АЛЕКСАНДР ПАНОВ

bhv®

РЕВЕРСИНГ и ЗАЩИТА ПРОГРАММ ОТ ВЗЛОМА

Наиболее распространенные способы защиты программ и способы их преодоления

Примеры работы с отладчиками и дизассемблерами

Подробное описание отладчика OllyDBg

Методика написания средств защиты программ от взлома

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Александр Панов

РЕВЕРСИНГ и ЗАЩИТА ПРОГРАММ ОТ ВЗЛОМА

Санкт-Петербург

«БХВ-Петербург»

2006

УДК 681.3.06
ББК 32.973.26-018.1
П16

Панов А. С.

П16 Реверсинг и защита программ от взлома. — СПб.: БХВ-Петербург, 2006. — 256 с.: ил.

ISBN 5-94157-889-X

Подробно изложены современные основные методики защиты программного обеспечения, начиная с составления программ и заканчивая их отладкой. Рассмотрены примеры взлома стандартных защит и даны рекомендации для предотвращения такого взлома. Приведено большое количество рабочих примеров, которые должны помочь программисту решить возникшие перед ним проблемы в защите его интеллектуальной собственности. Подробно описана работа с отладчиком OllyDBg. На сопроводительном компакт-диске находятся описанные в книге программы.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Татьяна Темкина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.06.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 20,64.
Тираж 2500 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-889-X

Оглавление

Глава 1

Введение

Эта книга уникальна. До сих пор ничего подобного не издавалось. Это первая книга, посвященная только крекерам и разнообразным анализам защиты программ, а также способам ее преодоления.

Многие, наверное, задавались вопросом: как же появляются на свет различные креки (crack) и патчи (patch) к программам, откуда они берутся? Для того чтобы ответить на этот вопрос, и была написана эта книга. Прочитав ее, вы поймете разницу между хакером и крекером, узнаете, насколько увлекателен реверсинг (reverse code engineering), и научитесь снимать защиту с программ и создавать ее.

Надеюсь, книга вам понравится.

Для кого предназначена эта книга

Книга предназначена в первую очередь для всех тех, кто сам занимается написанием защиты программного обеспечения, всем программистам и тем, кто хотел бы защитить написанную им программу от изменения.

На страницах книги вы найдете множество рекомендаций по усилению защиты ваших программных продуктов, примеры и способы разработки разнообразных защит и т. д.

Книгу можно использовать как справочник по отладчику OllyDBg. Это первая книга, в которой вы найдете описание данного отладчика.

Книга также пригодится тем, кто интересуется, как именно взламываются программы, какие защиты существуют и что вообще такое "реверсинг".

Благодарности

В первую очередь я говорю огромное СПАСИБО моей жене Пановой Ольге Анатольевне, за терпение, поддержку и внимание, проявленные в процессе работы над материалом книги и ее подготовки к изданию. Если бы не она, книга никогда не была бы закончена.

Большое спасибо моим родителям за то, что они вырастили меня таким, какой я есть.

Большое спасибо и моему брату, Панову Владимиру Сергеевичу, именно с ним мы начинали осваивать компьютер.

Также хочу поблагодарить Масло Татьяну Викторовну за ее помощь и советы по изложению материала.

Я хочу сказать спасибо моему другу nice за его советы и помощь в написании книги. Без них книга не была бы подготовлена в срок.

Также хочу поблагодарить Bad_guy и всех жителей сайта **cracklab.ru** за помощь в написании этой книги.

Особая благодарность — высокопрофессиональному коллективу редакции издательства "БХВ-Петербург", предоставившего возможность издать данную книгу.

Кто такой хакер и как им стать

Для начала поговорим немного о терминах.

Сейчас в мире происходит путаница. Многие СМИ под "хакерами" подразумевают большой пласт компьютерных профессионалов — хакеров, крекеров, программистов, компьютерных пиратов и т. д., что неверно по сути, поскольку у них совершенно разные направления деятельности. Это то же самое, что путать директора предприятия и дворника, инженера и секретаршу. Конечно, ничто не мешает одному человеку выполнять все обязанности, но чаще это разные люди. Упомянутые области программмирования настолько обширны, что досконально знать их все, быть везде профессионалом просто невозможно. Для того чтобы стать профессионалом, необходима постоянная и долгая практика. Человек пока, к сожалению, не может одновременно делать несколько больших дел, например, читать техническую документацию, чертить чертеж и писать письмо любимой.

Итак, кто же такой хакер? *Хакер* — профессионал, который разрабатыват или взламывает сетевую защиту. Главный интерес хакера — это сайты и серверы в Интернете. Хакеру интересен сам процесс исследования защиты, но он никогда

не сделает что-либо, вызывающее крах исследуемой им системы. Закончив исследование системы защиты, хакер составит подробный отчет о проделанной им работе, чтобы администратор данного сайта смог лучше защитить данный сайт от взлома. Этим он отличается от компьютерного пирата. *Компьютерный пират* тоже исследует защиту сайтов, но главная цель пирата — порча или уничтожение информации. Этим обычно занимаются обиженные на весь мир недалекие подростки. Им кажется, что так они обратят на себя внимание, покажут свою "крутость". Они ничего не придумывают сами — просто скачивают из Интернета программы и пытаются ими воспользоваться, не понимая принцип их действия. К сожалению, как показывает практика, они не догадываются, что после того, как на них "обратят внимание", им уже больше ничего не захочется.

Как стать хакером, т. е. профессионалом, занимающимся исключительно защитой сайтов и серверов? Ответить на этот вопрос не так просто, как кажется. В первую очередь, необходимо постоянно читать техническую документацию по серверам, программным защитам сайтов, межсетевым экранам. Это связано с тем, что все в мире постоянно изменяется, и вам необходимо быть в курсе последних разработок. Информация даже годичной давности сегодня уже не принесет большой пользы. Во-вторых, необходимо приобрести знания администратора компьютерной сети. Это позволит понять, как "думают" администраторы, изучить их сильные и слабые стороны. Еще нужно изучить языки программирования, начав с Perl, HTML, Java (к другим языкам сетевого программирования лучше обратиться уже после того). Знать ассемблер хакеру, скорее всего, не нужно, как и Delphi, C++ и т. д.

Главное, что необходимо, — это упорный труд и желание, без этого никакие знания вам не помогут.

Кто такой крекер и как им стать

Теперь поговорим о том, кто такой крекер (cracker).

Cracker переводится как "взломщик программной защиты", т. е. тот, кто исследует защиту программ. Это разнообразные триал-защиты, защиты Shareware и т. д. Конечно, совершенно необязательно взлому иметь негативный характер, это может быть и взлом, заказанный автором программы с целью исследования того, насколько хороша его защита. После исследования программы крекер составляет подробный отчет, который он передает автору с рекомендациями по усилению (если оно необходимо) защиты программы. Крекер никоим образом не занимается сетевой защитой сайтов, серверов и т. д. Его стихия — это программы. Он копирует программу на свой компьютер

(в отличие от хакера) и исследует ее. Его компьютер — это своеобразная лаборатория по исследованию программ. Крекеру в первую очередь необходимо знание ассемблера. Это своего рода специфика работы, т. к. программы доходят до крекера главным, если не единственным, образом — в виде исполняемого модуля. Крекер также должен знать какой-либо язык программирования высокого уровня, в идеале C++ или Delphi. Это необходимо для написания разнообразных утилит, облегчающих жизнь крекера. Конечно, никто не запрещает писать эти утилиты на ассемблере, но крекеры очень ценят свое время, поэтому все, что может быть автоматизировано, будет ими автоматизировано с наикратчайшие сроки.

Из всего сказанного понятно, что крекерство — не то занятие, в котором обиженные на весь мир подростки могут выказывать свою "крутость", применяя скачанные из Интернета программы. Это тяжелый труд. Но от этого он не становиться менее увлекательным. Эта профессия затягивает. В ней очень много интересного. Это как противоборство интеллектов человека и машины.

Как стать крекером? Ответить на этот вопрос тоже не так просто. Для начала необходимо изучить ассемблер. В этом вам может помочь моя книга "Assembler: Экспресс-курс" ("БХВ-Петербург", 2006). Изучив ассемблер и написав на нем пару программ (чтобы убедиться в том, что вы его знаете и понимаете), загружайте с сопроводительного компакт-диска этой книги или из Интернета (например, с моего сайта **www.abyss-soft.narod.ru** или с сайта **cracklab.ru**) разнообразные программы типа crackme, созданные специально для того, чтобы их взломали. Начинать лучше всего с простых программ (например, с моих или с crackme Fantoma). После этого вам придется изучить язык программирования высокого уровня (например C++); это потребуется для автоматизации процесса реверсинга.

На моем сайте вы найдете программу для удаления мусора с дисков, написанную мной по заказу клубов; на нее я поставил защиту, чтобы вы смогли потренироваться. На сопроводительном диске книги тоже есть несколько программ, которые я написал специально для этого. На таких программах я демонстрирую способы защиты программного обеспечения, а на страницах этой книги показываю, как преодолеваются такие защиты, можно сказать, "веду за ручку" начинающего крекера. Прочтя данную книгу, вы "проскочите" тот трудный участок в начале пути, когда, разбираясь в основах, не у кого спросить об элементарных вещах, до обсуждения которых не снисходят более опытные "братья по оружию". Вы справитесь с этим играючи.

Также хочу сказать, что лучший сайт, посвященный реверсингу, какой я когда-либо видел, это **cracklab.ru**. На этом сайте есть много профессионалов (привет вам, nice, Bad_guy и Ara), которые помогут вам в этом нелегком деле.

Что понадобится для работы с книгой

Для работы с этой книгой вам понадобится несколько вещей. В первую очередь, это отладчик. Рекомендую вам очень мощный отладчик OllyDBg, подробнее о котором я расскажу немного позднее. Сейчас скажу только, что для большинства исследований он очень удобен. Найти его можно на просторах Интернета. Но вы можете взять и любой отладчик, который вам больше нравится, например, SoftICE. Еще вам будут нужны разнообразные программные мониторы — RegMon, FileMon, Regsnap. Также вам понадобится дизассемблер IDA, самый мощный из существующих (про него я тоже расскажу позднее).

Глава 2

Особенности ассемблера для исследования работы программ

В этой главе кратко описаны принципы работы персонального компьютера и ассемблера (языка Assembler). Для более подробного изучения языка ассемблер я рекомендую вам прочитать мою книгу "Assembler: Экспресс курс". Это пригодится вам для того, чтобы лучше понять, как именно можно спроектировать защиту программы или как ее можно снять (это зависит от целей, которые вы перед собой ставите).

Желающим побыстрее начать что-то делать рекомендую перейти сразу к четвертой главе; если понадобится, вы сможете вернуться к этой главе позднее. Но если вы ступили на путь реверсинга программ впервые, то я рекомендую вам читать все по порядку. Это поможет вам понять многие принципы, сэкономив много времени.

Приведенные здесь основы ассемблера позволят лучше понять, что же именно мы будем делать, как устроена защита той или иной программы и т. д. Я не буду объяснять весь синтаксис ассемблера. Это нам ни к чему.

Мы рассмотрим принципы работы компьютера и микропроцессора на примере микропроцессора 8086. Сейчас у многих имеются компьютеры с гораздо более совершенной архитектурой (AMD Athlon XP, Intel Pentium и др.) но все, что я расскажу о микропроцессоре 8086, справедливо и для всех остальных моделей процессоров (естественно, речь идет только об архитектуре PC).

Структурная схема компьютера

Структурная схема персональной ЭВМ, работающей на базе микропроцессора, представлена на рис. 2.1.

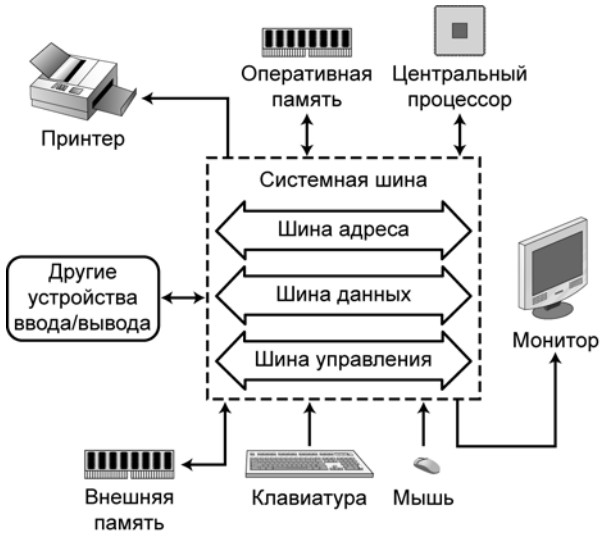

Рис. 2.1. Структурная схема микрокомпьютера

При работе персонального компьютера (также называемого *микрокомпьютером*) выполняются следующие действия:

1. Программа или данные вводятся с помощью устройств ввода из внешней памяти (жесткий диск, дискета и т. д.) в оперативную память.
2. Микропроцессор выбирает команды программы из оперативной памяти в строгом порядке, в котором их следует выполнять. Микропроцессор считывает указанные в этих командах данные и производит указанные командами действия. Адреса и данные передаются через *системную шину*, соединяющую центральный процессор, память и устройства ввода/вывода (рис. 2.1).
3. Данные, полученные в результате обработки команд, микропроцессор пересылает в оперативную память или на устройство вывода (дисплей, принтер, внешняя память и т. д.).

Микропроцессор расположен на *системной плате*, где также расположены микросхемы оперативной памяти и микросхемы *постоянного запоминающего устройства (ПЗУ)*, содержащего программу начальной загрузки памяти ЭВМ и другие вспомогательные программы. Различные компоненты компьютера подсоединяют к системной шине, состоящей из адресной шины, шины данных и шины управления. Связь между процессором, памятью и устройствами ввода/вывода осуществляется тоже через системную шину. Процессор подсоединяется к шине через блок логики управления шиной, устройства — через интерфейсы.

Архитектура микропроцессора

Главное, что должен делать центральный процессор, — упрощать работу со следующими объектами вычислительной системы:

- присваивания и арифметические выражения;
- безусловные/условные переходы;
- логические выражения;
- циклы;
- массивы/структуры данных;
- подпрограммы;
- устройства ввода/вывода.

Стандартная архитектура центрального процессора, обладающего такими возможностями, должна включать:

- набор регистров для адресации данных и выполнения арифметических операций;
- устройство управления для исполнения команд;

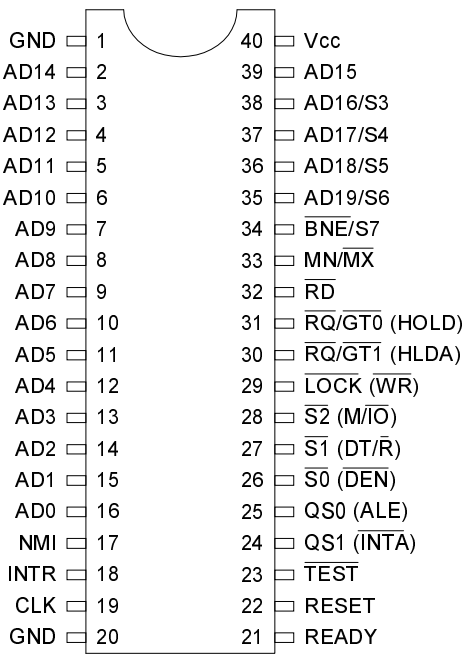

Рис. 2.2. Разводка контактов микропроцессора Intel 8086

- арифметико-логическое устройство (АЛУ) для выполнения арифметических и логических операций;
- секцию управления вводом/выводом.

Под *регистром* мы будем понимать запоминающее устройство для временного хранения целого числа байт, расположенное внутри микропроцессора.

Рассмотрим эти компоненты на примере микропроцессора Intel 8086 (рис. 2.2).

В компьютерах на базе данных процессоров используется системная шина, называемая *мультиплексной*, поскольку по ее линиям передаются как адреса, так и данные.

Сигналы принимаются и передаются через мультиплексную шину микропроцессором, который делится на две части — операционное устройство и шинный интерфейс (рис. 2.3).

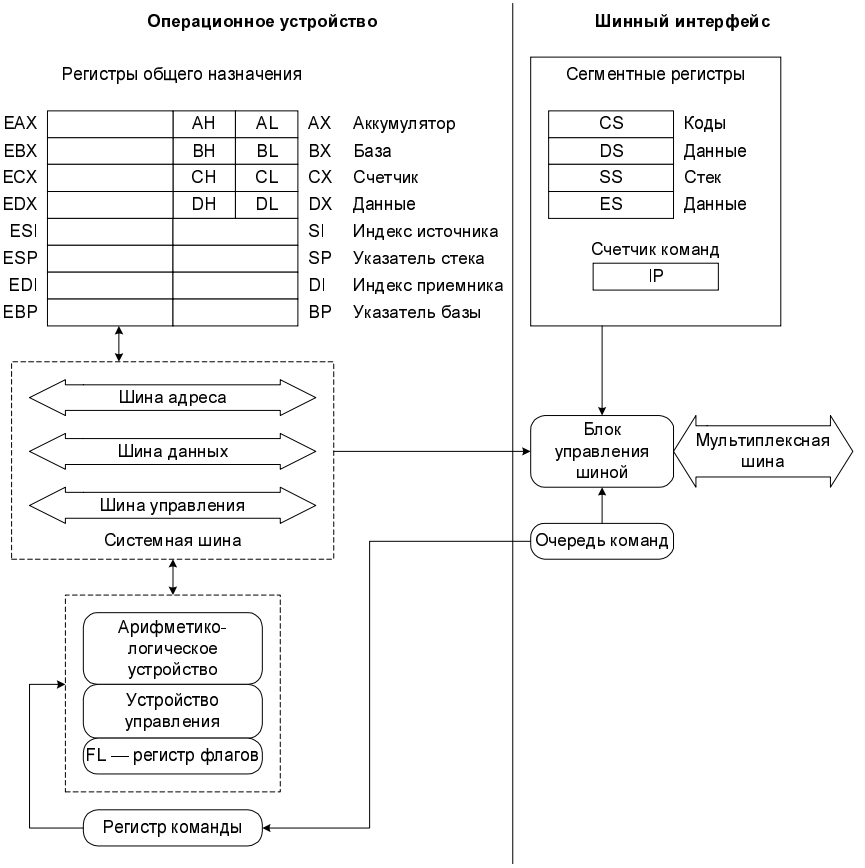

Рис. 2.3. Внутренняя архитектура микропроцессора Intel 8086

Операционное устройство предназначено для выполнения команд. Оно содержит АЛУ, устройство управления и 10 регистров.

Шинный интерфейс подготавливает команды и данные для выполнения и состоит из блока управления шиной, очереди команд и сегментных регистров. Он управляет передачей данных на операционное устройство, в память и на устройства ввода/вывода. В шинном интерфейсе адрес формируется с помощью сегментных регистров. Шинный интерфейс также имеет доступ к командам, находящимся в оперативной памяти, осуществляет выборку этих команд и помещает их в очередь команд. При этом шинный интерфейс должен как бы "заглядывать вперед" и выбирать команды так, чтобы всегда существовала непустая очередь команд, готовых для выполнения.

Согласно рис. 2.3 микропроцессор 8086 имеет 14 16-разрядных регистров, доступных для программиста. Начиная с микропроцессора i80386 пользовательских регистров стало 16 — добавились два сегментных регистра FS и GS. Остальные регистры расширились до 32 бит. Основные регистры микропроцессоров семейства 80x86 показаны на рис. 2.4. Регистры позволяют свести операции к командам, имеющим не более одного операнда, находящегося в памяти.

Рис. 2.4. Регистры микропроцессоров семейства Intel 80x86

Организация памяти

Внутренняя память микрокомпьютера делится на два типа.

Первый тип — это память, доступная только для чтения и называемая *постоянной памятью* (Read Only Memory, ROM). ROM-память представляет собой специальную микросхему, находящуюся на материнской плате, данные в которую записываются на заводе — изготовителе материнской платы. Основное назначение ROM-памяти — поддержка процедуры начальной загрузки. Для программиста наиболее важным элементом ROM-памяти является *базовая система ввода/вывода* (Basic Input/Output System, BIOS).

Память второго типа называется *памятью произвольного доступа* (Random Access Memory, RAM), или *оперативной памятью*. Эта память доступна для чтения и записи и является памятью, с которой работает программист. Содержимое оперативной памяти теряется при выключении питания компьютера. Поэтому для сохранения данных и программ требуются средства *внешней памяти* (обычно жесткий диск или дискета).

Память ЭВМ состоит из последовательности ячеек, каждая из которых имеет *адрес*. Таким образом, *память* — это область хранения информации, доступ к которой осуществляется по шине адреса.

Минимальная компьютерная единица информации называется *битом*. Бит может принимать только одно значение из двух — 0 или 1. Память разбивается на блоки различных размеров — байт, слово, сегмент, страница и др. Рассмотрим их подробнее.

Байтом называется минимальная адресуемая единица информации. Байт состоит из восьми бит. Таким образом, чтобы изменить один бит памяти, надо записать, по крайней мере, восемь бит в байт, содержащий этот бит. Каждому байту соответствует его собственный физический адрес.

Словом называется последовательность бит, число которых равно размеру регистра-аккумулятора. Например, в микропроцессоре 8086 регистр AX имеет длину 16 бит, т. е. слово состоит из 16 бит, или содержит 2 байта. В более современных компьютерах аккумулятором является регистр EAX, а это значит, что слово состоит из 32 бит (4 байта). В этом случае его называют *двойным словом*, оставляя название "слово" за областью памяти, состоящей из 16 бит.

Примечание

В 16-разрядных процессорах: слово — 16 бит, двойное слово — 32 бита, в 32-разрядных процессорах: слово — 32 бита, двойное слово — 64 бита.

Микропроцессор работает в одном из следующих режимов:

- *реальный режим* — режим, в котором работал процессор 8086;
- *защищенный режим* — режим, позволяющий максимально реализовать возможности процессоров 80*x*86 (начиная с процессора 80286 в этом режиме физический адрес формируется не так, как в реальном режиме);
- *режим виртуального 8086* — режим, возможный при работе процессора в защищенном режиме; в этом режиме могут работать несколько программ, разработанных для 8086; процесс формирования физического адреса для этих программ производится как в реальном режиме;
- *режим системного управления* — режим, появившийся в микропроцессоре Pentium; функционирование процессора в этом режиме по общим принципам подобно его работе в реальном режиме.

Мы только начинаем изучать язык ассемблера, поэтому для простоты работы будем рассматривать адресацию памяти в реальном режиме.

Сегментом называется независимый, поддерживаемый на аппаратном уровне блок памяти, имеющий размер не больше 2^{16} байт (поскольку $2^{16} = 2^6 \cdot 2^{10} = 64 \cdot 1024$, а 1024 байт = 1 Кбайт, максимальный размер сегмента будет равен 64 Кбайт).

Параграфом называется минимальный размер сегмента 16 байт.

Адресная шина процессора 8086 состоит из 20 линий, поэтому максимальный адресуемый объем памяти в реальном режиме равен 2^{20} байт = 2^{10} Кбайт = = 1 Мбайт. В защищенном режиме для процессора 80286 он равен 2^{24} байт = 16 Мбайт, а для процессоров 80386 и выше — 2^{32} байт = 4 Гбайт.

Память разделяется на сегменты. Адрес ячейки формируется подобно почтовому адресу (сегментный адрес указывает "улицу", а смещение — "номер дома"). Физический адрес байта, имеющего смещение (т. е. номер байта в сегменте) *x* относительно начала сегмента, указанного с помощью сегментного регистра S, будет равен $16 \times S + x$. Например, регистр CS (сегмент кодов) используется для указания сегмента, в котором находится код программы. Так как содержимое счетчика команд IP равно смещению текущей команды относительно начала сегмента, физический адрес EA текущей команды будет равен $16 \cdot CS + IP$.

Вернемся к рис. 2.3 и рассмотрим процессы выполнения команд и обмена данными между микропроцессором и оперативной памятью.

1. По содержимому регистров CS и IP, хранящих адрес команды в виде базового адреса, т. е. адреса начала сегмента (CS), и смещения (номера байта в сегменте) (IP), определяется местонахождение (EA) команды, выполняемой на следующем шаге.

2. Вычисленный адрес запоминается в стеке.
3. Из оперативной памяти считывается команда с этим адресом и пересылается в процессор.
4. Команды помещаются в очередь команд и выполняются в порядке очередности.
5. Местоположение данных, участвующих в командах, определяется в шинном интерфейсе по информации, содержащейся в адресной части команды, в регистрах сегментов DS, SS и ES, а также в регистрах SP, BP, SI, DI.
6. Данные, прочитанные из памяти, пересылаются в АЛУ или регистр данных (DS) и обрабатываются текущей командой, адрес которой был найден на шаге 2. Результаты обработки данных помещаются в оперативную память или в какой-либо из регистров.

Циклическое повторение описанных процедур составляет выполнение программы. Номер сегмента называется *адресом сегмента (сегментным адресом)*, а номер байта в сегменте — *смещением*. На практике указывают лишь смещение, а также по умолчанию предполагается, что значение регистра DS известно.

Надо помнить, что любой сегмент имеет объем до 64 Кбайт и имеется 4 сегментных регистра; таким образом, доступна память 256 Кбайт. Но в действительности в программе возможно любое количество сегментов. Для того чтобы адресовать любой сегмент, достаточно изменить значение сегментного регистра.

Память RAM включает в себя первые три четверти внутренней памяти, а ROM — последнюю четверть. В памяти объема 1 Мбайт ROM начинается с 768-го килобайта. Распределение памяти при работе в операционной системе MS-DOS показано на рис. 2.5 (слева указаны начальные адреса блоков памяти).

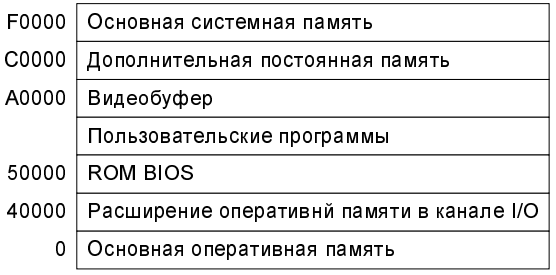

Рис. 2.5. Распределение памяти в MS-DOS

Представление данных

Данные в памяти ЭВМ имеют двоичное представление. В самом общем случае мы можем отвести под целое число любое количество соседних байтов. Программисту приходится работать с числами разных форматов (не только с целыми). Рассмотрим некоторые наиболее часто используемые типы данных:

- целые числа;
- двоично-десятичные числа;
- неупакованный BDC-формат;
- упакованный BDC-формат;
- числа с плавающей точкой;
- символьный тип.

Для быстрого старта этого достаточно, а подробную информацию о любых типах данных вы можете получить из специальных источников.

Целые числа

Система команд ЭВМ поддерживает работу с целыми числами *без знака* и *со знаком*, занимающими байт, слово или двойное слово.

Целое число без знака в двоичной записи состоит из k бит, где $k = 8$, 16 или 32.

Байтом 1010 1100 представляется число $2^7 + 2^5 + 2^3 + 2^2 = 172$, а словом 0111 0111 0000 1001 — число $7 \cdot 16^3 + 7 \cdot 16^2 + 9 = 30\,473$.

Самое большое число, которое можно представить байтом, равно 255, а наибольшее число, представляемое словом, — $2^{16} - 1 = 65\,535$.

Диапазоны значений целых чисел со знаком, имеющих разную длину:

- байт: от –128 до 127;
- слово: от –32 768 до 32 767;
- двойное слово: –2 147 483 648 до 2 147 483 647.

Целые числа со знаком записываются в дополнительном коде: положительное число записывается как беззнаковое (его старший бит равен 0), а отрицательное число $-x$ (при $x > 0$) представляется как $2^k - x$, где k — количество разрядов ячейки памяти, отведенной под число (k = 8,16 или 32). Если в этой ячейке записано некоторое беззнаковое число z, старший бит которого равен 1, то это число представляет число со знаком, равное $-(2^k - z)$.

Например, байт 10101100 является беззнаковым числом 172, которое превращается в $-(256 - 172) = -84$, если рассматривать его как число со знаком.

Итак, отрицательное число $-x$ ($x > 0$) представляется как $2^k - x$; его старший разряд должен содержать 1. Это представление можно получить с помощью следующих действий:

1. Записать двоичное представление числа x.
2. Инвертировать двоичное число, т. е. в разряды, содержащие единицы, записать нули, а в разряды, содержащие нули, — единицы.
3. К полученному числу прибавить 1.

Например, для числа –84 (байт) выполним следующие действия:

1. $84 = 2^6 + 2^4 + 2^2$, следовательно, двоичное представление: 01010100.
2. Инвертируем все разряды: 10101011.
3. Прибавляем 1: 10101100.

Двоично-десятичные числа

Затраты на перевод чисел из десятичной формы записи в двоичную и наоборот могут быть чрезмерно велики. Поэтому предусмотрено специальное представление целых чисел — двоично-десятичный код (binary coded decimal, BCD), строящийся по следующему принципу: берется десятичная запись числа и каждая его цифра заменяется на четыре двоичные цифры. Например, число 326 заменяется на последовательность бит 0011 0010 0110. Такой формат называется *BCD-форматом*. Различают *неупакованный* и *упакованный* BCD-форматы.

Неупакованный BCD-формат. На каждую десятичную цифру числа отводится один байт. Значение байта равно значению его младшей тетрады (четырех бит). Для обозначения знака применяются неиспользованные тетрады (старший байт). Если использовать ASCII-коды для представления цифр и знаков, то число будет состоять из символов:

00110000 = '0'	00110110 = '6'
00110001 = '1'	00110111 = '7'
00110010 = '2'	00111000 = '8'
00110011 = '3'	00111001 = '9'
00110100 = '4'	00101011 = '+'
00110101 = '5'	00101101 = '-'

Например, число –326 будет представлено такой последовательностью байт: '6', '2', '3', '-'. В программе это записывается с помощью директивы `db`:

```
db      '623-'
```

Запись числа в неупакованном BCD-формате может состоять из произвольного фиксированного числа байт. При выполнении арифметических операций отрицательные числа могут быть представлены в дополнительном коде — если число байт формата равно n, то отрицательное число записывается как $10^n - x$. Например, при $n = 4$ число –326 будет представлено символьной строкой '4769'.

Упакованный BCD-формат. В каждом байте записываются две десятичные цифры. Число состоит из 20 цифр и занимает 10 байт, старший байт отводится под знак.

Такое число определяется с помощью директивы `dt`. Например, число $x = 12\,345$ будет определено так:

```
x       dt      12345
```

и состоять из двадцати цифр — 00000000000000012345, каждая из которых записана четырьмя битами. В этом случае цифры располагаются в памяти парами в виде такой последовательности байт: 01000101, 00100011, 00000001, 00000000, 00000000, … (т. е. 45, 23, 01, 00 и т. д.).

Примечание

Байты в памяти компьютера располагаются в обратном порядке. Это связано с некоторыми особенностями строения персональных компьютеров. А для упакованного BCD-формата число условно разбивается на двузначные числа (недостающие разряды дополняются нулями). Например, число 47 281 будет разбито на пары 81, 72, 04.

Первый байт будет иметь смещение (относительный адрес) x, второй — $x + 1$, третий — $x + 2$ и т. д.

Самый старший бит десятого байта используется для хранения знака числа.

Например, запись

```
x       dt      -12345
```

будет транслироваться как строка, значение старшего байта которой — 80, затем идут шесть нулевых байтов, а затем — байты с значениями 01, 23 и 45.

Также надо иметь в виду, что для форматов BCD может применяется противоположный порядок хранения байтов в памяти. Формат хранения целиком зависит от квалификации и предпочтения программиста. Более подробную информацию вы можете найти в справочной литературе.

Числа с плавающей точкой

Как мы знаем, к вещественному типу относятся числа с фиксированной и плавающей точкой. Для ассемблера числа с фиксированной точкой и целые числа являются одним и тем же. Поэтому мы рассмотрим только числа с плавающей точкой, которые будем называть вещественными.

Число с плавающей точкой определяется значениями его знака, порядка и мантиссы. В частности, если под число отводится 4 байта, то эти 32 бита распределяются так:

- 1 бит (31-й) — знак (S);
- 8 бит (с 23-го по 30-й) — порядок (E);
- 23 бита (с 0-го по 22-й) — мантисса (M).

В этом случае 32 бита представляют число с плавающей точкой, значение которого равно $(-1)^S \cdot 2^{E-127} \cdot 1.M$.

Примечание

Тем, кто хочет подробнее разобраться в этом, я рекомендую обратиться к соответствующей литературе, в которой рассматривается программирование сопроцессора (например, [6]).

Следовательно, наибольшее положительное и наименьшее отрицательное значения такого числа равны $\pm 2^{255-127} \cdot (2 - 2^{-23})$. Наименьшее положительное число, записанное в формате с плавающей точкой, равно 2^{-127}.

Например, число 1 в формате с плавающей точкой будет представлено битами 00111111 10000000 00000000 00000000, т. е. $1 = (-1)^0 \cdot 2^{127-127} \cdot 1.0$.

Дробное число переводится в двоичную форму записи следующим образом: сначала целая часть переводится в двоичную форму, затем ставится точка и вычисляются разряды дробной части. С этой целью дробная часть последовательно сравнивается с числом $1/2^i$, начиная с $i = 1$, т. е. сначала с 1/2, затем с 1/4, 1/8 и т. д. (пока полученное число удовлетворяет необходимой нам разрядности). Если дробная часть меньше, записывается 0, а если больше или равна, то записывается 1 и из дробной части вычитается это число. Затем остаток дробной части сравнивается со следующим числом $1/2^i$.

Например, пусть $x = 1/3$:

1. x меньше 1/2, поэтому записываем 0.
2. x больше 1/4, поэтому записываем 1 и вычисляем новое значение:
3. $x = 1/3 - 1/2 = 1/12$.
4. x меньше 1/8, записываем 0.
5. x больше 1/16, поэтому записываем 1 и вычисляем новое значение $x = 1/12 - 1/16$.

и т. д.

В результате получаем: 1/3 = 0,0101… (двоичное число).

Символьные данные

Символьные данные используются, например, для работы с текстовыми строками.

Символьные данные заслуживают отдельного описания, они как бы стоят в стороне. На них надо обратить особое внимание при реверсинге. Дело в том, что при вводе пароля или серийного номера в программу программа что-то делает с введенными данными, а потом сравнивает с данными, находящимися в самой программе. Из этого следует, что в программе уже есть эти данные и в большинстве своем они представлены именно как "символьные".

Обычно в ЭВМ используются восьмиразрядные коды символов ASCII. Отметим следующие особенности.

- Коды латинских букв упорядочены согласно алфавиту и идут без пропусков. Это очень важно! В частности, если к коду буквы `'a'` прибавить 1, то получится код буквы `'b'`, `'a' + 2` будет равно `'c'` и т. д.
- Коды цифр упорядочены по возрастанию и идут без пропусков, значит, все x в пределах `'0'` $\le x \le$ `'9'` будут цифрами.

Представление данных сопроцессора

Математический сопроцессор серии 80x87 может обрабатывать 7 типов данных:

- целое число со знаком;
- короткое целое со знаком;
- длинное целое со знаком;
- целое число в упакованном BCD-формате;
- короткое вещественное число;
- длинное вещественное число;
- расширенное вещественное число.

Диапазоны изменения чисел первых трех типов соответственно:

$-2^{15} \le x \le 2^{15} - 1$;

$-2^{31} \le x \le 2^{31} - 1$;

$-2^{63} \le x \le 2^{63} - 1$.

Поскольку в упакованном BCD-формате старший байт отводится под знак, в этом формате число состоит из 18 десятичных цифр и, значит, лежит в диапазоне $-10^{18} \le x \le 10^{18} - 1$.

Короткое вещественное число занимает 32 бита и было описано ранее.

В длинном формате порядок E занимает 11 бит, мантисса M — 53 бита. Значение числа, представленного в длинном формате, равно $(-1)^S \cdot 2^{E-1023} \cdot 1.M$.[1]

В расширенном формате порядок E занимает 15 бит, мантисса M — 64 бита. Значение такого числа: $(-1)^S \cdot 2^{E-16\,383} \cdot 1.M$.

Регистры микропроцессора

Регистром процессора называется ячейка процессора, отведенная для хранения чисел или предназначенная для специальных операций.

Другими словами, регистры — это ячейки, расположенные в центральном процессоре и доступные из программы. В программах на языке ассемблера регистры используются постоянно для различных целей — временного хранения данных, аргументов или результатов различных операций. В регистре может, например, храниться, флаг, сигнализирующий о том, зарегистрирована программа или нет, и т. д. Большинство регистров имеют определенное функциональное назначение. Но есть и регистры, которые можно без ограничения использовать для любых операций.

Поскольку для доступа к регистру адрес не нужен, есть возможность организовать операции между регистром и памятью с помощью команд, имеющих лишь один операнд, находящийся в памяти. Другое назначение регистров — адресация данных с помощью сегментных, базовых и индексных регистров. Наконец, информация о состоянии процессора также находится в одном из регистров.

Регистры общего назначения

При обработке данных компьютером значительная часть времени микропроцессора тратится на передачу данных между микропроцессором и памятью. Время доступа к данным значительно уменьшается, если часто используемые операнды и результаты вычислений хранить в самом микропроцессоре. Четыре регистра — EAX (AX), EBX (BX), ECX (CX) и EDX (DX) — предназначены специально для этих целей. Тут надо немного сказать о самих регистрах. Когда на заре компьютерной индустрии был изобретен процессор 8086—80286, все регистры были 16-битные и назывались они AX, BX, CX, DX. Каждый из них состоял из 8-битных старшего и младшего байтов. Так как программистам довольно часто приходилось использовать эти регистры,

[1] Бит знака хранится не в самом числе — для этого используется флаг переноса.

они получили свои имена — старшие байты этих регистров стали называть AH, BX, CX, DX, а младшие — AL, BL, CL, DL (рис. 2.6). После появления на свет процессора 80386, уже 32-битного, программистам стало довольно сложно и неудобно работать с этими регистрами — вместо одной команды пересылки 32-битных данных приходилось использовать две и больше. Вот для того чтобы облегчить труд программистов, и появились в процессоре 80386 32-битные регистры — EAX, EBX, ECX, EDX.[1] Таким образом, сейчас получается (см. рис. 2.6), что младшие 16 бит каждого 32-битного регистра могут использоваться как самостоятельные регистры. Все эти регистры называются *регистрами общего назначения*.

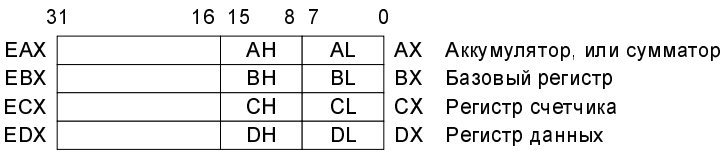

Рис. 2.6. Регистры общего назначения

Регистр AX (EAX). Является основным сумматором и применяется для арифметических операций, строковых операций, операций ввода/вывода. Команды умножения (`mul` и `imul`), деления (`div` и `idiv`), преобразования (`xlat`), коррекции (`aaa`, `aad`, `aam`, `aas`, `daa`, `das`) используют регистр AL — младший байт регистра AX. Старший байт регистра AX — регистр AH используется для задания функций обслуживающих подпрограмм (обработки прерываний).

Регистр BX (EBX). Применяется для вычислительных операций и в различных методах адресации. Это единственный регистр общего назначения, который можно использовать в качестве индекса элементов массива (для косвенно-регистровой адресации). Применяется в команде преобразования (`xlat`) для указания начала таблицы.

Регистр CX (ECX). Необходим для управления числом повторений в командах цикла (счетчик). Применяется в командах сдвига для указания числа бит, на которое сдвигается содержимое операнда. Содержит число повторений строковых операций при наличии префикса повторения.

Регистр DX (EDX). Применяется в командах ввода/вывода (`in` и `out`), умножения (`mul` и `imul`), деления (`div` и `idiv`), использующих регистровую пару DX:AX.

[1] E в названии регистра означает "32-битный".

Регистровые указатели и индексные регистры

Регистровые указатели SP и BP обеспечивают доступ к данным в сегменте стека. Регистр BP используется также в различных методах адресации.

Индексные регистры SI и DI применяются в строковых операциях и в различных методах адресации (рис. 2.7).

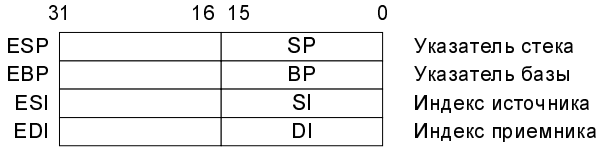

Рис. 2.7. Регистровые указатели и индексные регистры

Регистр SP (ESP). Указатель стека обеспечивает использование стека в памяти.

Стеком называется свободная область памяти, предназначенная для временного сохранения данных с целью их дальнейшего восстановления.

Адрес начала стека равен 16 × SS + SP, где SS — содержимое регистра сегмента стека.

Регистр BP (EBP). Облегчает доступ к параметрам, значения которых передаются с помощью сохранения данных в стеке.

Регистр SI (ESI). Является указателем источника. Применяется в строковых операциях. В строковых операциях связан с регистром сегмента данных DS. Адрес источника равен 16 · DS + SI.

Регистр DI (EDI). Является указателем приемника. В строковых операциях связан с дополнительным регистром сегмента данных ES.

Адрес приемника равен 16 · ES + DI.

Сегментные регистры

Как уже говорилось, физический адрес любой ячейки памяти состоит из 16-битного значения адреса сегмента и 16-битного смещения внутри сегмента. Каждый сегментный регистр используется для адресации определенного типа. Имеется четыре сегментных регистра: CS, DS, ES, SS. Микропроцессоры фирмы Intel, начиная с 80486, имеют два новых дополнительных сегментных регистра — FS и GS.

Регистр CS. Определяет сегмент кода — сегмент программы, содержащий выполняемые команды. Устанавливается при загрузке программы автоматически.

Регистр DS. Определяет сегмент данных или некоторую область памяти. В строковых операциях связан с регистром источника SI. Команды обработки данных по умолчанию, как правило, используют для адресации регистр DS. При загрузке программы значение регистра DS устанавливается программистом.

Регистр ES. Определяет дополнительный сегмент данных и применяется в тех случаях, когда требуется обратиться к произвольному сегменту памяти. В строковых операциях связан с регистром приемника DI. Инициализируется программистом.

Регистр SS. Регистр сегмента стека, связан с регистром SP. Содержит начальный адрес сегмента стека (в то время как SP устанавливается на конец стека). Регистры SS и SP при загрузке программы устанавливаются автоматически.

Регистры состояния и управления

В микропроцессоре присутствует несколько регистров, содержащих информацию о состоянии микропроцессора и программы, команды которой находятся в очереди команд ("загружены на конвейер"). К этим регистрам относятся счетчик команд IP (EIP), регистр флагов FL (EFL) и системные регистры.

Счетчик команд IP (EIP). Называется также *командным указателем*. Он связан с регистром сегмента кодов CS и содержит смещение команды, которая должна быть выполнена. При выполнении команды значение счетчика увеличивается за исключением тех случаев, когда команда относится к командам управления — перехода, цикла, вызова подпрограммы и возвращения из подпрограммы (включая программные прерывания).

Регистр флагов FL (EFL). Содержит 16 (32) бит. Отдельные биты имеют определенное функциональное назначение и называются *флагами*. Различают три типа флагов:

- системные флаги — отражают текущее состояние компьютера в целом и чаще используются операционной системой, а не программами пользователя;
- флаги состояния — изменяются после выполнения каждой команды;
- флаги управления.

Формат регистра флагов показан на рис. 2.8, назначение флагов приведено в табл. 2.1—2.3. Отметим, что бит 1 регистра флагов всегда равен единице. Остальные неиспользованные для флагов биты равны нулю (это относится, в частности, к битам 22—31).

Рис. 2.8. Регистр флагов EFL

Как мы знаем, микропроцессор может работать в одном из трех режимов — в реальном, защищенном и в режиме виртуального 8086 (V86). В разных режимах процессора флаги используются по-разному.

Таблица 2.1. *Системные флаги*

Флаг	Название	Бит	Назначение
ID	Флаг идентификации	21	Служит для проверки того, выполняет ли процессор команду `cpuid`. Если в программе можно установить и сбросить флаг ID, то данный процессор поддерживает команду `cpuid`, предоставляющую программисту информацию о продавце, модели и поколении данного процессора

Таблица 2.1 *(окончание)*

Флаг	Название	Бит	Назначение
VIP	Ожидание виртуального прерывания	20	Как и флаг VIF, позволяет каждой прикладной программе в многозадачном режиме иметь виртуальную версию флага IF
VIF	Флаг виртуального прерывания	19	Является виртуальным подобием флага IF и используется совместно с флагом VIP
AC	Контроль выравнивания	18	Предназначен для контроля выравнивания при обращениях к памяти. Используется совместно с битом AM в системном регистре CR0. К примеру, микропроцессор Pentium разрешает размещать команды и данные с любого адреса. Если требуется контролировать выравнивание данных и команд по адресам, кратным 2 или 4, то установка данных битов приведет к тому, что все обращения по некратным адресам будут возбуждать исключительную ситуацию
VM	Виртуальный режим	17	Признак работы микропроцессора в режиме виртуального 8086 (V86) — процессор работает в режиме V86 либо в реальном, или защищенном режиме (режим V86 является специальным подмножеством защищенного режима)
RF	Флаг возобновления	16	Служит для временного выключения обработки исключительных ситуаций отладки для того, чтобы команда, вызвавшая такую ситуацию, могла быть перезапущена и не стала бы причиной новой исключительной ситуации. Отладчик устанавливает этот флаг с помощью команды `iretd` при возврате в прерванную программу
IF	Разрешение прерываний	9	1 — микропроцессор воспринимает и соответственно реагирует на запрос прерывания по входу `intr`; 0 — прерывания по этому входу запрещаются. Значение флага IF не влияет на восприятие немаскируемых прерываний, выполняемых по команде `int`. Устанавливается командой `sti`, сбрасывается с помощью команды `cli`
TF	Флаг трассировки	8	1 — микропроцессор переходит в пошаговый режим работы, при котором генерируется прерывание 1 после выполнения каждой команды. Флаг можно установить с помощью команд `popf`, `popfd` или `iret`