краткое описание магистралей MULTIBUS I и II было дано в гл. 12. В настоящее приложение включен небольшой обзор, но в основном оно посвящено совместному использованию этих магистралей и МП 80386.

MULTIBUS I и MULTIBUS II используют концепцию «ведущий – ведомый». *Ведущим* является любой модуль, обладающий средствами управления магистралью. Ведущий с помощью логики доступа к магистрали захватывает магистраль, затем генерирует сигналы управления и адреса и сами адреса памяти или устройств ввода-вывода. Для выполнения этих действий ведущий оборудуется либо блоком центрального процессора, либо логикой, предназначенной для передачи данных по магистрали к местам назначения и от них. *Ведомый* – это модуль, декодирующий состояние адресных линий и действующий на основании сигналов, полученных от ведущих; ведомый *не* может управлять магистралью. Процедура обмена сигналами между ведущим и ведомым позволяет модулям различного бестродействия взаимодействовать через магистраль. Ведущий магистрали может отменить действия логики управления магистралью, если ему необходимо гарантировать для себя использование циклов магистрали. Такая операция носит название «блокирования» магистрали; она временно предотвращает использование магистрали другими ведущими.

Другой важной особенностью магистрали является возможность подключения многих ведущих модулей с целью образования многопроцессорных систем.

MULTIBUS I позволяет передать 8- и 16-разрядные данные и оперировать с адресами длиной до 24 разрядов.

MULTIBUS II воспринимает 8-, 16- и 32-разрядные данные, а адреса длиной до 32 разрядов. Протоколы магистралей MULTIBUS I и II подробно описаны в документации фирмы Intel, которую следует тщательно изучить перед использованием этих магистралей в какой-либо системе.

MULTIBUS I

MULTIBUS I фирмы Intel представляет собой 16-разрядную многопроцессорную систему, согласующуюся со стандартом IEEE 796. На рис. П4.1 приведена структурная схема сопряжения с магистралью MULTIBUS I. На рисунке не показана локальная шина и локальные ресурсы МП 80386

Пример интерфейса магистрали MULTIBUS I

Один из способов организации взаимодействия между МП 80386 и магистралью MULTIBUS I заключается в генерации всех сигналов MULTIBUS I с помощью

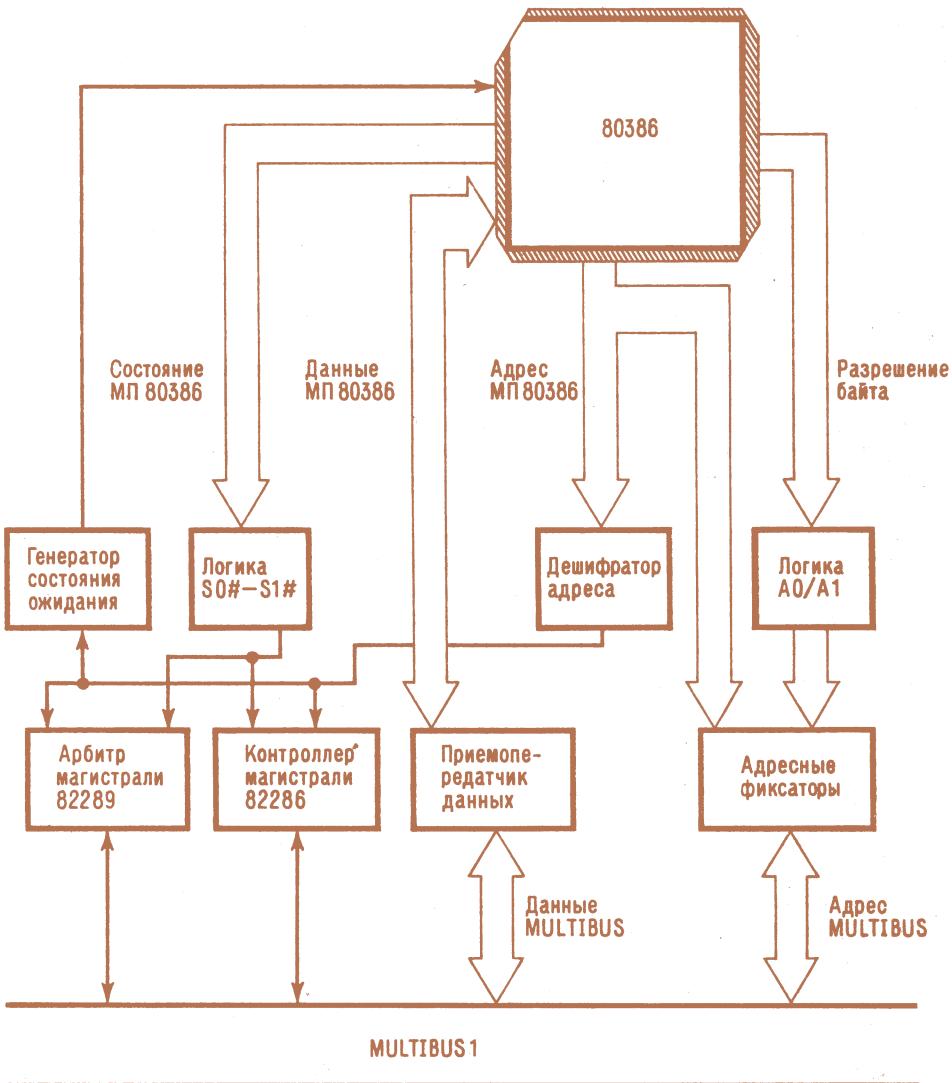


Рис. П4.1. Структурная схема сопряжения МП 80386 с магистралью MULTIBUS I.

программируемых логических матриц (ПЛМ) и схем ТТЛ. Проще использовать интерфейс, совместимый с МП 80286. Основные черты этого интерфейса описаны ниже.

Интерфейс магистрали MULTIBUS I состоит из совместимого с МП 80286 арбитра магистрали 82289 и контроллера магистрали 82288. Контроллер может работать как в режиме локальной магистрали, так и в режиме MULTIBUS I; резистор на входе MB схемы 82288, подключенный к источнику питания, активизирует режим MULTIBUS I. Выходной сигнал MBEN дешифратора адреса на ПЛМ служит сигналом выбора обеих микросхем 82288 и 828289. Сигнал AEN # с выхода 82289 открывает выходы контроллера 82288.

Взаимодействие между процессором 80386 и этими двумя устройствами осуществляется с помощью ПЛМ, в которые записаны программы генерации и преобразования необходимых сигналов. Арбитр 82289 вместе с арбитрами магистрали других вычислительных подсистем координирует управление магистралью MULTIBUS I, обеспечивая управляющие сигналы, необходимые для получения доступа к ней.

В системе MULTIBUS I каждая вычислительная подсистема претендует на использование общих ресурсов. Если подсистема запрашивает доступ к магистрали, когда другая система уже использует магистраль, первая подсистема должна ожидать ее освобождения. Логика арбитража магистрали управляет доступом к магистрали всех подсистем. Каждая вычислительная подсистема имеет собственный арбитр магистрали 82289. Арбитр подключает свой процессор к магистрали и разрешает доступ к ней ведущим с более высокими или более низкими приоритетами в соответствии с установленной заранее схемой приоритетов.

Возможны два варианта процедуры управления занятием магистрали: с последовательным и с параллельным приоритетом. Схема последовательного приоритета реализуется путем соединения цепочкой входов приоритета магистрали (BPRN #) и выходов приоритета магистрали (BPRO #) всех арбитров магистрали в системе. Задержка, возникающая при таком соеднинении, ограничивает число подключаемых арбитров. Схема параллельного приоритета требует наличия внешнего арбитра, который принимает входные сигналы BPRN# от всех арбитров магистрали и возвращает активный сигнал BPRO# запрашивающему арбитру с максимальным приоритетом. Максимальное число арбитров, участвующих в схеме с параллельным приоритетом, определяется сложностью схемы дешифрации.

После завершения цикла MULTIBUS I арбитр, занимающий магистраль, либо продолжает ее удерживать, либо освобождает с передачей другому арбитру. Процедура освобождения магистрали может быть различной. Арбитр может освобождать магистраль в конце каждого цикла, удерживать магистраль до тех пор, пока она не будет затребована ведущим с более высоким приоритетом, или освобождать магистраль при поступлении запроса от ведущего с любым приоритетом.

Система MULTIBUS I оперирует с 24 линиями адреса и 16 линиями данных. Адреса системы расположены в диапазоне 256 кбайт (между F00000H и F3FFFFH), причем используются все 24 линии. 16 линий данных представляют младшую половину (младшие 16 разрядов) 32-разрядной шины данных МП 80386. Адресные разряды MULTIBUS I нумеруются в шестнадцатеричной системе; A23–A0 в МП 80386 становятся ADR17#–ADR0# в системе MULTIBUS I. Инвертирующие адресные фиксаторы поразрядно преобразуют выходные сигналы адреса МП 80386 в адресные сигналы с низким активным уровнем для магистрали MULTIBUS I.

Дешифратор адреса. Система MULTIBUS I обычно включает и общую, и локальную память. Устройства ввода-вывода (УВВ) также могут быть расположены как на локальной магистрали, так и на MULTIBUS I. Отсюда следует, что:

1) пространство адресов МП 80386 должно быть разделено между MULTIBUS I и локальной магистралью и 2) должен использоваться дешифратор адресов для выбора одной из двух магистралей. Для выбора магистрали MULTIBUS I требуются два сигнала:

1. Сигнал разрешения MULTIBUS I (MBEN) служит сигналом выбора контроллера магистрали 82288 и арбитра магистрали 82289 в схеме сопряжения с MULTIBUS I. Другие выходы ПЛМ дешифратора служат для выбора памяти и УВВ на локальной магистрали.

2. Для обеспечения 16-разрядного цикла магистрали процессору 80386 должен быть возвращен активный сигнал размера шины BS16#. К уравнению ПЛМ, описывающему условия возбуждения сигнала BS16#, могут быть добавлены дополнительные члены для других устройств, требующих 16-разрядной шины (пример уравнения ПЛМ см. в гл. 7. Обращайтесь на фирму Intel для получения уравнений ПЛМ).

Ресурсы ввода-вывода, подключенные к магистрали MULTIBUS I, могут быть либо отображены на отдельное пространство адресов ввода-вывода, независимых от физического расположения устройств на магистрали I, либо отображены на пространство адресов памяти МП 80386. Адреса УВВ, отображенных на пространство памяти, должны декодироваться для возбуждения правильных команд ввода-вывода. Это декодирование должно осуществляться для всех обращений к памяти, попадающих в область отображения адресов ввода-вывода.

Адресные фиксаторы и приемопередатчики данных. Адрес во всех циклах магистрали должен фиксироваться, потому что по протоколу MULTIBUS I на адресных выходах должен удерживаться достоверный адрес по крайней мере 50 нс после того, как команда MULTIBUS I становится пассивной. Сигнал разрешения адреса (AEN#) на выходе арбитра магистрали 82289 становится активным, как только арбитр получает управление магистралью MULTIBUS I. Сигнал AEN# действует как разрешающий для фиксаторов MULTIBUS I. Как показано на рис. П4.2, выходной сигнал ALE# контроллера магистрали 82288 фиксирует адрес от МП 80386.

Разряды данных MULTIBUS I нумеруются в шестнадцатеричной системе, так что D15–D0 превращается в DATF#–DAT0#. Инвертирующие фиксаторы и приемопередатчики вырабатывают активный низкий уровень для магистрали MULTIBUS I. Данные фиксируются только в циклах записи. Во время цикла *записи* адресными фиксаторами и фиксаторами-приемопередатчиками данных управляют входные сигналы ALE#, DEN и DT/R# от контроллера 82288. В циклах чтения

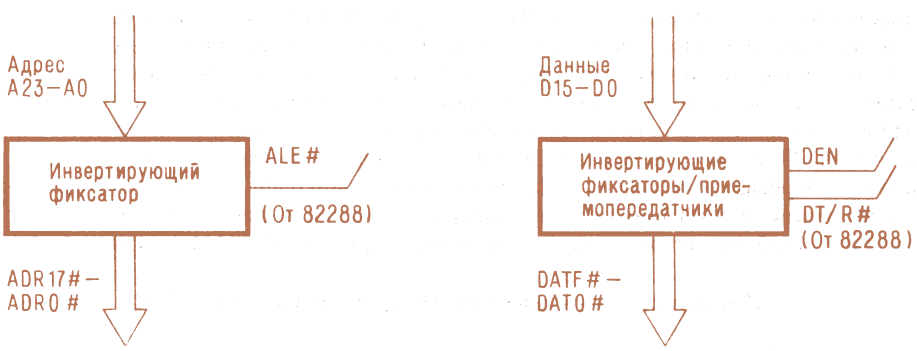


Рис. П4.2. Пример адресных фиксаторов и приемопередатчиков данных магистрали MULTIBUS I.

фиксаторы-приемопередатчики управляются сигналом локальной магистрали RD#. Если при использовании сигнала DEN за локальным циклом записи немедленно последует цикл чтения MULTIBUS I, на локальной магистрали МП80386 возникнет конфликтная ситуация.

Магистраль расширения ввода-вывода iSBX

Магистраль iSBX независима от типа процессора или платы. Каждый интерфейс расширения непосредственно поддерживает до шестнадцати 8-разрядных портов ввода-вывода. Посредством ведомых процессоров или процессоров с плавающей точкой обеспечивается расширение адресных возможностей. Кроме того, каждый интерфейс расширения может при необходимости поддерживать канал ПДП со скоростью передачи до 2 Мслов/с.

Магистраль iSBX включает два основных элемента: базовую плату и модуль расширения. *Базовая плата* – это любая плата с одним или несколькими интерфейсами расширения ввода-вывода (коннекторами), удовлетворяющими электрическим и механическим требованиям спецификации Intel. Естественно, базовая плата всегда является ведущим устройством, она генерирует все адреса, сигналы выбора и команды.

Модуль расширения магистрали iSBX представляет собой небольшую специализированную плату ввода-вывода, подключенную к базовой плате. Модуль может иметь одинарную или двойную ширину. Назначение модуля расширения – преобразование протокола основной магистрали в протокол конкретного устройства ввода-вывода.

Расширение функций, реализуемых каждой системной платой, подключенной к магистрали MULTIBUS I, повышает производительность системы, потому что для доступа к таким резидентным функциям не требуется арбитраж магистрали.

Многоканальная магистраль

Многоканальная магистраль представляет собой специализированный электрический и механический протокол, действующий как составная часть системы MULTIBUS I. Эта магистраль предназначена для скоростной блочной пересылки данных между системой MULTIBUS I и взаимосвязанными периферийными устройствами. В тех случаях, когда требуется переслать группу байтов или слов, расположенных (или располагаемых) по последовательным адресам, протокол блочной пересылки данных уменьшает непроизводительные потери. Передача осуществляется в асинхронном режиме с использованием протокола подтверждений и с проверкой четности, обеспечивающей правильность передачи данных.

Улучшению характеристик системы MULTIBUS I способствует уменьшение влияния на ее производительность периферийного оборудования пакетного типа. Потоки данных от пакетных устройств могут использовать интерфейс общего назначения. Протокол многоканальной магистрали специально приспособлен для пакетных пересылок данных. Максимальный выигрыш в производительности получается при использовании двухпортовой памяти с доступом как со стороны многоканальной магистрали, так и со стороны интерфейса MULTIBUS I.

Магистраль локального расширения iLBX

Магистраль iLBX предназначена для непосредственных скоростных передач данных между ведущими и ведомыми и обеспечивает: 1) максимум два ведущих на магистрали, что упрощает процедуру арбитража; 2) асинхронный по отношению к

передаче данных арбитраж магистрали; 3) минимум два и максимум пять устройств, связанных этой магистралью; 4) ведомые устройства, определяемые как ресурсы памяти с байтовой адресацией, и 5) ведомые устройства, функции которых непосредственно контролируются сигналами линий магистрали iLBX.

Увеличение локальных (на плате) ресурсов памяти высокопроизводительного процессора улучшает характеристики всей системы. Что же касается других специальных функций, то наличие на процессорной плате памяти повышает производительность, поскольку процессор может адресовать память непосредственно, не ожидая результатов арбитража магистрали. С другой стороны, в силу пространственных ограничений на процессорной плате удается разместить память лишь небольшого объема. Магистраль iLBX позволяет снизить эти пространственные ограничения. При использовании магистрали iLBX нет необходимости в размещении дополнительной памяти на процессорной плате. Вся память (объемом много Мбайт), адресуемая процессором, доступна через магистраль iLBX и представляется процессору размещенной на процессорной плате. Наличие в системе памяти двух портов – одного для обмена с магистралью iLBX, а другого для обмена с интерфейсом MULTIBUS I – делает эту память доступной другим компонентам системы. К магистрали iLBX можно подключить до пяти устройств. В число устройств должны входить первичный ведущий и один ведомый. Остальные три устройства могут включать дополнительные ведомые и один вторичный ведущий; эти устройства не являются обязательными. Первичный ведущий управляет магистралью iLBX и организует доступ вторичного ведущего к ресурсам ведомой памяти. Вторичный ведущий, если он есть, предоставляет дополнительные возможности доступа к ведомым ресурсам по магистрали iLBX.

Замечания по проектированию систем на базе MULTIBUS I

Используя MULTIBUS I с МП 80386, следует принимать во внимание ряд соображений, относящихся к обработке прерываний, пересылкам 8-разрядных данных, защите по тайм-ауту и обработке сбоев питания. Детальное описание этих вопросов можно найти в последнем справочнике фирмы Intel по архитектуре системы MULTIBUS I. Ниже изложена только суть дела.

Обработка прерываний. МП 80386, получив запрос на прерывание, генерирует цикл подтверждения прерывания, в процессе которого получает 8-разрядный вектор прерывания от устройства типа программируемого контроллера прерываний 8259A. Контроллер располагается либо на магистрали MULTIBUS I, либо на локальной магистрали. Несколько контроллеров 8259A (один ведущий и до 8 ведомых) можно включить последовательно и обрабатывать до 64 прерываний.

МП 80386, получив активный сигнал INTR, выполняет два цикла магистрали. В первом цикле ведущий контроллер прерываний определяет, какой из его ведомых контроллеров должен вернуть вектор прерывания. Во втором цикле МП 80386 считывает вектор, выставленный выбранным контроллером прерываний, и использует его для обработки прерывания.

8-разрядные пересылки. Все байтовые пересылки по магистрали MULTIBUS I выполняются с использованием младших восьми разрядов данных независимо от адреса этих данных. МП 80386 перед пересылкой данных в магистраль MULTIBUS I должен переместить данные со старших 24 линий данных (D8 – D15, D16 – D23 или D24 – D31) на младшие 8 линий (D0 – D7). Получив байт данных из магистрали MULTIBUS I, МП 80386 должен переместить данные со своих младших линий данных на соответствующую восьмерку старших линий. Это перемещение байтов выполняется одинаково на 8-, 16- и 32-разрядных системах, совместно использующих одну и ту же магистраль MULTIBUS I.

Во всех циклах магистрали MULTIBUS I генерируется и возвращается процессору 80386 сигнал BS16≠. МП 80386 автоматически перемещает данные на своей магистрали и, если требуется, добавляет дополнительный цикл магистрали для завершения пересылки данных. Таким образом, требуется только добавить логику перемещения данных с линий D8–D15 на линии D0–D7.

Тайм-ауты магистрали. Сигнал XACK≠ (подтвержение пересылки) магистрали MULTIBUS I завершает цикл магистрали МП 80386, запуская генератор состояния ожидания. Если МП 80386 адресует на магистрали MULTIBUS I несуществующее устройство, сигнал XACK# не возбуждается. При отсутствии в системе схемы защиты по тайм-ауту МП 80386 может войти в состояние бесконечного ожидания активного сигнала READY#, что лишит другие процессоры возможности использовать магистраль MULTIBUS I.

Обработка сбоев питания. В состав протокола MULTIBUS I входит сигнал прерывания по сбою питания (PFIN#), индицирующий ожидаемое нарушение питания системы. Обычно сигнал PFIN# подключается ко входам немаскируемых прерываний всех МП 80386. Программа обработки немаскируемого прерывания должна немедленно сохранить среду МП 80386, пока снижающееся напряжение питания и сигнал защиты памяти (MPRO#) не приведут к прекращению дальнейшего функционирования памяти. Если в системе используется память с резервным питанием или энергонезависимая память, сохраненная среда может быть восстановлена при повторном включении питания.

Процедура включения питания МП 80386 должна проверить состояние фиксатора датчика сбоя питания магистрали MULTIBUS I (PFSN #). Если перед этим имел место сбой питания, МП 80386 может передать управление программе, которая сбросит фиксатор с помощью сигнала сброса датчика сбоя питания (PFSR#) и восстановит сохраненную среду МП 80386 с целью продолжения выполнения задачи.

MULTIBUS II

Архитектура системы MULTIBUS II является процессорно-независимой. Она отличается наличием 32-разрядной параллельной системной магистрали с максимальной скоростью передачи 40 Мбайт/с, недорогой последовательной системной магистрали и быстродействующей локальной магистрали для доступа к отдельным платам памяти. MULTIBUS II включает пять магистралей Intel: 1) локального расширения (iLBX II), 2) многоканальную прямого доступа к памяти, 3) параллельную системную (iPSB), 4) последовательную системную (iSSB) и 5) параллельную расширения ввода-вывода (iSBX).

Структура с несколькими магистралями имеет преимущества перед одномагистральной системой. В частности, каждая магистраль оптимизирована для выполнения определенных функций, а операции на них выполняются параллельно. Кроме того, магистрали, не используемые в конкретной системе, могут быть исключены из ее архитектуры, что избавляет от неоправданных затрат. Три магистрали из перечисленных кратко описаны ниже; пользователям и проектировщикам, желающим ознакомиться с деталями, рекомендуется приобрести последний справочник фирмы Intel по архитектуре MULTIBUS II.

Параллельная системная магистраль iPSB

Параллельная системная магистраль iPSB используется для межпроцессорных пересылок данных и взаимосвязи процессоров. Магистраль поддерживает пакетную предачу с максимальной постоянной скоростью 40 Мбайт/с.

Связной магистрали представляет собой плату, объединяющую функциональную подсистему. Каждый связной магистрали должен иметь средства передачи данных между МП 80386, его регистрами межсоединений и магистралью iPSB. Магистраль iPSB предоставляет каждому связному магистрали четыре пространства адресов: 1) обычного ввода-вывода, 2) обычной памяти, 3) пространство объемом 255 адресов для передачи сообщений и 4) пространство межсоединений. Последнее обеспечивает географическую адресацию, при которой идентификация связного магистрали (платы) осуществляется по номеру позиции, на которой установлена плата. Поскольку МП 80386 имеет доступ только к пространствам памяти или ввода-вывода, пространства сообщений и межсоединений следует отображать на первые два пространства.

Операции на магистрали iPSB осуществляются посредством трех циклов магистрали. Цикл арбитража определяет следующего владельца магистрали. Этот цикл состоит из двух фаз: фазы принятия решения, на которой определяется приоритет для управления магистралью, и фазы захвата, когда связной с наивысшим приоритетом начинает цикл пересылки.

Второй цикл магистрали iPSB – цикл пересылки, реализует пересылку данных между владельцем магистрали и другим связным. Третий цикл iPSB – цикл исключения, указывает на возбуждение исключения в течение цикла пересылки.

Магистраль локального расширения iLBX II

Магистраль локального расширения iLBX II является быстродействующей магистралью, предназначенной для быстрого доступа к памяти, расположенной на отдельных платах. Одна магистраль iLBX II поддерживает либо две процессорные подсистемы плюс четыре подсистемы памяти, либо одну процессорную подсистему плюс пять подсистем памяти. При необходимости иметь большой объем памяти система MULTIBUS II может включать более одной магистрали iLBX II. В системе на базе МП 80386 с тактовой частотой 16 МГц типичный цикл доступа iLBX требует 6 циклов ожидания.

Для магистрали iLBX характерны 32-разрядная шина данных и 26-разрядная шина адресов. Поскольку эти шины разделены, возникает возможность конвейерных операций в цикле пересылки. К дополнительным особенностям магистрали iLBX относятся: 1) однонаправленное подтверждение при быстрой передаче данных, 2) пространство межсоединений (для каждого связного магистрали), через которое первичный запрашивающий связной инициализирует и настраивает всех остальных связных магистрали, и 3) средство взаимного исключения, позволяющее управлять многопортовой памятью.

Последовательная магистраль iSSB

Относительно дешевая последовательная системная магистраль iSSB может использоваться вместо параллельной системной магистрали iPSB в тех случаях, когда не требуется высокая производительность последней. Магистраль iSSB может содержать до 32 связных магистрали, распределенных на длине максимум 10 м. Управление магистралью ведется с помощью стандартного протокола множественного доступа с опросом несущей и разрешением конфликтов (CSMA/CD). Связные магистрали используют этот протокол для передачи данных по мере своей готовности. В случае одновременного инициирования передачи двумя или несколькими связными вступает в действие алгоритм разрешения конфликтов, обеспечивающий справедливое предоставление доступа всем запрашивающим связным.

Приложение 5

Команды защищенного режима и пример программы

Программа, листинг которой приведен ниже, устанавливает среду МП 80386, затем переключается на задачу виртуального МП 8086 (задача V86). После возврата из задачи V86 осуществляется выход в операционную систему MS-DOS. Ввод-вывод и маскирование прерываний ввода-вывода осуществляются средствами MS-DOS. На рис. П5.1 приведена блок-схема программы, а в следующем разделе дано словесное описание логики ее выполнения.

Приведенная программа просто демонстрирует использование команд МП 80386 и отнюдь не является примером наиболее эффективного программирования. В ней, однако, использована большая часть команд, которые встретились бы и в «рабочей» программе. Подробная документация, включенная в программу, поможет программисту при ее изучении.

Наиболее важные элементы документации программы извлечены из листинга и приведены в несколько расширенном виде в следующем разделе этого приложения. Программа написана с помощью макроассемблера Microsoft, версия 4.0, корпорации Microsoft. Еще не существует (пока) ассемблера МП 80386, поэтому в программе использованы 16-разрядные операнды и адреса.

Программа разрабатывалась на вычислительной системе, построенной на базе МП 80286. Использовалась память с длительностью цикла 125 нс. Микросхема 80286 была удалена, и в гнездо, предназначенное для МП 80286, вставлена адаптерная плата с МП 80386.

Логический ход программы

Большинство программных прерываний отражается на обработчике прерываний MS-DOS. Команды IRET и INT вызывают прерывание 13. Команда IRET эмулируется, как и команды INT, за исключением INT 00, которая используется для проверки команд.

Шаг 1. Макроопределения для команд МП 80386, префиксов и т. д., используемых в примерах команд.

- Преобразование значения регистра сегмента МП 8086 в линейный адрес.
- Запись линейного адреса TSS в дескриптор.
- Преобразование адреса сегмента МП 8086 в линейный адрес в дескрипторе сегмента.

Шаг 2. Описание структур данных, используемых программой (TSS описан после дескрипторов, чтобы иметь возможность ссылаться на них в процессе его инициализации). Определение дескриптора. Этот дескриптор имеет «родовой» характер. Дескрипторы определяют сегменты программы, данных, стека и т. д.

Рис. П5.1. Блок-схема программы.

Описание	Структура	Пояснения
LIMIT	Слово	Относительный адрес последнего байта или (если не TSS) младшие 16 разрядов границы сегмента
BASE_L	Слово	Разряды 0–15 32-разрядного адреса
BASE_M	Байт	Разряды 16–23 адреса
ACCESS	Байт	Привилегии доступа
GRAN	Байт	Дробность и граница (LIMIT)
BASE_H	Байт	Разряды 24–31 базы (BASE)

• Описание стека обработчика прерываний уровня 0 для задачи в режиме V86 (виртуального МП 8086). Заметьте, что в программе описание расположено «вверх ногами», так как оно используется как область данных с адресацией от младших адресов к старшим, а не как стек.

Шаг 3. Определение сегментированной структуры программы. Была выбрана простая модель с единственным реальным сегментом, потому что она легко преобразуется в файл «COM».

Шаг 4. Объявление сегментов стеков и данных.

Шаг 5. Определение области таблицы глобальных дескрипторов (GDT).

• Пустой LDT уровня 3.

• Пустой LDT уровня 0.

• Ссылка на динамические данные, используемые в обработчике прерываний.

• Ссылка на реальную ячейку 0 для векторов прерываний MS-DOS.

• Дескрипторы, используемые обработчиками прерываний.

• Дескрипторы, используемые задачей выполнения команд.

Шаг 6. Определение области таблицы дескрипторов прерываний (IDT). Адреса инициализируются в процессе настройки. Используется селектор дескриптора программы супервизора.

Шаг 7. В этой секции описывается сегмент состояния задачи (TSS). Имена некоторых полей содержат префикс «t», чтобы они не совпадали со стандартными именами системы МП 80386.

Шаг 8. В этой секции определяются различные TSS и области данных.

Определение TSS

• Первый TSS получает управление по INT 00 из задачи V86. Задача выполняется как задача защищенного режима на уровне 0.

• Второй TSS принадлежит задаче МП 80386, которая выключается, чтобы стартовала задача V86.

• Задача V86, под управлением которой будет работать MS-DOS. Все 8K битовой карты ввода-вывода сброшены в нуль, чтобы разрешить ввод-вывод устройствам 0000-FFFF. Байт, заполненный единицами, завершает карту ввода-вывода.

Локальные данные и рабочие области

• Управление отладочным дампом:
«1» в байте разрешает входы трассировки,
«2» вызывает нормальный дамп,
«3» вызывает аварийный дамп,
«0» обозначает отсутствие вывода.

• Определение смещения в стеке для обработчика прерываний.

• Установка дескрипторов сегментов.

Шаг 9. Компоновка и заполнение дескрипторов сегментов для задачи МП 80836.

• Преобразование содержимого регистров сегментов реального режима в линейные адреса в дескрипторах МП 80396.

• Запись смещений обработчиков прерываний в их дескрипторы. Мы полагаем, что дескрипторы и обработчики прерываний генерируются отдельными группами и внутри каждой группы примыкают друг к другу.

• Преобразования линейных адресов областей данных для заполнения дескрипторов сегментов, например, дескрипторов TSS и LDT.

Шаг 10. Запись селекторов сегментов обработчиков прерываний в TSS задачи МП 80386.

Шаг 11. Заполнение регистров программы, стека и флагов.

• Код 0002 в регистре EFLAGS нам подходит, так что оставляем его в том виде, как он был инициализирован в TSS.

Шаг. 12. Настройка TSS задачи V86. Запись значений дескрипторов в TSS задачи V86. В этом случае мы копируем среду реального режима МП 8086, которая будет теперь активной.

- Установка селектора и указателя сегмента стека обработчика прерываний уровня 0. Нужен только уровень 0, но для полноты устанавливаем уровни 1 и 2. Устанавливаем также только младшие 16 разрядов ESP, так как знаем, что старшие 16 разрядов будут всегда содержать нули.
- Установка значений регистров для продолжения выполнения в режиме 8086 (IP и SP).
- Установка EFLAGS в TSS, чтобы продолжать в режиме V86. VM = 1, IOPL = 3, прерывания разрешены.
- Установка значения LDT в TSS задачи B86.

Шаг 13. Установка регистров МП 80836 для выполнения в защищенном режиме.

- Установка регистров GDT и IDT в ЦП 80386.
- Вычисляем линейный адрес для таблицы дескрипторов и записываем его в память. Записываем размер таблицы и загружаем в регистр таблицы дескрипторов эти значения.

Шаг 14. Среда МП 80386 установлена. Теперь переходим в защищенный режим. Устанавливаем бит защищенного режима в CR0 и выполняем внутрисегментный переход, чтобы очистить конвейер команд.

- Теперь мы в защищенном режиме.
- Средства отладки могут выполнить дамп только что скомпонованных областей данных.

Шаг 15. Установка дескрипторов формата задачи МП 80386, регистра задачи и выполнения межсегментного перехода, чтобы загрузить CS. В этой точке все дескрипторы имеют формат задачи МП 80386, даже CS.

- Теперь установим флаг вложенной задачи (NT) и перейдем на задачу V86.
- Переходим в режим V86, используя команду IRET. Цепочка возврата в текущем TSS и флаг NT вызывают переход. Конечный результат заключается в том, что, начиная с этой точки, MS-DOS работает как задача виртуального МП 8086. Войти в защищенный режим можно с помощью команды INT 00.

Примечание. Этот переход в MS-DOS оставляет пользователя в памяти. Это и удачно, и запланировано, поскольку продолжение программы реально контролирует работу ЦП. Область программы до этой точки может быть теперь занята чем угодно; программа выполнила свои функции.

Дополнительная обслуживающая программа для защищенного режима.

- Генерация начальных точек входа для обработчиков прерываний. Эти входы соответствуют таблице векторов прерываний МП 8086. Они имеют ту же длину (4 байт), и для генерации требуемого смещения в этой таблице используется адрес возврата из вызванной процедуры.
- Обработка всех прерываний и ловушек. Обработчик отражает прерывание на MS-DOS. Эти начальные строки программы обеспечивают определенный формат содержимого стека для дальнейшего использования. Мы также сохранили код ошибки прерывания (если он был), а номер вектора хранится в удобной для вычислений форме. Если номер прерывания отличен от 13, выполняются следующие 5 шагов.

Порядок действий таков:

Если «код ошибки» был сохранен в стеке, он извлекается оттуда:

1. Сохранение содержимого регистров прерванной задачи.
2. Установка селекторов используемых дескрипторов.
3. Пересылка содержимого IP, CS и регистра флагов в стек прерванной задачи и выравнивание ее SP.

4. Пересылка вектора MS-DOS в стек (текущий) обработчика прерываний.

5. Восстановление сохраненного содержимого регистров и выполнение команды IRET.

Если возбуждено прерывание с номером 13, оно при нормальном ходе выполнения задачи было инициировано командами INT nn или IRET. В этом случае шаг 3 может выполняться по-разному.

Проверка значения по адресу CS:IP на коды операций INT или IRET. Если обнаружен какой-то другой код – запуск аварийного дампа. Под таким номером не может быть прерывания от устройства или МП 8086.

Если обнаружен код INT nn – получение значения nn. Увеличение содержимого IP в стеке на 2. Если nn не нуль, считаем nn*4 указателем вектора и начинаем выполнение «нормального» шага 3. Если nn = 0, копируем CS, AX и DX задачи V86 в полях CX, AX и DX сегмента TSS супервизора. Затем выполняем оператор CALL для вызова супервизора и после возврата продолжаем с шага 5.

Если обнаружен код IRET – пересылка флагов, CS и IP из стека V86 на текущий стек, выравнивание SP задачи V86 и продолжение с шага 5.

- Мы должны получить CS задачи V86 в качестве дескриптора, чтобы можно было проанализировать код операции: INT или IRET.
- Задача 00 супервизора выполняется как задача защищенного режима МП 80386. Обработчик прерываний задачи V86 вызывает эту задачу, когда он обнаруживает INT 00. Эта задача (которая носит чисто иллюстративный характер) выполняет команды, на которые указывает DS:DX задачи V86. За этими командами должна следовать команда RET FAR.

Программа отладки

Вывод отладочных сообщений на печать, а не на дисплей связан с тем, что ошибки в такого рода программах часто вызывают перезагрузку системы. В этом случае сообщения системы выводятся на экран поверх отладочных, стирая их. Ниже описаны различные процедуры, используемые при отладке.

- Первая процедура заполняет дескриптор в GDT адресом сегмента (адрес сегмента находится в BX) и загружает значение дескриптора в BP. Вызывающая программа переместит его (обычно) в DS, в результате чего линейный адрес загрузится из GDT, стерев любое предварительно кэшированное значение.
- Вывод на печатающее устройство байта из AL.

Примечание: содержимое AL разрушается, и только AL.

- Преобразование шестнадцатеричной цифры в код символа, воспринимаемый печатающим устройством, и вывод его на печать.
- Печать байта из AL в виде упорядоченной пары шестнадцатеричных цифр.
- Преобразование слова в AX в шестнадцатеричную форму и вывод его на печать.
- Преобразование двойного слова в EAX в шестнадцатеричную форму и вывод его на печать.
- Вывод «возврата каретки» (CR) и «перевод строки» (LF).
- Дамп 16 байтов памяти по адресу DS:AX, затем вывод CR и LF. В AX возвращается следующий адрес после дампа. ES установлен как стандартный DS для выполнения.
- Получение линейного адреса для DS в виде значения для МП 8086 или 80386. Заметьте, что выбор зависит от значения DS, которое при правильном выполнении этой программы превышает размер GDT для МП 8086.
- Печать содержимого AX в шестнадцатеричной форме, затем пробела.
- Дамп всех регистров.

• Дамп TSS и его стека (64 байт). При входе в процедуру в CX находится смещение к TSS.

• Дамп содержимого стека задачи. Сначала надо скомпоновать DS для отображения SS задачи, который может иметь формат либо задачи МП 80386, либо задачи V86. Затем можно выполнить дамп данных.

• Дамп GDT, IDT и всех TSS. Смещение к конкретному TSS находится в CX.

• Минимум дампа при трассировке.

Программное обеспечение МП 80386

Перечисленное ниже программное обеспечение можно приобрести в TAB BOOKS Inc., P. O. Box 40, Blue Ridge Summit, PA 17214.

Требования к системе

1. Компьютер, построенный на базе МП 80386.
2. Система MS-DOS (или совместимая с ней), версия по меньшей мере 3.1.
3. Любой стандартный дисплей.
4. Для ассемблирования исходного текста макроассемблер Microsoft или совместимый с ним.
5. Память: гораздо меньше 256К.
6. Печатающее устройство, настроенное на адреса «стандартного» печатающего устройства фирмы IBM.

Содержимое дискет

VITR86L. ASM Исходный текст монитора виртуального МП 8086 для МП 80386.

VIRT86L. EXE Выполнимый файл, полученный после ассемблирования и компоновки редактором связей файла ASM.

VIRT86L. LST Листинг ассемблирования файла ASM.

VIRT86L. CRF Таблица перекрестных ссылок того же ассемблирования.

DOINST. ASM Исходный текст иллюстративной программы, которая использует монитор VIRT86L для выполнения команд.

DOINST. EXE Выполнимый файл, полученный после ассемблирования и компоновки редактором связей файла DOINST. ASM.

DOINST. LST Листинг ассемблирования файла DOINST. ASM.

DOINST. CRF Таблица перекрестных ссылок того же ассемблирования.

Листинг программы

Ниже приведены листинги и таблицы перекрестных ссылок программ VIRT86L и DOINST. Такие листинги можно получить на печатающем устройстве: 1) с шириной 132 символа или 2) имеющем средства «сжатия» текста и печатания его на бумаге размером 8,5 × 11 дюймов. Мы использовали печатающее устройство C. Itoh Prowriter, сжимающее текст.

```
1
2                                       PAGE    59,132 ;Установить размеры страницы
3                                       .286P
4       ; c.1987 Don Brumm and Penn Brumm
5       ;       P.O. Box 9888, San Jose, CA  95157
6       ;         .XLIST
7       ;  V86L.ASM
```

```
 8      ; Программа устанавливает задачу 80386 и переключается
 9      ; на виртуальную задачу 8086 (V86). Из V86 выполняется
10      ; возврат и выход в  MS-DOS. Ввод-вывод и маскирование
11      ; прерываний ввода-вывода осуществляется средствами MS-DOS.
12      ;
13      ; Большая часть прерываний отрахается на обработчики
14      ; MS-DOS. IRET и INT nn вызывают прерывания 13.
15      ; IRET и INT nn эмулируются, за исключением INT OO, которое
16      ; используется для проверки команд.
17      ;
18      ; Демонстрационная программа написана на макроассемблере
19      ; Microsoft (r), версия 4.0 корпорации Microsoft.
20      ; Пока еще не существует ассемблера для МП 80386,
21      ; поэтому программа использует 16-разрядные операнды и адреса.
22      ;
23      ; "Шаги", на которые разделена программа, более подробно
24      ; описаны в книге П.Брамм и Д.Брамма "Архитектура микро-
25      ; процессора 80386" (Penn Brumm, Don Brumm. "80386 Architecture").
26      ;
27      ;
28      ; Шаг 1:
29      ; Макроопределения префиксов команд 80386, областей
30      ; данных и повторяющихся команд 80386, используемых
31      ; в программе.
32      ;
33                          opsize      MACRO               ;Заменить размер операнда
34                          db                  66h
35                                      ENDM
36
37
38      ;
39      ; Преобразовать значение в регистре сегмента  в  линейный адрес.
40      ;
41                          segadr      MACRO   segr
42                                      db      66H
43                                      sub     AX,AX       ;Очистить весь регистр
44                                      mov     AX,&segr    ;Взять регистр сегмента
45                                      db      66H
46                                      shl     AX,4        ;Преобразовать в адрес
47                                      ENDM
48      ;
49      ; Установить линейный адрес TSS в дескрипторе.
50      ;
51                          settssa     MACRO   reg,tss,seg
52                                      db      66h
53                                      lea     reg&X,&tss ;Смещение к TSS
```

```
                                db      66h
                                add     reg&X,AX    ;Линейный адрес в регистр
                                mov     seg&.base_l,reg&X ;Мл. 16 бит лин. адр.
                                db      66h
                                SHR     reg&X,8     ;Сдвинуть биты 16-23 в H
                                mov     seg&.base_m,reg&H ;Средняя часть
                                ENDM
;
; Преобразовать адрес сегмента 8086 в линейный адрес в дескрипторе сегмента
;
                        segcvt  MACRO   segr,segm
                                segadr  segr
                                mov     segm&.base_l,AX   ;База данных супер-
                                db      66H               ;визора
                                shr     AX,8              ;Загрузить ее в AH
                                mov     segm&.base_m,AH   ;и в средний байт
                                ENDM
;
; Шаг 2:
; Описание структур данных, используемых программой
; (TSS описан после дескрипторов, чтобы иметь возможность
; ссылаться на них в процессе его инициализации.)
;
; Определить дескриптор. Описаны поля и их размеры.
;
                        desc    struc
0000  0000              limit   dw      0         ;Смещение последнего байта
0002  0000              base_l  dw      0         ;Мл. 16 бит 32-разр. адреса
0004  00                base_m  db      0         ;Биты 16-23
0005  00                access  db      0         ;Привилегии доступа
0006  00                gran    db      0         ;Дробность и граница
0007  00                base_h  db      0         ;Биты 24-31 базы
0008                    desc    ends

;
; Описание стека обработчика прерываний L0 для задачи в режиме V86
; (заметьте, что описание расположено "вверх ногами", так как оно исполь-
; зуется как область данных с адресацией от младших адресов к старшим).
;
                        stkdef  struc
0000  0000              oldip   dw      0         ;Мл. 16 бит EIP
0002  0000                      dw      0         ;
0004  0000              oldcs   dw      0         ;Значение CS
0006  0000                      dw      0         ;
0008  0000              oldfl   dw      0         ;Мл. 16 бит EFLAGS
000A  0000                      dw      0         ;
```

```
000C  0000            oldsp     dw      0          ;Мл. 16 бит ESP
000E  0000                      dw      0          ;
0010  0000            oldss     dw      0          ;Значение SS
0012  0000                      dw      0          ;
0014  0000            oldes     dw      0          ;Значение ES
0016  0000                      dw      0          ;
0018  0000            oldds     dw      0          ;Значение DS
001A  0000                      dw      0          ;
001C  0000            oldfs     dw      0          ;Значение FS
001E  0000                      dw      0          ;
0020  0000            oldgs     dw      0          ;Значение GS
0022  0000                      dw      0          ;
0024                  stkdef    ends

                                PAGE
    ;
    ; Шаг 3:
    ; Определение сегментированной структуры программы.
    ;
    ; Была выбрана простая модель с единственным реальным
    ; сегментом, потому что она легко преобразуется в файл .COM.
    ;
    ; Программа устанавливает среду защищенного режима и сегмент
    ; задачи виртуального 8086, затем возвращается в MS-DOS в V86.
    ; Дальнейшее выполнение MS-DOS идет под управлением обработчика
    ; прерываний защищенного режима. Интерфейс для выполнения
    ; команд в защищенном режиме обеспечивается использованием INT00.
    ;
0000                  code      SEGMENT para public 'CODE'
                                ASSUME  cs:code, ds:code, ss:code
0100                            org     100h       ;Оставить место для префикса
    ;
    ; При входе осуществить переход через стек и области данных.
    ;
0100                  begin     PROC    far
0100  E9 4B25 R                jmp     segsetup   ;Перейти к области кодов
0110                            org     begin+16   ;Выравнивание параметров
    ;
    ; Шаг 4:
    ; Объявление областей стека и данных.
    ;
0110  2000[                     DB      2000h DUP(?);Место для стека
          ??
               ]
```

```
146
147     ;
148     ; Шаг 5:
149     ; Определение области таблицы глобальных дескрипторов (GDT).
150     ;
151 2110  0000           gdt_def    desc    <>          ;Первый - пустой, его
152 2112  0000
153 2114  00
154 2115  00
155 2116  00
156 2117  00
157
158                                                    ;можно использовать, как 0
159     ;
160     ; Пустой LDT уровня 3.
161     ;
162 2118  0007           ref_l3     desc    <7,0,0,11100010b,0,0> ;Наличие, 3,
163 211A  0000                                                    ;тип 2,
164 211C  00                                                      ;граница = 7
165 211D  E2
166 211E  00
167 211F  00
168
169 = 000B               refl3      equ     ref_l3-gdt_def+3
170     ;
171     ; Пустой LDT уровня 0.
172     ;
173 2120  0007           ref_l0     desc    <7,0,0,10000010b,0,0> ;Наличие, 0,
174 2122  0000                                                    ;тип 2,
175 2124  00                                                      ;граница = 7
176 2115  82
177 2126  00
178 2127  00
179
180 = 0010               refl0      equ     ref_l0-gdt_def
181     ;
182     ; Ссылка на динамические данные, используемые в обработчике.
183     ;
184 2128  FFFF           ref_d      desc    <0FFFFh,0,0,10010010b,00001111b,0>
185 212A  0000                                          ;Данные
186 212C  00
187 212D  92
188 212E  0F
189 212F  00
190
191 = 0018               ref_dta    equ     ref_d-gdt_def
```

```
192     ;
193     ; Ссылка на динамические данные, используемые в TSS дампа.
194     ;
195 2130  FFFF          ref_dd     desc    <0FFFFh,0,0,1001001b,00001111b,0>
196 2132  0000                                        ;Данные
197 2134  00
198 2135  92
199 2136  0F
200 2137  00
201
202 = 0020              ref_dtad   equ     ref_dd-gdt_def
203     ;
204     ; Ссылка на реальную ячейку 0 для векторов прерываний MS-DOS.
205     ;
206 2138  FFFF          ref_zer    desc    <0FFFFh,0,0,10010010b,00001111b,0>
207 213A  0000                                        ;Нуль
208 213C  00
209 213D  92
210 213E  0F
211 213F  00
212
213 = 0028              ihzero     equ     ref_zer-gdt_def
214 2140  FFFF          stk_data   desc    <0FFFFh,0,0,10010010b,00001111b,0>
215 2142  0000                                        ;Стек, как данные
216 2144  00
217 2145  92
218 2146  0F
219 2147  00
220
221 = 0030              stkdata    equ     stk_data-gdt_def
222 2148  0000                     desc    <0,0,0,10010110b,0,0>  ;Стек
223 214A  0000
224 214C  00
225 214D  96
226 214E  00
227 214F  00
228
229 2150  0067          ref_0      desc    <sz386t-1,0,0,10001001b,00000000b,0>
230 2152  0000                                        ;Задача 80386
231 2154  00
232 2155  89
233 2156  00
234 2157  00
235
236 = 0040              ref0d      equ     ref_0-gdt_def
237 2158  0067          ref_1      desc    <sz386t-1,0,0,10001001b,00000000b,0>
```

```
238 215A  0000                                        ;Задача 80386
239 215C  00
240 215D  89
241 215E  00
242 215F  00
243
244 = 0048                ref1d       equ     ref_1-gdt_def
245 2160  2068            v86_0       desc    <szv86t,0,0,11101011b,00000000b,0>
246 2162  0000                                      ;Задача V86
247 2164  00
248 2165  EB
249 2166  00
250 2167  00
251
252 = 0053                v860d       equ     v86_0-gdt_def+3
253     ;
254     ; Дескрипторы, используемые обработчиками прерываний.
255     ;
256     ;
257 2168  FFFF            sup_c       desc    <0FFFFh,0,0,10011010b,00001111b,0>
258 216A  0000                                      ;Программа
259 216C  00
260 216D  9A
261 216E  0F
262 216F  00
263
264 = 0058                supcd       equ     sup_c-gdt_def ;DPL=0, несогласованный,
265                                                         ;с доступом для чтения
266     ;
267 2170  FFFF
268 2172  0000
269 2174  00
270 2175  92
271 2176  0F
272 2177  00
273
274 = 0060                supdd       equ     sup_d-gdt_def ;DPL=0,
275                                                         ;с доступом для записи
276 2178  0000            sup_s       desc    <0,0,0,10010110b,00000000,0>  ;Стек
277 2172  0000
278 217C  00
279 217D  96
280 217E  00
281 217F  00
282
283 = 0068                supsd       equ     sup_s-gdt_def ;Расширение вниз,
```

```
284                                                    ;с доступом для записи
285      ;
286      ; Дескрипторы, используемые задачей выполнения команд.
287      ;
288      ;
289 2180  FFFF
290 2182  0000
291 2184  00
292 2185  9A
293 2186  0F
294 2187  00
295
296 = 0070                execd       equ     exe_c-gdt_def ;DPL=0, несогласованный
297                                                    ;С доступом для чтения
298      ;
299 2188  FFFF            exe_d       desc    <0FFFFh,0,0,10010010b,00001111b,0>
300 218A  0000                                         ;Данные
301 218C  00
302 218D  92
303 218E  0F
304 218A  00
305
306 = 0078                exedd       equ     exe_d-gdt_def ;DPL=0, с доступом
307                                                    ;для записи
308 2198  0000            exe_s       desc    <0,0,0,10010110b,00000000b,0> ;Стек
309 2192  0000
310 2194  00
311 2195  96
312 2196  00
313 2197  00
314
315 = 0080                exesd       equ     exe_s-gdt_def ;Расширение вниз,
316                                                    ;с доступом для записи
317      ;
318      ; Конец GDT
319      ;
320 = 0088                gdt_size    equ     &-gdt_def
321      ;
322      ; Шаг 6:
323      ; Определение области таблицы дескрипторов прерываний (IDT).
324      ; Адреса инициализируются в процессе настройки.
325      ; Используется селектор дескриптора программы супервизора.
326      ;
327                                    .SALL
328 = 2198                 idt_def     equ     &
329                                    REPT    256
```

```
330                                     desc    <0,supcd,0,8Eh,0,0>
331                                     ENDM
332 = 0800                  idt_size    equ     &-idt_def
333
334
335     ; Шаг 7 (шаг 2A):
336     ; В этой секции описывается сегмент состояния задачи (TSS).
337     ; Имена некоторых полей содержат префикс t, чтобы они не
338     ; совпадали со стандартными системными именами МП 80386
339     ; (Это описание приводится именно здесь, так как оно
340     ; содержит ссылки на дескрипторы, описанные выше).
341     ;
342                         tss_d       struc
343 0000  0000              back        dw      0    ;Обратная ссылка на предыдущий TSS
344 0002  0000                          dw      0    ;Зарезервировано
345 0004  00 04 00 00       esp0        dd      400h ;SP для прерывания уровня 0
346 0008  0068              ss0         dw      supsd;и SS
347 000A  0000                          dw      0    ;Зарезервировано
348 000C  00 04 00 00       esp1        dd      400h ;Стек обработчика прерыв. уровня 1
349 0010  0068              ss1         dw      supsd
350 0012  0000                          dw      0    ;Зарезервировано
351 0014  00 04 00 00       esp2        dd      400h ;Стек обработчика прерыв. уровня 2
352 0018  0068              ss2         dw      supsd
353 001A  0000                          dw      0    ;Зарезервировано
354 001C  00 00 00 00       tcr3        dd      0
355 0020  0000              teip        dw      0    ;Указатель команд
356 0022  0000                          dw      0
357 0024  002               teflags     dw      2    ;EFLAGS
358 0026  0000                          dw      0
359 0028  0000              teax        dw      0    ;AX для инициализации
360 002A  0000                          dw      0
361 002C  0000              tecx        dw      0    ;CX для инициализации
362 002E  0000                          dw      0
363 0030  00 00 00 00       tedx        dd      0
364 0034  00 00 00 00       tebx        dd      0
365 0038  1000 0000         tesp        dw      100h,0
366 003C  00 00 00 00       tebp        dd      0
367 0040  00 00 00 00       tesi        dd      0
368 0044  00 00 00 00       tedi        dd      0
369 0048  0060              tes         dw      supdd;Селектор сегмента расширения
370 004A  0000                          dw      0    ;Зарезервировано
371 004C  0058              tcs         dw      supcd;Селектор сегмента программы
372 004E  0000                          dw      0    ;Зарезервировано
373 0050  0068              tss         dw      supsd;Селектор сегмента стека
374 0052  0000                          dw      0    ;Зарезервировано
375 0054  0060              tds         dw      supdd;Селектор сегмента данных
```

```
376 0056  0000                    dw      0     ;Зарезервировано
377 0058  0000          tfs       dw      0     ;Селектор сегмента F
378 005A  0000                    dw      0     ;Зарезервировано
379 005C  0000          tgs       dw      0     ;Селектор сегмента G
380 005E  0000                    dw      0     ;Зарезервировано
381 0060  0010          tldt      dw      ref10;Таблица локальных дескрипторов
382 0062  0000                    dw      0     ;Заразервировано
383 0064  0000                    dw      0     ;Заразервировано
384 0066  0068          iomap     dw      104   ;Смещение к базе карты в/в
385 0068                tss_d     ends
386     ;
387     ; Шаг 8:
388     ; Определение сегментов состояния задачи (TSS).
389     ;
390     ; Эта задача получает управление по INT 00 из задачи V86.
391     ; Она выполняется, как задача защищенного режима на уровне 0.
392     ;
393 2998  0000          supv      tss_d   <>    ;Место для задачи супервизора
394 299A  0000
395 299C  00 04 00 00
396 29A8  0068
397 29A2  0000
398 29A4  00 04 00 00
399 29A8  0068
400 29A2  0000
401 29AC  00 04 00 00
402 29B0  0068
403 29B2  0000
404 29B4  00 00 00 00
405 29B8  0000
406 29BA  0000
407 29BC  0002
408 29BE  0000
409 29C0  0000
410 29C2  0000
411 29C4  0000
412 29C6  0000
413 29C8  00 00 00 00
414 29CC  00 00 00 00
415 29D0  0001[
416           1000
417           0000
418                    ]
419 29D4  00 00 00 00
420 29D8  00 00 00 00
421 29DC  00 00 00 00
```

```
422 29E0  0060
423 29E2  0000
424 29E4  0058
425 29E6  0000
426 29E8  0068
427 29EA  0000
428 29EC  0060
429 29EE  0000
430 29F0  0000
431 29F2  0000
432 29F4  0000
433 29F6  0000
434 29F8  0010
435 29FA  0000
436 29FC  0000
437 29FE  0068
438
439 = 0068              sz386t     equ     $-supv
440     ;
441     ; Эта задача выключается и запускается задача V86.
442     ;
443 2A00  0053          supv2      tss_d   <v860d>    ;Задача переключена
444 2A02  0000
445 2A04  00 04 00 00
446 2A08  0068
447 2A0A  0000
448 2A0C  00 04 00 00
449 2A10  0068
450 2A12  0000
451 2A14  00 04 00 00
452 2A18  0068
453 2A1A  0000
454 2A1C  00 00 00 00
455 2A20  0000
456 2A22  0000
457 2A24  0002
458 2A26  0000
459 2A28  0000
460 2A2A  0000
461 2A2C  0000
462 2A2E  0000
463 2A30  00 00 00 00
464 2A34  00 00 00 00
465 2A38  0001[
466             1000
467             0000
```

```
468                     ]
469 2A3C  00 00 00 00
470 2A40  00 00 00 00
471 2A44  00 00 00 00
472 2A48  0060
473 2A4A  0000
474 2A4C  0058
475 2A4E  0000
476 2A50  0068
477 2A52  0000
478 2A54  0068
479 2A56  0000
480 2A58  0000
481 2A5A  0000
482 2A5C  0000
483 2A5E  0000
484 2A60  0010
485 2A62  0000
486 2A64  0000
487 2A66  0068
488
489     ;
490     ;
491     ; Это TSS для задачи V86, под управлением
492     ; которой будет работать MS-DOS.
493     ;
494 2A68  0000          v86       tss_d   <>        Определить TSS v86
495 2A6A  0000
496 2A6C  00 04 00 00
497 2A70  0068
498 2A72  0000
499 2A74  00 04 00 00
500 2A78  0068
501 2A7A  0000
502 2A7C  00 04 00 00
503 2A80  0068
504 2A82  0000
505 2A84  00 00 00 00
506 2A88  0000
507 2A8A  0000
508 2A8C  0002
509 2A8E  0000
510 2A90  0000
511 2A92  0000
512 2A94  0000
513 2A96  0000
```

```
514 2A98  00 00 00 00
515 2A9C  00 00 00 00
516 2AA0  0001[
517             1000
518             0000
519                         ]
520 2AA4  00 00 00 00
521 2AA8  00 00 00 00
522 2AAC  00 00 00 00
523 2AB0  0060
524 2AB2  0000
525 2AB4  0058
526 2AB6  0000
527 2AB8  0068
528 2ABA  0000
529 2ABC  0060
530 2ABE  0000
531 2AC0  0000
532 2AC2  0000
533 2AC4  0000
534 2AC6  0000
535 2AC8  0010
536 2ACA  0000
537 2ACC  0000
538 2ACE  0068
539
540 2AD0  1000[                     dw      4096 dup (0);Битовая карта ввода-
541             0000                                    ;вывода, все биты равны 0
542                         ]
543
544     ; Разрешение в/в всем устройствам 0000 - FFFF
545 = 2068                szv86t      equ     $-v86
546 4AD0  FFFF                        dw      -1    ;Байт единиц - конец карты ввода-
547                                   PAGE          ;вывода
548     ; Некоторые локальные данные и рабочие области.
549     ;
550 4AD2  0000            dtsize      dw      0     ;Размер таблицы дескрипторов
551 4AD4  00 00 00 00     dtload      dd      0     ;Адрес таблицы дескрипторов
552 4AD8  0000            intcd1      dw      0     ;Из стека, при сохранении кода int
553 4AD2  0000            intcd2      dw      0     ;То же
554     ;
555     ; Управление отладочным дампом: 1 в байте разрешает входы трассировки,
556     ;                                2 вызывает нормальный дамп,
557     ;                                3 вызывает аварийный дамп и
558     ;                                0 обозначает отсутствие вывода.
559     ;
```

```
4ADC  00              debugf     db      0    ;До входа в защищенный режим
4AD0  00              debudf0    db      0    ;Общая точка входа обработчика
4ADE  00              debugf0a   db      0    ;прерываний и выход
4ADF  02              debugf1    db      02   ;Точка входа INT13
4AE0  00              debugf1a   db      0    ;и выход INTnn
4AE1  00              debugf1b   db      0    ;Выход IRET и все остальные
4AE2  0000            retdisp    dw      0    ;Адрес вызова обработчика преры-
                                              ;ваний (см. int_hand0)
4AE4  0000 0000       dsaddr     dw      0,0  ;Адрес области данных
4AE8  0000            dumpcal    dw      0    ;Адрес вызова отладочного дампа
4AEA  56 69 72 74 75  success    db      'Virtual 8086 mode is established! '
      61 6C 20 38 30
      38 36 20 6D 6F
      64 65 20 69 73
      20 65 73 74 61
      62 6C 69 73 68
      65 64 21 20
4B0C  45 6E 74 65 72              db      'Enter G (go) or D - $
      20 47 20 28 67
      6F 29 20 6F 72
      20 44 20 2D 20
      24
4B21  0D 8A 24          newline   db      0dh,0ah,'$'
4B24  00                endad     db      0
= 2A24                  datasiz equ $-gdt_def+15 ;Размер области данных
    ; Определение смещения в стеке для обработчика прерываний.
= 1800                  ihspv     equ     1800h      ;Начальное значение SP
= 17DC                  ihspn1    equ     ihspv-size stkdef;В стеке нет кодов INT
= 17D8                  ihspn2    equ     ihspn1-4   ;Коды прерывания сохранены
                                  PAGE
    ;
    ; Установка дескрипторов сегментов (мы в реальном режиме)
    ;
4B25                    segsetup:
4B25  8C C8                       mov     AX,CS      ;Адрес сегмента в AX
4B27  8E D0                       mov     SS,AX      ;Установить SP на 2000h
4B29  BC 2000                     mov     SP,2000h
4B2C  8E D8                       mov     DS,AX      ;Установить DS
4B2E  8E C0                       mov     ES,AX      ;и ES

    ;
    ; Шаг 9:
    ; Компоновка и заполнение дескрипторов сегментов для задачи 80386.
    ;
4B30  FC                          cld                ;Направление строк вверх
4B31  FA                          cli                ;Выключить прерывания
```

```
606     ;
607     ; Шаг 9A:
608     ; Преобразование содержимого регистров различных сегментов
609     ; в линейные адреса в дескрипторах 80386.
610     ;
611                                 segcvt  DS,sup-d    ;Установить сегмент данных
612                                 segcvt  CS,sup-c    ;Установить сегмент программы
613                                 segcvt  SS,sup-s    ;Установить сегмент стека
614                                 segcvt  SS,stk-data;как сегмент данных
615                                 segadr  DS          ;Линейный адрес DS в EAX
616                                 opsize
617 4B8C  A3 4AE4 R                 mov     dsaddr,AX   ;Сохранить его на будущее
618     ;
619     ; Шаг 9B:
620     ; Запись смещений обработчиков прерываний в их дескрипторы.
621     ; Предполагается, что дескрипторы и обработчики генерируются отдельными
622     ; группами и внутри каждой группы примыкают друг к другу.
623     ;
624 4B8F  2B DB                      sub     BX,BX         ;Очистить индекс, получить
625 4B91  8D 06 4D7B R               lea     AX,int_hand0  ;первое смещение
626 4B95  B9 0100                    mov     CX,256        ;и установить счетчик
627
628 4B98                  fillihd1:
629 4B98  89 87 2198 R               mov     idt_def.limit[BX],AX ;Установить сме-
630 4B9C  05 0004                    add     AX,4          ;щение, затем увеличить его
631 4B9F  83 C3 08                   add     BX,8          ;на его величину. Увеличить
632 4BA2  E2 F4                      loop    fillihd1      ;индекса дескриптора тоже
633                                                        ;Повторить это 256 раз
634                                 segadr  DS            ;Линейный адрес DS в EAX
635     ;
636     ; Шаг 9C:
637     ; Преобразование линейных адресов областей данных для заполнения
638     ; дескрипторов сегментов, например, дескрипторов TSS и LDT.
639     ;
640                                 settssa D,supv,ref_0  ;Указать на супервизор
641                                 settssa D,supv2,ref_1 ;Указать на супервизор 2
642                                 settssa D,v86,v86_0   ;Указать на TSS V86
643
644                                 settssa D,gdt_def,ref_13;LDT уровня 3 на 0 GDT
645                                 settssa D,gdt_def,ref_10;LDT уровня 0 тоже
646
647     ;
648     ; Шаг 10:
649     ; Запись селекторов сегментов обработчиков прерываний в TSS 80386.
650     ;
651 4C11  C7 06 29E4 R 0058         mov     supv.tcs,supcd  ;Дескриптор программы
```

```
652 4C17  B8 0060                 mov     AX,supdd        ;Селектор сегмента дан-
653 4C1A  A3 29EC R               mov     supv.tds,AX     ;ных в TSS данных,
654 4C1D  A3 29E0 R               mov     supv.tes,AX     ;в экстра TSS,
655 4C20  A3 29F0 R               mov     supv.tfs,AX     ;в TSS FS,
656 4C23  A3 29F4 R               mov     supv.tgs,AX     ;в TSS GS.
657 4C26  C7 06 29E8 R 0068       mov     supv.tss,supsd  ;Селектор стека в TSS
658 4C2C  B8 0010                 mov     AX,ref10        ;Получить указатель LDT
659 4C2F  A3 29F8 R               mov     supv.tldt,AX    ;Установить его в обеих
660 4C32  A3 2A60 R               mov     supv2.tldt,AX   ;их задачах супервизора
661     ;
662     ; Шаг 11:
663     ; Заполнение регистров программы, стека и флагов.
664     ;
665                               opsize
666 4C36  2B C0                   sub     AX,AX           ;Очистить EAX
667 4C38  B8 1200                 mov     AX,01200h       ;Получить значение SP
668                               ;(в SS 2000h), 80386 получает младшие 1800h
669                               opsize                  ;Размер - двойное слово
670 4C3C  A3 29D0 R               mov     supv.tesp,AX    ;установить в TSS
671 4C3F  B8 5386 R               mov     AX,offset instexe;Получить смещение
672                               opsize                  ;Размер - двойное слово
673 4C43  A3 29B8 R               mov     supv.teip,AX    ;установить в TSS
674     ;
675     ; Код 0002 в EFLAGS подходит, так что оставляем его как есть.
676     ;
677 4C46  C7 06 29A0 R 0068       mov     supv.ss0,supsd  ;SS уровня 0 для прер.
678 4C4C  C7 06 299C R 0800       mov  word ptr supv.esp0,800h;Установить SP
679
680 4C52  C7 06 29A8 R 0068       mov     supv.ss1,supsd      ;SS уровня 1
681 4C58  C7 06 29A4 R 0800       mov  word ptr supv.esp1,800h;Установить SP
682
683 4C5E  C7 06 29B0 R 0068       mov     supv.ss2,supsd      ;SS уровня 2
684 4C64  C7 06 29AC R 0800       mov  word ptr supv.esp2,800h;Установить SP
685     ;
686     ; Шаг 12:
687     ; Настройка TSS виртуального 8086.
688     ; Запись значений дескрипторов в TSS задачи V86. В этом случае
689     ; копируем активную среду реального режима 8086, который сейчас
690     ; установлен, в TSS задачи V86.
691
692 4C6A  8C D8                      mov     AX,DS       ;Получить DS в AX
693 4C6C  A3 2AB0 R                  mov     v86.tes,AX  ;Установить для выполнения
694 4C6F  A3 2ABC R                  mov     v86.tds,AX  ;V86 во всех указателях
695 4C72  A3 2AC0 R                  mov     v86.tfs,AX  ;сегментов данных
696 4C75  A3 2AC4 R                  mov     v86.tgs,AX  ;
697 4C78  8C D0                      mov     AX,SS       ;Получить значение SS теку-
```

```
698 4C7A  A3 2AB8 R                   mov     v86.tss,AX ;щего стека и установить его
699 4C7D  8C C8                       mov     AX,CS ;Получить значение CS и тохе
700 4C7F  A3 2AB4 R                   mov     v86.tcs,AX ;установить его
701     ;
702     ; Установка селектора и указателя сегмента стека обработчика прерываний
703     ; уровня 0. Нам нухен только уровень 0, но для полноты устанавливаем
704     ; уровни 1 и 2. Устанавливаем только младшие 16 разрядов
705     ; ESP, так как старшие 16 разрядов будут всегда содержать нули.
706     ;
707 4C82  C7 06 2A70 R 0068              mov     v86.ss0,supsd ;Установить SS уровня 0
708 4C88  C7 06 2A6C R 1800 .           mov  word ptr v86.esp0,ihspv ;и значение SP
709
710 4C8E  C7 06 2A78 R 0068              mov     v86.ss1,supsd ;Установить SS уровня 1
711 4C94  C7 06 2A74 R 1600             mov  word ptr v86.esp1,1600h ;и значение SP
712
713 4C9A  C7 06 2A80 R 0068              mov     v86.ss2,supsd ;Установить SS уровня 2
714 4CA0  C7 06 2A7C R 1400             mov  word ptr v86.esp2,1400h ;и значение SP
715     ;
716     ; Установка значений регистров для продолхения рехима 8086 (IP и SP).
717     ;
718                                   opsize
719 4CA7  2B C0                       sub     AX,AX           ;Очистить вершину
720 4CA9  B8 4D4A R                   mov     AX,offset resume ;Получить адрес resume
721                                   opsize
722 4CAD  A3 2A88 R                   mov     v86.teip,AX     ;и установить его
723 4CB0  8B C4                       mov     AX,SP           ;Получить значение SP
724                                   opsize
725 4CB3  A3 2AA0 R                   mov     v86.tesp,AX     ;и установить его
726     ;
727     ; Установка EFLAGS в TSS, чтобы продолхать в рехиме V86.
728     ;
729 4CB6  C7 06 2A8C R 3202           mov     v86.teflags,0011001000000010b;Биты 0-15
730     ;                              Имена флагов =====> NIOODITSZ A P C
731     ;                                                    PL
732     ;
733 4CBC  C7 06 2A8E R 0002           mov     v86.teflags+2,0000000000000010b;и 16-31
734     ;                              Имена флагов =====>                VR
735     ;                                                                 M
736     ; VM установлен, IOPL = 3, прерывания разрешены.
737     ;
738 4CC2  C7 06 2ACE R 0068           mov     v86.iomap,sz386t;Смещение к таблице в/в
739     ;
740     ; Установка значения LDT в TSS задачи.
741     ;
742 4CC8  C7 06 2AC8 R 000B           mov     v86.tldt,ref13  ;LDT уровня 3
743     ;
```

```
744     ; Шаг 13:
745     ; Установка регистров 80386 для выполнения в защищенном режиме.
746     ;
747     ; Установка регистров GDT и IDT в центральном процессоре 80386.
748     ;
749                                 opsize
750 4CCF  A1 4AE4 R                 mov     AX,dsaddr  ;Загрузить 32 бита адреса
751     ;
752     ; Используем линейный адрес DS для получения значений регистров
753     ; GDT и IDT. Сначала загружаем в память значение GDT.
754     ;
755                                 opsize
756 4CD3  2B DB                    sub     BX,BX      ;Очистить EBX
757 4CD5  8D 1E 2110 R              lea     BX,gdt_def ;Смещение к GDT
758                                 opsize
759 4CDA  03 D8                    add     BX,AX      ;Линейный адрес GDT
760                                 opsize
761 4CDD  89 1E 4AD4 R              mov     word ptr dtload,BX ;Занести значение
762 4CE1  C7 06 4AD2 R 0087         mov     dtsize,gdt_size-1  ;и размер таблицы
763
764 4CE7  0F 01 16 4AD2 R           LGDT    qword ptr dtsize   ;Параметры таблицы
765
766     ;
767     ; Теперь загружаем в память значение IDT.
768     ;
769                                 opsize
770 4CED  2B DB                    sub     BX,BX      ;Очистить EBX
771 4CEF  8D 1E 2198 R              lea     BX,idt_def ;Смещение к IDT
772                                 opsize
773 4CF4  03 D8                    add     BX,AX      ;Линейный адрес IDT
774                                 opsize
775 4CF7  89 1E 4AD4 R              mov     word ptr  dtload,BX ;Занести значение
776 4CFB  C7 06 4AD2 R 07FF         mov     dtsize,idt_size-1   ;и размер таблицы
777
778 4D01  0F 01 1E 4AD2 R           LIDT    qword ptr  dtsize   ;Параметры таблицы
779
780     ;
781     ; Шаг 14:
782     ; Среда 80386 установлена.
783     ; Теперь переходим в защищенный режим.
784     ;
785     ;                           mov     EAX,CR0
786 4D06  0F 20 C0                 db      0fh,20h,0c0h   ;Переместить CR0 в AX
787 4D09  0C 01                    or      AL,1           ;Защищенный режим
788     ;                           mov     CR0,EAX
789 4D08  0F 22  C0                db      0fh,22h,0c0h   ;Переместить CR0 из AX
```

```
790 4D0E  7A  00                    jp      local           ;Очистить очередь
791 4D10                  local:
792     ;
793     ; Теперь программа выполняется в защищенном режиме.
794     ;
795
796
797 4D10  B8 0010                    mov     AX,ref10 ;Указать на пустой дескриптор
798 4913  0F 00 D0                   lldt    AX       ;LDT и установить LDT
799
800     ;
801     ; Средства отладки.
802 4D16  80 3E 4ADC R 00            cmp     debugf,0 ;Дамп нужен?
803 4D1B  74 06                      je      skip01   ;Нет, обойти
804
805 4D1D  E8 55D2 R                  call    edump    ;Получить необходимое
806 4D20  E8 5464 R                  call    newln    ;Перевод строки
807 4D23                  skip01:
808
809 4D23  B8 0060                    mov     AX,supdd ;Загузить дескриптор данных
810 4D26  8E D8                      mov     DS,AX    ;386
811 4D28  8E C0                      mov     ES,AX
812                       ;          mov     FS,AX
813 4DA2  8E E0                      db      8eh,0e0h
814                       ;          mov     GS,AX
815 4DAC  8E E8                      db      8eh,0e8h ;во все дескрипторы данных
816 4D2E  B8 0068                    mov     AX,supsd ;Дескриптор стека 386
817 4D31  8E D0                      mov     SS,AX
818 4D33  EA                         db      0eah
819 4D34  4D38 0058                  dw      godoit-begin+100h,supcd
820
821     ;
822     ; Шаг 15:
823     ; В этой точке все дескрипторы имеют формат 80386, даже CS.
824     ;
825     ; Теперь установим флаг NT и перейдем на задачу V86.
826     ;
827 4D38                  godoit:
828 4D38  B9 0048                    mov     CX,ref1d ;Получить значение для регис-
829 493B  0F 00 D9                   ltr     CX       ;тра задачи и установить его
830                                  opsize
831 4D3F  9C                         pushf            ;EFLAGS в стек
832                                  opsize
833 4D41  58                         pop     AX       ;Загрузить его в EAX
834 4D42  80 CC 40                   or      AH,40h   ;Флаг вложенной задачи
835                                  opsize
```

```
836 4D46  50                        push    AX          ;Опять EFLAGS в стек, затем
837                                 opsize
838 4D48  9D                        popf                ;его же в регистр флагов
839     ;
840     ; Теперь перейдем в режим V86, используя IRET. Цепочка
841     ; возврата в нашем TSS, и флаг NT вызывает переход.
842     ;
843 4D49  CF                        iret                ;и перейти в V86
844
845     ;
846     ; Шаг 16:
847     ; Теперь мы в режиме V86. Программа выдает сообщение и ожидает
848     ; ввода с клавиатуры символа G (от go, пуск). После его получения
849     ; программа осуществляет вызов системы MS-DOS "Завершить выполнение
850     ; задачи, но оставить ее в памяти".
851     ;
852     ; Конечный результат заключается в том, что  начиная с этой точки,
853     ; MS-DOS работает, как V86. Вход в защищенный режим через INT 00.
854     ;
855 4D4A                  resume:
856 = 4D4A                 resumed   equ     resume-begin+100h
857 4D4A  FC                        cld                 ;Прямое направление
858 4D4B                  resloop:
859 4D4B  8D 16 4AEA R             lea     DX,success  ;Указатель на сообщение
860 4D4F  B4 09                     mov     AH,9        ;Установить код записи
861 4951  CD 21                     int     21h         ;Пусть DOS выведет его
862 4953  B4 01                     mov     AH,1        ;Получить символ с клави-
863 4D55  CD 21                     int     21h         ;атуры
864 4D57  50                        push    AX          ;Сохранить символ
865 4D58  8D 16 4B21 R              lea     DX,newline  ;Указатель на новую строку
866 4D5C  B4 09                     mov     AH,9        ;Установить код записи
867 4D5E  CD 21                     int     21h         ;Вызвать DOS
868 4D60  58                        pop     AX          ;Получить символ назад
869 4D61  3C 47                     cmp     AL,'G'      ;Пуск?
870 4D63  74 0B                     je      todos       ;Да - в DOS
871 4D65  3C 44                     cmp     AL,'D'      ;Нужен дамп?
872 4D67  75 E2                     jne     resloop     ;Нет, пробуем снова
873 4D69  C6 06 4ADD R 03           mov     debugf0,3   ;Дамп по следующему прерыва-
874 4D6E  EB DB                     jmp     resloop     ;нию. Попробуем снова
875
876 4D70                  todos:
877 4970  8D 16 5853 R              lea     DX,totalsize+20fh  ;Верхний адрес
878 4D74  C1 EA 04                  shr     DX,4        ;Преобразовать в параграфы
879 4D77  B4 31                     mov     AH,31h      ;Оставить в памяти
880 4D79  CD 21                     int     21h         ;Предположим, что закончили
881
```

```
882     ; Выполненный выше переход в MS-DOS оставляет нас в памяти. (Это
883     ; и удачно, и запланировано, поскольку продолжение этой программы
884     ; реально контролирует работу центрального процессора!)
885     ;
886     ; Область программы до этой точки может быть теперь занята чем
887     ; угодно - программа выполнила свои функции.
888     ;
889                                 PAGE
890     ;
891     ; ОБСЛУЖИВАЮЩАЯ ПРОГРАММА
892     ; 1.
893     ;
894     ; Генерация начальных точек входа для обработчиков прерываний.
895     ; Эти входы соотвествуют таблице векторов прерываний 8086.
896     ; Они имеют ту же длину (4 байт) и для генерации требуемого
897     ; смещения в таблице используется адрес возврата из вызванной
898     ; процедуры (в этой точке можно работать с CS и нашими SP/SS)
899     ;
900     ;
901 4D7B                int_hand0:                      ;Начальное смещение
902                                 REPT    256         ;Прерывания 0 - 255
903                                 call    ih_common   ;3-байтовый вызов
904                                 iret                ;1-байтовый возврат из прер.
905                                 ENDM
906     ; 2.
907     ;
908     ; Обработка всех прерываний и ловушек.
909     ; Обработчик отражает прерывание на MS-DOS. Выполняется это
910     ; просто:
911     ;       0) Если в стеке "код ошибки", извлечение его из стека
912     ;          (если бы мы обрабатывали прерывание, здесь можно
913     ;          было бы проанализировать и использовать этот код).
914     ;       1) Сохранение регистров прерванной задачи и
915     ;       2) установка селекторов дескрипторов.
916     ; Для прерываний, отличных от 13:
917     ;       3) Пересылка IP, CS и флагов в стек прерванной
918     ;          задачи и выравнивание ее SP.
919     ;       4) Пересылка вектора MS-DOS в стек обработчика
920     ;          прерываний (наш текущий стек).
921     ;       5) Восстановление сохраненных регистров и выполнение IRET.
922     ;
923     ; Прерывание 13 обычно должно быть INT nn или IRET,
924     ; поэтому шаг 3 может выполняться по-разному:
925     ;       3) Проверка значения по адресу в CS, IP на коды операций INT или IRET.
926     ;          Если ни то, ни другое, аварийный дамп -
927     ;          под таким номером не может быть прерывания
```

```
928     ;       от устройства или 8086.
929     ;     3-INT) Получение значения nn, увеличение IP на 2 в нашем
930     ;       стеке. Если nn не 00, считаем nn*4 указателем вектора
931     ;       и начинаем выполнение шага 3 (строка 917).
932     ;       Если nn=00, копируем CS, AX и DX задачи V86 в поля
933     ;       CX, AX и DX сегмента TSS супервизора. Затем вызываем
934     ;       задачу супервизора и после возврата продолжаем
935     ;       с шага 5.
936     ;     3-IRET) Пересылка флагов, CS и IP из стека V86 на наш стек,
937     ;       выравнивние SP задачи V86 и продолжение с шага 5.
938
939 517B                ih_common:
940 517B  50                        push    AX          ;Сохранить регистр
941 517C  B8 0060                   mov     AX,supdd    ;Получить адресуемость
942 517F  8E D8                     mov     DS,AX       ;данных
943 5181  58                        pop.    AX          ;SP теперь указывает на
944                                         ;возврат вызова процедуры прерывания
945 5182  8F 06 4AE2 R              pop     retdisp     ;Из стека в область хранения
946 5186  81 FC 17D8                cmp     SP,ihspn2   ;В стеке код прерывания?
947 518A  75 08                     jne     ihcommn     ;Нет, не будем его трогать
948 518C  8F 06 4AD8 R              pop     intcd1      ;Снимем со стека слова 1 и 2
949 5190  8F 06 4ADA R              pop     intcd2      ;кода прерывания
950     ;
951     ; Выполненный только что фрагмент дал нам формат стека. Мы также
952     ; сохранили код ошибки прерывания (если он был), а номер вектора
953     ; хранится в удобной для вычислений форме (неплохо для 11 команд!)
954     ;
955 5194                ihcommn:
956 5194  60                        pusha               ;Сохранить регистры
957
958 5195  B8 0030                   mov     AX,stkdata ;Загрузить стек как дескрип-
959 5198  8E C0                     mov     ES,AX       ;тор данных в ES
960 519A  8B DC                     mov     BX,SP       ;Загрузить SP в BX
961 519C  26: 8B 47                 mov     AX,ES:[BX+26] ;Старшее слово EFLAGS
962 51A0  24 02                     and     AL,2        ;Выделить бит VM
963 51A2  75 0A                     jnz     stillok     ;Если есть - все нормально
964     ;
965     ; Прерывание не из режима V86 считается здесь ошибкой. Если бы
966     ; это был обработчик прерываний общего назначения, мы обрабаты-
967     ; вали бы это прерывание, как обычное событие 80386. Однако
968     ; здесь мы инициируем дамп и игнорируем это неожиданное явление.
969     ;
970 51A4  C6 06 4ADD R 03              mov     debugf0,3  ;Выдать дамп при аварии
971 51A9  C6 06 4ADE R 03              mov     debugf0a,3
972 51AE                stillok:
973 51AE  8C DA                     mov     DX,DS       ;Установить селектор
```

```
 974 51B0  8E C2                    mov     ES,DX       ;дескриптора в ES
 975 51B2  80 3E 4ADD R 00          cmp     debugf0,0   ;Нужен дамп?
 976 51B7  74 42                    je      skip02      ;Нет, обойти
 977 51B9  80 3E 4ADD R 01          cmp     debugf0,1   ;Тип трассировки?
 978 51BE  75 09                    jne     fulld02     ;Нет, сделать по максимуму
 979 51C0  A1 4AE2 R                mov     AX,retdisp  ;Получить номер прерывания
 980 51C3  E8 5633 R                call    mindump     ;и трассировать
 981 51C6  EB 33 90                 jmp     skip02      ;Выполнено
 982 51C9                  fulld02:
 983 51C9  80 3E 4ADD R 02          cmp     debugf0,2   ;Нужен обычный дамп?
 984 51CE  74 28                    je      fulld02e    ;Да, на дамп
 985 51D0  C6 06 4ADD R 00          mov     debugf0,0   ;Выполнить аварийный дамп
 986     ; Мы должны иметь дополнительный стек на случай аварийной ситуации.
 987 51D5  B8 0030                  mov     AX,stkdata  ;Получить область стека
 988 51D8  8E D8                    mov     DS,AX       ;Дамп стека
 989 51DA  B8 0780                  mov     AX,780h     ;Сначала первую часть -
 990 51DD  B9 0008                  mov     CX,8        ;128 байт
 991 51E0                  badstk0:
 992 51E0  E8 5471 R                call    dumpln
 993 51E3  E2 FB                    loop    badstk0     ;Получить весь
 994
 995 51E5  E8 5464 R                call    newln       ;Промежуток
 996 51E8  B8 1780                  mov     AX,1780h    ;Теперь старшую часть -
 997 51EB  B9 0008                  mov     CX,8        ;128 байт
 998 51EE                  badstk1:
 999 51EE  E8 5471 R                call    dumpln
1000 51F1  E2 FB                    loop    badstk1     ;Получить весь
1001 51F3  E8 5464 R                call    newln       ;и промежуток
1002 51F6  8E DA                    mov     DS,DX       ;Селектор дескриптора
1003
1004 51F8                  fulld02e:
1005 51F8  E8 55D2 R                call    edump
1006 51FB                  skip02:
1007 51FB  8B 36 4AE2 R             mov     SI,retdisp  ;Смещение возврата как сме-
1008 51FF  81 EE 4D78               sub     SI,int_hand0-begin+100h;щение к векто-
1009 5203  A1 E6 FFFC               and     SI,0fffch   ;рам прерываний MS-DOS
1010 5207  89 36 4AE2 R             mov     retdisp,SI  ;Очистить для отладки
1011
1012 520B  B8 0030                  mov     AX,stkdata  ;Пекреслать стек как селек-
1013 520E  8E C0                    mov     ES,AX       ;тор данных в ES
1014 5210  8B DC                    mov     BX,SP       ;Отправить SP в BX
1015 5212  83 C3 10                 add     BX,16       ;Указатель на ячейку с IP
1016 5215  83 FE 34                 cmp     SI,13*4     ;Это прерывание 13?
1017 5218  75 03                    jne     notint13    ;Нет, просто продолжить
1018 521A  E9 52DC R                jmp     isint13     ;Да, специальная обработка
1019
```

```
1020 521D                notint13:
1021 521D  26: 8B 47 10          mov    AX,ES:[BX].oldss ;SS задачи V86
1022 5221  8A CC                 mov    CL,AH       ;Старший байт в CL
1023 5223  C1 E0 04              shl    AX,4        ;Преобразовать дескриптор в
1024 5226  C0 E9 04              shr    CL,4        ;линейный адрес в AX и CL
1025 5229  A3 212A R             mov    ref_d.base_l,AX;Биты 0-15 в дескрипторе
1026 522C  88 0E 212C R          mov    ref_d.base_m,CL;Биты 16-23 (Биты 24-31
1027                                                    ;игнорируем)
1028 5230  26: 8B 4F 0C          mov    CX,EX:[BX].oldsp;Получить SS V86, нас-
1029 5234  83 E9 06              sub    CX,6            ;троить его на лохное
1030 5237  26: 89 4F 0C          mov    ES:[BX].oldsp,CX;прерывание и назад
1031
1032 523B  26: 8B 2F             mov    BP,ES:[BX].oldip;Получить IP,
1033 523E  26: 8B 7F 04          mov    DI,ES:[BX].oldcs;CS
1034 5242  26: 8B 47 08          mov    AX,ES:[BX].oldfl;и флаги до прерывания
1035
1036 5246  BA 0018               mov    DX,ref_dta ;Получить дескриптор
1037 5249  8E DA                 mov    DS,DX      ;в DS
1038 524B  87 D9                 xchg   BX,CX      ;Отправить SP V86 в BX и со-
1039                                               ;хранить дескриптор стека
1040 524D  89 2F                 mov    [BX],BP    ;IP стека,
1041 524F  89 7F 02              mov    [BX]+2,DI  ;CS и
1042 5252  89 47 04              mov    [BX]+4,AX  ;флаги в стек V86
1043 5255  25 FCFF               and    AX,ofcffh  ;Очистить флаги прерываний
1044                                               ;и ловушек
1045 5258  BB 0028               mov    BX,ihzero  ;Селектор дескриптора нуле-
1046 525B  8E DB                 mov    DS,BX      ;вой базы как  DS
1047 525D  8B DE                 mov    BX,SI      ;Индекс в таблице векторов
1048 525F  8B 2F                 mov    BP,[BX]    ;Получить IP вектора
1049 5261  8B 7F 02              mov    DI,[BX]+2  ;и CS
1050
1051 5264  8B D9                 mov    BX,CS      ;Получить смещение стека
1052 5266  26: 89 47 08          mov    ES:[BX].oldfl,AX  ;Установить флаги,
1053 526A  26: 89 2F             mov    ES:[BX].oldip,BP  ;IP и
1054 526D  26: 89 7F 04          mov    ES:[BX].oldcs,DI  ;CS на нашем стеке
1055
1056 5271                rejoin:
1057 5271  BA 0060               mov    DX,supdd   ;Получить адресуемость
1058 5274  8E DA                 mov    DS,DX      ;данных
1059 5276  8E C2                 mov    ES,DX      ;
1060 5278  80 3E 4ADE R 00       cmp    debugf0a,0 ;Нужен дамп завершения?
1061 527D  74 54                 je     skip03     ;Нет, обойти
1062 527F  80 3E 4ADE R 01       cmp    debugf0a,1 ;Просто трассировка?
1063 5284  75 08                 jne    fulld02a   ;Нет, полностью
1064 5286  8B C5                 mov    AX,BP      ;Получить адрес назначения
1065 5288  E8 5633 R             call   mindump    ;Трассировать
```

```
1066 528B  EB 46 90                jmp     skip03     ;Выполнено
1067 528E                  fulld02a:
1068 528E  E8 55D2 R               call    edump
1069
1070 5291  B8 0030                 mov     AX,stkdata ;Переслать стек как селектор
1071 5294  8E D8                   mov     DS,AX      ;данных в DS
1072 5296  BB 17DC                 mov     BX,ihspn1  ;Смещение к стеку PL0
1073 5299  8B 47 10                mov     AX,[BX].oldss   ;Получить SS задачи V86
1074 529C  8B 5F 0C                mov     BX,[BX].oldsp   ;и SP
1075 529F  8E DA                   mov     DS,DX           ;Восстановить DS
1076 52A1  8A CC                   mov     CL,AH           ;Приготовиться
1077 52A3  C0 E9 04                shr     CL,4            ;установить
1078 52A6  88 0E 212C R            mov     ref_d.base_m,CL ;биты 16-23,
1079 52AA  C1 E0 04                shl     AX,4            ;затем биты 0-15 адре-
1080 52AD  A3 212A R               mov     ref_d.base_l,AX ;са в дескрипторе
1081 52B0  B8 0018                 mov     AX,ref_dta ;Получить селектор дескрип-
1082 52B3  8E D8                   mov     DS,AX      ;тора в DS
1083 52B5  8B C3                   mov     AX,BX      ;Установить SP как смещение
1084 52B7  B9 0004                 mov     CX,4       ;Приготовиться к дампу
1085 52BA                  dumpit380:
1086 52BA  E8 5471 R               call    dumpln     ;Выводить по 16 за раз
1087 52BD  E2 FB                   loop    dumpit380
1088
1089 52BF  B8 0060                 mov     AX,supdd   ;Вернуть нормальное значе-
1090 52C2  8E D8                   mov     DS,AX      ;ние DS в DS
1091
1092 52C4  E8 5464 R               call    newln      ;Перевод строки
1093 52C7  80 3E 4ADE R 03         cmp     debugf0a,3 ;Выполняем аварийный дамп?
1094 52CC  75 05                   jne     skip03     ;Нет, оставляем как есть
1095 52CE  C6 06 4ADE R 00         mov     debugf0a,0 ;Прекратить
1096 52D3                  skip03:
1097 52D3  2B C0                   sub     AX,AX      ;Получить 0
1098 52D5  8E D8                   mov     DS,AX      ;для DS
1099 52D7  8E C0                   mov     ES,AX      ;и ES
1100 52D9  61                      popa               ;Восстановить все флаги
1101       ; Поскольку мы используем 16-разрядные операнды, нам нужен
1102       ; 32-разрядный IRET для распознавания переключения в V86.
1103                               opsize
1104 52DB  CF                      iret
1105
1106       ;
1107       ; 3.
1108       ;
1109       ; Прерывание 13. Надо получить CS V86 в качестве дескриптора,
1110       ; чтобы проверить: INT или IRET.
1111       ;
```

```
1112 52DC                isint13:
1113 52DC  26: 8B 47 04          mov     AX,ES:[BX].oldcs ;Загрузить CS в AX
1114 52E0  26: 8B 37             mov     SI,ES:[BX].oldip ;Отправить IP в SI
1115 52E3  8A CC                 mov     CL,AH       ;Для вычисления средних бит
1116 52E5  C1 E0 04              shl     AX,4        ;Получить дескриптор (0-15)
1117 52E8  C0 E9 04              shr     CL,4        ;и средние 8 бит
1118 52EB  A3 212A R             mov     ref_d.base_l,AX ;Биты 0-15 и средние 8
1119 52EE  88 0E 212C R          mov     ref_d.base_m,CL ;бит линейного адреса
1120 52F2  B8 0018               mov     AX,ref_dta ;Селектор только что создан-
1121 52F5  8E D8                 mov     DS,AX       ;ного дескриптора в DS
1122 52F7  FC                    cld                 ;Убедиться в инкременте SI
1123 52F8  AC                    lodsb               ;Получить код команды, выз-
1124 52F9  3C CF                 cmp     AL,0cfh     ;вавшей отказ. Это IRET?
1125 52FB  74 4A                 je      prociret    ;Да, на эмуляцию IRET
1126 52FD  3C CD                 cmp     AL,0cdh     ;Это INT nn?
1127 52FF  74 10                 je      procint     ;Да, на эмуляцию INT nn
1128       ;
1129       ; Это неожиданно - сделаем дамп.
1130       ;
1131 5301  8E DA                 mov     DS,DX       ;Обычный дескриптор данных
1132 5303  E8 55D2 R             call    edump       ;Теперь дамп
1133 5306  C6 06 4ADE R 03       mov     debugf0a,3 ;Вынужденный дапм при выходе
1134 530B  BE 0034               mov     SI,13*4     ;Установим обычное смещение
1135 530E  E9 521D R             jmp     notint13    ;и на обработку обычного
1136                                                    ;прерывания
1137       ;
1138       ; 4.
1139       ;
1140       ; Обрабатываем IRET.
1141       ;
1142 5311                procint:
1143 5311  AC                    lodsb               ;Получить nn
1144 5312  26: 89 37             mov     ES:[BX].oldip,SI ;Новый IP в стек
1145 5315  8E DA                 mov     DS,DX       ;Установить обычный DS
1146 5317  B4 00                 mov     AH,0        ;Очистить старший байт AX
1147 5319  03 C0                 add     AX,AX       ;Получить nn*2
1148 531B  03 C0                 add     AX,AX       ;и nn*4
1149 531D  8B F0                 mov     SI,AX       ;как вектор прерывания
1150 531F  75 23                 jnz     notint00    ;Переход если не INT 00
1151       ;
1152       ; Был запрос на выполнение команды. Копируем DS, AX и DS вызывающей
1153       ; программы в TSS супервизора, как указатели параметров.
1154       ;
1155 5321  26: 8B 47 18          mov     AX,ES:[BX].oldss ;Поместить DS в CX
1156 5325  A3 29C4 R             mov     supv.tecx,AX     ;супервизора
1157 5328  8B CB                 mov     CX,BX       ;Сохранить указатель в стеке
```

```
1158 532A  8B DC                    mov     BX,SP       ;Указатель на регистры
1159 532C  26: 8B 47 0E             mov     AX,ES:[BX+14] ;Поместить AX вызывающей
1160 5330  A3 29C0 R                mov     supv.teax,AX  ;задачи в AX супервизора
1161 5333  26: 8B 47 0A             mov     AX,ES:[BX+10] ;И DX тоже
1162 5337  A3 29C8 R                mov     word ptr supv.tedx,AX
1163 533A  8B D9                    mov     BX,CX       ;Восстановить обычный индекс
1164                            ;   call    ref0d:0     ;Вызвать задачу 80386
1165 533C  9A                       db      09ah
1166 533D  0000 0040                dw      0,ref0d
1167 5341  E9 5271 R                jmp     rejoin      ;К строкам обработки int 00
1168 5344                   notint00:
1169 5344  E9 521D R                jmp     notint13    ;На обработку обычного
1170                                                    ;прерывания
1171      ; 5.
1172      ;
1173      ; Обработка IRET.
1174      ;
1175 5347                   prociret:
1176 5347  8E DA                    mov     DS,DX       ;CS из V86 больше не нужен
1177 5349  26: 8B 47 10             mov     AX,ES:[BX].oldss ;Нужен адрес возврата
1178 534D  8A CC                    mov     CL,AH       ;Вычислим дескриптор
1179 534F  C0 E9 04                 shr     CL,4
1180 5352  C1 E0 04                 shl     AX,4
1181 5355  A3 212A R                mov     ref_d.base_l,AX ;Биты 0-15
1182 5358  88 0E 212C R             mov     ref_d.base_m,CL ;и биты 16-23
1183 535C  26: 8B 4F 0C             mov     CX,ES:[BX].oldsp;Получить SP V86
1184 5360  B8 0018                  mov     AX,ref_dta ;Получить SS V86
1185 5363  8E D8                    mov     DS,AX       ;как наш DS
1186 5365  87 D9                    xchg    BX,CX       ;Установить SP V86 в BX
1187 5367  8B 07                    mov     AX,[BX]     ;Получить IP возврата,
1188 5369  8B 77 02                 mov     SI,[BX+2]   ;CS и
1189 536C  8B 7F 04                 mov     DI,[BX+4]   ;флаги
1190 536F  83 C3 06                 add     BX,6        ;Настроить SP
1191 5372  87 D9                    xchg    BX,CX       ;и отправить его в CX
1192 5374  26: 89 4F 0C             mov     ES:[BX].oldsp,CX ;Теперь поместим зна-
1193 5378  26: 89 07                mov     ES:[BX].oldip,AX ;чения в наш стек
1194 537B  26: 89 77 04             mov     ES:[BX].oldcs,SI
1195 537F  26: 89 7F 08             mov     ES:[BX].oldfl,DI
1196 5383  E9 5271 R                jmp     rejoin    ;К строкам завершения-возврата
1197
1198      ;
1199      ; 6.
1200      ;
1201      ; Эта программа выполняется, как задача защищенного режима 80386.
1202      ; Обработчик прерываний задачи V86 вызывает эту задачу, когда он
1203      ; обнаруживает INT 00. Эта задача (которая носит чисто иллюстративный
```

```
1204      ; характер) выполняет команды, на которые указывает DS:DX задачи V86.
1205      ; За этими командами должна следовать команда RET FAR.
1206      ;
1207      ; Обработчик прерываний записывает в TSS значения DS и DX перед
1208      ; вызовом задачи.
1209      ;
1210 5386                    instexe    PROC
1211 5386                    instexb:
1212 5386  80 3E 4ADF R 00              cmp     debugf1,0  ;Дамп нужен?
1213 538B  74 12                        je      skip04     ;Нет, обойти
1214 538D  80 3E 4ADF R 01              cmp     debugf1,1  ;Только трассировка?
1215 5392  75 08                        jne     fulld04    ;Нет, полностью
1216 5394  B8 0E0D                      mov     AX,oe0dh   ;Получить номер int
1217 5397  E8 5633 R                    call    mindump    ;Трассировать
1218 539A  EB 03                        jmp     short skip04 ;Выполнено
1219 539C                    fulld04:
1220 539C  E8 55D2 R                    call    edump      ;Получить данные
1221 539F                    skip04:
1222 539F  50                           push    AX         ;Сохранить контрольную
1223                                    opsize             ;информацию
1224 53A1  2B C0                        sub     AX,AX      ;Скопировать дескриптор
1225 53A3  8B C1                        mov     AX,CX      ;в чистый регистр
1226                                    opsize
1227 53A6  2B C9                        sub     CX,CX      ;Очистить и ECX
1228 53A8  8B CA                        mov     CX,DX      ;Смещение к программе
1229                                    opsize
1230 53AB  C1 E0 04                     shl     AX,4       ;Получить дескриптор, как
1231                                    opsize
1232 53AF  03 C1                        add     AX,CX      ;линейный адрес прогорраммы
1233 53B1  A3 2182 R                    mov     exe_c.base_l,AX ;Установить младшие 16
1234 53B4  A3 218A R                    mov     exe_d.base_l,AX ;бит в дескрипторах
1235 53B7  A3 2192 R                    mov     exe_s.base_l,AX ;программы, данных и
1236                                    opsize                  ;стека
1237 53BB  C1 E8 08                     shr     AX,8            ;Биты 16-24 в AH
1238 53BE  88 26 2184 R                 mov     exe_c.base_m,AH ;Установить средние 8
1239 53C2  88 26 218C R                 mov     exe_d.base_m,AH ;бит в тех же
1240 53C6  88 26 2194 R                 mov     exe_s.base_m,AH ;дескрипторах
1241 53CA  B8 0078                      mov     BX,exedd   ;Установить селектор данных
1242 53CD  B9 0080                      mov     CX,exesd   ;в BX, а стека - в CX
1243 53D0  58                           pop     AX         ;Восстановить AX
1244                                    call    execs:0    ;и вызвать программу
1245 53D1  9A                           db      09ah
1246 53D2  0000 0070                    dw      0,execd
1247 53D6  75 03                        jnz     skipdmp    ;Если возвращен не 0 - обход
1248 53D8  E8 55D2 R                    call    edump      ;Иначе дамп нашей среды
1249 53DB                    skipdmp:
```

```
1250 53DB  B8 0060                    mov     AX,suppdd  ;Получить наш селектор дан-
1251 53DE  8E D8                      mov     DS,AX      ;ных и обеспечить следующий
1252 53E2  8E C0                      mov     ES,AX      ;вход
1253 53E2  CF                         iret               ;Возврат
1254 53E3  EB A1                      jmp     instexb    ;На начало
1255                          instexe ENDP
1256                                  PAGE
1257
1258      ;
1259      ; ПРОГРАММА ОТЛАДКИ
1260      ;
1261      ; Вывод отладочных сообщений на печать, а не на дисплей, связан с тем,
1262      ; что ошибки в такого рода программах часто вызывают перезагрузку системы.
1263      ; В этом случае отладочные сообщения сохраняются лишь в виде недолго-
1264      ; вечного изображения на сетчатке глаза. Хотелось бы иметь более
1265      ; долговечные данные.
1266      ;
1267
1268 53E5                getdsv    PROC
1269      ;
1270      ; Процедура заполняет дескриптор в GDT линейным адресом сегмента
1271      ; (адрес сегмента в BX) и загружает значение дескриптора в BP.
1272      ; Вызывающая программа переместит его (обычно) в DS, в результате
1273      ; чего линейный адрес загрузится из GDT (стерев любое предвари-
1274      ; тельно кэшированное значение).
1275      ;
1276                                    opsize
1277 53E6  50                          push    AX
1278                                    opsize
1279 53EB  2B C0                      sub     AX,AX      ;Очистить EAX
1280 53E2  8B C3                      mov     AX,BX      ;Получить регистр сегмента
1281                                    opsize
1282 53ED  C1 E0 04                   shl     AX,4       ;Линейный адрес сегмента
1283 53F0  A3 2132 R                  mov     ref_dd.base_l,AX ;Установить биты 0-15
1284                                    opsize
1285 53F4  C1 E8 08                   shr     AX,8       ;и загрузить биты 16-23 AH
1286 53F7  88 26 2134 R               mov     ref_dd.base_m,AH ;в дескриптор
1287 53FB  BD 0020                    mov     BP,ref_dtad;Получить значение селектора
1288                                    opsize
1289 53FF  58                         pop     AX
1290 5400  C3                         ret
1291                          getdsv  ENDP
1292
1293 5401                     printc   PROC
1294      ;
1295      ; Вывод на печать байта из AL.
```

```
1296      ; Примечание: содержимое AL разрушается, но только  AL.
1297      ;
1298 5401  51                        push    CX          ;Сохранить используемые
1299 5402  53                        push    BX          ;регистры
1300 5403  52                        push    DX
1301 5404  BA 03BC                    mov     DX,03bch    ;Адрес порта (данных)
1302 5407  EE                        out     DX,AL       ;Напечатать байт
1303 5408  B3 10                     mov     BL,10h      ;Установить счетчик задержки
1304 540A  2B C9                     sub     CX,CX       ;и счетчик цикла
1305 540C  42                        inc     DX          ;Адрес порта состояния
1306 540D                 busy1:
1307 540D  EC                        in      AL,DX       ;Получить состояние
1308 540E  A8 80                     test    AL,80h      ;Проверить бит занятости
1309 5410  75 06                     jnz     strobe     ;Сообщить о готовности данных
1310 5412  E2 F9                     loop    busy1       ;64K раз
1311 5414  FE CB                     dec     BL          ;16 раз проверяем бит
1312 5416  75 F5                     jnz     busy1       ;занятости
1313 5418                 strobe:
1314 5418  B0 0D                     mov     AL,0dh      ;Данные имеются
1315 541A  42                        inc     DX          ;Используем порт строба
1316 541B  EE                        out     DX,AL       ;Послать сигнал
1317 541C  B9 00FF                   mov     CX,255      ;Короткая задержка
1318 541F                 signl:
1319 541F  E2 FE                     loop    signl
1320 5421  FE C8                     dec     AL          ;Убрать бит строба
1321 5423  EE                        out     DX,AL
1322 5424  5A                        pop     DX          ;Восстановить регистры
1323 5425  5B                        pop     BX
1324 5426  59                        pop     CX
1325 5427  C3                        ret
1326
1327                      printc      ENDP
1328
1329
1330 5428                 hexit       PROC
1331      ;
1332      ; Преобразование шестнадцатеричной цифры в код символа и печать.
1333      ;
1334 5428  0C 30                     or      AL,30h      ;Сделаем цифру?
1335 542A  3C 3A                     cmp     AL,3ah      ;Получилось?
1336 542C  7C 02                     jl      yes1        ;Если меньше - это 1
1337 542E  04 07                     add     AL,7        ;Преобразуем в A-F
1338 5430                 yes1:
1339 5430  E8 5401 R                 call    printc      ;Напечатать символ
1340 5433  C3                        ret
1341                      hexit       ENDP
```

```
1342
1343 5434                prtalx    PROC
1344      ;
1345      ;Печать байта из AL в виде упорядоченной пары шестнадцатеричных цифр.
1346      ;
1347 5434  50                       push    AX          ;Сохранить регистр
1348 5435  C0 E8 04                 shr     AL,4        ;Старший полубайт
1349 5438  E8 5428 R                call    hexit       ;Вывести его как цифру
1350 543B  58                       pop     AX          ;Получить символ назад
1351 543C  50                       push    AX          ;но сохранить регистр
1352 543D  24 0F                    and     AL,0fh      ;Попробуем младший полубайт
1353 543F  E8 5428 R                call    hexit       ;Напечатать цифру
1354 5442  58                       pop     AX
1355 5443  C3                       ret
1356                     prtalx     ENDP
1357
1358 5444                prtaxx     PROC
1359      ;
1360      ;Преобразование слова в AX в шестнадцатеричную форму и печать.
1361      ;
1362 5444  50                       push    AX          ;Сохраним
1363 5445  50                       push    AX          ;дважды
1364 5446  8A C4                    mov     AL,AH       ;Сначала старший байт
1365 5448  E8 5434 R                call    prtalx      ;Напечатать его
1366 544B  58                       pop     AX
1367 544C  E8 5434 R                call    prtalx      ;Затем младший байт
1368 544F  58                       pop     AX
1369 5450  C3                       ret
1370                     prtaxx     ENDP
1371
1372 5451                prteaxx    PROC
1373      ;
1374      ;Преобразование двойного слова в EAX в шестнадцатеричную форму и печать.
1375      ;
1376                                 opsize
1377 5452  50                       push    AX          ;Сохранить
1378                                 opsize
1379 5454  50                       push    AX          ;дважды
1380                                 opsize
1381 5456  C1 E8 10                 shr     AX,16       ;Сначала старшее слово
1382 5459  E8 5444 R                call    prtaxx      ;Напечатать его
1383                                 opsize
1384 545D  58                       pop     AX
1385 545E  E8 54E3 R                call    prt16       ;Затем младшее слово
1386                                 opsize
1387 5462  58                       pop     AX
```

```
1388 5463  C3                          ret
1389                        prteaxx    ENDP
1390
1391
1392 5464                   newln      PROC
1393      ;
1394      ;Вывод возврата каретки и перевода строки.
1395      ;
1396 5464  50                          push    AX
1397 5465  B0 0D                       mov     AL,0dh      ;Вывести возврат каретки
1398 5467  E8 5401 R                    call    printc
1399 546A  B0 0A                       mov     AL,0ah      ;и перевод строки
1400 546C  E8 5401 R                    call    printc
1401 546F  58                          pop     AX
1402 5470  C3                          ret
1403                        newln      ENDP
1404
1405 5471                   dumpln     PROC
1406      ;
1407      ; Дамп 16 байтов памяти по адресу DS:AX, затем возврат каретки и перевод
1408      ; строки. В AX возвращается следующий адрес после дампа.
1409      ; ES установлен в соответствии с нашим DS для выполнения.
1410      ;
1411 5471  53                          push    BX          ;Сохранить регистры
1412 5472  51                          push    CX
1413 5473  50                          push    AX
1414      ;
1415      ; Получение линейного адреса для DS в виде значения для 8086 или
1416      ; 80386. Выбор зависит от того, превышает ли значение DS gdtsize
1417      ; (что обычно должно быть для этой программы).
1418      ;
1419                                  opsize
1420 5475  2B C0                       sub     AX,AX       ;Очистить EAX
1421                                  opsize
1422 5478  2B DB                       sub     BX,BX       ;и EBX
1423 547A  8C DB                       mov     BX,DS       ;Получить селектор сегмента
1424 547C  B8 0088                     mov     AX,gdt_size;Сделать как в 80386
1425                                  opsize
1426 5480  3B D8                       cmp     BX,AX       ;Проверить формат
1427 5482  7F 17                       jg      do8086      ;Должен быть формат 8086
1428      ;
1429      ; Мы могли ошибиться и получить здесь мусор, но обычно все хорошо.
1430      ;
1431 5484  06                          push    ES          ;Сохранить текущий ES
1432 5485  B8 0060                     mov     AX,supdd    ;Загрузить стандартный се-
1433 5488  8E C0                       mov     ES,AX       ;лектор данных в ES
```

```
1434 548A  26: 8A A7 2114 R          mov     AH,ES:gdt_def.base_m[BX] ;Старший байт
1435                                 opsize
1436 5490  C1 E0 08                  shl     AX,8        ;в старшую половину, затем
1437 5493  26: 8B 87 2112 R          mov     AX,ES:gdt_def.base_l[BX] ;биты 0-15
1438 5498  07                        pop     ES          ;Восстановить ES
1439 5499  EB 06                     jmp     short dolinear ;Скомбинировать сегмент
1440 549B                  do8086:
1441 549B  8B C3                     mov     AX,BX       ;Дескриптор в AX
1442                                 opsize
1443 549E  C1 E0 04                  shl     AX,4        ;Умножить на 16 в EAX
1444 54A1                  dolinear:
1445                                 opsize
1446 54A2  2B DB                     sub     BX,BX       ;Очистить EBX
1447 54A4  5B                        pop     BX          ;Получить смещение
1448 54A5  53                        push    BX          ;Сохранить
1449                                 opsize
1450 54A7  03 C3                     add     AX,BX       ;Получили линейный адрес
1451
1452 54A9  E8 5451 R                 call    prteaxx     ;Вывести адрес
1453 54AC  B0 20                     mov     AL,' '      ;и два пробела
1454 54AE  E8 5401 R                 call    printc
1455 54B1  B9 0008                   mov     CX,8        ;8 слов
1456 54B4                  nxtword:
1457 54B4  8B 07                     mov     AX,[BX]     ;Байты переставлены
1458 54B6  83 C3 02                  add     BX,2        ;Сместить указатель
1459 54B9  86 C4                     xchg    AL,AH       ;Переставить байты
1460 54BB  E8 54E3 R                 call    prt16       ;Напечатать
1461 54BE  E2 F4                     loop    nxtword     ;для всех 8 слов
1462 54C0  B0 20                     mov     AL,' '      ;Получить еще пробел
1463 54C2  E8 5401 R                 call    printc      ;и напечатать байт
1464 54C5  5B                        pop     BX          ;Получить указатель
1465 54C6  B9 0010                   mov     CX,16       ;Получить счетчик байтов
1466 54C9                  nxtbyte:
1467 54C9  8A 07                     mov     AL,[BX]     ;Получить байт
1468 54CB  43                        inc     BX          ;и сместить указатель
1469 54CC  3C 20                     cmp     AL,' '      ;Сравнить с пробелом
1470 54CE  7C 04                     jl      isactl      ;Если меньше - сделать точку
1471 54D0  3C 7E                     cmp     AL,07eh     ;Слишком много для печати?
1472 54D2  7E 02                     jng     nothigh     ;Нет, использовать как есть
1473 54D4                  isactl:
1474 54D4  B0 2E                     mov     AL,'.'      ;Да, заменить точкой
1475 54D6                  nothigh:
1476 54D6  E8 5401 R                 call    printc      ;Напечатать символ
1477 54D9  E2 EE                     loop    nxtbyte     ;Повторить 16 раз
1478
1479 54DB  E8 5464 R                 call    newln       ;Возврат каретки, перевод
```

```
1480                                                    ;строки
1481 54DE  8B C3                    mov     AX,BX      ;Возвращаемое значение
1482 54E0  59                       pop     CX
1483 54E1  5B                       pop     BX
1484 54E2  C3                       ret
1485                        dumpln  ENDP
1486
1487 54E3                   prt16   PROC
1488     ;
1489     ; Процедура для печати AX в шестнадцатеричной форме и пробела.
1490     ;
1491 54E3  E8 5444 R                call    prtaxx     ;Напечатать значение в AX
1492 54E6  B0 20                    mov     AL,' '     ;Затем пробел
1493 54EB  E8 5401 R                call    printc     ;(вывести)
1494 54EB  C3                       ret
1495                        prt16   ENDP
1496
1497 54EC               dumprg   PROC
1498     ;
1499     ; Дамп всех регистров.
1500     ;
1501                                 opsize
1502 54ED  50                       push    AX
1503
1504 54EE  E8 5464 R                call    newln      ;Возврат каретки, перевод
1505                                                    ;строки
1506                                 opsize
1507 54F2  9C                       pushf              ;Сохранить EFLAGS
1508 54F3  E8 5451 R                call    prteaxx    ;Вывести AX,
1509                                 opsize
1510 54F7  8B C3                    mov     AX,BX
1511 54F9  E8 5451 R                call    prteaxx    ;затем BX,
1512                                 opsize
1513 54FD  8B C1                    mov     AX,CX
1514 54FF  E8 5451 R                call    prteaxx    ;затем CX,
1515                                 opsize
1516 5503  8B C2                    mov     AX,DX
1517 5505  E8 5451 R                call    prteaxx    ;затем DX,
1518                                 opsize
1519 5509  8B C5                    mov     AX,BP
1520 550B  E8 5451 R                call    prteaxx    ;затем BP,
1521                                 opsize
1522 550F  8B C4                    mov     AX,SP
1523 5511  E8 5451 R                call    prteaxx    ;затем SP,
1524                                 opsize
1525 5515  8BC6                     mov     AX,SI
```

```
1526 5517  E8 5451 R              call    prteaxx    ;затем SI
1527                              opsize
1528 551B  8B C7                  mov     AX,DI
1529 551D  E8 5451 R              call    prteaxx    ;и наконец DI
1530
1531 5520  E8 5464 R              call    newln      ;Перевод строки,
1532
1533 5523  8C C8                  mov     AX,CX
1534 5525  E8 54E3 R              call    prt16      ;затем вывести CS,
1535 5528  8C D8                  mov     AX,DS
1536 552A  E8 54E3 R              call    prt16      ;затем DS,
1537 552D  8C C0                  mov     AX,ES
1538 552F  E8 54E3 R              call    prt16      ;затем ES,
1539 5532  8C D0                  mov     AX,SS
1540 5534  E8 54E3 R              call    prt16      ;затем SS,
1541 5537  8C E0                  db      8ch,0e0h
1542 5539  E8 54E3 R              call    prt16      ;затем FS,
1543 553C  8C E8                  db      8ch,0e8h
1544 553E  E8 54E3 R              call    prt16      ;затем GS,
1545 5541  0F 00 C0               sldt    AX
1546 5544  E8 54E3 R              call    prt16      ;затем LDT
1547 5547  0F 00 C8               str     AX
1548 554A  E8 54E3 R              call    prt16      ;и регистр задачи
1549
1550 554D  E8 5464 R              call    newln      ;Перевод строки
1551
1552                        ;     mov     AX,CR0
1553 5550  0F 20 C0               db      0fh,20h,0c0h ;Получить CR0
1554 5553  E8 5451 R              call    prteaxx      ;и напечатать его
1555 5556  0F 20 D0               db      0fh,20h,0d0h ;Получить CR2
1556 5559  E8 5451 R              call    prteaxx      ;и напечатать его
1557 555C  0F 20 D8               db      0fh,20h,0d8h ;Получить CR3
1558 555F  E8 5451 R              call    prteaxx      ;и напечатать его
1559 5562  0F  01 06 4AD2 R       sgdt    qword ptr dtsize;Сохранить регистр GDT,
1560                              opsize
1561 5568  A1 4AD4 R              mov     AX,word ptr dtload ;получить адрес
1562 5568  E8 5451 R              call    prteaxx            ;и вывести
1563 556E  0F 01 0E 4AD2 R        sidt    qword ptr dtsize;Сохранить регистр IDT,
1564                              opsize
1565 5574  A1 4AD4 R              mov     AX,word ptr dtload;получить адрес
1566 5577  E8 5451 R              call    prteaxx           ;и вывести
1567                              opsize
1568 557B  58                     pop     AX         ;Получить регистр EFLAGS
1569 557C  E8 5451 R              call    prteaxx    ;и вывести его
1670
1571 557F  E8 5464 R              call    newln      ;Перевод строки
```

```
1572
1573                                    opsize
1574 5583  58                          pop     AX
1575 5584  C3                          ret
1576                                    dumprg  ENDP
1577
1578 5585                               dmptss  PROC
1579      ;
1580      ; Дамп TSS и его стека (64 байта).
1581      ; При входе в процедуру в CX смещение к TSS.
1582      ;
1583 5585  50                          push    AX          ;Сохранить используемые
1584 5586  53                          push    BX          ;регистры
1585 5587  51                          push    CX          ;Сохранить смещение к TSS
1586 5588  8B C1                       mov     AX,CX       ;Установить смещение к TSS
1587 558A  B9 0007                     mov     CX,7        ;Приготовиться к дампу 70h
1588 558D                   dumpit2:
1589 558D  E8 5471 R                   call    dumpln      ;Вывести по 16 на строке
1590 5590  E2 FB                       loop    dumpit2
1591
1592 5592  E8 5464 R                   call    newln       ;Перевод строки
1593
1594      ;
1595      ; Теперь выполняем дамп содержимого стека задачи.
1596      ; Сначала нам надо скомпоновать DS для отображения SS задачи,
1597      ; который может иметь формат либо 80386, либо V8086.
1598      ; Затем можно выполнить дамп данных.
1599      ;
1600 5595  5B                          pop     BX          ;Получить смещение к этому
1601 5596  53                          push    BX          ;TSS и снова сохранить его
1602 5597  8B 4F 50                    mov     CX,[BX].tss;Получить значение SS зада-
1603 559A  8B 57 38                    mov     DX,[BX].tesp;чи и SP задачи (16 бит)
1604 55D9  8B 47 26                    mov     AX,[BX].teflags+2;Получить старшие 16
1605 55A0  24 02                       and     AL,2  ;бит EFLAGS и выделить флаг V86
1606 55A2  75 11                       jnz     short gotss;Если V86 - все хорошо
1607 55A4  80 E1 F8                    and     CL,0f8h     ;Нельзя для локальных стеков
1608 55A7  8D 1E 211                   lea     BX,gdt_def ;Получить расположение GDT
1609 55AB  03 D9                       add     BX,CX       ;Смещение к Sset
1610                                    opsize
1611 55AE  8B 4F 02                    mov     CX,[BX].base_1;Младшие 24 бита в ECX
1612                                    opsize
1613 55B2  C1 E9 04                    shr     CX,4        ;и дескриптор 8086 в CX
1614 55B5                   gotss:
1615 55B5  8B D9                       mov     BX,CX       ;В регистр для процедуры
1616 55B7  E8 53E5 R                   call    getdsv      ;Построен дескриптор BP
1617 55BA  8E DD                       mov     DS,BP       ;Установить как DS
```

```
1618 55BC  8B C2                    mov     AX,DX       ;Установить значение SP
1619 55B2  B9 0004                  mov     CX,4        ;Установить счетчик
1620 55C1                  dumpiSS:
1621 55C1  E8 5471 R                call    dumpln      ;Вывести по 16 на строке
1622 55C4  E2 FB                    loop    dumpiSS
1623
1624 55C6  B8 0060                  mov     BX,supdd    ;Получить назад DS
1625 55C9  8E DB                    mov     DS,BX       ;в DS
1626
1627 55CB  E8 5464 R                call    newln       ;Перевод строки
1628
1629 55CE  59                       pop     CX          ;Восстановить сохраненные
1630 55CF  5B                       pop     BX          ;регистры
1631 55D0  58                       pop     AX
1632 55D1  C3                       ret
1633                       dmptss   ENDP
1634
1635 55D2                  edump    PROC
1636      ;
1637      ;
1638      ; Дамп GDT, IDT и всех TSS.
1639      ; Смещение к конкретному TSS в CX.
1640      ;
1641 55D2  8F 06 4AE8 R             pop     dumpcal     ;Адрес вызова
1642 55D6  FF 36 4AE8 R             push    dumpcal     ;Вернуть его в стек
1643 55DA  50                       push    AX          ;Сохранить AX,
1644 55DB  53                       push    BX          ;BX,
1645 55DC  51                       push    CX          ;CX
1646 55DD  52                       push    DX          ;и DX
1647
1648 55DE  E8 54EC R                call    dumprg      ;Сначала напечатать регистры
1649
1650 55E1  E8 5464 R                call    newln       ;Перевод строки
1651 55E4  8D 06 4AD2 R             lea     AX,dtsize   ;Указатель на отладочные
1652 55E8  E8 5471 R                call    dumpln      ;данные и сделать дамп
1653 55EB  E8 5471 R                call    dumpln
1654 55EE  E8 5464 R                call    newln
1655
1656 55F1  8D 06 2110 R             lea     AX,gdt_def ;Установить смещение к GDT
1657 55F5  B9 000A                  mov     CX,<idt_def-gdt_def+31>/16;Дамп всего
1658 55F8                  dumpitg:
1659 55F8  E8 5471 R                call    dumpln      ;Вывести по 16 на строке
1660 55FB  E2 FB                    loop    dumpitg
1661
1662 55FD  E8 5464 R                call    newln       ;Перевод строки
1663 5600  8D 0E 2998 R             lea     CX,supv     ;Указатель на TSS супервизора
```

```
1664 5604  E8 5585 R                call    dmptss
1665 5607  8D 0E 2A00 R             lea     CX,supv2    ;Указатель на TSS supv2
1666 560B  E8 5585 R                call    dmptss
1667 560E  8D 0E 2A68 R             lea     CX,v86      ;Указатель на TSS v86
1668 5612  E8 5585 R                call    dmptss
1669
1670 5615  E8 5464 R                call    newln       ;Перевод строки
1671
1672 5618  B9 0006                  mov     CX,6        ;Приготовиться к дампу
1673 561B  8B C4                    mov     AX,SP       ;из SP
1674 561D  8C D3                    mov     BX,SS       ;и SS
1675 561F  8E DB                    mov     DS,BX
1676 5621                  dumpit30:
1677 5621  E8 5471 R                call    dumpln      ;Вывести по 16 на строке
1678 5624  E2 FB                    loop    dumpit30
1679
1680 5626  B8 0060                  mov     AX,supdd    ;Обычное значение DS назад
1681 5629  8E D8                    mov     DS,AX       ;в DS
1682
1683 562B  E8 5464 R                call    newln       ;Перевод строки
1684
1685 562E  5A                       pop     DX          ;Восстановить регистры
1686 562F  59                       pop     CX
1687 5630  5B                       pop     BX
1688 5631  58                       pop     AX
1689 5632  C3                       ret                 ;Возврат
1690                       edump    ENDP
1691
1692 5633                  mindump  PROC
1693       ;
1694       ; Минимум дампа, чтобы не мешал при трассировке.
1695       ;
1696                                opsize
1697 5634  C1 E0 10                 shl     AX,16       ;Сдвинуть AX в конец EAX
1698 5637  58                       pop     AX          ;Получить адрес возврата
1699 5638  50                       push    AX          ;Назад в стек
1700 5639  E8 5451 R                call    prteaxx     ;Вывести
1701 563C  E8 5464 R                call    newln       ;Пустая строка
1702                                opsize
1703 5640  C1 E8 10                 shr     AX,16       ;Получить AX назад
1704 5643  C3                       ret
1705                       mindump  ENDP
1706 5644                  totalsize:
1707                       begin    ENDP
1708 5644                  code     ENDS
1709                                END     begin
```

```
Macros:                    Макроопределения:

N a m eLines               ИмяЧисло_строк

OPSIZE.....................1
SEGADR.....................5
SEGCVT.....................5
SETTSSA....................8

Structures and Records:    Структуры и записи:

                Name   Имя          Width  # fields   Ширина Число полей
                Shift  Width  Mask  Initial   Сдвиг ширина маска Начальное_значение

DESC.....................00080006
  LIMIT..................0000
  BASE_L.................0002
  BASE_M.................0004
  ACCESS.................0005
  GRAN...................0006
  BASE_H.................0007
STKDEF...................00240012
  OLDIP..................0000
  OLDCS..................0004
  OLDFL..................0008
  OLDSP..................000C
  OLDSS..................0010
  OLDES..................0014
  OLDDS..................0018
  OLDFS..................001C
  OLDGS..................0020
TSS_D....................0068002A
  BACK...................0000
  ESP0...................0004
  SS0....................0008
  ESP1...................000C
  SS1....................0010
  ESP2...................0014
  SS2....................0018
  TCR3...................001C
  TEIP...................0020
  TEFLAGS................0024
  TEAX...................0028
  TECX...................002C
  TEDX...................0030
  TEBX...................0034
```

```
  TESP.....................0038
  TEBP.....................003C
  TESI.....................0040
  TEDI.....................0044
  TES......................0048
  TCS......................004C
  TSS......................0050
  TDS......................0054
  TFS......................0058
  TGS......................005C
  TLDT.....................0060
  IOMAP....................0066

Segments and groups:      Сегменты и группы:

                Name  Имя  SizeAlignCombine Class  РазмерВыравниваниеКомбинир Класс

CODE.....................5644PARAPUBLIC'CODE'

Symbols:                  Символы:

                Name  Имя   TypeValueAttr          ТипЗначениеАтрибуты

BADSTK0..................L NEAR51E0CODE
BADSTK1..................L NEAR51EECODE
BEGIN....................F PROC0100CODELength=5544
BUSYL....................L NEAR540DCODE

DATASIZ..................Number2A24
DEBUGF...................L BYTE 4ADCCODE
DEBUGF0..................L BYTE 4ADDCODE
DEBUGF0A.................L BYTE 4ADECODE
DEBUGF1..................L BYTE 4ADFCODE
DEBUGF1A.................L BYTE 4AE0CODE
DEBUGF1B.................L BYTE 4AE1CODE
DMPTSS...................N PROC5585CODELength=004D
D08086...................L NEAR549BCODE
DOLINEAR.................L NEAR54A1CODE
DSADDR...................L WORD 4AE4CODE
DTLOAD...................L DWORD4AD4CODE
DTSIZE...................L WORD 4AD2CODE
DUMPCAL..................L WORD 4AE8CODE
DUMPISS..................L NEAR55C1CODE
DUMPIT2..................L NEAR558DCODE
DUMPIT30.................L NEAR5621CODE
DUMPIT380................L NEAR52BACODE
```

```
DUMPITG.................L NEAR55F8CODE
DUMPLN..................N PROC5471CODELength=0072
DUMPRG..................N PROC54ECCODELength=0099

EDUMP...................N PROC55D2CODELength=0061
ENDAD...................L BYTE 4B24CODE
EXECD...................Number0070
EXEDD...................Number0078
EXESD...................Number0080
EXE_C...................L 00082180CODE
EXE_D...................L 00082188CODE
EXE_S...................L 00082190CODE

FILLIHD1................L NEAR4B98CODE
FULLD02.................L NEAR51C9CODE
FULLD02A................L NEAR528ECODE
FULLD02E................L NEAR51F8CODE
FULLD04.................L NEAR539CCODE

GOT_DEF.................L 00082110CODE
GDT_SIZE................Number0088
GDTSV...................N PROC53E5CODELength=001C
GODOIT..................L NEAR4D38CODE
GOTSS...................L NEAR55B5CODE

HEXIT...................N PROC5428CODELength=000C

IDT_DEF.................NEAR 2198CODE
IDT_SIZE................Number0800
IHCOMMN.................L NEAR5194CODE
IHSPN1..................Number17DC
IHSPN2..................Number17D8
IHSPV...................Number1800
IHZERO..................Number0028
IH_COMMON...............L NEAR517BCODE
INSTEXB.................L NEAR5386CODE
INSTEXE.................N PROC5386CODELength=005F
INTCD1..................L WORD 4AD8CODE
INTCD2..................L WORD 4ADACODE
INT_HANDO...............L NEAR4D7BCODE
ISACTL..................L NEAR54D4CODE
ISINT13.................L NEAR52DCCODE

LOCAL...................L NEAR4D10CODE

MINDUMP.................N PROC5633CODELength=0011
```

```
NEWLINE.................L BYTE 4B21CODE
NEWLN...................N PROC5464CODELength=000D
NOTHIGH.................L NEAR54D6CODE
NOTINTOO................L NEAR5344CODE
NOTINT13................L NEAR521DCODE
NXTBYTE.................L NEAR54C9CODE
NXTWORD.................L NEAR54B4CODE

PRINTC..................N PROC5401CODELength=0027
PROCINT.................L NEAR5311CODE
PROCIRET................L NEAR5347CODE
PRT16...................N PROC54E3CODELength=0009
PRTALX..................N PROC5434CODELength=0010
PRTAXX..................N PROC5444CODELength=000D
PRTEAXX.................N PROC5451CODELength=0013

REF0D...................Number0040
REF1D...................Number0048
REFL0...................Number0010
REFL3...................Number000B
REF_0...................L 00082150CODE
REF_1...................L 00082158CODE
REF_D...................L 00082128CODE
REF_DD..................L 00082130CODE
REF_DTA.................Number0018
REF_DTAD................Number0020
REF_L0..................L 00082120CODE
REF_F3..................L 00082118CODE
REFZER..................L 00082138CODE
REJOIN..................L NEAR5271CODE
RESLOOP.................L NEAR4D4BCODE
RESUM...................L NEAR4D4ACODE
RESUMED.................Number4D4A
RETDISP.................L WORD 4AE2CODE

SEGSETUP................L NEAR4B25CODE
SIGNL...................L NEAR541FCODE
SKIP01..................L NEAR4D23CODE
SKIP02..................L NEAR51FBCODE
SKIP03..................L NEAR52D3CODE
SKIP04..................L NEAR539FCODE
SKIPDMP.................L NEAR53DBCODE
STILLOK.................L NEAR51AECODE
STKDATA.................Number0030
STK_DATA................L 00082140CODE
STROBE..................L NEAR5418CODE
```

```
SUCCESS.................L BYTE 4AEACODE
SUPCD...................Number0058
SUPDD...................Number0060
SUPSD...................Number0068
SUPV....................L 00682998CODE
SUPV2...................L 00682A00CODE
SUP_C...................L 00082168CODE
SUP_D...................L 00082170CODE
SUP_S...................L 00082178CODE
SZ386T..................Number0068
SZV86T..................Number2068

TODOS...................L NEAR4D70CODE
TOTALSIZE...............L NEAR5644CODE

V86.....................L 00682A68CODE
V860D...................Number0053
V86_0...................L 00082160CODE

YES1....................L NEAR5430CODE

    1459 Source Lines     1459 строк исходного текста
    2375 Total  Lines     2375 строк всего
     191 Symbols           191 символов

   39670 Bytes symbol space free  39670 байтов свободного пространства

       0 Worning Errors   0 предупреждений
       0 Severe  Errors   0 серьезных ошибок
Symbol Cross-Reference (# is definition) Таблица перекрестных ссылок (#-определение)

ACCESS.............    83     83#

BACK...............   343    343#
BADSTK0............   991    991#   993
BADSTK1............   998    998#  1000
BASE_H.............    85     85#
BASE_L.............    81     81#   612    613    614    615    641    642    643    645
                     646   1025   1080   1118   1181   1233   1234   1235   1283   1437
                    1611

BASE_M.............    82     82#   612    613    614    615    641    642    643    645
                     646   1026   1078   1119   1182   1238   1239   1240   1286   1434

BEGIN..............   134    134#   136    819    856   1008   1709
BUSYL..............  1306   1306#  1310   1312
```

CODE..... 128 128# 128 129 129 129 1708

DATASIZ..... 584 584#
DEBUGF..... 560 560# 802
DEBUGF0..... 561 561# 873 970 975 977 983 985
DEBUGF0A..... 562 562# 971 1060 1062 1093 1095 1133
DEBUGF1..... 563 563# 1212 1214
DEBUGF1A..... 564 564#
DEBUGF1B..... 565 565#
DESC..... 79# 86
DMPTSS..... 1578 1578# 1664 1666 1668
DO8086..... 1427 1440 1440#
DOLINEAR..... 1439 1444 1444#
DSADDR..... 568 568# 617 750
DTLOAD..... 553 553# 761 775 1561 1565
DTSIZE..... 552 552# 762 764 776 778 1559 1563 1651
DUMPCAL..... 569 569# 1641 1642
DUMPISS..... 1620 1620# 1622
DUMPIT2..... 1588 1588# 1590
DUMPIT30..... 1676 1676# 1678
DUMPIT380..... 1085 1085# 1087
DUMPITG..... 1658 1658# 1660
DUMPL..... 992 999 1086 1405 1405# 1589 1621 1652 1653 1659
DUMPRG..... 1497 1497# 1648

EDUMP..... 805 1005 1068 1132 1220 1248 1635 1635#
ENDAD..... 583 583#
ESP0..... 345 345# 678 708
ESP1..... 348 348# 681 711
ESP2..... 351 351# 684 714
EXECD..... 296 296# 1246
EXEDD..... 306 306# 1241
EXESD..... 315 315# 1242
EXE_C..... 289 289# 296 1233 1238
EXE_D..... 299 299# 306 1234 1239
EXE_S..... 308 308# 315 1235 1240

FILLIHD1..... 628 628# 632
FULLD02..... 978 982 982#
FULLD02A..... 1063 1067 1067#
FULLD02E..... 984 1004 1004#
FULLD04..... 1215 1219 1219#

GDT_DEF..... 151 151# 169 180 191 202 213 221 236 244
252 264 274 283 296 306 315 320 582 645
646 757 1434 1437 1608 1656 1657

```
GDT_SIZE.....  320   320#  762   1424
GETDSV.......  1268  1268#  1616
GODOIT.......  819   827   827#
GOTSS........  1606  1614  1614#
GRAN.........  84    84#

HEXIT........  1330  1330#  1349  1353

IDT_DEF......  328   328#  332   629   771   1657
IDT_SIZE.....  332   332#  776
IHCOMMN......  947   955   955#
IHSPN1.......  587   587#  588   1072
IHSPN2.......  588   588#  946
IHSPV........  586   586#  587   708
IHZERO.......  213   213#  1045
IH_COMMON....  906   906   906   906   906   906   906   906   906   906
               906   906   906   906   906   906   906   906   906   906
               906   906   906   906   906   906   906   906   906   906
               
```

INT_HANDO..... 625 901 901# 1008
IOMAP..... 384 384# 738
ISACTL..... 1470 1473 1473#
ISINT13..... 1018 1112 1112#

LIMIT..... 80 80# 629
LOCAL..... 790 791 791#

MINDUMP..... 980 1065 1217 1692 1692#

NEWLINE..... 582 582# 865
NEWLN..... 806 995 1001 1092 1392 1392# 1479 1504 1531 1550
1571 1592 1627 1650 1654 1662 1670 1683 1701

NOTHIGH..... 1472 1475 1475#
NOTINT00..... 1150 1168 1168#
NOTINT13..... 1017 1020 1020# 1135 1169
NXTBYTE..... 1466 1466# 1477
NXTWORD..... 1456 1456# 1461

OLDCS..... 96 96# 1033 1054 1113 1194
OLDDS..... 106 106# 1155
OLDES..... 104 104#
OLDFL..... 98 98# 1034 1052 1195
OLDFS..... 108 108#
OLDGS..... 110 110#
OLDIP..... 94 94# 1032 1053 1114 1144 1193
OLDSP..... 100 100# 1028 1030 1074 1183 1192
OLDSS..... 102 102# 1021 1073 1077
OPSIZE..... 616 665 669 672 718 721 724 749 755 758
760 769 772 774 830 832 835 837 1103 1223
1226 1229 1231 1236 1276 1278 1281 1284 1288 1376
1378 1380 1383 1386 1419 1421 1425 1435 1442 1449
1501 1506 1509 1512 1515 1518 1521 1524 1527 1560
1564 1567 1573 1610 1612 1696 1702

PRINTC..... 1293 1293# 1339 1398 1400 1454 1463 1476 1493
PROCINT..... 1127 1142 1142#
PROCIRET..... 1125 1175 1175#
PRT16..... 1385 1460 1487 1487# 1534 1536 1538 1540 1542 1544
1546 1548
PRTALX..... 1343 1343# 1365 1367
PRTAXX..... 1358 1358# 1382 1491
PRTEAXX..... 1372 1372# 1452 1508 1511 1514 1517 1520 1523 1526
1529 1554 1556 1558 1562 1566 1569 1700

```
REFOD.............  236   236#  1166
REF1D.............  244   244#  828
REFLO.............  180   180#  381   658   797
REFL3.............  169   169#  742
REF_0.............  229   229#  236   641   641
REF_1.............  237   237#  244   642   642
REF_D.............  184   184#  191   1025  1026  1078  1080  1118  1119  1181
                   1182
REF_DD............  195   195#  202   1283  1286
REF_DTA...........  191   191#  1036  1081  1120  1184
REF_DTAD..........  202   202#  1287
REF_LO............  173   173#  180   646   646
REF_L3............  162   162#  169   645   645
REF_ZER...........  206   206#  213
REJOIN............  1056  1056#  1167  1196
RESLOOP...........  858   858#  872   874
RESUME............  720   855   855#  856
RESUMED...........  856   856#
RETDISP...........  566   566#  945   979   1007  1010

SEGADR............  612   613   614   615   615   634
SEGCVT............  611   612   613   614
SEGSETUP..........  135   593   593#
SETTSSA...........  640   641   642   644   645
SIGNL.............  1318  1318#  1319
SKIP01............  803   807   807#
SKIP02............  976   981   1006  1006#
SKIP03............  1061  1066  1094  1096  1096#
SKIP04............  1213  1218  1221  1221#
SKIPDMP...........  1247  1249  1249#
SS0...............  346   346#  677   707
SS1...............  349   349#  680   710
SS2...............  352   352#  683   713
STILLOK...........  963   972   972#
STKDATA...........  221   221#  958   987   1012  1070
STKDEF............   93#  112   587
STK_DATA..........  214   214#  221   615   615
STROBE............  1309  1313  1313#
SUCSESS...........  570   570#  859
SUPCD.............  264   264#  332   332   332   332   332   332   332   332
                   332   332   332   332   332   332   332   332   332   332
                   332   332   332   332   332   332   332   332   332   332
                   332   332   332   332   332   332   332   332   332   332
                   332   332   332   332   332   332   332   332   332   332
                   332   332   332   332   332   332   332   332   332   332
                   332   332   332   332   332   332   332   332   332   332
```

	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	332	332
	332	332	332	332	332	332	332	332	371	651
	819									
SUPDD.....	274	274#	369	375	652	809	941	1057	1089	1250
SUPSD.....	283	283#	346	349	352	373	657	677	680	683
	707	710	713	816						
SUPV.....	393	393#	439	641	651	653	654	655	656	657
	659	670	673	677	678	680	681	683	684	1156
	1160	1162	1163							
SUPV2.....	443	443#	642	660	1665					
SUP_C.....	257	257#	264	613	613					
SUP_D.....	267	267#	274	612	612					
SUP_S.....	276	276#	283	614	614					
SZ386T.....	229	237	439	439#	738					
SZV86T.....	245	545	545#							
TCR3.....	354	354#								
TCS.....	371	371#	651	700						
TDS.....	375	375#	653	694						
TEAX.....	359	359#	1160							
TEBP.....	366	366#								
TEBX.....	364	364#								
TECX.....	361	361#	1156							
TEDI.....	368	368#								
TEDX.....	363	363#	1162							
TEFLAGS.....	357	357#	729	733	1604					

```
TEIP.............   355    355#   673    722
TES..............   369    369#   654    693
TESI.............   367    367#
TESP.............   365    365#   670    725   1603
TFS..............   377    377#   655    695
TGS..............   379    379#   656    696
TLDT.............   381    381#   659    660    742
TODOS............   870    876    876#
TOTALSIZE........   877   1706   1706#
TSS..............   373    373#   657    698   1602
TSS_D............   342#   385

V86..............   494    494#   545    643    693    694    695    696    698    700
                    707    708    710    711    713    714    722    725    729    733
                    738    742   1667

V860D............   252    252#   443
V86_0............   245    245#   252    643    643

YES1.............  1336   1338   1338#

170 Symbols   170 символов

                                    PAGE    59,132
                                    .286P

                ;
                ; Отладочная тестовая программа для выполнения в защищенном режиме 80386.
                ; Предполагается, что под управлением программы VIRT86L работает система
                ; MS-DOS, как задача виртуального 8086. Выход по INT 00 используется для
                ; организации  связи в защищенном режиме в точке goback.
                ;
0000            code        SEGMENT para 'CODE'
                            ASSUME  CS:code, DS:code, SS:code
0100                        org     100h
0100            begin       PROC    far
0100  E9 0210 R              jmp     startex
0110                        org     begin+16   ;Выровнять на границу
0110  0100[                 db      256 dup (0);параграфа
            00
                        ]
                stkwork     equ     $
0210            startex:
0210  BC 0210               mov     SP,100h+stkwork+begin ;Установить SP
0213  8D 16 0230 R          lea     DX,goback  ;Смещение к программе
```

```
24 0217  8C C9                    mov     CX,CX       ;Получить значение для на-
25 0219  8E D9                    mov     DS,CX       ;шего DS (мы используем CS)
26 021B  CD 00                     int     0          ;Вызов в защищенном режиме
27       ;
28       ; Реальная программа могла бы делать здесь что-то другое. Чувствуйте
29       ; себя свободным в выборе любых необходимых или желательных действий.
30       ;
31 021D  B4 4C                    mov     AH,4ch      ;Установить код 'завершения'
32 021F  CD 21                    int     21h         ;и закончить вызовом MS-DOS
33 = 0020              orgpt      equ     $-startex+15 and 0fff0h
34 0230                           org     startex+orgpt ;Выровнять
35       ;
36       ; Определение программы защищенного режима. Опять это можно
37       ; расширить применительно к конкретным требованиям.
38       ;
39 = 0230              goback     equ     $
40       ;
41       ; При входе сюда указывает сегмент программы. Значение в BX является
42       ; селектором дескриптора данных и указывает сюда. Значение в CX
43       ; является селектором сегмента стека и тоже указывает сюда.
44       ;
45       ; После возврата из этой программы нам нужны исходные значения
46       ; DS, SS и SP. Если мы установим флаг нуля, программа при
47       ; завершении выполняет дамп.
48
49 0230  8C D6                      mov     SI,SS       ;Сохранить SS монитора
50 0232  8B FC                    mov     DI,SP       ;и SP тоже
51 0234  BA 0200                  mov     DX,200h     ;Установить SP для себя
52 0237  8E D1                    mov     SS,CX       ;Установить наш переданный
53 0239  8B E2                    mov     SP,DX       ;SS и наш новый SP
54 023B  60                       pusha               ;Сохранить все регистры
55 023C  1E                       push    DS          ;Теперь сохраним DS и уста-
56 023D  8E DB                    mov     DS,BX       ;новим наше значение DS
57 023F  90                       nop
58 0240  90                       nop
59       ;
60       ; Выполняйте здесь все, что угодно.
61       ;
62 0241  66                       db      66h         ;Замена размера
63 0242  60                       pusha               ;Сохранить расширенные
64 0243  66                       db      66h         ;регистры
65 0244  9C                       pushf               ;и EFLAGS
66 0245  E8 0257 R                call    instspec    ;На выполнение команды
67 0248  66                       db      66h
68 0249  9D                       popf
69 024A  66                       db      66h
```

```
 70 024B  61                        popa                ;Наш стек возвращен
 71
 72     ;
 73     ; Действия, обратные входным строкам, для восстановления требуемой
 74     ; для нашего возврата среды.
 75     ;
 76 024C  1F                        pop     DS          ;Восстановить DS
 77 024D  61                        popa                ;Получить назад регистры
 78 024E  8E D6                     mov     SS,SI       ;Восстановить SS
 79 0250  8B E7                     mov     SP,DI       ;и SP
 80     ;
 81     ; Установите (сбросьте) флаг нуля для получения (подавления) дампа.
 82     ;
 83 0252  B0 01                     mov     AL,1        ;Установить нужное значение
 84 0254  0A C0                     or      AL,AL       ;в AL и коды условий
 85 0256  CB                        ret
 86     ;
 87     ; Предусмотрите программу для выдачи команды.
 88     ;
 89 0257                 instspec   PROC
 90 0257  90                        nop
 91 0258  90                        nop
 92 0259  90                        nop
 93 025A  90                        nop
 94 025B  90                        nop
 95 025C  90                        nop
 96 025D  90                        nop
 97 025E  90                        nop
 98 025F  90                        nop
 99 0260  90                        nop
100 0261  90                        nop
101 0262  90                        nop
102 0263  C3                        ret
103                      instspec   ENDP
104 0264  0100[                     dw      256 dup(0) ;Обеспечить место для стека
105           0000                                     ;и прочего
106                      ]
107
108                      begin      ENDP
109 0464                 code       ENDS
110                                 END     begin

Segments and groups:   Сегменты и группы:

            N a m e   SizeAlignCombine Class  Имя РазмерВыравнивКомбинир Класс
```

```
CODE..................... 0464PARANONE'CODE'

Symbols:                 Символы:

            N a m e   TypeValueAttr          Имя ТипЗначениеАтрибуты

BEGIN.................... F PROC0100CODELength = 0364

GOBACK................... NEAR 0230CODE

INSTSPEC................. N PROC0257CODELength = 000D

ORGPT.................... Number0020

STARTEX.................. L NEAR0210CODE
STKWORK.................. NEAR 0210CODE

    104 Source  Lines    104 строки исходного текста
    104 Total   Lines    104 строки всего
     28 Symbols          28 символов

  49746 Bytes symbol space free     49746 байтов свободного пространства

      0 Warning Errors   0 предупреждений
      0 Severe  Errors   0 серьезных ошибок

Symbol Cross-Reference (# is definition) Таблица перекрестных ссылок (#-определение)

BEGIN.....................   13     13#    15     22     110

CODE......................   10     10#    10     11     11     11     109

GOBACK....................   23     39     39#

INSTSPEC..................   66     89     89#

ORGPT.....................   33     33#    34

STARTEX...................   14     21     21#    33     34
STKWORK...................   20     20#    22

  7 Symbols              7 символов
```

Пробные прогоны

Загрузите вашу систему. Получив запрос, вставьте дискету в дисковод. На нашей системе имеется лишь один дисковод гибких дисков; в то же время программа записана на жестком диске (дисковод С), поэтому все запросы в приведенном ниже диалоге будут иметь вид «С >».

Системные запросы напечатаны прямым шрифтом, ваши ответы – наклонным.

Прогон 1 – запуск программы МП 80386.

```
C > VIRT86L
Virtual 8086 mode established! Enter G(go) or D – G [1)]
C >
```

Примечание. Теперь вы находитесь в режиме виртуального МП 8086. Система MS-DOS работает под контролем программы VIRT86L на МП 80386. Вы можете вводить любые команды MS-DOS. Для выхода из этого режима вы должны заново загрузить систему, для чего обычно нажимаются клавиши CNTL-ALT-DEL.

Прогон 2 – в этом примере ответьте на запросы «*D*» и получите дамп.

```
C > VIRT86L
Virtual 8086 mode is istablished! Enter G(go) or D – D
```

Примечание. Полученный вами дамп имеет вид секций, состоящих из шестнадцатеричных чисел, разделенных пустыми строками. Перед выводом дампа проверьте адреса портов вашего печатающего устройства. Секции дампа:

1) стек, используемый обработчиком прерывания INT 00;
2) обычный стек программы;
3) регистры;
4) сегмент управления дампом;
5) таблица глобальных дескрипторов и элементы таблицы дескрипторов прерываний;
6) TSS обработчика прерывания INT 00 задачи V86;
7) стек сегмента состояния задачи (TSS) из п. 6;
8) TSS задачи, выключенной при входе в режим V86;
9) стек TSS из п. 6;
10) TSS режима V86;
11) стек TSS из п. 10;
12) стек, используемый задачей, инициировавшей дамп.

Прогон 3 – если вы хотите войти в VIRT86L и запустить программу DOINST в режиме V86.

```
C > VIRT86L
Virtual 8086 mode is established! Enter G(go) or D – G
C > DOINST
```

Примечание. Этот прогон выдает дамп, состоящий из следующих секций:

1) регистры;
2) сегмент управления дампом;
3) таблица глобальных дескрипторов и элементы таблицы дескрипторов прерываний;
4) сегмент управления дампом;

[1)] Режим виртуального МП 8086 установлен! Вводите *G* (пуск) или *D*. – *Прим. перев.*

5) таблица глобальных дескрипторов и элементы таблицы дескрипторов прерываний;

6) TSS обработчика прерывания INT 00 задачи V86;

7) стек TSS из п. 6;

8) TSS задачи, выключенной при входе в режим V86;

9) стек TSS из п. 8;

10) TSS режима V86;

11) стек TSS из п. 10;

12) стек, используемый задачей, инициировавшей дамп.

Если вы наберете «DOINTS» в ответ на «C» в то время, как управление не принадлежит программе VIRT86L, то получите ошибку «переполнение при делении».

Глоссарий [1)]

Абсолютный адрес (Absolute address)
Полностью определенный адрес, соответствующий номеру ячейки памяти.
Адресное пространство (Address space)
Пространство памяти, доступное программе.
Аккумулятор (Accumulator)
Один или несколько регистров для хранения промежуточных результатов и операндов в арифметико-логическом устройстве.
Арбитр прерываний (Priority interrupt control unit)
Устройство, анализирующее приоритеты одновременно действующих запросов на прерывание, и отбирающее наиболее приоритетный запрос.
Архитектура (Architecture)
Упорядоченная организация подсистем, удовлетворяющая задачам, поставленным перед системой в целом.
Асинхронные операции (Asynchronous operations)
Режим операций взаимодействующих систем, при котором каждая система функционирует независимо от внутренних временных ограничений всех остальных систем.
Ассембер (Assembler)
Программа, преобразующая язык ассемблера в двоичный машинный язык.
Арифметико-логическое устройство (Arithmetic logic unit)
Устройство, выполняющее все арифметические и логические операции в микропроцессоре.
Базовый адрес (Base address)
Физический адрес начала сегмента.
Бит маски (Mask bit)
Бит, используемый для того, чтобы скрыть или заблокировать какое-либо условие.
Битовая карта разрешения ввода-вывода (I/O permission bit map)
Механизм, позволяющий микропроцессору 80386 селективно возбуждать исключения при обращении программы к конкретным адресам устройств ввода-выво-

---

[1)] Составленный авторами глоссарий несомненно окажет помощь читателю при изучении книги, однако содержащиеся в нем толкования терминов нельзя считать точными. Некоторые статьи (*Подтверждение связи, Программируемый параллельный интерфейс, Селектор* и др.) практически не несут в себе никакой информации, другие (*Коды, исправляющие ошибки, Реентерабельная программа, Передача памяти, Сдвиг, Сопряжение*) настолько неточны, что могут ввести читателя в заблуждение относительно истинного смысла термина. Значительно более взвешенные и точные определения можно найти в книге: Борковский А. Б. Англо-русский словарь по программированию и информатике (с толкованиями).– М.: Русский язык, 1987.– *Прим. перев.*

да. Битовая карта разрешения ввода-вывода размещается в сегменте состояния задачи (TSS). Карта представляет собой битовый вектор и ее размеры и расположение в TSS могут изменяться. МП 80386 локализует карту с помощью поля базы карты ввода-вывода, которое размещается в определенном месте TSS.

Ближний указатель (Near pointer)

В МП 80386–32-разрядный логический адрес. Ближний указатель представляет собой относительный адрес в сегменте.

Блокирование (Lock)

В многопроцессорной системе это сигнал от одного из процессоров, запрещающий другим процессорам доступ к памяти. Процессор имеет исключительный доступ к памяти до тех пор, пока он поддерживает сигнал LOCK.

Блок памяти (Memory unit)

Часть ЭВМ, в которой хранится информация – команды и операнды.

Буфер (Buffer)

Запоминающие элементы (регистры и ячейки памяти) для временного хранения информации до момента использования ее системой, которой эта информация предназначена, например периферийным устройством.

Буфер ассоциативной трансляции (Translation lookaside buffer (TLB))

TLB представляет собой четырехвходовую ассоциативную кэш-память и используется для трансляции линейных адресов в физические. Алгоритм поиска в TLB может быть изменен.

Вентиль (Gate)

Простейшая логическая схема, выходное напряжение которой принимает высокое или низкое значение в зависимости от состояния входов и типа вентиля. В МП 80386 используются вентили четырех типов: вызова, ловушки, прерывания и задачи.

Векторное прерывание (Vector interrupt)

Прерывание определяет адрес первой команды программы обработки этого прерывания.

Виртуальный адрес (Virtual address)

Адрес, состоящий из селектора и относительного адреса (смещения) в режиме защищенного адреса.

Виртуальная память (Virtual memory)

Способ выполнения программ, размер которых превышает размер доступной физической памяти. Части программы хранятся на диске и пересылаются в память только по мере надобности. Пересылка осуществляется автоматически операционной системой и невидима для программы.

Виртуальный режим – см. режим защищенного виртуального адреса.

Время выполнения (Execution time)

Время, требуемое микропроцессору для выполнения машинной команды. Время выполнения для различных команд неодинаково.

Время доступа (Access time)

Интервал времени между требованием на получение данных из блока памяти и их реальным получением процессором.

Выборка (Fetch)

Получение информации из блока памяти.

Дальний указатель (Far pointer)

В МП 80386 48-разрядный логический адрес, состоящий из двух компонентов: 16-разрядного селектора сегмента и 32-разрядного относительного адреса.

Дамп (Dump)

Вывод содержимого памяти системы на внешнюю среду, например магнитную ленту.

Двоично-десятичное кодирование (Binary coded decimal)

Алгоритм кодирования, предусматривающий представление каждой десятичной цифры от 0 до 9 эквивалентным 4-разрядным двоичным числом.

Двунаправленный (Bi-directional)

Термин, используемый для описания коммуникационных линий, которые могут передавать сигналы в обоих направлениях.

Дескриптор (Descriptor)

8-разрядная величина, определяющая защищенный объект

Дешифратор адреса (Address decoder)

Схема, преобразующая адрес МП 80386 в сигналы выбора микросхем, которые посылаются в логическую схему управления магистралью.

Дешифрация адреса (Address decoding)

Сжатие адреса на магистрали в один сигнал, который либо выбирает, либо выключает конкретное устройство и задает число циклов ожидания для этого устройства.

Диспетчеризация задач (Task dispatching)

Выбор и запуск задачи.

Диспетчер памяти (Memory management unit)

Устройство, выполняющее отображение пространства адресов задачи на доступную память.

Задача (Task)

Определенная логически единая функция, выполняемая ЭВМ. Задачи называют также процессами.

Задача прерывания (Interrupt task)

Задача, активизируемая в ответ на прерывание.

Задержка (Delay)

Замедление распространения сигнала, либо умышленное с целью предотвращения изменения состояния входов одновременно с тактовыми сигналами, либо ненамеренное и вызванное конечными временами нарастания и спада сигнала в транзисторной схеме.

Задержка прерывания (Interrupt latency)

Время, прошедшее до обслуживания запроса на прерывание.

Замаскированное исключение (Masked exception)

Исключение, результатом которого является не прерывание, а выдача кода исключения.

Запрашиваемый уровень привилегий (Requested privilege level (RPL))

Поле в селекторе, которое указывает степень «доверия», или привилегий, данных программе селектором. RPL определяется двумя младшими разрядами селектора.

Захват циклов магистрали (Cycle stealing)

Контроллер или мультиплексор системы сопряжения может захватить («украсть») цикл памяти для передачи слова из внешнего запоминающего устройства в основную память или наоборот.

Зашитая логика (Hard-wired logic)

Логическая структура, использующая в качестве логических элементов непрограммируемые микросхемы.

Индексный регистр (Index register)

Один из 16-разрядных регистров SI или DI. Как правило, индексный регистр содержит относительный адрес в текущем сегменте данных или дополнительном сегменте.

Инициализация (Initialization)

Операция установки или сброса определенных регистров и счетчиков в процессе подготовки к выполнению задачи.

Инициализация системы (System initialization)

Последовательность операций, выполняемых во взаимодействии аппаратурой и программами после первого включения вычислительной системы или в результате использования ключа установки исходного состояния.

Интервальный таймер (Interval timer)

Компонент, вызывающий прерывание процессора по истечении определенного интервала времени. Программа может установить значение интервала, а также режим однократных или повторных прерываний.

Интерфейс – см. сопряжение

Исключение (Exception)

Условие, возникающее, если команда нарушает правила нормального выполнения программы.

Кадр стека (Stack frame)

Место для хранения значений местных переменных процедуры. Соответствует конкретному вызову процедуры.

Кадр страницы (Page frame)

Блок смежных адресов физической памяти объемом 4 кбайт. Адрес кадра страницы определяет начальный физический адрес страницы; младшие 12 бит адреса всегда равны нулю.

Квантование времени (Time-slicing)

Прием, позволяющий нескольким задачам совместно использовать ЭВМ. ЭВМ переключается от одной задачи на другую, не занимаясь любой задачей дольше небольшого и определенного интервала времени.

Код (Code)

Способ представления информации в цифровой форме путем назначения каждому элементу информации определенного сочетания битов.

Код исключения (Exception value)

Специальный код, образуемый командой, если она вызывает замаскированное исключение.

Код операции (КОП) (Operation code, opcode)

Часть команды, определяющая требуемую операцию в противоположность объектам, над которыми выполняется эта операция.

Код ошибки (Error value)

См. код исключения

Коды, исправляющие ошибки (Error correctig codes (ECC))

Анализ бита четности, позволяющий обнаружить неправильные биты в слове кода.

Команда (Instruction)

Единичный шаг в программе, одна строка программного кода.

Компаратор (Comparator)

Компонент, сравнивающий два двоичных числа.

Компилятор (Compiler)

Программа, преобразующая (транслирующая) исходный текст на языке высокого уровня в машинный язык.

Контроллер (Controller)

Элемент или группа элементов вычислительной системы, задающие последовательность операций и посылающие другим узлам ЭВМ сигналы, требующие выполнения этих операций.

Контроллер прерываний (Interrupt controller)

Компонент, сортирующий множественные запросы на прерывания по приоритетам.

Косвенная адресация (Inderect adressing)

Способ локализации ячейки памяти, при котором сначала из какой-то другой ячейки памяти или из регистра извлекается требуемый адрес.

Коэффициент разветвления по входу (Fan-in)

Полное число входов конкретного вентиля или функции.

Коэффициент разветвления по выходу (Fan-out)

Полное число единиц нагрузки, подключаемых к конкретному вентилю или функции.

Кэш-память (Cache)

Быстродействующая память буферного типа, заполняемая с умеренной скоростью из основной памяти. Кэш-память по сравнению с другими видами запоминающих устройств вычислительной системы имеет минимальное время доступа и используется для хранения только тех данных, которые могут понадобиться процессору в ближайшем будущем.

Кэш-память дескрипторов (Descriptor cache)

См. теневой регистр.

Линейное пространство адресов (Linear adress space)

Пространство адресов от байта 0 до максимального физического адреса, к которому процессор имеет доступ. В МП 80386 линейное пространство адресов простирается до 4 Гбайт.

Логический адрес (Logic address)

В системе МП 80386 логический адрес состоит из селектора и относительного адреса. Селектор указывает на дескриптор некоторого сегмента, который содержит линейный базовый адрес сегмента. Относительный адрес указывает, на каком расстоянии от начала сегмента находится требуемый байт.

Ложный сигнал (Glitch)

Ложный цифровой сигнал.

Локальное пространство адресов (Local address space)

Набор сегментов, доступных задаче посредством таблицы локальных дескрипторов.

Локальность программы (Program locality)

Тенденция программы обращаться при очередной операции к ячейкам памяти, расположенным поблизости от адресуемых в предыдущей операции. Именно локальность программы объясняет высокую эффективность кэш-памяти.

Магистраль, шина (Bus)

Набор линий, передающих сигналы.

Маскирование (Masking)

Способ анализа лишь определенных битов в слове. Обычно реализуется путем выполнения операции логического И над словом и маской, содержащей 1 в требуемых разрядах.

Машинный язык (Machine language)

Формат представления команд в виде двоичных чисел, непосредственно воспринимаемых процессором.

Метки (Labels)

Идентификаторы или номера, используемые для описания позиций триггеров или логических вентилей.

Микропрограмма (Microprogram)

Последовательность микрокоманд, задающих самые элементарные операции процессора.

Микросхема (Chip)

Небольшая пластина из полупроводникового материала, содержащая миниатюризованную электронную схему; интегральная схема.

Микропроцессор (Microprocessor)

Центральный процессор (ЦП), выполненный на микросхемах с помощью технологии сверхбольших интегральных схем.

Мнемоника (Mnemonic)

Символическое имя, используемое, в частности, для обозначения кодов операций.

Многозадачность (Multitasking)

Создание и одновременное использование на ЭВМ множества задач.

Мультиплексирование (Multiplexing)

Распределение и совместное использование несколькими пользователями общих ресурсов ЭВМ.

Набор команд – см. система команд.

Начальная загрузка (Bootstrap)

Способ запуска вычислительной системы с помощью небольшой программы, хранящейся в ее памяти.

Немаскируемое прерывание (Nonmaskable interrupt (NMI))

Сигнал процессору от внешнего устройства, говорящий о возникновении ситуации, требующей обслуживания.

Непосредственный операнд (Immediate operand)

Константа, содержащаяся в команде и используемая в качестве операнда.

Не число (Not-a-number (NAN))

Код исключения, возбуждаемого арифметическим сопроцессором в ответ на замаскированное исключение по неправильной операции.

Нуль-селектор (Null selector)

Селектор, в котором все биты равны 0.

Обработчик прерывания (Interrupt handler)

Процедура или задача, вызываемая в ответ на прерывание.

Объектная программа (Object program)

Программа на двоичном машинном языке, результат работы ассемблера.

Операнд (Operand)

Данное, над которым процессор выполняет арифметическую или логическую операцию.

Операция реального времени (Real time operation)

Выполнение процессором программы параллельно с развитием внешнего процесса, с которым он взаимодействует.

Опрос (Polling)

Методика идентификации источника сигнала прерывания, заключающаяся в периодическом опросе каждого внешнего устройства.

Останов (Shutdown)

Состояние бездействия, из которого процессор может быть выведен установкой исходного состояния (Reset) или немаскируемым прерыванием.

Относительный адрес (Offset)

Число, характеризующее позицию байта внутри сегмента.

Отказ страницы (Page fault)

Процессор 80386 возбуждает прерывание 14 при обнаружении одного из следующих условий: 1) текущая процедура не имеет необходимых привилегий для доступа к указанной странице; 2) биты присутствия в необходимых для трансляции адреса элементах таблицы страниц или каталога страниц равны нулю. Другими словами, требуемая страница не загружена из вспомогательной памяти.

Отсутствие (Not present)

В системе виртуальной памяти этот термин описывает сегмент, находящийся на диске, а не в основной памяти.

Очистка – см. сброс

Память (Memory)

Среда для хранения программ и данных.

Параметр (Parameter)

Элемент данных, передаваемых процедуре из вызывающей программы.

Передача памяти (Memory passing)

Способ межзадачной коммуникации, по которому данные передаются из одной задачи в другую.

Передача управления (Transfer of control)

Указание на необходимость выполнения последовательности команд, начинающихся в какой-то новой ячейке, вместо выполнения команды, непосредственно следующей за только что выполненной. Процессор 80386 осуществляет передачу управления с помощью исключений, прерываний и команд CALL, JMP, INT, IRET и RET. «Ближняя» передача управления приводит к переходу на точку внутри текущего сегмента программы. «Дальняя» передача осуществляет переход в другой сегмент.

Переключение задачи (Task switch)

Смена одной задачи на другую.

Переход (Jump)

Команда, вызывающая передачу управления из одной части программы в другую.

Перечислимый (обычный) тип (Ordinal)

Беззнаковое двоичное число, содержащееся в байте, 16-разрядном слове или 32-разрядном двойном слове.

Периферийное устройство (Peripheral device)

Электронное устройство, подключенное к процессору; периферийные устройства взаимодействуют с процессором посредством команд ввода-вывода, прерываний, или ввода-вывода, отображенного на память.

Повторно выполняемая команда (Restartable instruction)

Команда является повторно выполняемой по отношению к исключению, если после ликвидации причины исключения, возбужденного при выполнении этой команды, программу можно правильно продолжить, выполнив команду повторно.

Подтверждение связи (Handshaking)

Режим передачи данных между асинхронными устройствами.

Поле битов (Bit field)

Непрерывная последовательность битов. В МП 80386 поле битов может начинаться в любом разряде байта и иметь длину до 32 бит.

Положительная логика (Positive logic)

Логические операции, в которых более положительное напряжение соответствует состоянию логической 1.

Последним вошел – первым вышел (Last-in-first-out (LIFO))

Алгоритм обработки элементов данных, в котором элемент, помещенный в стек или блок последним, обрабатывается в первую очередь.

Последовательные данные (Serial data)

Данные, образующие последовательность битов и поступающие бит за битом.

Пошаговое выполнение (Single stepping)

Выполнение программы команда за командой с паузой после каждой команды. Этим средством программисты обычно пользуются для исследования результата выполнения программной строки.

Преобразование адреса – см. трансляция адреса.

Прерывание (Interrupt)

Приостановка выполения процессором текущей программы с целью обслуживания одного или нескольких периферийных устройств. Также вынужденный вызов

процедуры, не содержащийся в явном виде в программе и активизируемый исключением, сигналом от внешнего по отношению к процессору устройства или с помощью специальной команды прерывания.

Префикс (Prefix)

Байт, предшествующий коду операции команды. Префикс указывает на необходимость повторного выполнения команды, или ее блокирования, или использования другого сегмента.

Приоритет (Priotity)

Характеризует отношения очередности применительно к одновременным процессам или событиям.

Программируемая логическая матрица (ПЛМ) (Programmable array logic (PAL))

Интегральная схема, допускающая программирование для задания ей требуемых логических функций. ПЛМ состоит из программируемой матрицы элементов И, выходы которых подключены к фиксированной матрице элементов ИЛИ.

Программируемый параллельный интерфейс (Programmable parallel input/output (PIO))

Устройство сопряжения

Программное обеспечение (Software)

Комплекс программ вместе с относящейся к ним документацией.

Продолжительность цикла (Cycle time)

Фиксированный временной интервал.

Процедура (Procedure)

Часть программы, выполняющая конкретные функции. Процедура может быть вызвана из нескольких точек программы.

Процедура обработки прерывания (Interrupt procedure)

Процедура, вызываемая в результате прерывания.

Процесс (Process)

См. задача

Прямая адресация (Direct addressing)

Определение ячейки памяти с помощью адреса, включенного в команду.

Прямой доступ к памяти (ПДП) (Direct memory access (DMA))

Способ пересылки данных в память или из нее без нарушения хода программы, выполняемой процессором.

Разброс (Scatter read)

Чтение блока данных в несмежные ячейки памяти.

Разгрузка (Unload)

Извлечение больших массивов информации.

Рандеву (Rendezvous)

Способ синхронизации двух задач. Задача, первой дошедшая до точки синхронизации, ожидает вторую.

Распределитель прерываний (Iinterrupt distributor)

Системный обработчик прерываний, передающий управление той или иной пользовательской программе обработки прерываний в зависимости от того, какая именно оказалась прервана.

Расслоение памяти (Interleaving)

Метод повышения производительности памяти вычислительной системы, основанный на назначении последовательных ячеек памяти разным банкам. Распределение ячеек памяти по имеющимся банкам осуществляется циклически. В результате при последовательном обращении к ячейкам памяти в полной мере реализуются преимущества параллелизма, обеспечиваемого множеством банков.

Расширение вверх (Expand up)

Свойство сегмента, определяющее характер проверки процессором относитель-

ного адреса при обращении к этому сегменту. Относительный адрес не должен превышать границу сегмента. Все сегменты, кроме сегментов стека, обладают этим свойством.

Расширение вниз (Expand down)

Свойство сегмента, определяющее характер проверки процессором относительного адреса при обращении к этому сегменту. Относительный адрес должен превышать границу сегмента. Этим свойством обладают сегменты стека.

Реальный адрес (Real address)

Адрес, состоящий из селектора и относительного адреса. Используется в режиме реального адреса.

Реальный режим – см. режим реального адреса.

Регенерация (Refresh)

Процесс считывания и перезаписи назад данных из ячеек памяти динамического запоминающего устройства.

Регистр (Register)

Быстродействующая память для временного хранения информации обычно входит в состав процессора.

Регистр задачи (Task register (TR))

Этот регистр является указателем информации, требуемой процессору для определения текущей задачи.

Регистр команд (Instruction register)

Регистр в микропроцессоре, содержащий текущую команду.

Регистр общего назначения (General register)

Один из 16-разрядных регистров: AX, BX, CX или DX

Регистр сегмента (Segment register)

Один из регистров: CS, DS, ES или SS.

Регистр таблицы глобальных дескрипторов (Global descriptor table register (GDTR))

Регистр процессора 80386, содержащий 32-разрядный линейный базовый адрес и 16-разрядную границу таблицы глобальных дескрипторов.

Регистр-указатель (Pointer register)

Один из регистров BP или BX. Обычно в регистре-указателе содержится относительный адрес в текущем сегменте стека.

Регистры отладки (Debug registers)

В МП 80386 имеются 6 программно-доступных регистров: DR0 – DR3, а также DR6 и DR5. Регистры DR0 – DR3 определяют четыре линейные точки останова. DR4 и DR5 зарезервированы фирмой Intel для будущего использования. В DR6 записывается текущее состояние точек останова, а DR7 используется для установки точек останова.

Регистры тестирования (Test registers)

Регистры TR6 и TR7 используются для тестирования ассоциативной памяти в буфере ассоциативного преобразования.

Регистры управления (Control registers)

В МП 80386 эти регистры используются для хранения данных глобального характера о состоянии машины. Регистрам присвоены обозначения CR0, CR1, CR2 и CR3. Младшие 16 разрядов CR0 называются словом состояния машины для совместимости с МП 80286.

Редактор текста (Editor)

Программа, используемая для обработки в интерактивном режиме исходного текста программы.

Реентерабельная программа (Re-entrant code)

Программа, которая может быть прервана во время выполнения.

Режим адресации (Addressing mode)

Описание операнда в команде.

Режим защиты виртуального адреса (Protected virtual address mode)

Режим работы процессора, в котором обеспечиваются многозадачность, расширенные средства защиты и виртуальная память.

Режим реального адреса (Real address mode)

Режим работы процессора, в котором имитируются действия микросхемы «нижнего уровня» из того же семейства процессоров. Например, МП 80386 может имитировать работу МП 8086.

Сброс (Clear/Reset)

Процесс записи в рассматриваемые данные двоичных нулей.

Свопинг (Swapping in/out)

Пересылка сегмента с диска в память (подкачка) или из памяти на диск (откачка).

Сдвиг (Shift)

Процесс перемещения данных из одного места на другое.

Сегментированная память (Segmented memory organization)

Пространство адресов, включающее до 16383 линейных адресных пространств размером до 4 Гбайт каждое. Полное адресное пространство, используемое в программе для МП 80386, может достигать 2^{46} байт (64 Тбайт)

Сегмент состояния задачи (Task state segment (TSS))

Сегмент, в котором хранится содержимое регистров задачи, пока процессор выполняет другую задачу.

Селектор (Selector)

Число, характеризующее сегмент.

Семафор (Semaphore)

Переменная, используемая процессорами или задачами совместно с целью взаимной синхронизации. Нуль означает разрешение выполнения, не нуль требование ожидания.

Сигнал управления магистралью (Bus command)

Сигнал, определяющий выполнение на магистрали конкретной операции.

Сигнатура (Signature)

Конкретный отличительный сигнал данной схемы.

Символ (Character)

Собирательный термин, обозначающий буквы, цифры, специальные знаки, знаки пунктуации, математические операторы, а также соответствующие этим знакам двоичные коды.

Синхронная операция (Synchronous operation)

Переключение схемы с помощью тактового генератора. Все цепи в схеме переключаются одновременно. Все действия выполняются синхронно с тактовыми импульсами.

Система команд (Instruction set)

Набор всех команд машинного языка, выполняемых данным процессором.

Системный сброс (Reset)

Сигнал, по которому аппаратура ЭВМ приводится в исходное состояние. Является элементом инициализации системы.

Системный тактовый сигнал (System clock)

Основной синхронизирующий сигнал в системе.

Слово (Word)

Машинно-зависимая единица памяти, обычно соответствующая ширине шины данных и длине внутренних регистров.

Смещение (Displacement)

16-разрядная величина, определяемая в команде и используемая для вычисления относительного адреса.

Согласованный сегмент (Conforming segment)

Сегмент, каждая процедура которого выполняется на уровне привилегий вызывающей процедуры.

Сопроцессор (Coprocessor)

Вспомогательный процессор, работающий согласованно с ЦП и предоставляющий архитектурные возможности, которые при ограничениях сегодняшней технологии невозможно реализовать другими способами. Сопроцессоры обеспечивают аппаратное выполнение функций, которые иначе пришлось бы реализовать программно. Кроме того, сопроцессоры расширяют систему команд основного процессора.

Сопряжение (Interface)

Общая граница двух систем, через которую системы обмениваются информацией.

Состояние ожидания (Wait state)

Дополнительный цикл процессора, добавляемый к циклу магистрали для получения ответа от подключенных к магистрали медленных устройств.

Сплошная память (Flat memory)

Адресное пространство, состоящее (в МП 80386) из единственного массива размером до 4 Гбат. МП 80386 отображает сплошное пространство размером 4 Гбайт на физическое пространство адресов с помощью механизма трансляции адресов.

Стек (Stack)

Буфер, для доступа к содержимому которого обычно используется алгоритм «последним вошел – первым вышел». См. также соответствующий термин.

Страница (Page)

Набор последовательных байтов. Страницы начинаются на границе 4 кбайт. Страничная организация памяти приводит к разбиению программ на множество страниц одного размера, не имеющих непосредственной связи с логической структурой программы.

Строб (Strobe)

Сигнал, подаваемый на вход регистра-фиксатора и обеспечивающий запоминание текущего входного состояния этого регистра. Введенное в регистр значение будет действовать на его выходе до прихода следующего сигнала строба, даже если состояние входа регистра изменилось.

Строка (String)

Слитная последовательность байтов, слов или двойных слов. Строка может содержать от 0 байт до $2^{32} - 1$ байт, или 4 Гбайт.

Строка битов (Bit string)

Слитная последовательность битов. Строка битов в МП 80386 может начинаться с любого разряда байта и иметь размер до 32 бит.

Сумматор (Adder)

Переключательная схема, комбинирующая входные биты так, чтобы образовать сумму и перенос этих битов.

Схема на дискретных компонентах (Descrete circuit)

Электронная схема, состоящая из транзисторов, диодов, регистров, конденсаторов и других компонентов, выпускаемых по отдельности и собираемых затем на общей плате.

Счетчик (Counter)

Устройство, изменяющее свое состояние в заданной последовательности при получении соответствующих входных сигналов.

Таблица глобальных дескрипторов (Global descriptor table (GDT))

Таблица, находящаяся в памяти и содержащая дескрипторы сегментов, используемых совместно всеми задачами.

Таблица дескрипторов прерываний (Interrupt descriptor table (IDT))

Таблица, находящаяся в памяти и содержащая вентили соответствующих обработчиков прерываний. Индексы вентилей в таблице соответствуют номерам прерываний.

Таблица дескрипторов сегментов (Segment descriptor table)

Массив 8-битовых ячеек памяти, содержащих дескрипторы. Таблица дескрипторов МП 80386 может включать до 8192 дескрипторов.

Таблица локальных дескрипторов (Local descriptor table (LDT))

Таблица, находящаяся в памяти и содержащая дескрипторы сегментов, локальных для данной задачи.

Таблица страниц (Page table)

Массив 32-разрядных описателей страниц. Таблица занимает 4 кбайт, содержит до 1К 32-разрядных элементов и сама представляет страницу.

Тактовый генератор (Clock)

Генератор периодических электрических импульсов, синхронизирующих работу электронных переключательных элементов ЭВМ. Частота генератора ограничивается временем срабатывания интегральных схем, использованных в системе. При прочих равных условиях чем выше частота тактового генератора, тем больше операций может выполнить ЭВМ за то же время.

Теневой регистр (Shadow register)

Скрытый регистр, связанный с видимым регистром. Теневой регистр содержит дескриптор, соответствующий селектору, который хранится в видимом регистре.

Текущий уровень привилегий (Current privilege level (CPL))

Уровень привилегий выполняемой программы. CPL можно определить, анализируя два младших бита регистра программного сегмента (CS).

Тестовая программа (Benchmark)

Программа, служащая для исследований и оценки производительности различных систем.

Трансляция адреса (Address translation)

Преобразование селектора и относительного адреса в физический адрес.

Указатель (Pointer)

Адрес, используемый программой и состоящий из селектора и относительного адреса. В реальном режиме указатели представляют собой реальные адреса, в виртуальном режиме – виртуальные адреса.

Указатель команд (Instruction pointer (IP))

Регистр, содержащий относительный адрес выполняемой команды. Селектор сегмента, в который входит эта команда, записан в регистре CS. Указатель команд (EIP) МП 80386 представляет собой 32-разрядный регистр, содержащий относительный адрес команды, следующей за выполняемой. Младшие 16 разрядов регистра EIP образуют регистр IP.

Указатель стека (Stack pointer)

Регистр, в котором находится адрес вершины стека (адрес последнего элемента, записанного в стек).

Упакованное двоично-десятичное число (Packed BCD)

Упакованное в байт представление двух десятичных цифр, каждое в диапазоне от 0 до 9. В каждом полубайте хранится по одной цифре, причем в старшем полубайте находится старшая цифра.

Уровень привилегий (Privilege level)

Число в обусловленном заранее диапазоне, указывающее степень защиты, или степень привилегий.

Уровень привилегий дескриптора (Descriptor privilege level (DPL))

Поле в дескрипторе, определяющее степень его защиты. DLP является наименее привилегированным уровнем, на котором задача получает доступ к конкретному дескриптору и связанному с ним сегменту.

Физический адрес (Physical address)

Число, передаваемое аппаратуре управления памятью с целью обращения к конкретной ячейке памяти.

Фиксатор (Latch)

Компонент, запоминающий по команде состояние своего входа. Также петля обратной связи в симметричной цифровой схеме (триггере), обеспечивающая сохранение состояния схемы.

Фиксатор адреса (Address latch)

Схема, сохраняющая свое содержимое в течение определенного времени. Фиксатор используется для удерживания адреса устройств ввода-вывода на время цикла магистрали.

Флаг (Flag bit)

Бит, указывающий одно из двух взаимно исключающих условий или состояний.

Центральный процессор (ЦП) (Central processing unit (CPU))

Состоит из микропроцессора, арифметико-логического устройства, различных регистров, а также управляющих и синхронизирующих схем.

Цикл (Looping)

Многократное выполнение некоторой последовательности команд.

Цикл магистрали (Bus cycle)

Единичная передача информации по магистрали.

Цифровая схема (Digital cicuit)

Полупроводниковая схема, выполняющая функции переключателя.

Четность (Parity)

Указание на четность или нечетность количества двоичных 1 в числе.

Шина – см. магистраль.

Шина данных (Data bus)

Часть магистрали, служащая для пересылки данных между процессором и вспомогательными микросхемами.

Шина управления (Control bus)

Часть магистрали, служащая для пересылки сигналов управления и состояния между процессором и вспомогательными микросхемами.

Шинный формирователь (Bus driver)

Источник электрического тока, используемый для поддержания или передачи сигналов по магистрали.

Эмулировать (Emulate)

Создавать с помощью микропрограммирования на процессоре команды машинного языка другого процессора.

Эффективный уровень привилегий (Effective privilege level (EPL))

Наименее привилегированный из двух уровней: запрашиваемого и дескрипторного.

Язык ассемблера (Assembly language)

Символические обозначения машинных команд.

Литература

Ashborn J., Intel Microprocessors: A Tradition of Innovation, *Solutions*, Nov./Dec. 1985, pp. 8–9.
Ashborn J., MULTIBUS II Adds Punch to 80386-Based Designs, *Solutions*, Nov./Dec. 1985, p. 10.
Augarten S., *Bit by Bit*, Ticknor & Fields, New York, NY, 1984.
Augarten S., *State of The Art*, Ticknor & Fields, New York, NY, 1983.
Carr J.J., *Microprocesser Interfacing*, TAB Books Inc., Blue Ridge Summit, PA, 1982.
Chorafas D.N., *Microprocessor For Manadgement*, Petrocelli Books, New York, NY, 1984.
Ciarcia S., *Ciarcia's Circuit Cellar*, McGraw-Hill Book Company, New York, NY, 1984.
Gluley J.C., *Computer Interfacing and On-Line Operation*, Grane Russak, New York, NY, 1975.
Coli V.J., Introduction to Programmable Array Logic, *Byte*, Jan. 1987, pp. 207–219.
Crosswy, Caldwell, Perez M., Upward to the 80386, *PC Tech Journal*, Feb. 1987, pp. 51–66
Fawcette J.E., 80386: The Megabyte Manager, *PC World*, Feb. 1986, pp. 238–243.
Frederiksen T.M., *Intuitive IC CMOS Evolution*, National Semiconductor Corporation, Sunnyvale, CA, 1984.
Glorioso R.M., Osorio F.C.C., *Engineering Intelligent Systems*, Digital Press, Bedford, MA, 1980.
Graham N., *Introduction to Computer Science*, West Publishing Co., New York, NY, 1979.
Hayes N.M., Chip Set Geared To Unix Simplifies High-End Designs, *Computer Design*, Jan. 1, 1986, pp. 65–72.
Hindin H.J., 32-bit Parts and Architectures Vie for Attention, *Computer Design*, Jan. 1, 1986, pp. 49–61.
Intel Corporation, *Microprocessor and Peripherals Handbook*, Intel Corporation, 1983.
Intel Corporation, *80386 Hardware Reference Manual*, Intel Corporation, 1986.
Intel Corporation, *80386 Programmer's Reference Manual*, Intel Corporation, 1986.
Lau E.J., *Performance Improvement of Virtual Memory Systems*, UMI Research Press, Ann Abor, MI, 1982.
Lewis P.H., 32-bit Chip: Powerhouse in the Wings, *New York Times*, Aug. 29, 1985.
Markhoff N., Complete Systems Now Possible with 32-bit Chip Sets, *Computer Designs*, Jul. 1, 1985, pp. 77–89.
McGrievy D.J., Pickar K.A., *VLSI Technologies Through the 80s and Beyond*, IEEE Computer Society Press, Silver City, MD, 1982.
Mead C., Conway L., *Introduction to VLSI System*, Addison-Wesley, Reading, MA, 1980.
Middleton R.G., *New Handbook of Troubleshooting Techniques*, Prentice-Hall, Englewood Cliffs, NJ, 1984.
Monds F., *The Business of Electronic Product Development*, Peter Peregrinus Ltd. on behalf of the Institution of Electrical Engineers, London, UK, 1984.
Philips D., Mainframe Tricks Raise Performance of 32-bit Micros, *Computer Designs*, Jul. 1, 1985, pp. 95–103.
Pooch U.W., Chattergy R., *Designing Microcomputer Systems*, Hayden Book Company, Rochelle Park, NJ, 1979.
Rafiquzzaman M., *Microprocessor and Microcomputer Development Systems*, Harper and Row, New York, NY, 1984.
Rant J., Extending the Legacy of Leadership: The 80386 Arrives, *Solutions*, Nov./Dec. 1985, pp. 2–7.
Shires G., 80386 Cache Design, *Solutions*, Nov./Dec. 1985, pp. 12–27.
Short K.L., *Microprocessors and Programmed Logic*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

Theaker C. J., Brookes G. R., *A Practical Course on Operating Systems*, Springer-Verlag, New York, NY, 1983.

Thomplait C., Memory Management Boosts Efficiency of Powerful Micros, *Computer Design*, Jul. 1, 1986, pp. 105–109.

Thompson R., Uberoi A., Processor Offers Code Compatibility, VAX-Like Architecture, *Computer Designe*, Jan. 1, 1986, pp. 76–80.

Tseng V. (ed.), *Microprocessor Development and Development Systems*, McGraw-Hill Book Company, New York, NY, 1982.

Zaks R., *From Chips To Systems*, Sybex, Berkley, CA, 1981.

Предметный указатель

Адрес 10
Адресное поле 117
Адресное пространство 46
Аппаратная прозрачность 58
Аппаратура:
  изготовление и отладка 107
  шаговое построение 108
Арифметико-логическое устройство 340
Арифметические команды 119
Ассоциативное запоминающее устройство 24

База 28
Бит дробности 29
– обращения 29
– сегмента 29
Битовое поле 17
Блок декодирования команд 12
– исполнительный 11
– предвыборки 11
– сопряжения с магистралью 339
– управления сегментами 12
Блокированные циклы 67
Буфер ассоциативной трансляции 24
Буфера 297
Буферизация со сквозной записью 55

Вентиль 9
Виртуальная память 317
Виртуальный процессор 349
Виртуальный режим МП 8086:
  адресация 126
  вентили прерываний и ловушек 125
  ввод-вывод 128
  виртуальная задача 127
  вход и выход 124
  отличия от МП 8086 129
  формирование адреса 127
  чувствительные команды 128
Выход из процесса 35

Граница 28

Дескрипторы 9
– сегментов 28
Динамическое ОЗУ 51
Диспетчер памяти 14

Задача 11
Задержка прерывания 65
Запоминающее устройство с произвольной выборкой 24
Запрашиваемый уровень привилегий 27
Защита 69
– каталогов 84
– страниц 84
Защищенная среда 9
Защищенный режим 15
––, инициализация 138

Индекс 27
Индексный регистр 118
Индикатор таблицы 27
Интегральные схемы 286
Интерфейс 11
– ввода-вывода 68
– с локальной магистралью 61
– с памятью 49
Исключения 32
Исполнительный адрес 117

Кадр страницы 48
Коды ошибок 38
Команды 25
Команды ввода-вывода 69
– логические 121
– десятичной коррекции 189
– загрузки 214
– защиты 122
– операций над строками 122
– пересылки 226
– – данных 120
– переходов 209
– преобразования 178, 188
– прерывания 206
– проверки битов 172
– сброса флагов 179
– сдвигов 121
– сканирования битов 170
– сравнения 183
– управления программой 121
– установки флагов 271
– циклического сдвига 248
Конвейеризация 85
– адресов 85
Контроллер 8259A 93
– ПДП 82258 96
– прерываний 36
– прямого доступа в память 49
Контроллеры 93
– последовательные 8274 93
– прерываний 8259A 93
– прямого доступа в память 82258 96
Косвенная адресация 121
Кэш-память 52
– – полностью ассоциативная 54

Линейное пространство адресов 10
Ловушки 35
Логика управления магистралью 74
Логические адреса 10
Локальность программы 54
Локальность ссылок 54

Маскируемые прерывания 33
Матричное расположение выводов 93
Микрокод 11
Микросхема 4004 334
– 8008 334
– 8080 334
– 8080A 335
– 8085A 336
– 8086 339
– 8087 342
– 80186 343
– 80286 345
– –, выполнение программы 130
– –, защищенный режим 130
– –, ограничения префикса LOCK 131
– –, циклический возврат адреса 131
– 80386 347
– –, архитектура реального и защищенного режимов 14
– –, введение 8
– –, виртуальная среда МП 8086 124
– –, диспетчер памяти 14
– –, защищенный режим МП 80286 130
– –, интерфейс ввода-вывода 68
– –, интерфейс с локальной магистралью 61
– – и MULTIBUS I/II 358
– –, исключения 133
– –, конвейеризация 85
– –, контроллеры 93
– –, кэш-память 52
– –, многозадачность 87
– –, мультиобработка 91
– –, назначение выводов 94
– –, описание 32-разрядного процессора 11
– –, основные блоки 11
– –, основные определения 9
– –, память 11
– –, подсистемы 93
– –, прерывания и исключения 32
– –, привилегии и защита 75
– –, программирование 116
– –, производительность 109
– –, регистры 11
– –, режим реального адреса 132
– –, режимы адресации 117
– –, сводка сигнальных выводов 63
– –, селекторы и индикаторы 27
– –, система команд 118
– –, сопроцессоры 97
– –, структурная схема 13
– –, тактовый генератор 102
– –, тепловые характеристики 107
– –, тестирование и отладка 139
– –, типы данных 15
– –, требования к питанию и заземлению 106
– –, элементы языка 117
Многозадачность 329
Модификатор 148
Мультипрограммирование 329
Мультипротокольный контроллер последовательной связи 93

Некэшируемая память 58
Немаскируемые прерывания 34
Неопределенные биты 8
Непосредственная адресация 117
Неупакованный двоично-десятичный тип 16
Нуль-селектор 27

Обычный тип 17
Ограниченный сигнал LOCK 131
Операции над битами 119
Отказы 35
Относительная адресация 118
Отображение 10
Очередь команд 12

Память 144
–, адресация 313
–, буфера 323
– виртуальная 317
–, защита 312
–, интерфейс 49
–, кэш 319
–, организация 312
–, сегментация 316
–, системы 311
Переключение задач 90
Периферийное устройство 297
Поддержка языка высокого уровня 120
Порты ввода-вывода 302
Предвыборка 11
Прерывание 32
Приемопередатчики данных 74
Привилегии 69
–, иерархия 349
–, команды 77
–, уровни 74
–, чувствительные команды 77
Приоритеты 36
Проверка границы 79
Проверка поля TYPE 79
Программируемая логическая матрица 110
– –, интерфейс 110
– – состояния ДОЗУ 115
– – счетчика адресов регенерации 115
– – счетчик интервалов регенерации 115
– – управления ДОЗУ 115
– – управления локальной магистралью 112
Программируемый контроллер прерываний 93
Программное обеспечение:
инициализация защищенного режима 138
– режима реального адреса 137
разработка и проектирование 116
системное проектирование 116
указания программисту 141
Программные модули 117
Производительность системы 325
Прямая адресация 117
Прямой доступ к памяти 315

Реальная память 310
Реальный режим 14
Регистр задачи 90
– сегмента 21
– системных флагов 18
Регистры 17
– общего назначения 18
– отладки 24
– системных адресов 22
–, перечень 351
– тестирования (проверки) 25
– управления 22
– – сегментированной памятью 22
Режимы адресации 117
Режим реального адреса 132
– –, вход и выход 132
и МП 8086 134

––– и МП 80286 135
–––, программная инициализация 137
–––, формирование физического адреса 133

Сброс и инициализация 136
Сверхбольшие интегральные схемы 291
Сегмент состояния задачи 11
Сегментированное пространство адресов 46
Сегмент 9
Селектор 27
– адреса 73
Сигнал LOCK 92
Система команд 118
Система MULTIBUS 305
Системы MULTIBUS I/II 358
Согласованность кэш-памяти 58
Согласованный сегмент 82
Сопроцессоры 97
– арифметические 97
–, коды ошибок 43
– локальной вычислительной сети 102
– 8087 342
– 80287 346
Состояния магистрали 64
Сплошное адресное пространство 45
Статическое ОЗУ 51
Страничная организация 11
Строка 17
– битов 17

Таблица 9
– глобальных дескрипторов 30
– дескрипторов 30
– – прерываний 31
– локальных дескрипторов 31
– страниц 48
Тактовый генератор 82384 102
Тестирование и отладка 139
Тип 29
Типы данных 16
Трансляция страниц 46

Узлы управления 325
Указанный адрес 117
Указатель ближний 17
– дальний 17
– команд 22
Универсальный синхронно-асинхронный приемопередатчик 303
Упакованный двоично-десятичный тип 17
Уровень привилегий дескриптора 29

Физический адрес 11
Фиксатор адреса 73
Флаги 11
–, перечень 351

Целый тип 17
Цикл 65
– останова/прекращения работы 66
– подтверждения прерывания 65

Шины 229
– с тремя состояниями 299

Эффективный адрес 117

Оглавление

Предисловие 6

Глава 1. Введение в МП 80386 фирмы Intel 8

1.1. Основные определения 9
1.2. Обзор 32-разрядного микропроцессора 11
1.2.1. Основные блоки 11
1.2.2. Устройство управления памятью 14
1.2.3. Архитектура режима реальных адресов и защищеного режима 14
1.3. Типы данных 15
1.4. Регистры 17
1.4.1. Регистры общего значения (РОН) 18
1.4.2. Регистр системных флагов 18
1.4.3. Регистры сегментов 21
1.4.4. Регистры управления сегментированной памятью 22
1.4.5. Указатель команд 22
1.4.6. Регистры управления 22
1.4.7. Регистр отладки 24
1.4.8. Буфер ассоциативной трансляции 24

Глава 2. Селекторы и дескрипторы 27

2.1. Селекторы 27
2.2. Дескрипторы сегментов 28
2.3. Таблицы дескрипторов 30
2.3.1. Таблица глобальных дескрипторов 30
2.3.2. Таблицы локальных дескрипторов 31
2.3.3. Таблица дескрипторов прерываний 31

Глава 3. Прерывания и исключения 32

3.1. Маскируемые прерывания 33
3.2. Немаскируемые прерывания 34
3.3. Таблицы дескрипторов прерываний 35
3.4. Исключения 35
3.5. Контроллер прерываний 36
3.6. Приоритеты прерываний и исключений 36
3.7. Задачи и процедуры прерываний 36
3.8. Коды ошибок 38
3.8.1. Коды ошибок сопроцессора 43

Глава 4. Память 45

4.1. Организация памяти 45
4.1.1. Сегментация 45
4.1.2. Адресное пространство 46
4.1.3. Разбиение на страницы и их трансляция 46
4.1.4. Контроллер прямого доступа к памяти 49
4.2. Интерфейс с памятью 49
4.2.1. Интерфейс с основной памятью 50
4.2.2. Интерфейс со статическим ОЗУ 51
4.2.3. Инерфейс с динамическим ОЗУ 51
4.3. Кэш-память в МП 80386 52
4.3.1. Организация кэш-памяти 54
4.3.2. Обновление кэш-памяти 55
4.3.3. Кэш-память в МП 80386 58

Глава 5. Интерфейс: локальная магистраль, ввод и вывод 61

5.1. Интерфейс с локальной магистралью 61
5.1.1. Состояния магистрали 64
5.1.2. Циклы чтения и записи 65
5.1.3. Прерывания 65
5.1.4. Блокированные циклы и захват магистрали 67
5.2. Интерфейс с устройствами ввода-вывода 68
5.2.1. Команды ввода-вывода 69
5.2.2. Защита, привилегии и ввод-вывод 69
5.2.3. Основной интерфейс ввода-вывода 71

Глава 6. Привилегии и защита, конвейеризация, многозадачность и мультиобработка 75

6.1. Привилегии и защита 75
6.1.1. Привилегии 76
6.1.2. Уровни и правила привилегий 76
6.1.3. Привилегированные команды 77
6.1.4. Чувствительные команды 77
6.1.5. Защита 77
6.2. Конвейеризация с целью повышения производительности 85
6.2.1. Конвейеризация адресов 85
6.2.2. Замедление микропроцессора 86
6.3. Многозадачность 87
6.3.1. Сегмент состояния задачи 88
6.3.2. Дескриптор сегмента состояния задачи 88
6.3.3. Переключение задач 90
6.4. Мультиобработка 91
6.4.1. Префикс LOCK и сигнал LOCK# 92

Глава 7. Аппаратные средства и подсистемы 93

7.1. Контроллеры 93
7.1.1. Контроллер последовательной связи 8274 93
7.1.2. Контроллер прерываний 8259A 93
7.1.3. Контроллер прямого доступа к памяти 82258 96
7.2. Сопроцессоры 97
7.2.1. Арифметические сопроцессоры 80287 и 80387 97
7.2.2. Сопроцессор локальной вычислительной сети 82586 102
7.3. Тактовый генератор 82384 102
7.3.1. Тактовые частоты и синхронизация 104

7.3.2. Подача тактовых импульсов и оконечная нагрузка 104
7.4. Требования к питанию и заземлению 106
7.5. Тепловые характеристики 107
7.6. Изготовление и отладка аппаратуры 107
7.6.1. Рекомендации по шаговому построению системы 108
7.6.2. Рекомендации по отладке 109
7.7. Вопросы производительности 109
7.8. Применение программируемых логических матриц 110
7.8.1 Интерфейс на основе ПЛМ 110
7.8.2. ПЛМ управления локальной магистралью 112
7.8.3. ПЛМ состояния ДОЗУ 115
7.8.4. ПЛМ управления ДОЗУ 115
7.8.5. ПЛМ счетчика интервалов регенерации 115
7.8.6. ПЛМ счетчика адресов регенерации 115

Глава 8. Программирование МП 80386 116

8.1. Разработка программных комплексов 116
8.2. Элементы языка 117
8.3 Режимы адресации 117
8.4. Обзор системы команд 118
8.5. Виртуальная среда МП 8086 124
8.5.1. Вход и выход из виртуального режима МП 8086 124
8.5.2. Адресация в режиме виртуального МП 8086 126
8.5.3. Формирование адреса 127
8.5.4. Задача виртуального МП 8086 127
8.5.5. Чувствительные команды в режиме виртуального МП 8086 128
8.5.6. Ввод-вывод в виртуальном режиме 128
8.5.7. Различия виртуального МП 8086 и МП 80386 129
8.6. Выполнение программы в защищенном режиме МП 80286 130
8.6.1. Ограничения префикса LOCK 131
8.6.2. Циклический возврат адреса в МП 80286 131
8.7. Режим реальных адресов МП 80386 132
8.7.1. Вход и выход из режима реальных адресов 132
8.7.2. Формирование физического адреса в реальном режиме 133
8.7.3. Новые исключения МП 80386 133
8.7.4. Различия реального режима и МП 8086 134
8.7.5. Различия реального режима и МП 80286 135
8.8. Сброс и инициализация 136
8.8.1. Регистры после запуска 136
8.8.2. Программная инициализация режима реальных адресов 137
8.8.3. Программная инициализация защищенного режима 138
8.9. Тестирование и отладка 139
8.9.1. Самотестирование 139
8.9.2. Буфер ассоциативной трансляции (TLB) 139

Глава 9. Замечания по программированию МП 80386 141

9.1. Указания программисту 141
9.2. Программирование МП 80386 142
9.2.1. Память 142
9.2.2. Дескрипторы 143
9.2.3. Команды программы 143
9.2.4. Регистры 144
9.2.5. Задачи 145
9.2.6. Привилегии и защита 145
9.2.7. Тестирование и отладка 146

Глава 10. Система команд МП 80386 148

10.1. Формат команд 148
10.1.1. Описание обозначений 152
10.1.2. Описание полей MOD и R/M кода команды 154
10.1.3. Примеры выполнения команд 155
10.1.4. Используемые флаги и регистры 155
10.2. Алфавитный список команд 156

Глава 11. Введение в интегральные схемы 286

11.1. Начало 286
11.2. Из чего сделаны ИС? 287
11.2.1 МОП-транзисторы с каналами *p*- и *n*-типов 290
11.2.2. КМОП-транзисторы 291
11.2.3. Сверхбольшие интегральные схемы (СБИС) 291
11.3. Упаковка ИС 292

Глава 12. Принцип сопряжения 295

12.1. Сопряжение 295
12.1.1. Уровни 295
12.1.2. Регистр 296
12.1.3. Буфер 297
12.1.4. Периферийные устройства 297
12.1.5. Память 298
12.2. Шины 299
12.2.1. Шина с тремя состояниями 299
12.2.2. Типы шин 300
12.2.3. Операции на магистрали 301
12.3. Порты ввода-вывода 302
12.3.1. Режимы работы портов ввода-вывода 302
12.4. Универсальный синхронно-асинхронный приемопередатчик 303
12.5. MULTIBUS 305
12.5.1. Ведущие 305
12.5.2. Ведомые 305
12.5.3. Операции на магистрали 305
12.5.4. Архитектура магистрали 306
12.6. Электрическое питание 308
12.7. Резюме 308

Глава 13. Организация запоминающих устройств 309

13.1. Реальная память 310
13.2. ПЗУ и ОЗУ 310
13.3. Системы памяти 311
13.3.1. Организация памяти 312
13.3.2. Защита памяти 312
13.3.3. Адресация памяти 313
13.3.4. Диспетчер памяти 314
13.3.5. Прямой доступ к памяти (ПДП) 315
13.3.6. Сегментация памяти 316
13.3.7. Виртуальная память 317
13.3.8. Кэш-память 319
13.4. Буфера 323
13.5. Синхронизация 324
13.5.1. Временные ограничения 324
13.6. Резюме 324

Глава 14. Повышение производительности системы 325

14.1. Узлы управления 325
14.2. Независимость процессов 326
14.3. Предвыборка 326
14.3.1. Эффективность предвыборки 327
14.3.2. Алгоритмы предвыборки 327
14.3.3. Выборка следующей страницы 327
14.4. Конвейерная обработка 327
14.4.1. Методы 327
14.4.2. Проблемы 328
14.4.3. Алгоритмы управления 329
14.5. Сопроцессор 329
14.6. Многозадачность и мультипрограммирование 329
14.6.1. Упорядочение событий 330
14.6.2. Свойства мультипрограммных систем 331
14.7. Резюме 331

Приложение 1. Интегральные схемы и фирма Intel 332

Приложение 2. Перечень регистров, битов и флагов 351

Приложение 3. Мнемонические обозначения и условные сокращения 353

Приложение 4. Магистрали MULTIBUS I/II и микропроцессор 80386 358

Приложение 5. Команды защищенного режима и пример программы 366

Глоссарий 425

Литература 438

Предметный указатель 440

Уважаемый читатель!

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присылать по адресу:
129820, ГСП, Москва, И-110, 1-й Рижский пер., д. 2, изд-во «Мир»

Научное издание

Пенн Брамм, Дон Брамм

МИКРОПРОЦЕССОР 80386 И ЕГО ПРОГРАММИРОВАНИЕ

Заведующий редакцией д-р техн. наук А. Л. Щёрс
Зам. заведующего редакцией Э. Н. Бадиков
Научный редактор С. В. Богдасаров
Младший научный редактор Ю. Л. Евдокимова
Художник А. В. Захаров
Художественные редакторы Н. М. Иванов, О. Н. Адаскина
Технический редактор И. И. Володина
Корректор Л. Д. Панова

ИБ № 7213

Сдано в набор 03.07.89. Подписано к печати 11.04.90. Формат $70 \times 100^{1}/_{16}$. Бумага офсетная № 1. Печать офсетная. Гарнитура таймс. Объем 14,0 бум. л. Усл. печ. л. 36,4. Усл. кр.-отт. 71,5. Уч.-изд. л. 34,82. Изд. № 6/6679. Тираж 20 000 экз. Зак. 854. Цена 3 р 90 к.

Издательство «Мир»
В/О «Совэкспорткнига» Государственного комитета СССР по печати
129820, ГСП, Москва, 1-й Рижский пер. 2.

Можайский полиграфкомбинат В/О «Совэкспорткнига» Государственного комитета СССР по печати
г. Можайск, ул. Мира, д. 93