Содержание

Используемый компилятор ... 9

Глава 1. Основы ... 11
1.1. Основные понятия ... 11
1.1.1. Что такое процессор? ... 11
1.1.2. Небольшая предыстория ... 14
1.1.3. Процессоры x86-64 ... 16
1.1.4. Регистры процессоров x86-64 ... 18
1.1.5. Память ... 19
1.1.6. Работа с внешними устройствами ... 21
1.1.7. Резюме ... 21
1.2. Основы ассемблера ... 22
1.2.1. Немного о языке ассемблера ... 22
1.2.2. Регистр флагов ... 23
1.2.3. Команда MOV ... 24
1.2.4. Формат хранения данных в памяти ... 26
1.2.5. Команды SUB и ADD ... 27
1.2.6. Логические операции ... 27
1.2.7. Сдвиги ... 28
1.2.8. Работа с флагами процессора ... 30
1.2.9. Работа со стеком ... 30
1.2.10. Резюме ... 30
1.3. Метки, данные, переходы ... 31
1.3.1. Данные ... 31
1.3.2. Метки ... 32
1.3.3. Переходы ... 35
1.3.4. Безымянные метки ... 38
1.3.5. Работа с битами ... 39
1.3.6. Резюме ... 39
1.4. Изучаем ассемблер подробнее ... 39
1.4.1. Работа с памятью и стеком ... 40
1.4.2. Работа с числами на ассемблере ... 41
1.4.3. Умножение и деление ... 44
1.4.4. Порты ввода-вывода ... 46
1.4.5. Циклы ... 46
1.4.6. Обработка блоков данных ... 47
1.4.7. Макросы ... 50

1.4.8. Структуры ... 52
1.4.9. Работа с MSR-регистрами ... 53
1.4.10. Команда CPUID ... 54
1.4.11. Команда UD2 ... 55
1.4.12. Включение файлов ... 55
1.4.13. Резюме ... 55

Глава 2. Защищённый режим ... 56
2.1. Введение в защищённый режим ... 56
2.1.1. Уровни привилегий ... 56
2.1.2. Сегменты в защищённом режиме ... 58
2.1.3. Глобальная дескрипторная таблица ... 61
2.1.4. Практика ... 63
2.1.5. Резюме ... 70
2.2. Прерывания в защищённом режиме ... 71
2.2.2. Дескрипторы шлюзов ... 72
2.2.3. Исключения ... 74
2.2.4. Коды ошибок ... 76
2.2.5. Программные прерывания ... 77
2.2.6. Аппаратные прерывания ... 77
2.2.7. Обработчик прерывания ... 79
2.2.8. Практика ... 80
2.2.9. Резюме ... 85
2.3. Механизм трансляции адресов ... 85
2.3.1. Что это такое? ... 85
2.3.2. Обычный режим трансляции адресов ... 87
2.3.3. Режим расширенной физической трансляции адресов ... 91
2.3.4. Обработчик страничного нарушения ... 94
2.3.5. Флаг WP в регистре CR0 ... 95
2.3.6. Практика ... 96
2.3.7. Резюме ... 102
2.4. Многозадачность ... 102
2.4.1. Общие сведения ... 102
2.4.2. Сегмент задачи (TSS) ... 103
2.4.3. Дескриптор TSS ... 105
2.4.4. Локальная дескрипторная таблица ... 105
2.4.5. Регистр задачи (TR) ... 106
2.4.6. Управление задачами ... 106
2.4.7. Шлюз задачи ... 109
2.4.8. Уровень привилегий ввода-вывода ... 109
2.4.9. Карта разрешения ввода-вывода ... 110
2.4.10. Включение многозадачности ... 110
2.4.11. Практическая реализация ... 111
2.4.12. Резюме ... 118

2.5. Механизмы защиты ... 119
2.5.1. Поля и флаги, используемые для защиты на уровне сегментов и страниц ... 119
2.5.2. Проверка лимитов сегментов ... 120
2.5.3. Проверки типов ... 120
2.5.4. Уровни привилегий ... 122
2.5.5. Проверка уровня привилегий при доступе к сегментам данных ... 123
2.5.6. Проверка уровней привилегий при межсегментной передаче управления ... 124
2.5.7. Шлюзы вызова ... 125
2.5.8. Переключение стека ... 128
2.5.9. Использование инструкций SYSENTER и SYSEXIT ... 129
2.5.10. Практика ... 130
2.5.11. Резюме ... 133

Глава 3. ПРОГРАММИРОВАНИЕ В WIN32 ... 134
3.1. Введение в Win32 ... 134
3.1.1. Основные сведения ... 135
3.1.2. Память в Win32 ... 135
3.1.3. Исполняемые компоненты Windows ... 136
3.1.4. Системные библиотеки и подсистемы ... 137
3.1.5. Модель вызова функций в Win32 ... 138
3.1.6. Выполнение программ в Win32: общая картина ... 138
3.1.7. Практика ... 139
3.1.8. Резюме ... 147
3.2. Программирование в третьем кольце ... 148
3.2.1. Общий обзор ... 148
3.2.2. Работа с объектами ... 149
3.2.3. Работа с файлами ... 149
3.2.4. Обработка ошибок API-функций ... 152
3.2.5. Консольные программы ... 152
3.2.6. GUI-программы ... 153
3.2.7. Динамически подключаемые библиотеки ... 156
3.2.8. Обработка исключений в программе ... 159
3.2.9. Практика ... 162
3.2.10. Резюме ... 171
3.3. Программирование в нулевом кольце ... 171
3.3.1. Службы ... 172
3.3.2. Общий обзор ... 173
3.3.3. Driver Development Kit (DDK) ... 174
3.3.4. Контекст потока и уровни запросов прерываний ... 175
3.3.5. Пример простого драйвера ... 176
3.3.6. Строки в ядре Windows ... 179

3.3.7. Подсистема ввода-вывода ... 180
3.3.8. Практика ... 186
3.3.9. Резюме ... 201

Глава 4. LONG MODE ... 202
4.1. Введение в long mode ... 202
4.1.1. Общий обзор ... 202
4.1.2. Сегментация в long mode ... 204
4.1.3. Механизм трансляции страниц ... 205
4.1.4. Переход в long mode ... 205
4.1.5. Практика ... 206
4.1.6. Резюме ... 208
4.2. Работа с памятью в long mode ... 208
4.2.1. Общий обзор ... 209
4.2.2. Страницы размером 4 Кб ... 209
4.2.3. Страницы размером 2 Мб ... 211
4.2.4. Страницы размером 1 Гб ... 212
4.2.5. Регистр CR3 ... 213
4.2.6. Проверки защиты ... 214
4.2.7. Практика ... 214
4.2.8. Резюме ... 221
4.3. Прерывания в long mode ... 221
4.3.1. Дескрипторы шлюзов ... 221
4.3.2. Таблица IDT, 64-битный TSS и механизм IST ... 222
4.3.3. Вызов обработчика прерывания ... 223
4.3.4. Практика ... 224
4.3.5. Резюме ... 230
4.4. Защита и многозадачность ... 230
4.4.1. Сегменты ... 231
4.4.2. Шлюзы вызова ... 231
4.4.3. Инструкции SYSCALL и SYSRET ... 232
4.4.4. Многозадачность ... 233
4.4.5. Практика ... 235
4.4.6. Резюме ... 238

Глава 5. ПРОГРАММИРОВАНИЕ В WIN64 ... 239
5.1. Введение в Win64 ... 239
5.1.1. Преимущества и недостатки ... 239
5.1.2. Память в Win64 ... 240
5.1.3. Модель вызова ... 240
5.1.4. Режим совместимости ... 242
5.1.5. Win64 API и системные библиотеки ... 242
5.1.6. Практика ... 243
5.1.7. Резюме ... 244

5.2. Программирование в Win64 ... 244
5.2.1. Изменения в типах данных ... 245
5.2.2. Выравнивание стека ... 245
5.2.3. GUI-приложения ... 246
5.2.4. Программирование драйверов ... 250
5.2.5. Отладка приложений в Win64 ... 254
5.2.6. Резюме ... 254

Глава 6. МНОГОПРОЦЕССОРНЫЕ СИСТЕМЫ ... 255

6.1. Работа с APIC ... 255
6.1.1. Общий обзор ... 255
6.1.2. Включение APIC ... 256
6.1.3. Local APIC ID ... 257
6.1.4. Локальная векторная таблица ... 257
6.1.5. Local APIC Timer ... 259
6.1.6. Обработка прерываний ... 261
6.1.7. Работа с I/O APIC ... 263
6.1.8. Практика ... 266
6.1.9. Резюме ... 270
6.2. Межпроцессорное взаимодействие ... 270
6.2.1. Общий обзор ... 270
6.2.2. Межпроцессорные прерывания ... 271
6.2.3. Синхронизация доступа к данным ... 273
6.2.4. Инициализация многопроцессорной системы ... 275
6.2.5. Практика ... 276
6.2.6. Резюме ... 280

ПРИЛОЖЕНИЯ ... 281

Приложение А. MSR-регистры ... 281
А.1. Регистр IA32_EFER ... 281
А.2. Регистры, используемые командами SYSENTER/SYSEXIT ... 281
А.3. Регистры, используемые командами SYSCALL/SYSRET ... 282
А.4. Регистры APIC ... 282
А.5. Регистры для управления сегментами в long mode ... 283
А.6. Вспомогательные регистры ... 283
Приложение Б. Системные регистры ... 283
Б.1. Регистр CR0 ... 283
Б.2. Регистры CR2 и CR3 ... 285
Б.3. Регистр CR4 ... 286
Б.4. Регистры GDTR и IDTR ... 287
Б.5. Регистры LDTR и TR ... 288
Б.6. Регистр флагов ... 288
Б.7. Регистр CR8 ... 290

Приложение B. Системные команды 290
B.1. Работа с системными регистрами 290
B.2. Системные команды 293
B.3. Работа с кэшем процессора 295
B.4. Дополнительные команды 295

Алфавитный указатель 297

Используемый компилятор

При компиляции всех примеров, приведённых в этой книге, необходимо использовать FASM. Почему я выбрал именно этот компилятор? По очень многим причинам.

1. Максимальный набор поддерживаемых команд. FASM поддерживает весь (или почти весь) набор команд x86-64.
2. Наличие Linux- и Windows-версий, а также поддержка широкого списка выходных файлов. Можно компилировать программы для Windows (формат PE) и для Linux (формат ELF).
3. Гибкий синтаксис и отсутствие совершенно бесполезных, но обязательных директив (например, .386, .486 и т. п.).
4. Полный контроль над размещением данных в исполняемом файле.
5. Мощнейший макросный движок, с помощью которого можно создать практически любой макрос и изменить текст программы и сам язык до неузнаваемости.
6. При компиляции через командную строку нет необходимости указывать множество опций компиляции. Для того чтобы получить исполняемый файл нужного типа, достаточно задать все необходимые параметры в самом исходнике.
7. Windows-версия FASM поставляется вместе с IDE, благодаря которой можно скомпилировать программу нажатием одной кнопки (или пункта меню).

Компилятор FASM можно найти на компакт-диске, прилагающемся к книге (папка fasm). Последнюю версию этого компилятора также можно бесплатно скачать на сайте flatassembler.net.

Глава 1. Основы

1.1. Основные понятия

Поскольку книга посвящена программированию на платформе x86-64 и все примеры в книге приведены на языке ассемблер, то было бы правильным первый раздел книги посвятить изложению основ программирования на ассемблере. Даже неподготовленный читатель после прочтения первого раздела книги сможет усвоить материал следующих разделов.

Ниже будет рассказано о принципе работы процессора, а также основных понятиях, необходимых при программировании на ассемблере и не только на ассемблере. Под *процессором* подразумеваются процессоры от Intel и его клоны (разумеется, главными клонами являются процессоры AMD). Итак, начнём.

1.1.1. Что такое процессор?

Что такое процессор? *Процессор* – это важнейшая часть любой компьютерной системы, именно на нём и происходит вся вычислительная деятельность (или почти вся).

Любая компьютерная система состоит из трёх частей: центральный процессор, системная память и внешние устройства. Все три компонента взаимодействуют друг с другом посредством шин; наиболее важная из них – системная шина: именно с её помощью процессор взаимодействует с системной памятью и с важнейшими контроллерами. Здесь нам понадобится ввести четвёртое понятие – *контроллер*. Контроллер может выполнять разные действия: быть посредником между процессором и внешними устройствами (контроллер ввода-вывода, контроллер прерываний, USB-контроллер и т. д.) либо выполнять некоторые специфичные операции (например, контроллер прямого доступа к памяти).

Основная задача памяти и контроллеров в целом понятна; какова же задача процессора? Основная задача процессора – это выполнение инструкций, которые находятся в памяти. Всё время после включения компьютера и до самого выключения процессор выполняет инструкции – за исключением некоторых случаев, когда он переходит в «подвешенное» состояние для экономии энергии. В зависимости от режима, в котором находится процессор, инструкции он выполняет поразному.

Что такое инструкция? *Инструкция*, или *машинная команда*, – это просто некоторый код, фактически обычное число, которое находится в памяти и обозначает некоторое действие. Инструкции бывают разные и разного размера – они могут

занимать от одного до нескольких байт. По сути, они дают указание процессору, что ему надо делать. Для того чтобы процессор «знал», откуда из памяти брать команды для выполнения, он использует специальный указатель инструкции. Выполнив очередную инструкцию, процессор обновляет этот указатель, так чтобы он указывал на следующую команду в памяти, и т. д. Если какая-либо инструкция не может быть выполнена, процессор генерирует исключение; если исключение не может быть обработано, то он перезагружается.

На рис. 1.1 приведена наиболее упрощённая схема устройства процессорной системы.

Линии, которыми соединяются блоки, – это шины: шина адреса, шина данных и шина управления. Все они объединяются в одну системную шину. Для расширения возможностей системы на системную шину «цепляются» контроллеры. Каждый контроллер исполняет команды, которые были посланы ему через системную шину.

Самая главная составляющая любой системы – это *память* (системная память). Память бывает двух видов: физическая и линейная.

Процессор тоже имеет свою память двух видов: кэш-память и регистровую. Дефакто *кэш-память* – это такая же системная память, просто находящаяся внутри кристалла процессора. Кэш-память является копией некоторой области основной памяти, к которой он часто обращается.

Регистровую память представляют собой регистры. *Регистр* – это такое устройство, которое хранит в себе некоторую информацию, т. е. некоторое значение. Разрядность значения определяет разрядность регистра. Одни регистры могут хранить только определённую информацию, другие – любую. Те регистры, которые могут хранить любую информацию, называются *регистрами общего назначения*. Остальные регистры напрямую управляют работой процессора; точнее сказать, сами они

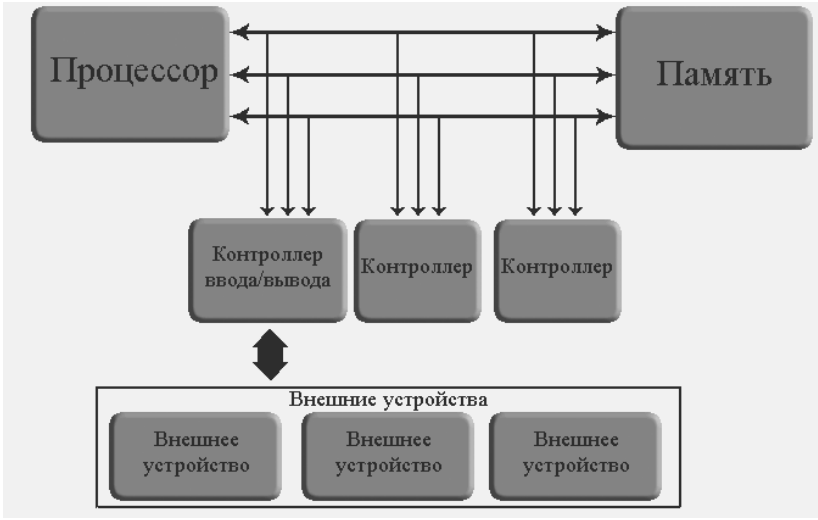

Рис. 1.1. Упрощённая схема устройства компьютера

не управляют, а процессор работает по-разному в зависимости от значений, которые принимают эти регистры. Регистры бывают разной разрядности, но большинство регистров 8-, 16-, 32- и 64-разрядные.

Для того чтобы получить значение, которое находится по определённому адресу памяти, процессор выставляет на шину адреса физический адрес, после чего на шине данных появляется значение. Конечно, появляется не мгновенно, но очень быстро – это зависит от скорости работы системной шины, памяти и других параметров. Если на шину адреса выведется значение памяти по адресу примерно большее 512 Мб, а на машине реально установлено 256 Мб, то в зависимости от материнской платы и памяти на шину данных может выйти просто «ерунда» или нулевое значение. Точно так же устанавливается и значение ячейки памяти. Вы спросите: «Как же материнская плата "узнает", что собирается сделать процессор?». На шину управления выставляется определённое значение, которое указывает материнской плате (или модулю памяти), что процессор хочет писать в память или читать из неё.

Всё вышесказанное само по себе вряд ли нам сможет пригодиться, но эта информация важна для понимания материала, который будет приводиться ниже. Обращения к определённым адресам памяти (которых обычно реально нет в системной памяти) могут перехватываться некоторыми котроллерами и перенаправляться к устройствам. Перехватив обращение к памяти, контроллер перенаправляет данные, которые должны быть помещены в память (или считаны из памяти) некоторому устройству. Таким образом, программа может взаимодействовать с внешними устройствами посредством записи/чтения из определённых адресов памяти. Это основной метод взаимодействия программ и устройств: например, так происходит взаимодействие с PCI-устройствами.

Как уже было сказано выше, адрес, который выставляется на шину адреса, – это и есть физический адрес. С виртуальной памятью всё будет сложнее. Виртуальная память может и не существовать реально, т. е. она «вроде бы есть, но её как бы нет». Если приложение операционной системы обращается к несуществующему адресу, процессор транслирует его в существующий адрес, а если памяти не хватает, то другая занятая память выгружается на внешнюю память (HDD, Flash и т. д.). Более подробно о виртуальной памяти будет говориться в следующих разделах, посвящённых программированию в защищённом режиме.

Как же происходит выполнение программ на процессоре? Все данные, которыми может оперировать процессор, находятся в памяти и только в памяти; если данные находятся не в памяти, а на внешних устройствах, то необходимо сначала загрузить эти данные в память и только потом с ними работать. Программа, дефакто, – это тоже данные, просто они представляют собой коды инструкций. Код инструкции называют *опкодом*. Опкод – это несколько байтов данных (от 1 до 10 и более), закодированных специальным образом, чтобы процессор мог понять, что от него «хотят». Иначе говоря, опкоды инструкций – это приказы процессору, которые тот беспрекословно выполняет, например: переслать данные из одного регистра в другой, из регистра в память, выполнить вычитание или сложение и т. д.

Процессор использует специальный регистр в качестве указателя инструкции EIP (IP, RIP). Этот регистр всегда содержит адрес следующей инструкции. После выполнения очередной инструкции процессор читает инструкцию из адреса, на который указывает регистр EIP, определяет её размер и обновляет адрес в регистре EIP (прибавляет к значению в регистре EIP размер текущей инструкции). После выборки инструкции из памяти и изменения регистра EIP процессор приступает к её выполнению. Инструкция может выполняться от одного до нескольких десятков процессорных тактов. В процессе выполнения инструкции может измениться адрес в регистре EIP. При изменении адреса в регистре EIP следующей будет выполнена именно та инструкция, адрес которой содержит регистр EIP – иными словами, произойдёт переход, или прыжок. После выполнения инструкции процессор читает следующую инструкцию, которая находится по адресу, содержащемуся в регистре EIP, и процесс повторяется. Вся сущность работы процессора заключается в выполнении команд. Однажды включившись, он всегда должен выполнять команды до тех пор, пока он не выключится или не перейдёт в специальное «подвешенное» состояние (ждущий режим).

Ещё одно важное понятие – это *стек*. Стек – это специальная область памяти, которая используется для хранения промежуточных данных. Представьте, что одна подпрограмма вызывает другую; вызываемая подпрограмма завершила свое выполнение, и теперь ей надо передать управление подпрограмме, которая её вызвала. Адрес команды, к которой надо вернуться после выполнения вызванной подпрограммы, находится на верхушке стека.

На рис. 1.2 приведён пример цепочки вызова подпрограмм и содержимого стека.

Стек – как магазин автомата: добавили адрес, добавили второй, добавили третий, и затем адреса достают в порядке, обратном добавлению: третий, второй, первый. Следовательно, подпрограмма может вызвать ещё одну подпрограмму, а та, в свою очередь, – ещё одну, и так сколько угодно – всё ограничивается только размером стека. На верхушку стека указывает определённый регистр; более подробно о стеке мы будем говорить в разделе 2.

На рис. 1.2 адреса возврата выделены рамкой. *Адрес возврата* – это адрес следующей инструкции после инструкции вызова (CALL). При выполнении команды возврата из подпрограммы (RET) адрес возврата извлекается из верхушки стека и происходит передача управления по этому адресу.

1.1.2. Небольшая предыстория

Исторически сложилось так, что все процессоры, которыми мы пользуемся, называются процессорами x86. Под процессорами x86 подразумевают следующие модели процессоров: 8086, 80186, 80286, 80386, 80486, 80586 (Pentium I), 80686 (Pentium II) и т. д. Процессоры 8086-80286 – 16-разрядные, все остальные – 32-разрядные. Процессоры Intel с поддержкой технологии EM64T и процессоры AMD с поддержкой технологии AMD64 являются 64-разрядными процессорами. Фактически технологии AMD64 и EM64T – это одно и то же; разработчиком этой технологии является компания AMD, а Intel всего лишь «лицензировала» (если это можно так назвать) технологию AMD64 у компании AMD.

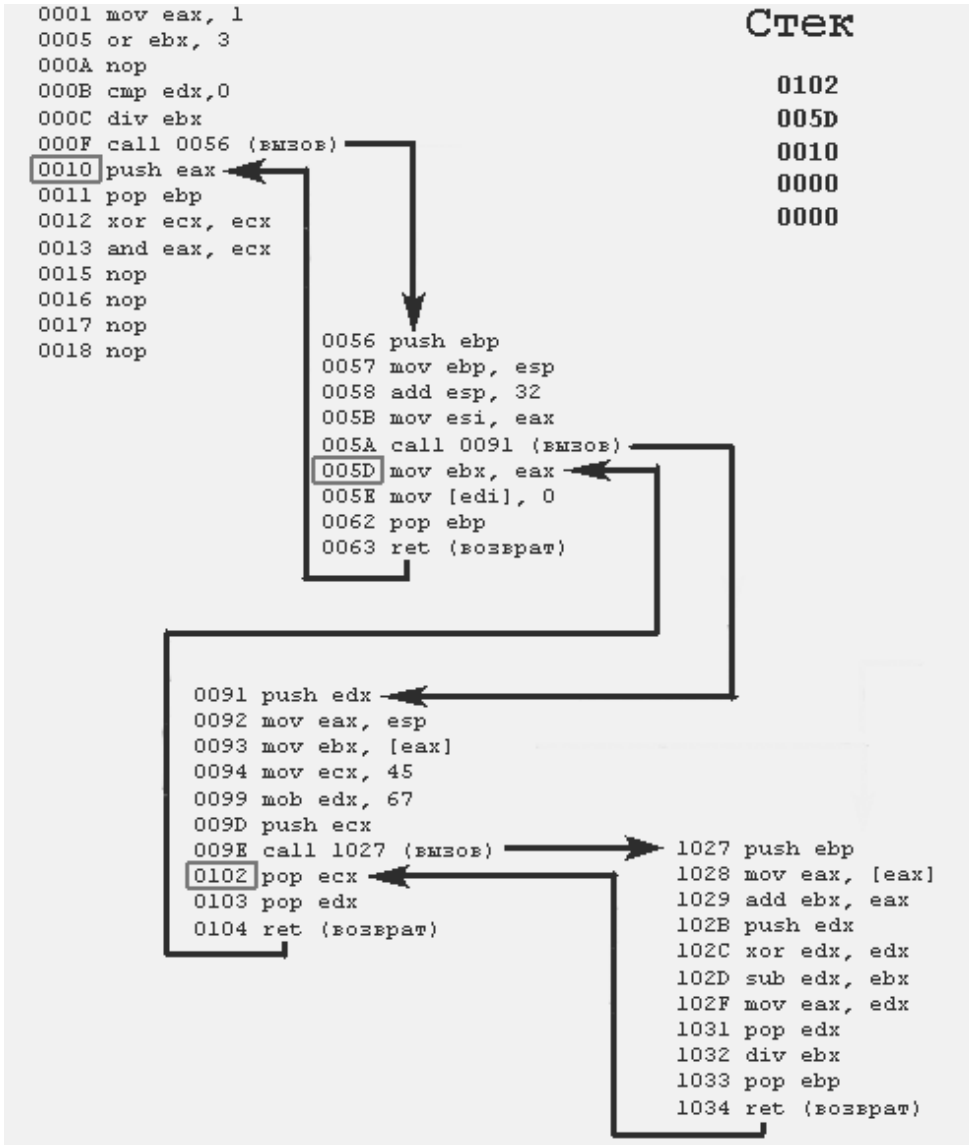

Рис. 1.2. Пример цепочки вызова подпрограмм и содержимое стека

Компания Intel тоже разработала свою 64-разрядную технологию под названием IA-64, но она настолько инновационная, что для её описания не хватит всей этой книги. По этой технологии компания Intel разработала свой процессор Itanium. Архитектура IA-64 (и процессоры Itanium) главным образом создавалась для использования на высокопроизводительных серверных системах; в этой книге она рассматриваться не будет.

Платформы, базирующиеся на процессорах с поддержкой AMD64 или EM64T, мы в дальнейшем будем называть x86-64 платформами. Часто платформа x86-64 упоминается под названием x64, но это более узкое понятие, которое подразумевает только 64-разрядный режим процессора. Платформа x86-64 полностью совместима с x86 и является её логическим развитием и продолжением. Платформы, базирующиеся на процессорах Itanium, будем называть IA-64. Не следует путать x86-64 и IA-64: это совершенно разные технологии.

1.1.3. Процессоры x86-64

Поскольку сейчас в основном выпускают только процессоры с поддержкой технологии AMD64 (EM64T), в этой книге речь пойдёт главным образом о работе процессоров x86-64. По сути, процессоры x86-64 – это почти все процессоры, которые можно встретить на рынке и при повседневном использовании.

Процессоры x86-64 в основном могут работать в трёх основных режимах: реальный режим, защищённый режим и 64-разрядный режим, или long mode (далее – long mode):

- *реальный режим* – это режим, в который переходит процессор после включения или перезагрузки. Это стандартный 16-разрядный режим, в котором доступно только 1 Мб физической памяти и возможности процессора почти не используются, а если и используются, то в очень малой степени. Иногда этот режим называют режимом реальных адресов, потому что в нем нельзя активировать механизм трансляции виртуальных адресов в физические. Это значит, что все адреса, к которым обращаются программы, являются физическими, т. е. без какого-либо преобразования будут выставлены на шину адреса. В этом режиме «родной» для процессора размер равен 2 байтам, или слову (WORD);
- *защищённый режим* (protected mode, или legacy mode по документации AMD) – это 32-разрядный режим; разумеется для процессоров x86 этот режим главный. В защищённом режиме 32-разрядная операционная система может получить максимальную отдачу от процессора – разумеется, если ей это потребуется. В этом режиме можно получить доступ к 4-гигабайтному физическому адресному пространству, если память, конечно, установлена на материнской плате, а при включении специального механизма трансляции адресов можно получить доступ к 64 Гб физической памяти. В защищённый режим можно перейти только из реального режима. Защищённый режим называется так потому, что позволяет защитить данные операционной системы от приложений. В этом режиме «родной» для процессора размер данных – это 4 байта, или двойное слово (DWORD). Все операнды, которые выступают в этом режиме как адреса, должны быть 32-битными;
- long mode («длинный режим», или IA-32e по документации Intel) – это собственно сам 64-разрядный режим. По своему принципу работы он почти полностью сходен с защищённым режимом, за исключением нескольких аспектов. В этом режиме можно получить доступ к 2^{52} байтам физической памяти и к 2^{48} байтам виртуальной памяти. В 64-разрядный режим можно

перейти только из защищённого режима. В этом режиме «родной» для процессора размер данных – это двойное слово (DWORD), но можно оперировать данными размером в 8 байт. Размер адреса всегда 8-байтовый.

Помимо приведённых выше режимов есть ещё один режим. Это режим системного управления (System Management Mode), в который процессор переходит при получении специального прерывания SMI. Режим системного управления предназначен для выполнения некоторых действий с возможностью их полной изоляции от прикладного программного обеспечения и даже операционной системы. Переход в этот режим возможен только аппаратно. Режим системного управления может использоваться для реализации системы управления энергосбережением компьютера или функций безопасности и контроля доступа.

Помимо вышеперечисленных режимов работы процессор поддерживает следующие два подрежима:

- *режим виртуального процессора 8086* – это подрежим защищённого режима для поддержки старых 16-разрядных приложений. Его можно включить для отдельной задачи в многозадачной операционной системе защищённого режима;
- *режим совместимости для long mode*. В режиме совместимости приложениям доступны 4 Гб памяти и полная поддержка 32-разрядного и 16-разрядного кода; «родной» для процессора размер данных – это двойное слово. Режим совместимости, можно сказать, представляет собой в long mode то же самое, что и режим виртуального 8086 процессора в защищённом режиме. Режим совместимости можно включить для отдельной задачи в многозадачной 64-битной операционной системе. В режиме совместимости размер адреса 32-битный, а размер операнда не может быть 8-байтовым.

Итак, мы узнали, в каких режимах может работать процессор, но в официальной документации производителей процессоров (Intel и AMD) немного другая терминология. Если следовать официальной документации, есть два режима работы процессора: 32-битный и 64-битный. 32-битный режим в документации от AMD называется legacy mode, а в документации от Intel носит название IA-32. Он включает в себя режим реальных адресов и защищённый режим. 64-битный режим в документации от AMD называется long mode, в документации от Intel – IA-32e; он включает в себя два подрежима: сам 64-разрядный режим и режим совместимости.

На рис. 1.3 изображены диаграмма режимов работы процессора и возможности перехода из одного режима в другой.

Из схемы видно, что в 64-битный режим можно перейти, только лишь предварительно включив защищённый режим. После перезагрузки процессора, в каком бы он режиме ни находился, он всё равно начнёт работать в режиме реальных адресов. Непосредственный переход в режим реальных адресов возможен только из защищённого режима, а находясь в других режимах это придется делать только посредством перезагрузки процессора.

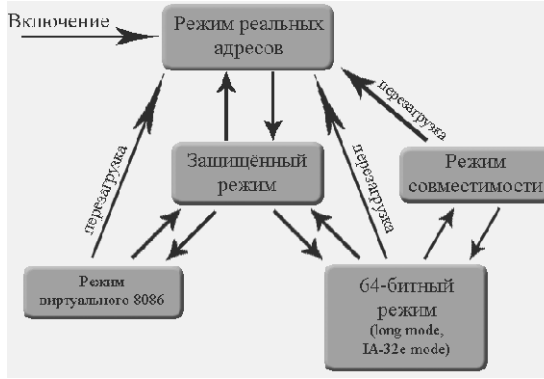

Рис. 1.3. Диаграмма режимов процессоров x86-64

В этой книге режим виртуального процессора 8086 и режим реальных адресов описаны не будут – литературы, где описывается программирование на ассемблере в режиме реальных адресов, существует предостаточно. Мы же рассмотрим только защищённый режим и 64-битный режим процессора.

1.1.4. Регистры процессоров x86-64

В защищённом режиме, в режиме реальных адресов и режиме совместимости доступны следующие регистры:

- регистры общего назначения: 32-разрядные EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP; 16-разрядные AX, BX, CX, DX, SI, DI, SP, BP (они являются младшими частями 32-разрядных регистров); 8-битные регистры AH, BH, CH, DH и AL, BL, CL, DL (старшие и младшие части 16-битных регистров соответственно);
- 32-разрядный EIP (IP в реальном режиме) – указатель инструкции;
- 16-разрядные сегментные регистры: CS, DS, SS, ES, FS, GS;
- 32-разрядный регистр флагов – EFLAGS;
- 80-битные регистры математического сопроцессора ST0-ST7 и др.;
- 64-битные MMX-регистры – MM0 – MM7;
- 128-разрядные XMM-регистры – XMM0 – XMM7 и 32-битный MXCSR;
- 32-разрядные регистры управления CR0 – CR4; регистры-указатели системных таблиц GDTR, LDTR, IDTR и регистр задачи TR;
- 32-разрядные регистры отладки – DR0 – DR3, DR6, DR7;
- MSR-регистры.

В режиме реальных адресов доступны не все вышеуказанные регистры, но регистры управления доступны в любом случае. В режиме реальных адресов нельзя использовать некоторые регистры размером более 16 бит.

При переключении процессора в 64-разрядный режим программе доступны следующие регистры:

- регистры общего назначения: 64-разрядные RAX, RBX, RCX, RDX, RSI, RDI, RSP, RBP и R8, R9, ..., R15; 32-разрядные EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, R8D – R15D (являются младшими частями 64-разрядных регистров); 16-разрядные AX, BX, CX, DX, SI, DI, SP, BP, R8W – R15W (являются младшими частями 32-разрядных регистров); 8-битные регистры AH, BH, CH, DH и AL, BL, CL, DL, SIL, DIL, SPL, BPL, R8L – R15L (старшие и младшие части 16-битных регистров соответственно);
- 64-разрядный RIP – указатель инструкции;
- 16-разрядные сегментные регистры: CS, DS, SS, ES, FS, GS;
- 64-разрядный регистр флагов – RFLAGS;
- 80-битные регистры математического сопроцессора ST0 – ST7;
- 64-битные MMX-регистры (MM0 – MM7);
- 128-разрядные XMM-регистры – XMM0 – XMM15 и 32-битный MXCSR;
- 64-разрядные регистры управления CR0 – CR4 и CR8; регистры-указатели системных таблиц GDTR, LDTR, IDTR и регистр задачи TR;
- 64-разрядные регистры отладки – DR0 – DR3, DR6, DR7;
- MSR-регистры.

Теперь немного пояснений. Регистры сегментов напрямую участвуют в формировании адресов, каждый сегментный регистр указывает на свой сегмент памяти, а именно: CS – сегмент кода, DS – сегмент данных, SS – сегмент стека; остальные три регистра дополнительные и могут не использоваться программой. Свободная работа с ними не всегда возможна; например в защищённом и 64-разрядном режимах загружать в них можно лишь определённые значения. В защищённом и 64-разрядном режимах доступность регистров зависит уровня привилегий, на котором выполняется программа.

Регистр общего назначения ESP (RSP) всегда указывает на верхушку стека, но при этом нам ничто не мешает использовать его в других целях, хотя тогда будет потеряна возможность нормальной работы со стеком. Вообще все регистры общего назначения можно свободно использовать в своих целях, но следует помнить, что некоторые регистры используются некоторыми командами: например, EBP (RBP) обычно указывает на начало фрейма в стеке, где хранятся локальные данные подпрограмм.

Указатель инструкции EIP (RIP) напрямую использовать нельзя – данный регистр используется самим процессором.

Регистры STn, MMn, XMMn используются математическим сопроцессором при работе с числами с плавающей точкой.

Регистры CRn, DRn, регистры-указатели системных таблиц, MSR-регистры являются системными и управляются ключевыми механизмами работы процессора.

Не следует сейчас заострять внимание на регистрах, их названиях и назначении: более подробно о регистрах будет рассказано ниже.

1.1.5. Память

Начнём с начала – с 70-х годов прошлого века. Тогда компьютер с памятью 64 Кб считался суперкомпьютером – если она была установлена, это было вообще чудо.

Для адресации такой памяти хватало 16 бит. После того как появились модули памяти 128 Кб и даже 256 Кб, регистров размером 16 бит стало не хватать для адресации памяти. Тогда ввели 16-битные сегментные регистры. Вся память делилась на сегменты размером 64 Кб. Сегмент задавался сегментным регистром, а адрес – обычным регистром или непосредственно 16-битным значением адреса. Но даже после этих новаций можно было адресовать только 1 Мб физической памяти из-за особенности формирования физического адреса и особенностей архитектуры процессора.

Всё кардинально изменилось, когда был выпущен первый полноценный 32-разрядный процессор (Intel 80386): он мог работать в двух режимах – в обычном 16-разрядном, как старые процессоры, и в защищённом, в котором можно было адресовать до 4 Гб физической памяти. На процессоре Intel 80386 появились новые 32-разрядные регистры, с помощью которых можно было адресовать эту память; также он поддерживал набор команд, позволяющий работать с данными размером 32 бита. Сегментные регистры остались, но они не слишком сильно влияли на формирование адреса, а предназначались для защиты определённых областей памяти, которые могла задавать операционная система; отсюда и название этого режима. Впрочем, для того чтобы получить доступ к 4 Гб памяти, необязательно надо было переходить в защищённый режим: такая возможность существовала и в режиме реальных адресов, но защищённый режим давал намного больше новых возможностей, чем просто расширение физического адресного пространства до 4 Гб.

После выпуска первых процессоров с поддержкой технологии AMD64 появился и 64-битный режим, который являлся вариантом защищённого режима, рассчитанным только на 64-битную архитектуру. В нём важность сегментных регистров была снижена до минимума, а защита данных операционной системы полностью возложена на механизм трансляции страниц.

В чем суть этого изложения краткой истории развития процессорной техники? Суть в том, чтобы показать, какие бывают адреса. Существует три вида адресов:

1. Физический.
2. Линейный (или виртуальный).
3. Логический.

С первым пунктом всё понятно: физический адрес – это адрес в системной памяти компьютера, именно тот адрес, который выставляется на шину адреса. Самое сложное – это третий пункт. Но начнём по порядку.

Для того чтобы получить доступ к некоторому значению в памяти (в любом режиме), приложение должно указать сегмент и адрес в этом сегменте. Сегмент указывается в сегментном регистре или непосредственно значением (это значение может быть только 16-битным), а адрес – в обычном регистре или непосредственно значением (это значение может быть 16-, 32-, 64-битным в зависимости от режима). Приложение может указать адрес разными способами, а сегмент можно вовсе не указывать. Если он не указан, то значение сегмента берётся из соответствующего (код, данные или стек) сегментного регистра. Хотите вы этого или

нет, адрес всегда будет в таком формате – «сегмент:смещение»; именно этот адрес и называется логическим.

После преобразования логического адреса (способ преобразования тоже зависит от режима процессора) получается абсолютный 20-, 32-, 64-битный адрес в зависимости от режима; этот адрес называется линейным. В режиме реальных адресов он сразу выставляется на шину адреса. В защищённом и 64-разрядном режимах этот адрес можно называть виртуальным, если активирован механизм трансляции страниц (подробнее об этом механизме рассказано в разделе 2.3). Если механизм трансляции страниц не активирован, то линейный адрес становится физическим, т. е. без преобразования выставляется на шину адреса. Если же механизм трансляции страниц включён, то линейный (он же виртуальный) адрес специальным образом преобразуется в физический адрес; способ преобразования задаётся самой операционной системой.

1.1.6. Работа с внешними устройствами

Одна из основных задач процессора – это взаимодействие с внешними устройствами. Если процессор не будет взаимодействовать с внешними устройствами, то какая от него польза? Теперь следует привести схему процессорной системы, немного отличную от той, которая была рассмотрена выше, в разделе 1.1.1 (см. рис. 1.4).

Все стрелки на схеме – это шины; некоторые из них находятся на системной шине, а некоторые – на шинах, связывающих устройства с процессором.

Процессор взаимодействует с внешними устройствами через порты ввода/вывода (не путайте их с портами LPT, PS/2 и т. д.) и через спроецированные на память регистры устройств. Это взаимодействие основано на том, что процессор выводит какие-либо данные в порты либо в определённые ячейки памяти, на которые спроецированы регистры устройств. На вышеприведённом рисунке посредником между процессором и устройствами выступает контроллер ввода-вывода (это общий случай; иногда посредником между процессором и внешним устройством может выступать какой-либо другой контроллер).

Но как процессор «узнаёт» о том, что устройство обработало данные или получило новые данные от пользователя? Если внешнее устройство хочет что-то «сказать» процессору (например, что оно обработало запрос или приняло от пользователя какие-либо данные), то оно через контроллер прерываний передаёт процессору сигнал о прерывании.

Итак, мы ввели новое понятие – *прерывание*. Прерывание – это сигнал процессору о том, чтобы он прервал выполнение текущей программы и передал управление специальной функции, называемой функцией-обработчиком прерывания. Программа через контроллер ввода-вывода получает необходимые данные для обработки этого запроса.

1.1.7. Резюме

В этом разделе мы познакомились с понятием «процессор», изучили основные режимы и принципы его работы. В разделе 1.2 речь пойдёт об основах ассемблера.