МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Курганский государственный университет»

Кафедра «Безопасность информационных и автоматизированных систем»

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Методические указания к выполнению лабораторных работ
для студентов очной и заочной формы обучения направления 09.03.04

Курган 2018

Кафедра: «Безопасность информационных и автоматизированных систем».
Дисциплина: «Архитектура вычислительных систем».
Составил: канд. техн. наук, В.А. Стукало.

Утверждены на заседании кафедры « 24 » ноября 2017 г.

Рекомендованы методическим советом университета «12 » декабря 2016 г.

Цель работы:

– углубление знаний программно-аппаратного обеспечения ПЭВМ, детальное изучение защищенного режима работы микропроцессора;

– получить минимальный опыт работы в защищенном режиме микропроцессора;

– закрепление навыков программирования на языке Assembler.

Оборудование: IBM-совместимая ПЭВМ.

ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Данная лабораторная работа включает в себя две части:

1 Освоение алгоритма перехода в защищенный режим и работа в защищенном режиме без прерываний.

2 Изучение работы прерываний в защищенном режиме.

Любой современный микропроцессор, находясь в реальном режиме, очень мало отличается от i8086. Это лишь его более быстрый аналог с увеличенным (до 32 бит) размером всех регистров, кроме сегментных. Чтобы получить доступ ко всем остальным архитектурным и функциональным новшествам микропроцессора, необходимо перейти в защищенный режим. Если бы мы могли проникнуть внутрь компьютера после перехода в защищенный режим, то увидели бы, что микропроцессор совершенно преобразился. Прежде всего, это стало бы заметно в изменении принципов работы микропроцессора с памятью. Она по-прежнему является сегментированной, но изменяются функции и номенклатура программно-аппаратных компонентов, участвующих в сегментации. Вспомните, что в реальном режиме работы микропроцессора сегмент был длиной не более 64 Кбайт, а адрес области памяти сегмента располагался в одном из сегментных регистров. Функциональное назначение сегмента определялось тем, в каком из шести сегментных регистров находился его адрес. Аппаратные средства контроля доступа к сегменту отсутствовали. Если и можно было организовать такой контроль, то только со стороны операционной системы. Реальный режим поддерживал выполнение всего одной программы. Для этого достаточно было простых механизмов распределения оперативной памяти и не было потребности в организации защиты программ от взаимного влияния и т.д. Поэтому все, что нужно было знать программе, — это адреса, по которым располагаются ее сегменты кода, данных и стека. Если бы вдруг появилась необходимость поместить в одну программно-аппаратную среду несколько независимых программ, то автоматически встал бы вопрос об их защите от взаимного влияния. Для решения этой проблемы микропроцессору уже недостаточно, ис-

пользуя сегментные регистры, знать, где располагаются сегменты программ. Для обеспечения совместной работы нескольких задач необходимо защитить их от взаимного влияния, а если возникает потребность во взаимодействии между ними, то оно должно обязательно регулироваться.

Чтобы ввести такое регулирование, нужно иметь больше информации о самих задачах. Можно предложить несколько вариантов структурной организации и размещения такой информации. Фирма Intel не стала нарушать принцип сегментации. Так как каждая задача в системе занимает один или несколько сегментов в памяти, то логично иметь больше информации о них, как об объектах, реально существующих в данный момент в системе. Если каждому из этих объектов присвоить определенные атрибуты, то часть контроля за доступом к ним можно переложить на сам микропроцессор. Что и было сделано. Любой сегмент памяти в защищенном режиме имеет следующие атрибуты:

- расположение сегмента в памяти;
- размер сегмента;
- уровень привилегий (определяет права данного сегмента относительно других сегментов);
- тип доступа (определяет назначение сегмента);
- некоторые другие.

Состав перечисленных атрибутов показывает, что в защищенном режиме микропроцессор поддерживает два типа защиты — по привилегиям и доступу к памяти. В отличие от реального режима, в защищенном режиме программа уже не может запросто обратиться по любому физическому адресу памяти. Для этого она должна иметь определенные полномочия и удовлетворять ряду требований.

Ключевым объектом защищенного режима является специальная структура — дескриптор сегмента, который представляет собой 8-байтовый дескриптор (краткое описание) непрерывной области памяти, содержащий перечисленные выше атрибуты (рисунок 1). Любая область памяти, которая логически может являться сегментом данных, стека или кода, должна быть описана таким дескриптором.

Все дескрипторы собираются вместе в одну из трех дескрипторных таблиц. В какую именно таблицу должен быть помещен дескриптор, определяется его назначением. Адрес, по которому размещаются эти дескрипторные таблицы, может быть любым; он хранится в специально предназначенном для этого адреса системном регистре.

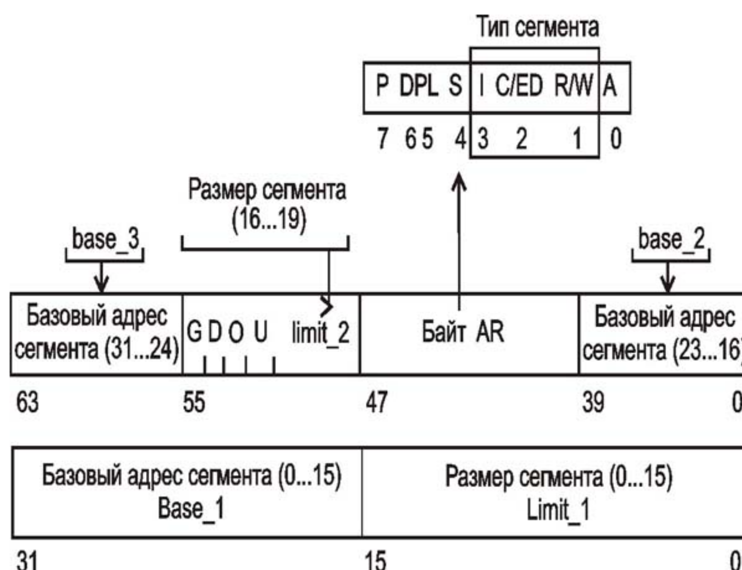

Рисунок 1 – Структура дескриптора сегмента защищенного режима

Это–ключевые моменты. Теперь подробно рассмотрим архитектуру микропроцессора в защищенном режиме и основные правила программирования этого режима.

Системные регистры микропроцессора

Само название этих регистров говорит о том, что они выполняют специфические функции в системе. Использование этих регистров жестко регламентировано. Именно они обеспечивают работу защищенного режима. Их также можно рассматривать как часть архитектуры микропроцессора, которая намеренно оставлена видимой для того, чтобы квалифицированный системный программист мог выполнить самые низкоуровневые операции.

Системные регистры можно разделить на три группы:

- четыре регистра управления;
- четыре регистра системных адресов;
- восемь регистров отладки.

В состав системных регистров микропроцессоров ряда Pentium введены следующие изменения:

- задействован ранее зарезервированный регистр управления cr4;
- введена группа MSR-регистров (MSR — Model Specific Register, модельно-зависимые регистры процессора), назначение и возможности которых привязаны к архитектуре конкретной модели микропроцессора. Ранее их функции частично выполняли тестовые регистры, вошедшие теперь в состав MSR-регистров. Для доступа к этим регистрам введены специальные команды.

Регистры управления

В группу регистров управления входят пять регистров: cr0, cr1, cr2, cr3, cr4. Эти регистры предназначены для общего управления системой. Регистры управления доступны только программам с уровнем привилегий 0. Хотя микропроцессор имеет пять регистров управления, доступными являются только четыре из них; исключается cr1, функции которого пока не определены (он зарезервирован для будущего использования).

Регистр cr0 содержит системные флаги, управляющие режимами работы микропроцессора и отражающие его состояние глобально, независимо от конкретных выполняющихся задач. Назначение системных флагов:

- pe (Protect Enable), бит 0 — разрешение защищенного режима работы. Состояние этого флага показывает, в каком из двух режимов — реальном (pe = 0) или защищенном (pe = 1) — работает микропроцессор в данный момент времени;
- mp (Math Present), бит 1 — наличие сопроцессора. Всегда 1;
- ts (Task Switched), бит 3 — переключение задач. Процессор автоматически устанавливает этот бит при переключении на выполнение другой задачи;
- am (Alignment Mask), бит 18 — маска выравнивания. Этот бит разрешает (am = 1) или запрещает (am = 0) контроль выравнивания;
- cd (Cache Disable), бит 30 — запрещение кэш-памяти. С помощью этого бита можно запретить (cd = 1) или разрешить (cd = 0) использование внутренней кэш-памяти (кэш-памяти первого уровня);
- pg (PaGing), бит 31 — разрешение (pg = 1) или запрещение (pg = 0) страничного преобразования. Регистр с г 0 используется при страничной модели организации памяти.

Регистр cr2 используется при страничной организации оперативной памяти для регистрации ситуации, когда текущая команда обратилась по адресу, содержащемуся в странице памяти, отсутствующей в данный момент времени в памяти. В такой ситуации в микропроцессоре возникает исключительная ситуация с номером 14, и линейный 32-битный адрес команды, вызвавшей это исключение, записывается в регистр cr2. Имея эту информацию, обработчик исключения 14 определяет нужную страницу, осуществляет ее подкачку в память и возобновляет нормальную работу программы.

Регистр cr3 также используется при страничной организации памяти. Это так называемый регистр каталога страниц первого уровня. Он содержит 20-битный физический базовый адрес каталога страниц текущей задачи. Этот каталог содержит 1024 32-битных дескриптора, каждый из которых содержит адрес таблицы страниц второго уровня. В свою очередь, каждая из таблиц стра-

ниц второго уровня содержит 1024 32-битных дескриптора, адресующих страничные кадры в памяти. Размер страничного кадра — 4 Кбайт.

Регистр cr4 содержит признаки, в основном разрешительного характера, которые характеризуют те или иные архитектурные элементы, впервые появившиеся в различных моделях микропроцессоров Pentium. В качестве примеров таких свойств -можно привести следующие: поддержка 36-разрядной адресации, использование отложенных прерываний в режиме виртуального i8086, поддержка страниц размером 4 Мбайт и т.д. Устанавливая в регистре cr4 те или иные биты, можно включать или отключать поддержку этих свойств.

Регистры системных адресов

Эти регистры еще называют регистрами управления памятью. Они предназначены для защиты программ и данных в мультизадачном режиме работы микропроцессора. При работе в защищенном режиме микропроцессора адресное пространство делится на:

- глобальное — общее для всех задач;
- локальное — отдельное для каждой задачи.

Этим разделением и объясняется то, что в архитектуре микропроцессора присутствуют следующие системные регистры:

- регистр таблицы глобальных дескрипторов gdtr (рисунок 2) (Global Descriptor Table Register) имеет размер 48 бит и содержит 32-битный (биты 16-47) базовый адрес глобальной дескрипторной таблицы GDT и 16-битное (биты 0–15) значение предела, представляющее собой размер в байтах таблицы GDT;

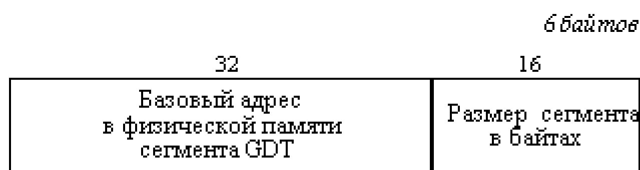

Рисунок 2 – Формат регистра GDTR

- регистр таблицы локальных дескрипторов ldtr (Local Descriptor Table Register) имеет размер 16 бит и содержит так называемый селектор дескриптора локальной дескрипторной таблицы LDT. Этот селектор является указателем в таблице GDT, который и описывает сегмент, содержащий локальную дескрипторную таблицу LDT;
- регистр таблицы дескрипторов прерываний idtr (Interrupt Descriptor Table Register) имеет размер 48 бит и содержит 32-битный (биты 16-47) базовый адрес дескрипторной таблицы прерываний IDT и 16-битное (биты 0-15) значение предела, представляющее собой размер в байтах таблицы IDT;

- 16-битный регистр задачи tr (Task Register) подобно регистру Idtr содержит селектор, то есть указатель на дескриптор в таблице GDT. Этот дескриптор описывает текущий сегмент состояния задачи (TSS — Task Segment Status). Этот сегмент создается для каждой задачи в системе, имеет жестко регламентированную структуру и содержит контекст (текущее состояние) задачи. Основное назначение сегментов TSS — сохранять текущее состояние задачи в момент переключения на другую задачу.

Регистры отладки

Это очень интересная группа регистров, предназначенных для аппаратной отладки. Средства аппаратной отладки впервые появились в микропроцессоре i486. Аппаратно микропроцессор содержит восемь регистров отладки, но реально из них используются только шесть.

Регистры dr0, dr1, dr2, dr3 имеют разрядность 32 бита и предназначены для задания линейных адресов четырех точек прерывания. Используемый при этом механизм следующий: любой формируемый текущей программой адрес сравнивается с адресами в регистрах dr0... dr3, и при совпадении генерируется исключение отладки с номером 1.

Регистр dr6 называется регистром состояния отладки. Биты этого регистра устанавливаются в соответствии с причинами, которые вызвали возникновение последнего исключения с номером 1. Перечислим эти биты и их назначение:

- b0 – если этот бит установлен в 1, то последнее исключение (прерывание) возникло в результате достижения контрольной точки, определенной в регистре dr0;
- b1 – аналогично b0, но для контрольной точки в регистре dr1;
- b2 – аналогично b0, но для контрольной точки в регистре dr2;
- b3 – аналогично b0, но для контрольной точки в регистре dr3;
- bd (бит 13) служит для защиты регистров отладки;
- bs (бит 14) устанавливается в 1, если исключение 1 было вызвано состоянием флага tf = 1 в регистре eflags;
- bt (бит 15) устанавливается в 1, если исключение 1 было вызвано переключением на задачу с установленным битом ловушки в TSS t = 1.

Все остальные биты в этом регистре заполняются нулями. Обработчик исключения 1 по содержимому dr6 должен определить причину, по которой произошло исключение, и выполнить необходимые действия.

Регистр dr7 называется регистром управления отладкой. В нем для каждого из четырех регистров контрольных точек отладки имеются поля, с помо-

щью которых можно уточнить следующие условия, при которых следует сгенерировать прерывание:

- место регистрации контрольной точки — только в текущей задаче или в любой задаче. Соответствующие биты занимают младшие восемь бит регистра dr7 (по два бита на каждую контрольную точку (фактически, точку прерывания), задаваемую регистрами dr0, drl, dr2, dr3). Первый бит из каждой пары — это так называемое локальное разрешение, его установка говорит о том, что точка прерывания действует, если она находится в пределах адресного пространства текущей задачи. Второй бит в каждой паре определяет глобальное разрешение, которое говорит о том, что данная контрольная точка действует в пределах адресных пространств всех задач, находящихся в системе;
- тип доступа, по которому инициируется прерывание: только при выборке команды, при записи или при записи/чтении данных. Биты, определяющие природу возникновения прерывания, локализуются в старшей части данного регистра.

Структуры данных защищенного режима

В защищенном режиме любой запрос к памяти как со стороны операционной системы, так и со стороны прикладных программ должен быть санкционирован. Микропроцессор аппаратно контролирует доступ программ к любому адресу в оперативной памяти. Для получения доступа целевой адрес, к которому хочет получить доступ программа, должен быть описан для программы. Это означает, что участок физической памяти, содержащий нужный адрес, должен быть описан с помощью некоторого дескриптора сегмента, который помещается в одну из трех дескрипторных таблиц. Локализация этих таблиц осуществляется с использованием одного из рассмотренных нами системных регистров — gdtr, ldtr или idtr. Программе, которая желает использовать данный участок памяти, должен быть сообщен указатель на соответствующий дескриптор в одной из двух дескрипторных таблиц — GDT или LDT. Что касается таблицы IDT, то работа с ней осуществляется по несколько иному принципу, поэтому о ней мы поговорим позже. Указатель на дескриптор сегмента в одной из таблиц GDT или LDT, в зависимости от функционального назначения описываемого дескриптором участка памяти (сегмента), помещается в один из шести сегментных регистров. Таким образом, в защищенном режиме меняется роль сегментного регистра – теперь он содержит уже не адрес, а селектор или индекс в таблице дескрипторов сегментов. Но само назначение сегментных регистров не меняется – они по-прежнему указывают на сегменты команд, данных и стека, но делают это, используя принципиально иные механизмы.

В защищенном режиме размер сегмента не фиксирован, его расположение можно задать в пределах 4 Гбайт, то есть он может занимать все возможное физическое пространство памяти. Как это возможно, если суммарный размер поля размера сегмента всего 20 бит, что соответствует величине 1 Мбайт? Секрет скрыт в поле гранулярности — бит G (рисунок 1). Если бит G = 0, то значение в поле размера сегмента означает размер сегмента в байтах, а если G = 1, то в страницах. Размер страницы составляет 4 Кбайт. Нетрудно подсчитать, что когда максимальное значение поля размера сегмента составляет 0ffffh, то это соответствует 1 М страниц, что и соответствует величине 1 М * 4 Кбайт = 4 Гбайт.

Из вышесказанного понятно, что выведение информации о базовом адресе сегмента и его размере на уровень микропроцессора позволяет аппаратно контролировать работу программ с памятью и предотвращать обращения по несуществующим адресам либо по адресам, находящимся вне предела, разрешенного полем размера сегмента limit.

Другой аспект защиты заключается в том, что сегменты неравноправны в правах доступа к ним. Информация об этом содержится в специальном байте AR, входящем в состав дескриптора. Формат этого байта приведен в виде выноски на рисунке 1.

Наиболее важные поля байта AR — это dpl и биты R/W, C/ED и I, которые вместе определяют тип сегмента. Поле dpl — часть механизма защиты по привилегиям. Суть этого механизма заключается в том, что конкретный сегмент может находиться на одном из трех уровней привилегированности с номерами 0, 1, 2 и 3. Самым привилегированным является уровень 0. Существует ряд ограничений на взаимодействие сегментов с различными уровнями привилегий.

Полю типа сегмента мы уделим больше внимания, так как оно понадобится нам при разработке программы. Это поле определяет целевое назначение сегмента. Возможны два принципиально разных вида сегментов: данных и кода. Сегмент стека является разновидностью сегмента данных, но с особой трактовкой поля размера сегмента. Это объясняется спецификой использования стека (он растет в направлении младших адресов памяти). Таким образом, видно, что поле типа сегмента ограничивает использование объявленных сегментов. В частности, программные сегменты не могут быть модифицированы без применения специальных приемов. Доступ к сегменту данных также может быть ограничен только на чтение.

Итак, в защищенном режиме перед использованием любой области памяти должна быть проведена определенная работа по инициализации соответ-

ствующего дескриптора. Эту работу выполняет операционная система или программа, сегменты которой также описываются подобными дескрипторами.

Интерес представляет то, каким образом микропроцессор переходит в защищенный режим. Сразу после включения питания или нажатия кнопки сброса микропроцессор начинает свою работу в реальном режиме. В этом режиме он производит действия по тестированию аппаратуры компьютера. После успешного завершения тестирования микропроцессор выполняет начальную загрузку системы, используя программу начальной загрузки, хранящейся на нулевой дорожке диска. Программа начальной загрузки считывает с диска программу инициализации операционной системы и передает ей управление. Действие этой программы зависит от того, в каком режиме работы микропроцессора будет осуществляться дальнейшее функционирование системы. Если в реальном режиме, то операционная система формирует среду и структуры данных для работы в этом режиме. Если же загружаемая операционная система собирается дальше работать в защищенном режиме, то она должна в него специальным образом перейти. Но прежде чем сделать это, операционная система формирует системные структуры данных (в частности, рассмотренные нами дескрипторные таблицы) для работы в защищенном режиме. Затем можно переходить в защищенный режим и выполнять дальнейшие действия.

Следует также учитывать несколько организационных моментов. Программа по переходу в защищенный режим должна начинать свою работу в реальном режиме, т.е. либо в операционной системе MS-DOS, либо в режиме эмуляции MS-DOS в Windows. Запустить программу прямо из многозадачной операционной системы, которой является, например, Windows 95, нельзя. Дело в том, что в целях защиты любая операционная система защищенного режима перекрывает на определенном уровне доступ к системным ресурсам со стороны программ пользователя с тем, чтобы не нарушить свою работу. В этом режиме наша программа, написанная как приложение MS-DOS, будет выполняться в режиме виртуального 8086 с уровнем привилегий 3. Чтобы получить доступ к ресурсам микропроцессора, управляющим сегментацией, ей необходимо иметь уровень привилегий 0. Кроме того, находясь в защищенном режиме, не имеет смысла говорить о проблеме перехода в него.

Следовательно есть две возможности — загрузить MS-DOS с помощью системной дискеты либо выполнить перезагрузку своего компьютера с тем, чтобы перевести Windows в режим эмуляции MS-DOS.

Но одной загрузки MS-DOS мало, необходимо проследить, чтобы в файле config.sys были отключены все драйверы расширенной памяти типа emm366.exe, так как они неявно переводят микропроцессор в защищенный режим работы, исключая доступ к структурам данных этого режима.

ЗАДАНИЕ НА ПЕРВУЮ ЧАСТЬ ЛАБОРАТОРНОЙ РАБОТЫ

Разработать программу по переводу микропроцессора в защищенный режим и обратно в реальный режим в соответствии со своим вариантом задания. А также выполнить какое-либо действие в защищенном режиме (уточнить действие у преподавателя). В отчете по работе поместить задание, исходный текст программы с комментариями, вывод по результату работы программы.

Защитить работу у преподавателя.

Варианты заданий

Таблицу глобальных дескрипторов GDT необходимо разместить в памяти по следующим абсолютным адресам:

Вариант 1: 02700h.

Вариант 2: 02800h.

Вариант 3: 02900h.

Вариант 4: 03000h.

Вариант 5: 04000h.

МЕТОДИКА ВЫПОЛНЕНИЯ РАБОТЫ

Рассмотрим фрагменты программы, которые подготавливают структуры данных защищенного режима, переводят микропроцессор в этот режим, имитируют работу путем вывода некоторого сообщения, после чего переводят микропроцессор обратно в реальный режим и заканчивают выполнение. Давайте перечислим и обсудим действия, необходимые для обеспечения функционирования этой программы:

1 Подготовка в оперативной памяти таблицы глобальных дескрипторов GDT.

2 Инициализация необходимых дескрипторов в таблице GDT.

3 Загрузка в регистр gdtr адреса и размера таблицы GDT.

4 Запрет обработки аппаратных прерываний.

5 Переключение микропроцессора в защищенный режим.

6 Организация работы в защищенном режиме:

- настроить сегментные регистры;
- выполнить собственно содержательную работу программы; в данном случае нужно просто обозначить сам факт успешного перехода в защищенный режим;
- подготовиться к возврату в реальный режим;
- запретить аппаратные прерывания.

7 Переключение микропроцессора в реальный режим.

8 Настройка сегментных регистров для работы в реальном режиме.

9 Разрешение прерываний.

10 Стандартное для MS-DOS завершение работы программы.

Рассмотрим каждый этап программы более подробно.

Подготовка таблиц глобальных дескрипторов GDT

Для того чтобы собрать информацию о всех программных объектах, находящихся в данный момент в памяти и в системе в целом, их дескрипторы собираются в таблицы, которые представляют собой массивы 8-байтовых элементов.

Микропроцессор аппаратно поддерживает три типа дескрипторных таблиц.

1 Таблица GDT (Global Descriptor Table) — глобальная дескрипторная таблица. Это основная общесистемная таблица, к которой допускается обращение со стороны программ, обладающих достаточными привилегиями. Расположение таблицы GDT в памяти произвольно; оно локализуется с помощью специального регистра gdtr. В таблице GDT могут содержаться следующие типы дескрипторов:

- дескрипторы сегментов кодов программ;
- дескрипторы сегментов данных программ;
- дескрипторы стековых сегментов программ;
- дескрипторы TSS (Task Segment Status) − специальные системные объекты, называемые сегментами состояния задач;
- дескрипторы для таблиц LDT;
- шлюзы вызова;
- шлюзы задач.

2 Таблица LDT (Local Descriptor Table) −локальная дескрипторная таблица. Для любой задачи в системе может быть создана своя дескрипторная таблица, подобно общесистемной GDT. Тем самым адресное пространство задачи локализуется в пределах, установленных набором дескрипторов таблицы LDT. Для связи между, таблицами GDT и LDT в таблице GDT создается дескриптор, описывающий область памяти, в которой находится LDT. Расположение таблицы LDT в памяти также произвольно и локализуется с помощью специального регистра ldtr. В таблице LDT могут содержаться следующие типы дескрипторов:

- дескрипторы сегментов кодов программ;
- дескрипторы сегментов данных программ;
- дескрипторы стековых сегментов программ;
- шлюзы вызова;
- шлюзы задач.

3 Таблица IDT (Interrupt Descriptor Table) — дескрипторная таблица прерываний. Данная таблица также является общесистемной и содержит дескрипторы специального типа, которые определяют местоположение программ обработчиков всех видов прерываний. В качестве аналогии можно привести таблицу векторов прерываний реального режима. Расположение таблицы IDT в памяти произвольно и локализуется с помощью специального регистра idtr. Элементы данной таблицы называются шлюзами. Шлюзы бывают трех типов:

- шлюзы задач;
- шлюзы прерываний;
- шлюзы ловушек.

Каждая из дескрипторных таблиц может содержать до 8192 (2^{13}) дескрипторов. Данное значение определяется размерностью поля в сегментном регистре. В защищенном режиме роль дескрипторов изменяется по сравнению с реальным режимом, и это отражается даже на их названии — в защищенном режиме они называются селекторами сегментов.

Так как наша программа претендует лишь на роль фрагмента операционной системы, то вполне достаточно определить пока только одну дескрипторную таблицу — глобальную дескрипторную таблицу GDT. Таблицу LDT есть смысл применять, когда в системе работают несколько задач и необходимо изолировать их друг от друга. В данном случае этого не требуется. Тем не менее, отметим следующее.

1 Дескрипторы, описывающие сегменты некоторой программы, могут содержаться как в глобальной (GDT), так и в локальной (LDT) дескрипторных таблицах. Сегментные регистры содержат селекторы, которые являются указателями на дескрипторы, описывающие соответствующие области памяти.

2 Структура сегментного регистра в защищенном режиме представляется тремя полями. Поле RPL, занимающее два младших бита 0 и 1 (Request Privilege Level — запрашиваемый уровень привилегий), используется в механизме ограничения доступа по привилегиям. А вот состояние однобитового поля TI, занимающего бит 2 сегментного регистра, как раз и определяет, с какой именно таблицей идет работа:

- если TI = 0, то сегментный регистр содержит селектор на дескриптор в глобальной дескрипторной таблице GDT;
- если TI = 1, то сегментный регистр содержит селектор на дескриптор в локальной дескрипторной таблице LDT.

3 Сегментный регистр содержит поле селектора. Оно определяет число, кратное восьми (так как три младшие бита заняты под поля RPL и TI), являющееся указателем на дескриптор в одной из дескрипторных таблиц в соответствии со значением бита в поле TI. Управлять состоянием этого бита может либо сама

программа, но тогда она должна обладать достаточным уровнем привилегий, либо операционная система, обеспечивающая ее работу. Для самой же программы ничего не меняется. По-прежнему базовые адреса ее сегментов определяются с помощью сегментных регистров, хотя и по-иному, чем в реальном режиме, принципу. В каждый момент времени микропроцессор может работать только с одной дескрипторной таблицей: GDT или LDT. В данном случае будем считать, что все сегменты программы находятся в глобальной дескрипторной таблице GDT, то есть TI = 0 во всех используемых сегментных регистрах.

Определимся теперь с набором дескрипторов в таблице GDT, которые понадобятся для нашей программы:

- дескриптор для описания сегмента самой таблицы GDT;
- дескриптор для описания сегмента данных программы;
- дескриптор для описания сегмента команд программы;
- дескриптор для описания сегмента стека программы;
- дескриптор для описания сегмента, в котором будет находиться процедура для выдачи сигнала сирены;
- дескриптор для описания видеопамяти.

Формат дескриптора сегмента показан на рисунке 1. В программе его удобнее описать в виде структуры:

```
descr struc
limit     dw     0
base_1 dw     0
base_2 db      0
atr       db     0
lim_atr db     0
base_3 db      0
ends
```

Используя этот шаблон структуры, опишем таблицу GDT как массив структур:

```
gdt_seg    segment        para
gdt_0       descr <0,0,0,0,0,0>
gdt_gdt_8 descr <0,0,0,0,0,0>
gdt_ldt_10         descr <0,0.0,0,0.0>
gdt_ds_18 descr <0,0,0,0,0.0>
gdt_es_vbf_20    descr <0,0,0.0,0,0
gdt_ss_28 descr <0,0,0,0,0,0>
gdt_cs_30 descr <0,0,0,0,0,0>
gdt_size=$-gdt_0-l;       определение размера таблицы GDT
gdt_seg    ends
```

Инициализация дескрипторов в таблице GDT

Определившись с набором дескрипторов и сформировав глобальную дескрипторную таблицу, нужно разобраться с тем, где брать информацию для их заполнения и как это делать. Рассмотрим этот вопрос для каждого поля в отдельности:

- поле limit — размер сегмента. В это поле нужно поместить точный размер сегмента за вычетом единицы, так как адресация идет от нуля. Наиболее оптимальный способ заключается в использовании оператора $ — извлечь текущее значение счетчика адреса. Пример использования этого оператора для определения размера приведен выше при описании таблицы GDT;

- поля base_1, base_2, base_3 — поля 32-разрядного базового адреса. Этот адрес будет известен после загрузки программы в оперативную память, и поля придется загружать на этапе выполнения при подготовке к переходу в защищенный режим. Фрагмент программы заполнения поля базового адреса для дескриптора gdt_gdt_8 может выглядеть так:

```
xor     eax.eax
mov    ax,gdt_seg ;адрес сегмента в ax
 ;сдвигом на 4 разряда получим физический
;20-разрядный адрес сегмента gdt_seg:
shl  eax,4
mov  base_1,ax
rol  eax, 16;  меняем для получения оставшейся части адреса
mov  base_2,al
```

Так как эту операцию придется проделывать несколько раз, для каждого из сегментов программы, то для удобства работы и повышения наглядности оформим этот фрагмент в виде макроса load_addr:

```
load_addr       macro descr,seg_addr,seg_size
mov      descr.limit,seg_size
xor      eax,eax
mov       ax,seg_adr
shl       eax,4
mov       descr.base_l,ax
rol      eax,16
mov       descr.base_2,al endm
```

Заодно к операциям, выполняемым макросом, было добавлена и инициализация поля предела;

- поле atr — байт атрибутов. Понятно, что для конкретного сегмента его значение будет всегда константой. Чтобы не запутывать себя окончательно, значение этого байта в целом удобнее формировать как логическую сумму нужных значений его полей для определения типа конкретного сегмента. Сформируем элементарные константы полей этого байта.

;биты 0, 4, 5, 6, 7 – постоянная часть
;байта AR для всех типов сегментов
constequ 10010000b
;бит 1 – доступность сегментов по чтению/записи
code_r_n equ 00000000b ;сегмент кода
;чтение запрещено
code_r_y equ 00000010b ; сегмент кода
;чтение разрешено
data_wm_n equ 00000000b;сегмент данных
;модификация запрещена
data_wm_y equ 00000010b;сегмент данных
;модификация разрешена
;бит 2 - тип сегмента
code_n equ 00000000b ;обычный сегмент кода
code_p equ 00000100b ; подчиненный сегмент кода
_data equ 00000000b ;сегмент данных
_stack equ 00000l00b ;сегмент стека
;бит 3 – предназначение
_code equ 0000100b ;сегмент кода
data_stk equ 00000000b ; сегмент данных или стека

Теперь для получения значения атрибута для нужного сегмента достаточно подобрать нужные константы по битам и выполнить подсчет суммы, как, например, для дескриптора gdt_gdt_8:

atr=const or data_wm_y or _data or data_stk

Значение atr для дескриптора сегмента gdt_gdt_8 будет равно 10010010 = 92h. Для удобства использования можно создать макрос, который будет создавать константу с именем, состоящим из двух частей: приставки atr и имени дескриптора, для которого формируется байт атрибута (к примеру, для дескриптора gdt_gdt_8 это будет atr_gdt_gdt_8):

atr macro descr,bitl,bit2,bit3

```
atr&descr=const or bitl or bit2 or bit3
endm
```

Имена формируемых констант нужно указывать при инициализации структур для каждого дескриптора, к примеру, для gdt_gdt_8:

```
gdt_seg       segment       para
gdt_0 descr <0,0,0,0,0,0>
atr    gdt_gdt_8,data_wm_y,_data,data_stk
gdt_gdt_8     descr <0,0,0,atr_gdt_gdt_8,0,0>
gdt_seg       ends
```

- поле lim_atr — байт, состоящий из четырех старших битов размера сегмента и четырех атрибутов. В нашем случае размер сегмента небольшой, то есть четыре старших бита размера равны 0. И оставшиеся биты атрибутов для нашего случая также нулевые;

- поле base_3 содержит старший байт 4-байтового физического адреса сегмента. Так как работа начинается в реальном режиме, где размер максимального физического адреса не превышает 20 бит, то этот байт также будет нулевым.

Таким образом, в программе инициализации будут подлежать три поля: limit, три первых байта адреса base_l и base_2 и байт атрибута atr.

Загрузка регистра gdtr

Дескрипторные таблицы являются ключевыми для организации работы в защищенном режиме. Их местоположение в памяти может быть любым. Микропроцессор узнает о том, где находятся эти таблицы, по содержимому определенных системных регистров. Так, после того как была сформирована таблица GDT, ее адрес нужно поместить в регистр gdtr. Но этого мало, так как в этот же регистр нужно поместить и размер этой таблицы. Для загрузки именно этого регистра в системе команд микропроцессора есть специальная команда:

lgdt *адрес_ 48-битного_ поля* (Load GDT register) — загрузить регистр gdtr. Команда lgdt загружает системный регистр gdtr содержимым 6-байтового поля, адрес которого указан в качестве операнда.

Из описания команды следует, что вначале необходимо сформировать поле из шести байт со структурой, аналогичной формату регистра gdtr, а затем указать адрес этого поля в качестве операнда команды lgdt.

Для резервирования поля из шести байт (48 бит) TASM поддерживает специальные директивы резервирования и инициализации данных — dp и df. После выделения — с помощью одной из этих директив — области памяти в

сегменте данных необходимо сформировать в этой области указатель на начало таблицы GDT и ее размер. Но удобнее использовать структуру. Пример ее использования показан в следующем фрагменте программы:

```
point      struc
limdw       0
 adrdd      0
ends
data        segment
point_gdt point <gdt_size,0>
;…
code        segment
;…
xor       eax,eax
mov     ax,gdt_seg
shl        eax,4
mov     dword ptr point_gdt,adr,eax
lgdt     pword point_gdt
```

Запрет обработки аппаратных прерываний

Обработка прерываний в защищенном режиме принципиально отличается от обработки прерываний в реальном режиме (это будет рассмотрено позже). Поэтому, как только микропроцессор переключится в защищенный режим, первое же прерывание от таймера, которое происходит 18,2 раза в секунду, «подвесит» компьютер. Следовательно прерывания как программные, так и внешние нужно будет запрещать. Это можно осуществить двумя способами: прямым программированием контроллера прерываний и командой микропроцессора cli. Можно использовать любой, только не нужно их сочетать.

Переключение микропроцессора в защищенный режим

Теперь есть все условия, чтобы корректно перейти в защищенный режим. Специальных команд микропроцессора для выполнения такого перехода нет. О том, что микропроцессор находится в защищенном режиме, говорит лишь состояние бита pe в регистре cr0. Установить этот бит можно двумя способами:

- непосредственной установкой бита pe в регистре cr0. Состояние этого бита управляет режимами работы микропроцессора: если pe=0, то микропроцессор работает в реальном режиме, если pe=1, то микропроцессор работает в защищенном режиме;

- использованием функции 89h прерывания 15h BIOS. Опишем оба способа, но использовать будем первый.

Регистр cr0 программно доступен, поэтому установить бит pe можно, используя обычные команды ассемблера:

```
mov    eax,cr0
or     eax,0001h
mov    cr0,eax
```

Последняя команда mov переводит микропроцессор в защищенный режим.

Функция 89h прерывания 15h выполняет это и некоторые другие действия неявно. Прежде чем вызывать это прерывание, необходимо посредством регистров сообщить ему следующее:

- ah =89h;
- bl = новый номер для аппаратного прерывания уровня irq0. Уровни irq 1...7 будут иметь следующие по порядку номера;
- bh = новый номер для аппаратного прерывания уровня irq8. Уровни irq9...f будут иметь следующие по порядку номера;
- ds: si = адрес GDT для защищенного режима;
- cx = адрес первой выполняемой команды в защищенном режиме.

Эта функция предполагает, что дескрипторы в таблице GDT расположены в определенной последовательности:

- 0h − пустой дескриптор;
- 8h − дескриптор таблицы GDT;
- 10h − дескриптор таблицы LDT;
- 18h −дескриптор сегмента данных, на него указывает селектор в регистре ds;
- 20h − дескриптор дополнительного сегмента данных, на него указывает селектор в регистре es;
- 28h — дескриптор сегмента стека, на него указывает селектор в регистре ss;
- 30h — дескриптор сегмента кода, на него указывает селектор в регистре cs;
- остальные дескрипторы.

В нашей таблице GDT этот порядок следования соблюден, поэтому фрагмент программы перевода микропроцессора в защищенный режим может быть следующим:

```
code segment
mov     ah,89h
mov     bl,20h
```

```
mov     bh,28h
mov     ax,gdt_seg
mov     ds,ax
mov     si,0
lea     ex,protect
int     15h
protect:
 ...;работа в защищенном режиме
```

Работа в защищенном режиме

Настройка сегментных регистров. Содержимое сегментных регистров в реальном и защищенном режимах интерпретируется микропроцессором по-разному. Как только микропроцессор оказывается в защищенном режиме, первую же команду он пытается выполнить традиционно: по содержимому пары cs: ip определить ее адрес, выбрать ее и т.д. Но содержимое c s должно быть индексом, указывающим на дескриптор сегмента кода в таблице GDT. Но пока это не так, так как в данный момент c s все еще содержит физический адрес параграфа сегмента кода, как этого требуют правила формирования физического адреса в реальном режиме. То же самое происходит и с другими регистрами. Но если содержимое других сегментных регистров можно подкорректировать в программе, то в случае с регистром cs этого сделать нельзя, так как он в защищенном режиме программно недоступен. Нужно помочь микропроцессору сориентироваться в этой затруднительной ситуации. Вспомним, что действие команд перехода основано как раз на изменении содержимого регистров cs и ip. Команды ближнего перехода изменяют только содержимое eip\ip, а команды дальнего перехода – обоих регистров, cs и eip\ip. Воспользуемся этим обстоятельством, вдобавок существует и еще одно свойство команд передачи управления — они сбрасывают конвейер микропроцессора, подготавливая его тем самым к приему команд, которые сформированы уже по правилам защищенного режима. Это же обстоятельство заставляет нас впрямую моделировать команду межсегментного перехода, чтобы поместить правильное значение селектора в сегментный регистр cs. Это можно сделать так:

```
code        segment
;…
db      0eah ;машинный код команды jmp
dw      offset protect ;смещение метки перехода
     ;в сегменте команд
```

```
dw      30h
protect:
;загрузить селекторы для остальных дескрипторов
mov     ax,18h
mov     ds, ax   ;сегментный регистр данных
mov     ax,28h
mov     ss, ax   ;сегментный регистр стека
mov     ax,20h
mov     es, ax   ;дополнительный сегмент данных
        ;для указания на видеобуфер
```

Кроме того есть еще один момент, на который нужно обязательно обратить внимание. В микропроцессоре каждому сегментному регистру соответствует свой теневой регистр дескриптора. Этот регистр имеет размер 64 бита и формат дескриптора сегмента. Смена содержимого теневых регистров производится автоматически всякий раз при смене содержимого соответствующего сегментного регистра. Последние действия по изменению содержимого сегментных регистров привели к тому, что неявно было записано в теневые регистры микропроцессора содержимое соответствующих дескрипторов из GDT. Программисту теневые регистры недоступны, с ними работает только микропроцессор. Если есть необходимость изменить их содержимое, то для этого нужно сначала изменить сам дескриптор, а затем загрузить соответствующий ему селектор в нужный сегментный регистр.

Подготовка к возврату в реальный режим. Здесь возникает примерно та же проблема с сегментными регистрами, что была при входе в защищенный режим. Микропроцессора использует теневые регистры, даже работая в реальном режиме. При этом поля этих регистров заполнены, конечно, в соответствии с требованиями реального режима:

- предел должен быть равен 64 К = 0ffffh;
- бит G = 0 (значение размера в поле предела) — это значение в байтах;
- байт атрибута равен 10010010 = 92h;
- базовый адрес значения не имеет.

Следовательно, перед переходом в реальный режим нужно сформировать эти значения в соответствующих дескрипторах и сделать актуальными эти изменения в теневых регистрах, для чего нужно перезагрузить сегментные регистры. После этого можно смело переходить в реальный режим. В программе все эти действия могут выглядеть так:

```
<1>        code   segment
<2>        ;...
```

<3>   ;мы в защищенном режиме и начинаем переход в реальный режим

<4>   ;загрузим значение 0ffffh в поле предела
<5>   mov gdt_ds_18.limit,0ffffh
<6>   ;для регистров cs, ss, es, fs и gs аналогично
<7>   ;селектор – в регистр данных, чтобы обновить теневой регистр для ds
<8>   mov ax,18
<9>   mov ds,ax
<10>   ;для регистров ss, es, fs и gs аналогично
<11>   ;для регистра cs загрузку селектора проведем
<12>   ;особым способом – командой jmp far
<13>   db 0eah
<14>   dw   offset jump
<15>   dw   30h
<16>   jump:
<17>   ;можно переходить в реальный режим

Переключение микропроцессора в реальный режим. Для выполнения этой операции нужно сбросить нулевой бит регистра cr0. Это можно сделать несколькими способами. К примеру:

```
mov eax,cr0
and al,0feh
mov cr0,eax
```

После сброса бита pe микропроцессор снова оказался в реальном режиме.

Настройка сегментных регистров. После перехода в реальный режим опять возникает проблема с содержимым сегментных регистров. Решается она уже известным вам способом:

```
;…
;моделирование команды дальнего перехода для загрузки cs

db      0eah
dw      real_mode
dw      code
real_mode:
mov     ax,data
mov     ds, ax
mov     ax,stk
mov     ss, ax
```

Разрешение прерываний. Теперь можно разрешить прерывания. Так как в системе прерываний ничего не менялось, то выполняемые действия тоже были простейшими:

а) запрещение аппаратных прерываний для того, чтобы на время смены режима работы микропроцессора нас не беспокоило прерывание от таймера;

б) последующее разрешение прерываний после возврата в реальный режим.

Это осуществляется следующими командами:

cli ;флаг IF в 0 – запретить прерывания от аппаратуры
sti ;флаг IF в 1 – разрешить прерывания от аппаратуры

Использование прерываний в защищенном режиме. В первой части данной лабораторной работы мы перед входом в запрещенный режим запрещали прерывания. Это наиболее простой способ написания программ. Возможно и использование прерываний в защищенном режиме. Напомним основные моменты обработки прерываний в реальном режиме.

В реальном режиме имеются программные и аппаратные прерывания. Программные прерывания инициируются командой INT, аппаратные – внешними событиями, асинхронными по отношению к выполняемой программе. Обычно аппаратные прерывания инициируются аппаратурой ввода/вывода после завершения выполнения текущей операции.

Кроме того, некоторые прерывания зарезервированы для использования самим процессором – прерывания по ошибке деления, прерывания для пошаговой работы, немаскируемое прерывание и т.д.

Для обработки прерываний в реальном режиме процессор использует таблицу векторов прерываний. Эта таблица располагается в самом начале оперативной памяти, т.е. ее физический адрес – 00000.

Таблица векторов прерываний реального режима состоит из 256 элементов по 4 байта, таким образом ее размер составляет 1 килобайт. Элементы таблицы – дальние указатели на процедуры обработки прерываний. Указатели состоят из 16-битового сегментного адреса процедуры обработки прерывания и 16-битового смещения. Причем смещение хранится по младшему адресу, а сегментный адрес – по старшему.

Когда происходит программное или аппаратное прерывание, текущее содержимое регистров CS, IP а также регистра флагов FLAGS записывается в стек программы (который, в свою очередь, адресуется регистровой парой SS:SP).

Далее из таблицы векторов прерываний выбираются новые значения для CS и IP, при этом управление передается на процедуру обработки прерывания.

Перед входом в процедуру обработки прерывания принудительно сбрасываются флажки трассировки TF и разрешения прерываний IF. Поэтому если ваша процедура прерывания сама должна быть прерываемой, вам необходимо разрешить прерывания командой STI. В противном случае, до завершения процедуры обработки прерывания все прерывания будут запрещены.

Завершив обработку прерывания, процедура должна выдать команду IRET, по которой из стека будут извлечены значения для CS, IP, FLAGS и загружены в соответствующие регистры. Далее выполнение прерванной программы будет продолжено.

Что же касается аппаратных маскируемых прерываний, то в компьютере IBM AT и совместимых с ним существует всего шестнадцать таких прерываний, обозначаемых IRQ0-IRQ15. В реальном режиме для обработки прерываний IRQ0-IRQ7 используются вектора прерываний от 08h до 0Fh, а для IRQ8-IRQ15 – от 70h до 77h.

Сейчас рассмотрим вопрос об использовании прерываний в защищенном режиме. В дальнейшем нам понадобятся сведения о CMOS – памяти, питаемой от независимого источника. При запуске компьютера содержимое CMOS анализируется процедурами BIOS, которые извлекают оттуда информацию о конфигурации системы, а также текущую дату и время. Доступ к CMOS осуществляется через порты 70h и 71h.

Чтение из CMOS:

```
mov al, xxh
out 70H, al        ; Выбор номера ячейки CMOS
jmp $+2            ; Задержка
in al, 71h         ; Ввод байта из CMOS
```

Запись в CMOS:

```
mov al, xxh
out 70H, al        ; Выбор номера ячейки CMOS
jmp $+2            ; Задержка
mov al, yyh        ; Байт для ввода в CMOS
out 71h, al        ; Ввод байта из CMOS
```

Порт 70h служит не только для индексирования ячеек CMOS, но и для разрешения или запрещения NMI (немаскируемого прерывания). Если бит 7 равен 0, то NMI разрешается, в противном случаи запрещается. Управление NMI и индексацию можно совместить в одной команде. Если в AL бит 7 обнулен, то

команда OUT 70H, AL не только индексирует ячейку CMOS, но и размаскирует NMI. Адреса ячеек CMOS находятся в промежутке 00H-3FH. Даже, если адрес ячейки больше 3FH, то все равно учитываются только младшие шесть бит адреса.

Следует отметить, что ячейка CMOS с номером 0FH называется байтом состояния перезагрузки. Этот байт считывается после сброса центрального процессора, чтобы определить не был ли сброс вызван для выхода из защищенного режима. Если содержимое этого байта равно пяти, то автоматически выполняется команда JMP FAR PTR [0:467H].

В защищенном режиме все прерывания разделяются на два типа – обычные прерывания и исключения (exception – исключение, особый случай).

Обычное прерывание инициируется командой INT (программное прерывание) или внешним событием (аппаратное прерывание). Перед передачей управления процедуре обработки обычного прерывания флаг разрешения прерываний IF сбрасывается и прерывания запрещаются.

Исключение происходит в результате ошибки, возникающей при выполнении какой-либо команды, например, если команда пытается выполнить запись данных за пределами сегмента данных или использует для адресации селектор, который не определен в таблице дескрипторов. По своим функциям исключения соответствуют зарезервированным для процессора внутренним прерываниям реального режима. Когда процедура обработки исключения получает управление, флаг IF не изменяется. Поэтому в мультизадачной среде особые случаи, возникающие в отдельных задачах, не оказывают влияния на выполнение остальных задач.

Теперь перейдем к рассмотрению механизма обработки прерываний и исключений в защищенном режиме.

Таблица прерываний защищенного режима. Обработка прерываний и исключений в защищенном режиме по аналогии с реальным режимом базируется на таблице прерываний. Но таблица прерываний защищенного режима является таблицей дескрипторов, которая содержит так называемые вентили прерываний, вентили исключений и вентили задач.

Таблица прерываний защищенного режима называется дескрипторной таблицей прерываний IDT (Interrupt Descriptor Table). Так же как и таблицы GDT и LDT, таблица IDT содержит 8-байтовые дескрипторы. Причем это системные дескрипторы – вентили прерываний, исключений и задач. Поле TYPE вентиля прерывания содержит значение 6, а вентиля исключения – значение 7.

Формат элементов дескрипторной таблицы прерываний IDT показан на рисунке 3.

Рисунок 3 – Формат элементов дескрипторной таблицы прерываний IDT

Расположение дескрипторной таблицы прерываний IDT определяется содержимым 5-байтового внутреннего регистра процессора IDTR. Формат регистра IDTR полностью аналогичен формату регистра GDTR, для его загрузки используется команда LIDT. Так же, как регистр GDTR содержит 24-битовый физический адрес таблицы GDT и ее предел, так и регистр IDTR содержит 24-битовый физический адрес дескрипторной таблицы прерываний IDT и ее предел.

Регистр IDTR обычно загружают перед переходом в защищенный режим. Разумеется, это можно сделать и потом, находясь в защищенном режиме. Однако для этого программа должна работать в привилегированном нулевом кольце.

Исключения в защищенном режиме. Для обработки особых ситуаций-исключений зарезервировали 31 номер прерывания. В таблице 1 приведен полный список зарезервированных прерываний защищенного режима.

Таблица 1 – Зарезервированные прерывания защищенного режима

00h	Ошибка при выполнении команды деления
01h	Прерывание для пошаговой работы, используется отладчиками
02h	Немаскируемое прерывание
03h	Прерывание по точке останова для отладчиков
04h	Переполнение, генерируется командой INTO, если установлен флаг OF
05h	Генерируется при выполнении машинной команды BOUND, если проверяемое значение вышло за пределы заданного диапазона
06h	Недействительный код операции, или длина команды больше 10 байт
07h	Отсутствие арифметического сопроцессора

08h	Двойная ошибка, вырабатывается в том случае, если при обработке исключения возникло еще одно исключение. Если во время обработки этого прерывания возникает третье исключение, процессор переходит в состояние отключения, что приводит к перезапуску процессора
09h	Превышение сегмента арифметическим сопроцессором
0Ah	Недействительный сегмент состояния задачи TSS
0Bh	Отсутствие сегмента. Вырабатывается при попытке использовать для адресации дескриптор, у которого бит присуствия сегмента в памяти P сброшен в 0. Это прерывание используется для реализации механизма виртуальной памяти. В этом случае по прерыванию 0Bh операционная система может выполнить подкачку отсутствующего сегмента в память
0Ch	Исключение при работе со стеком. Может возникать в случае отсутствия сегмента стека в памяти или в случае переполнения (антипереполнения) стека
0Dh	Исключение по защите памяти. Возникает при любых попытках получения доступа к сегментам памяти, если программа обладает недостаточным уровнем привилегий
0Eh	Отказ страницы для процессоров i80386 или i80486, зарезервировано для i80286
0Fh	Зарезервировано.
10h	Исключение сопроцессора
11h – 1Ah	Зарезервированы

Перед тем, как передать управление обработчику исключения, для многих зарезервированных прерываний процессор помещает в стек 16-битовый код ошибки. Этот код ошибки программа может проанализировать и тем самым получить некоторую дополнительную информацию об ошибке.

Формат кода ошибки приведен на рисунке 4.

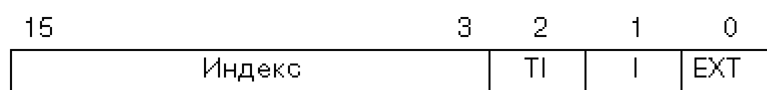

Рисунок 4 – Формат кода ошибки процессора

Поле индекса содержит индекс дескриптора, при обращении к которому произошла ошибка. Поле I, равное 1, означает, что этот индекс относится к таблице IDT. В этом случае произошла ошибка при обработке прерывания или исключения.

Если бит I равен 0, поле TI выбирает таблицу дескрипторов (GDT или LDT) по аналогии с соответствующим полем селектора.

Бит EXT устанавливается в том случае, когда ошибка произошла не в результате выполнения текущей команды, а по внешним относительно выполняемой программы причинам. Например, при обработке аппаратного прерывания от устройства ввода/вывода произошло обращение к отсутствующему в памяти сегменту (у которого в дескрипторе сброшен бит присутствия P).

Как мы только что говорили, коды ошибок включаются в стек не для всех исключений. Программа сможет проанализировать этот код только для следующих исключений:

08h – двойная ошибка;

0Ah – недействительный TSS;

0Bh – отсутствие сегмента в памяти;

0Ch – исключение при работе со стеком;

0Dh – исключение по защите памяти.

Заметим, что аналога коду ошибки для зарезервированных прерываний в реальном режиме нет.

Кроме того, новым при обработке прерываний в защищенном режиме является свойство повторной запускаемости исключений. Свойством повторной запускаемости обладают не все исключения.

Что такое повторная запускаемость?

Поясним это на конкретном примере. Пусть в нашей системе реализована виртуальная память. Программа в некоторый момент времени обратилась к отсутствующему в оперативной памяти сегменту, выдав какую-либо команду, например MOV или ADD.

Возникло исключение 0Bh – отсутствие сегмента в памяти. Обработчик этого исключения, входящий в состав операционной системы, выполнил свопинг соответствующего сегмента в оперативную память. Повторяем выполнение прерванной команды.

Это можно сделать, так как для всех повторно запускаемых исключений (кроме 03h – прерывание по точке останова и 04h – переполнение) в стек включается адрес не следующей за прерванной командой, а адрес первого байта команды, которая вызвала исключение. Выполнив команду IRET, программа обработки исключения вновь передаст управление прерванной команде.

Свойством повторной запускаемости обладает большинство зарезервированных прерываний, кроме следующих:

01h – прерывание для пошаговой работы;

08h – двойная ошибка;

09h – превышение сегмента сопроцессором;

0Dh – исключение по защите памяти;

10h – исключение сопроцессора.

Обработка аппаратных прерываний. Вспомните диапазон номеров прерываний, используемый в реальном режиме в компьютерах IBM PC: для обработки прерываний IRQ0-IRQ7 используются номера прерываний от 08h до 0Fh, а для IRQ8-IRQ15 – от 70h до 77h.

В защищенном режиме номера от 08h до 0Fh зарезервированы для обработки исключений.

Имеется простой способ перепрограммирования контроллера прерываний на любой другой диапазон номеров векторов аппаратных прерываний. Например, аппаратные прерывания можно расположить сразу за прерываниями, зарезервированными для обработки исключений.

После возврата процессора в реальный режим необходимо восстановить состояния контроллера прерываний. Если при подготовке к возврату в реальный режим мы записали в CMOS-память байт состояния отключения со значением 5, после сброса BIOS сам перепрограммирует контроллер прерываний для работы в реальном режиме и нам не надо об этом беспокоиться. В противном случае программа должна установить правильные номера для аппаратных прерываний реального режима.

Программа, которая работает с прерываниями. Пример программы, которая вам будет представлена, выполняет все необходимые действия, связанные с обработкой прерываний и исключений в защищенном режиме.

Программа из первой части работала в защищенном режиме с запрещенными прерываниями. Она как бы пролетала через не изведанное с завязанными глазами и сразу же возвращалась в хорошо освоенный реальный режим.

Теперь же программа имеет возможность «осмотреться» и может реагировать на такие события, как прерывания от клавиатуры и таймера. Кроме того, в случае возникновения исключений вам будет выдана некоторая диагностика, включающая код исключения, код ошибки и содержимое регистров процессора на момент возникновения исключения.

Таблица IDT содержит дескрипторы для обработчиков исключений и аппаратных прерываний. Вентили исключений содержат ссылки на программы с именами exc_00...exc_1F. Вслед за вентилями исключений в таблице IDT следуют вентили аппаратных прерываний. Из них задействованы только IRQ0 и IRQ1 – прерывания от таймера и клавиатуры.

```
IDT_BEG = $
; --------------------- Вентили исключений -------------------
idt     idt_struc <OFFSET exc_00,CS_DESCR,0,TRAP_ACC,0>
```

```
        idt_struc <OFFSET exc_01,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_02,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_03,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_04,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_05,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_06,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_07,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_08,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_09,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0A,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0B,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0C,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0D,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0E,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0F,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_10,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_11,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_12,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_13,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_14,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_15,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_16,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_17,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_18,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_19,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1A,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1B,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1C,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1D,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1E,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1F,CS_DESCR,0,TRAP_ACC,0>

; --------------- Вентили аппаратных прерываний ---------------
; int 20h-IRQ0
        idt_struc <OFFSET Timer_int,CS_DESCR,0,INT_ACC,0>
; int 21h-IRQ1
        idt_struc <OFFSET Keyb_int,CS_DESCR,0,INT_ACC,0>
; int 22h, 23h, 24h, 25h, 26h, 27h-IRQ2-IRQ7
        idt_struc 6 dup (<OFFSET dummy_iret0,CS_DESCR,0,INT_ACC,0>)
```

```
; int 28h, 29h, 2ah, 2bh, 2ch, 2dh, 2eh, 2fh-IRQ8-IRQ15
    idt_struc 8 dup (<OFFSET dummy_iret1,CS_DESCR,0,INT_ACC,0>)
; -------------------- Вентиль прерывания --------------------
; int 30h
        idt_struc     <OFFSET Int_30h_Entry,CS_DESCR,0,INT_ACC,0>
IDT_SIZE      = ($ - IDT_BEG)
```

Те аппаратные прерывания, которые нас не интересуют, приводят к выдаче в контроллеры прерывания команды конца прерывания. Для таких прерываний предусмотрены заглушки – процедуры с именами dummy_iret0 и dummy_iret1. Первая заглушка относится к первому контроллеру прерывания, вторая – ко второму.

```
; Посылаем сигнал конца прерывания в первый контроллер 8259A

    mov    al,EOI
    out    MASTER8259A,al
    pop    ax
    iret
ENDP   dummy_iret0
```

Вслед за вентилями аппаратных прерываний мы поместили в таблицу IDT вентиль программного прерывания int 30h, предназначенный для организации взаимодействия с клавиатурой на манер прерывания BIOS INT 16h. При выдаче прерывания int 30h вызывается процедура с именем Int_30h_Entry.

```
PROC   Int_30h_Entry   NEAR

    push   dx               ; запрещаем прерывания и
    cli                     ; сбрасываем признак
    mov    al, 0            ; готовности скан-кода
    mov    [key_flag], al   ; в буфере клавиатуры
```

После запуска программа переходит в защищенный режим и размаскирует прерывания от таймера и клавиатуры.

```
; Размаскируем прерывания от таймера и клавиатуры

    in     al,INT_MASK_PORT
    and    al,0fch
    out    INT_MASK_PORT,al
```

Далее она вызывает в цикле прерывание int 30h (ввод символа с клавиатуры), и выводит на экран скан-код нажатой клавиши и состояние переключающих клавиш (таких, как CapsLock, Ins, и т.д.).

```
; Ожидаем нажатия на клавишу <ESC>
charin:
      int    30h    ; ожидаем нажатия на клавишу
                    ; AX – скан-код клавиши,
                    ; BX – состояние переключающих клавиш
      cmp    al, 1  ; если <ESC> – выход из цикла
      jz     continue
```

Клавиша ESC означает – программа выходит из цикла. Далее в тексте программы следует ряд команд, закрытых символом комментария. Эти команды приводят к исключению. Вы можете попробовать их работу, удалив соответствующий символ комментария. При возникновении исключения на экран будет выведена диагностика.

Перед завершением работы программа устанавливает реальный режим процессора и стирает экран.

Обработчик аппаратного прерывания клавиатуры – процедура с именем Keyb_int. После прихода прерывания она выдает короткий звуковой сигнал, считывает и анализирует скан-код клавиши, вызвавшей прерывание. Скан-коды классифицируются на обычные и расширенные (для 101-клавишной клавиатуры). В отличие от прерывания BIOS INT 16h, мы для простоты не стали реализовывать очередь, а ограничились записью полученного скан-кода в глобальную ячейку памяти key_code. Причем прерывания, возникающие при отпускании клавиш игнорируются.

Запись скан-кода в ячейку key_code выполняет процедура Keyb_PutQ. После записи эта процедура устанавливает признак того, что была нажата клавиша – записывает значение 0FFh в глобальную переменную key_flag.

```
; ------------------------------------------------
; Запись скан-кода и расширенного скан-кода в
; «буфер», состоящий из одного слова.
; ------------------------------------------------

PROC    Keyb_PutQ      NEAR

      push    ax
      mov     [key_code], ax  ; записываемый код

; ------- Обрабатываем переключающие клавиши ---------
```

```
        cmp     ax, 002ah               ; L_SHIFT down
        jnz     @@kb1
        mov     ax, [keyb_status]
        or      ax, L_SHIFT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb1:
        cmp     ax, 00aah               ; L_SHIFT up
        jnz     @@kb2
        mov     ax, [keyb_status]
        and     ax, NL_SHIFT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb2:
         . . . . . . .  . . .
         . . . . . . .  . . .
         . . . . . . .  . . . .
@@kb20:

        test    ax, 0080h               ; фильтруем отжатия клавиш
        jnz     keyb_putq_exit

        mov     al, 0ffh                ; устанавливаем признак
        mov     [key_flag], al  ; готовности для чтения
                                ; символа из «буфера»
keyb_putq_exit:
        pop     ax
        ret
ENDP    Keyb_PutQ
```

Программное прерывание int 30h опрашивает состояние key_flag. Если этот флаг оказывается установленным, он сбрасывается, вслед за чем обработчик int 30h записывает в регистр AX скан-код нажатой клавиши, в регистр BX – состояние переключающих клавиш на момент нажатия клавиши, код которой передан в регистре AX.

Для того чтобы не загромождать программу второстепенными деталями, мы не стали перекодировать скан-код в код ASCII, вы сможете при необходимости сделать это сами.

Обработчик прерываний таймера – процедура с именем Timer_int. Эта процедура примерно раз в секунду выдает звуковой сигнал, чем ее действия и ограничиваются.

```
PROC    Timer_int       NEAR
        cli
        push    ax
; Увеличиваем содержимое счетчика времени
        mov     ax, [timer_cnt]
        inc     ax
        mov     [timer_cnt], ax
; Примерно раз в секунду выдаем звуковой сигнал
        test    ax, 0fh
        jnz     timer_exit
        call    beep
timer_exit:
; Посылаем команду конца прерывания
        mov     al,EOI
        out     MASTER8259A,al
        pop     ax
        sti
        iret
ENDP    Timer_int
```

Процедура rdump выводит на экран содержимое регистров процессора и может быть использована для отладки программы.

```
PROC    rdump NEAR
        pushf
        pusha
        mov     di, es
        mov     ax,[vir_crt]
        mov     es,ax
        mov     si,OFFSET reg_title
        mov     bx,1                ; (X,Y) = (AX,BX)
        mov     ax,6
        mov     cx,SIZE reg_title
        mov     dh,1fh              ; черный на голубом фоне
        call    writexy
; Выводим содержимое всех регистров
```

```
        mov     ax,cs           ; cs
        mov     bx, 0702h
        call    Print_Word
        mov     bp, sp
        mov     ax, [bp+18d]     ; ip
        mov     bx, 0708h
        call Print_Word
          . . . . . . . . . .
          . . . . . . . . . .
          . . . . . . . . . .
        mov     bx, 090eh
        mov     ax,ss           ; ss
        call Print_Word
        mov     bx, 0914h
        mov     ax, [bp+16d]    ; flags
        call Print_Word

; Восстанавливаем содержимое регистров

        popa
        popf
        ret
ENDP    rdump
```

Обратите внимание на использованный способ возврата процессора в реальный режим:

```
DATASEG

; Пустой дескриптор для выполнения возврата
; процессора в реальный режим через перевод
; его в состояние отключения.

null_idt idt_struc <>

CODESEG

PROC    set_rmode       NEAR

        mov     [real_sp],sp

; Переводим процессор в состояние отключения,
; это эквивалентно аппаратному сбросу, но
```

```
; выполняется быстрее.
; Сначала мы загружаем IDTR нулями, затем
; выдаем команду прерывания.

        lidt    [FWORD null_idt]
        int     3

rwait:
        hlt
        jmp     rwait

LABEL   shutdown_return FAR
```

Регистр IDTR загружается нулями, следовательно, предел дескрипторной таблицы прерываний равен нулю. После этого мы выдаем команду программного прерывания. При обработке прерывания возникает исключение, так как регистр IDTR инициализирован неправильно. Но это исключение не может быть обработано по той же причине, что вызывает новое исключение. Теперь процессор переходит уже в состояние отключения и выполняет рестарт в реальном режиме. Что нам и требовалось получить!

Для удобства создано два файла: lab8.inc, где происходит определение структур данных и константи lab8.asm, который и содержит саму программу.

Lab8.inc

```
; ---------------------------------------------------------
; Определения структур данных и констант
; ---------------------------------------------------------

STRUC       desc_struc                  ; структура дескриптора
        limit   dw          0       ; предел
        base_l      dw  0       ; мл. слово физического адреса
        base_h      db  0       ; ст. байт физического адреса
        access      db  0       ; байт доступа
        rsrv    dw  0       ; зарезервировано
ENDS        desc_struc

STRUC       idt_struc               ; вентиль прерывания
        destoff dw  0       ; смещение обработчика
        destsel dw  0       ; селектор обработчика
        nparams db  0       ; кол-во параметров
        assess  db  0       ; байт доступа
        rsrv    dw  0       ; зарезервировано
```

```
ENDS        idt_struc

STRUC       idtr_struc          ; регистр IDTR
        idt_lim dw 0        ; предел IDT
        idt_l   dw     0    ; мл. слово физического адреса
        idt_h   db     0    ; ст. байт физического адреса
        rsrv    db     0    ; зарезервировано
ENDS        idtr_struc

; ------------------------------------------------------------
; Биты байта доступа

ACC_PRESENT   EQU  10000000b ; сегмент есть в памяти
ACC_CSEG      EQU  00011000b ; сегмент кода
ACC_DSEG      EQU  00010000b ; сегмент данных
ACC_EXPDOWNEQU  00000100b ; сегмент расширяется вниз
ACC_CONFORMEQU  00000100b ; согласованный сегмент
ACC_DATAWR    EQU  00000010b ; разрешена запись
ACC_INT_GATE EQU  00000110b ; вентиль прерывания
ACC_TRAP_GATE EQU      00000111b ; вентиль исключения

; ---------------------------------------------------------
; Типы сегментов
; сегмент данных
DATA_ACC = ACC_PRESENT OR ACC_DSEG OR ACC_DATAWR
; сегмент кода
CODE_ACC = ACC_PRESENT OR ACC_CSEG OR ACC_CONFORM
; сегмент стека
STACK_ACC = ACC_PRESENT OR ACC_DSEG OR ACC_DATAWR OR
ACC_EXPDOWN
; байт доступа сегмента таблицы IDT
IDT_ACC       =       DATA_ACC
; байт доступа вентиля прерывания
INT_ACC       =       ACC_PRESENT OR ACC_INT_GATE
; байт доступа вентиля исключения
TRAP_ACC      =       ACC_PRESENT OR ACC_TRAP_GATE
; ---------------------------------------------------------
; Константы

STACK_SIZE    EQU  0400  ; размер стека
B_DATA_SIZE   EQU  0300  ; размер области данных BIOS
```

```
B_DATA_ADDR EQU  0400  ; адрес области данных BIOS
MONO_SEG        EQU  0b000; сегмент видеопамяти
                           ;  монохромного видеоадаптера
COLOR_SEG       EQU  0b800; сегмент видеопамяти
                           ; цветного видеоадаптера
CRT_SIZE  EQU  4000  ; размер сегмента видеопамяти
                           ;  цветного видеоадаптера
MONO_SIZE       EQU  1000  ; размер сегмента видеопамяти
                           ;  монохромного видеоадаптера

CRT_LOW  EQU  8000  ; мл. байт физического адреса
                           ;  сегмента видеопамяти
                           ;  цветного видеоадаптера
MONO_LOW        EQU  0000  ; мл. байт физического адреса
                           ;  сегмента видеопамяти
                           ;  монохромного видеоадаптера

CRT_SEG   EQU  0Bh   ; ст. байт физического адреса
                           ;  сегмента видеопамяти
CMOS_PORT       EQU  70h    ; порт для доступа к CMOS-памяти
PORT_6845       EQU  0063h; адрес области данных BIOS,
                           ; где записано значение адреса
                           ; порта контроллера 6845
COLOR_PORT      EQU  03d4h; порт цветного видеоконтроллера
MONO_PORT       EQU  03b4h; порт монохромного видеоконтроллера
STATUS_PORT     EQU  64h    ; порт состояния клавиатуры
SHUT_DOWN       EQU  0feh   ; команда сброса процессора
VIRTUAL_MODE EQU       0001h; бит перехода в защищенный режим
A20_PORT EQU  0d1h  ; команда управления линией A20
A20_ON      EQU  0dfh  ; открыть A20
A20_OFF     EQU  0ddh  ; закрыть A20
KBD_PORT_A      EQU  60h    ; адреса клавиатурных
KBD_PORT_B      EQU  61h    ;   портов
INT_MASK_PORT EQU      21h    ; порт для маскирования прерываний
EOI             EQU  20     ; команда конца прерывания
MASTER8259A     EQU  20     ; первый контроллер прерываний
SLAVE8259A      EQU  0a0    ; второй контроллер прерываний

; ------------------------------------------------------------
; Селекторы, определенные в таблице GDT
```

```
DS_DESCR=       (gdt_ds - gdt_0)
CS_DESCR=       (gdt_cs - gdt_0)
SS_DESCR =      (gdt_ss - gdt_0)
BIOS_DESCR      =       (gdt_bio - gdt_0)
CRT_DESCR       =       (gdt_crt - gdt_0)
MDA_DESCR       =       (gdt_mda - gdt_0)

; ------------------------------------------------------------
; Маски и инверсные маски для клавиш

L_SHIFT         equ     0000000000000001b
NL_SHIFT        equ     1111111111111110b

R_SHIFT         equ     0000000000000010b
NR_SHIFT        equ     1111111111111101b

L_CTRL          equ     0000000000000100b
NL_CTRL         equ     1111111111111011b

R_CTRL          equ     0000000000001000b
NR_CTRL         equ     1111111111110111b

L_ALT                   equ     0000000000010000b
NL_ALT          equ     1111111111101111b

R_ALT                   equ     0000000000100000b
NR_ALT          equ     1111111111011111b

CAPS_LOCK               equ     0000000001000000b
SCR_LOCK                equ     0000000010000000b

NUM_LOCK                equ     0000000100000000b
INSERT          equ     0000001000000000b
```

Lab8.asm

```
IDEAL
RADIX   16
P286
MODEL   LARGE

include 'tiny-os.inc'
```

```
STACK   STACK_SIZE
DATASEG
DSEG_BEG = THIS WORD

      real_ss dw      ?
      real_sp dw      ?
      real_es dw      ?
GDT_BEG = $
LABEL   gdtr          WORD

gdt_0   desc_struc <0,0,0,0,0>
gdt_gdt desc_struc <GDT_SIZE-1,,,DATA_ACC,0>
gdt_idt desc_struc <IDT_SIZE-1,,,IDT_ACC,0>
gdt_ds  desc_struc <DSEG_SIZE-1,,,DATA_ACC,0>
gdt_cs  desc_struc <CSEG_SIZE-1,,,CODE_ACC,0>
gdt_ss  desc_struc <STACK_SIZE-1,,,DATA_ACC,0>
gdt_bio desc_struc <B_DATA_SIZE-1,B_DATA_ADDR,0,DATA_ACC,0>
gdt_crt desc_struc <CRT_SIZE-1,CRT_LOW,CRT_SEG,DATA_ACC,0>
gdt_mda                 desc_struc                  <MONO_SIZE-
1,MONO_LOW,CRT_SEG,DATA_ACC,0>

GDT_SIZE = ($ - GDT_BEG)
; Область памяти для загрузки регистра IDTR
idtr    idtr_struc      <IDT_SIZE,,,0>
; Таблица дескрипторов прерываний
IDT_BEG = $

; ---------------------- Вентили исключений --------------------

idt     idt_struc <OFFSET exc_00,CS_DESCR,0,TRAP_ACC,0>
      idt_struc <OFFSET exc_01,CS_DESCR,0,TRAP_ACC,0>
      idt_struc <OFFSET exc_02,CS_DESCR,0,TRAP_ACC,0>
      idt_struc <OFFSET exc_03,CS_DESCR,0,TRAP_ACC,0>
      idt_struc <OFFSET exc_04,CS_DESCR,0,TRAP_ACC,0>
      idt_struc <OFFSET exc_05,CS_DESCR,0,TRAP_ACC,0>
      idt_struc <OFFSET exc_06,CS_DESCR,0,TRAP_ACC,0>
      idt_struc <OFFSET exc_07,CS_DESCR,0,TRAP_ACC,0>
      idt_struc <OFFSET exc_08,CS_DESCR,0,TRAP_ACC,0>
      idt_struc <OFFSET exc_09,CS_DESCR,0,TRAP_ACC,0>
      idt_struc <OFFSET exc_0A,CS_DESCR,0,TRAP_ACC,0>
      idt_struc <OFFSET exc_0B,CS_DESCR,0,TRAP_ACC,0>
```

```
        idt_struc <OFFSET exc_0C,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0D,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0E,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0F,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_10,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_11,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_12,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_13,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_14,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_15,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_16,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_17,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_18,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_19,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1A,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1B,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1C,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1D,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1E,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1F,CS_DESCR,0,TRAP_ACC,0>

; --------------- Вентили аппаратных прерываний ---------------
; int 20h -- IRQ0
        idt_struc <OFFSET Timer_int,CS_DESCR,0,INT_ACC,0>
; int 21h -- IRQ1
        idt_struc <OFFSET Keyb_int,CS_DESCR,0,INT_ACC,0>
; int 22h, 23h, 24h, 25h, 26h, 27h -- IRQ2-IRQ7
        idt_struc 6 dup (<OFFSET dummy_iret0,CS_DESCR,0,INT_ACC,0>)
; int 28h, 29h, 2ah, 2bh, 2ch, 2dh, 2eh, 2fh -- IRQ8-IRQ15
        idt_struc 8 dup (<OFFSET dummy_iret1,CS_DESCR,0,INT_ACC,0>)
; -------------------- Вентиль прерывания --------------------
; int 30h
            idt_struc       <OFFSET Int_30h_Entry,CS_DESCR,0,INT_ACC,0>
IDT_SIZE        = ($ - IDT_BEG)
CODESEG
PROC    start
        mov     ax,DGROUP
        mov     ds,ax
        call    set_crt_base
```

```
        mov     bh, 77h
        call    clrscr

; Устанавливаем защищенный режим
        call    set_pmode
        call    write_hello_msg
; Размаскируем прерывания от таймера и клавиатуры
        in      al,INT_MASK_PORT
        and     al,0fch
        out     INT_MASK_PORT,al
; Ожидаем нажатия на клавишу <ESC>
charin:
        int     30h     ; ожидаем нажатия на клавишу
                        ; AX – скан-код клавиши,
                        ; BX – состояние переключающих клавиш
        cmp     al, 1   ; если <ESC> – выход из цикла
        jz      continue
        push    bx              ; выводим скан-код на экран
        mov     bx, 0301h       ; координаты вывода
        call    Print_Word
        pop     bx
        mov     ax, bx          ; выводим состояние
        push    bx              ; переключающих клавиш
        mov     bx, 0306h
        call    Print_Word
        pop     bx
        jmp     charin
; Следующий байт находится в сегменте кода.
; Он используется нами для демонстрации возникновения
; исключения при попытке записи в сегмент кода.
wrong1  db ?
continue:
; После нажатия на клавишу <ESC> выходим в это место
; программы. Следующие несколько строк демонстрируют
; команды, которые вызывают исключение. Вы можете
; попробовать их, если уберете символ комментария
; из соответствующей строки.
;Попытка деления на нуль
         mov ax,10
```

```
        mov bl,0
        div bl
; Попытка записи за конец сегмента данных. Метка wrong
; находится в самом конце программы.
;       mov     [wrong], al
; Попытка записи в сегмент кода.
;       mov     [wrong1], al
; Попытка извлечения из пустого стека.
;       pop     ax
; Загрузка в сегментный регистр неправильного селектора.
;       mov     ax, 1280h
;       mov     ds, ax

; Прямой вызов исключения при помощи команды прерывания.
;       int     1
        call    set_rmode       ; установка реального режима
        mov     bh, 07h         ; стираем экран и
        call    clrscr          ; выходим в DOS
        mov     ah,4c
        int     21h
ENDP    start
MACRO setgdtentry
        mov     [(desc_struc bx).base_l],ax
        mov     [(desc_struc bx).base_h],dl
ENDM
; -----------------------------------------
; Установка защищенного режима
; -----------------------------------------
PROC    set_pmode       NEAR
        mov     ax,DGROUP
        mov     dl,ah
        shr     dl,4
        shl     ax,4
        mov     si,ax
        mov     di,dx
        add     ax,OFFSET gdtr
        adc     dl,0
        mov     bx,OFFSET gdt_gdt
        setgdtentry
```

```
; Заполняем дескриптор в GDT, указывающий на
; дескрипторную таблицу прерываний
        mov     ax,si   ; загружаем 24-битовый адрес сегмента
        mov     dx,di   ; данных
        add     ax,OFFSET idt   ; адрес дескриптора для IDT
        adc     dl,0
        mov     bx,OFFSET gdt_idt
        setgdtentry
; Заполняем структуру для загрузки регистра IDTR
        mov     bx,OFFSET idtr
        mov     [(idtr_struc bx).idt_l],ax
        mov     [(idtr_struc bx).idt_h],dl
        mov     bx,OFFSET gdt_ds
        mov     ax,si
        mov     dx,di
        setgdtentry
        mov     bx,OFFSET gdt_cs
        mov     ax,cs
        mov     dl,ah
        shr     dl,4
        shl     ax,4
        setgdtentry
        mov     bx,OFFSET gdt_ss
        mov     ax,ss
        mov     dl,ah
        shr     dl,4
        shl     ax,4
        setgdtentry
; готовим возврат в реальный режим
        push    ds
        mov     ax,40
        mov     ds,ax
        mov     [WORD 67],OFFSET shutdown_return
        mov     [WORD 69],cs
        pop     ds
        cli
        mov     al,8f
        out     CMOS_PORT,al
        jmp     del1
```

```
del1:
        mov     al,5
        out     CMOS_PORT+1,al
        mov     ax,[rl_crt]     ; сегмент видеопамяти
        mov     es,ax
        call    enable_a20      ; открываем линию A20
        mov     [real_ss],ss    ; сохраняем сегментные
        mov     [real_es],es    ; регистры
; -------- Перепрограммируем контроллер прерываний --------
; Устанавливаем для IRQ0-IRQ7 номера прерываний 20h-27h
        mov     dx,MASTER8259A
        mov     ah,20h
        call    set_int_ctrlr
; Устанавливаем для IRQ8-IRQ15 номера прерываний 28h-2Fh
        mov     dx,SLAVE8259A
        mov     ah,28h
        call    set_int_ctrlr
; Загружаем регистры IDTR и GDTR
        lidt    [FWORD idtr]
        lgdt    [QWORD gdt_gdt]
; Переключаемся в защищенный режим
        mov     ax,VIRTUAL_MODE
        lmsw    ax
;       jmp     far flush
        db      0ea
        dw      OFFSET flush
        dw      CS_DESCR
LABEL   flush   FAR
; Загружаем селекторы в сегментные регистры
        mov     ax,SS_DESCR
        mov     ss,ax
        mov     ax,DS_DESCR
        mov     ds,ax
; Разрешаем прерывания
        sti
        ret
ENDP    set_pmode
; ------------------------------
; Возврат в реальный режим
; ------------------------------
```

```
DATASEG
; Пустой дескриптор для выполнения возврата
; процессора в реальный режим через перевод
; его в состояние отключения.
null_idt idt_struc <>
CODESEG

PROC    set_rmode       NEAR
        mov     [real_sp],sp
; Переводим процессор в состояние отключения,
; это эквивалентно аппаратному сбросу, но
; выполняется быстрее.
; Сначала мы загружаем IDTR нулями, затем
; выдаем команду прерывания.
        lidt    [FWORD null_idt]
        int     3
; Это старый способ сброса процессора через
; контроллер клавиатуры.
;       mov     al,SHUT_DOWN
;       out     STATUS_PORT,al
rwait:
        hlt
        jmp     rwait
LABEL   shutdown_return FAR
        in      al,INT_MASK_PORT
        and     al,0
        out     INT_MASK_PORT,al
        mov     ax,DGROUP
        mov     ds,ax
        assume  ds:DGROUP
        cli
        mov     ss,[real_ss]
        mov     sp,[real_sp]
        mov     ax,000dh
        out     CMOS_PORT,al
        sti
        mov     es,[real_es]
        call    disable_a20
        ret
```

```
ENDP    set_rmode

; ------------------------------------------------
; Обработка исключений
; ------------------------------------------------
; Обработчики исключений. Записываем в AX номер
; исключения и передаем управление процедуре
; shutdown
LABEL   exc_00  WORD
        mov     ax,0
        jmp     shutdown
LABEL   exc_01  WORD
        mov     ax,1
        jmp     shutdown
LABEL   exc_02  WORD
        mov     ax,2
        jmp     shutdown
LABEL   exc_03  WORD
        mov     ax,3
        jmp     shutdown
LABEL   exc_04  WORD
        mov     ax,4
        jmp     shutdown
LABEL   exc_05  WORD
        mov     ax,5
        jmp     shutdown
LABEL   exc_06  WORD
        mov     ax,6
        jmp     shutdown
LABEL   exc_07  WORD
        mov     ax,7
        jmp     shutdown
LABEL   exc_08  WORD
        mov     ax,8
        jmp     shutdown
LABEL   exc_09  WORD
        mov     ax,9
        jmp     shutdown
LABEL   exc_0A  WORD
```

```
        mov     ax,0ah
        jmp     shutdown
LABEL   exc_0B  WORD
        mov     ax,0bh
        jmp     shutdown
LABEL   exc_0C  WORD
        mov     ax,0ch
        jmp     shutdown
LABEL   exc_0D  WORD
        mov     ax,0dh
        jmp     shutdown
LABEL   exc_0E  WORD
        mov     ax,0eh
        jmp     shutdown
LABEL   exc_0F  WORD
        mov     ax,0fh
        jmp     shutdown
LABEL   exc_10  WORD
        mov     ax,10h
        jmp     shutdown
LABEL   exc_11  WORD
        mov     ax,11h
        jmp     shutdown
LABEL   exc_12  WORD
        mov     ax,12h
        jmp     shutdown
LABEL   exc_13  WORD
        mov     ax,13h
        jmp     shutdown
LABEL   exc_14  WORD
        mov     ax,14h
        jmp     shutdown
LABEL   exc_15  WORD
        mov     ax,15h
        jmp     shutdown
LABEL   exc_16  WORD
        mov     ax,16h
        jmp     shutdown
LABEL   exc_17  WORD
```

```
        mov     ax,17h
        jmp     shutdown
LABEL   exc_18  WORD
        mov     ax,18h
        jmp     shutdown
LABEL   exc_19  WORD
        mov     ax,19h
        jmp     shutdown
LABEL   exc_1A  WORD
        mov     ax,1ah
        jmp     shutdown
LABEL   exc_1B  WORD
        mov     ax,1bh
        jmp     shutdown
LABEL   exc_1C  WORD
        mov     ax,1ch
        jmp     shutdown
LABEL   exc_1D  WORD
        mov     ax,1dh
        jmp     shutdown
LABEL   exc_1E  WORD
        mov     ax,1eh
        jmp     shutdown
LABEL   exc_1F  WORD
        mov     ax,1fh
        jmp     shutdown
DATASEG
exc_msg db      "Exception ...., ....:.... code ..... Press any key... "
CODESEG
; ----------------------------------------------
; Вывод на экран номера исключения, кода ошибки,
; дампа регистров и возврат в реальный режим.
; ----------------------------------------------
PROC    shutdown        NEAR
        call    rdump   ; дамп регистров процессора
        push    ax
        call    beep    ; звуковой сигнал

; Выводим сообщение об исключении
```

```
        mov     ax,[vir_crt]
        mov     es,ax
        mov     bx,1d
        mov     ax,4
        mov     si,OFFSET exc_msg
        mov     dh,74h
        mov     cx, SIZE exc_msg
        call    writexy
        pop     ax
        mov     bx, 040bh       ; номер исключения
        call    Print_Word
        pop     ax
        mov     bx, 0420h       ; код ошибки
        call    Print_Word
        pop     ax
        mov     bx, 0416h       ; смещение
        call    Print_Word
        pop     ax
        mov     bx, 0411h       ; селектор
        call    Print_Word
        call    set_rmode       ; возвращаемся в реальный режим
        mov     ax, 0   ; ожидаем нажатия на клавишу
        int     16h
        mov     bh, 07h
        call    clrscr
        mov     ah,4Ch
        int     21h
ENDP    shutdown

; ------------------------------------------------
; Перепрограммирование контроллера прерываний
;       На входе: DX – порт контроллера прерывания
;               AH – начальный номер прерывания
; ------------------------------------------------

PROC    set_int_ctrlr   NEAR
        mov     al,11
        out     dx,al
        jmp     SHORT $+2
```

```
        mov     al,ah
        inc     dx
        out     dx,al
        jmp     SHORT $+2
        mov     al,4
        out     dx,al
        jmp     SHORT $+2
        mov     al,1
        out     dx,al
        jmp     SHORT $+2
        mov     al,0ff
        out     dx,al
        dec     dx
        ret
ENDP    set_int_ctrlr

; -------------------------------
; Разрешение линии A20
; -------------------------------
PROC    enable_a20      NEAR
        mov     al,A20_PORT
        out     STATUS_PORT,al
        mov     al,A20_ON
        out     KBD_PORT_A,al
        ret
ENDP    enable_a20

; -------------------------------
; Запрещение линии A20
; -------------------------------

PROC    disable_a20     NEAR
        mov     al,A20_PORT
        out     STATUS_PORT,al
        mov     al,A20_OFF
        out     KBD_PORT_A,al
        ret
ENDP    disable_a20

; ---------- Обработчик аппаратных прерываний IRQ2-IRQ7
```

```
PROC    dummy_iret0     NEAR
        push    ax
; Посылаем сигнал конца прерывания в первый контроллер 8259A
        mov     al,EOI
        out     MASTER8259A,al
        pop     ax
        iret
ENDP    dummy_iret0

; ---------- Обработчик аппаратных прерываний IRQ8-IRQ15

PROC    dummy_iret1     NEAR
        push    ax
; Посылаем сигнал конца прерывания в первый
; и второй контроллеры 8259A
        mov     al,EOI
        out     MASTER8259A,al
        out     SLAVE8259A,al
        pop     ax
        iret
ENDP    dummy_iret1

; ------------------------------------------
; Процедура выдает короткий звуковой сигнал
; ------------------------------------------

PROC    beep    NEAR
        push    ax bx cx
        in      al,KBD_PORT_B
        push    ax
        mov     cx,80
beep0:
        push    cx
        and     al,11111100b
        out     KBD_PORT_B,al
        mov     cx,60
idle1:
        loop    idle1
        or      al,00000010b
```

```
        out     KBD_PORT_B,al
        mov      cx,60
idle2:
        loop    idle2
        pop     cx
        loop    beep0
        pop     ax
        out     KBD_PORT_B,al
        pop     cx bx ax
        ret
ENDP    beep

; ------------------------------------------------
; Процедура задерживает выполнение программы
; на некоторое время, зависящее от быстродействия
; процессора.
; ------------------------------------------------

PROC    pause           NEAR
        push    cx
        mov      cx,10
ploop0:
        push    cx
        xor     cx,cx
ploop1:
        loop    ploop1
        pop     cx
        loop    ploop0
        pop     cx
        ret
ENDP    pause

; ------------------------------------------
;       Процедуры для работы с клавиатурой
; ------------------------------------------

DATASEG
        key_flag        db      0
        key_code        dw      0
```

```
        ext_scan        db      0
        keyb_status     dw      0
CODESEG

; --------------------------------------------
; Обработчик аппаратного прерывания клавиатуры
; --------------------------------------------

PROC    Keyb_int        NEAR
        call    beep    ; выдаем звуковой сигнал
        push    ax
        mov     al, [ext_scan]  ; расширенный скан-код
        cmp     al, 0           ; или обычный ?
        jz      normal_scan1

; --------- обработка расширенного скан-кода ------------

        cmp     al, 0e1h        ; это клавиша <Pause>?
        jz      pause_key
        in      al, 60h         ; вводим скан-код
        cmp     al, 2ah         ; игнорируем префикс 2Ah
        jz      intkeyb_exit_1
        cmp     al, 0aah        ; игнорируем отпускание
        jz      intkeyb_exit_1  ; клавиш
        mov     ah, [ext_scan]  ; записываем скан-код и
        call    Keyb_PutQ       ; расширенный скан-код
                                ; в «очередь», состоящую
                                ; из одного слова
        mov     al, 0           ; сбрасываем признак
        mov     [ext_scan], al  ; получения расширенного
        jmp     intkeyb_exit    ; скан-кода
pause_key:                      ; обработка клавиши <Pause>
        in      al, 60h         ; вводим скан-код
        cmp     al, 0c5h        ; если это код <Pause>,
        jz      pause_key1      ; записываем его в очередь,
        cmp     al, 45h         ; иначе игнорируем
        jz      pause_key1
        jmp     intkeyb_exit
pause_key1:
        mov     ah, [ext_scan]  ; запись в очередь
```

```
       call    Keyb_PutQ       ; кода клавиши <Pause>
       mov     al, 0           ; сбрасываем признак
       mov     [ext_scan], al  ; получения расширенного
       jmp     intkeyb_exit    ; скан-кода

; --------- обработка обычного скан-кода -------------

normal_scan1:
       in      al, 60h         ; вводим скан-код
       cmp     al, 0feh        ; игнорируем FEh
       jz      intkeyb_exit
       cmp     al, 0e1h        ; расширенный скан-код?
       jz      ext_key         ; если да, то на обработку
                               ; расширенного скан-кода
       cmp     al, 0e0h
       jnz     normal_scan

ext_key:
       mov     [ext_scan], al  ; устанавливаем признак
       jmp     intkeyb_exit    ; расширенного скан-кода
; Сброс признака расширенного скан-кода и выход
intkeyb_exit_1:
       mov     al, 0
       mov     [ext_scan], al
       jmp     intkeyb_exit
; Запись нормального скан-кода в очередь и выход
normal_scan:
       mov     ah, 0
       call    Keyb_PutQ
intkeyb_exit:
       in      al, 61h         ; разблокируем клавиатуру
       mov     ah, al
       or      al, 80h
       out     61h, al
       xchg    ah, al
       out     61h, al
       mov     al,EOI          ; посылаем сигнал конца
       out     MASTER8259A,al  ; прерывания
       pop     ax
       sti
```

```
        iret
ENDP    Keyb_int

; ------------------------------------------------
; Запись скан-кода и расширенного скан-кода в
; «буфер», состоящий из одного слова.
; ------------------------------------------------

PROC    Keyb_PutQ       NEAR
        push    ax
        mov     [key_code], ax  ; записываемый код
; ------- Обрабатываем переключающие клавиши ---------
        cmp     ax, 002ah       ; L_SHIFT down
        jnz     @@kb1
        mov     ax, [keyb_status]
        or      ax, L_SHIFT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb1:
        cmp     ax, 00aah       ; L_SHIFT up
        jnz     @@kb2
        mov     ax, [keyb_status]
        and     ax, NL_SHIFT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb2:
        cmp     ax, 0036h       ; R_SHIFT down
        jnz     @@kb3
        mov     ax, [keyb_status]
        or      ax, R_SHIFT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb3:
        cmp     ax, 00b6h       ; R_SHIFT up
        jnz     @@kb4
        mov     ax, [keyb_status]
        and     ax, NR_SHIFT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
```

```
@@kb4:
      cmp     ax, 001dh       ; L_CTRL down
      jnz     @@kb5
      mov     ax, [keyb_status]
      or      ax, L_CTRL
      mov     [keyb_status], ax
      jmp     keyb_putq_exit
@@kb5:
      cmp     ax, 009dh       ; L_CTRL up
      jnz     @@kb6
      mov     ax, [keyb_status]
      and     ax, NL_CTRL
      mov     [keyb_status], ax
      jmp     keyb_putq_exit
@@kb6:
      cmp     ax, 0e01dh      ; R_CTRL down
      jnz     @@kb7
      mov     ax, [keyb_status]
      or      ax, R_CTRL
      mov     [keyb_status], ax
      jmp     keyb_putq_exit
@@kb7:
      cmp     ax, 0e09dh      ; R_CTRL up
      jnz     @@kb8
      mov     ax, [keyb_status]
      and     ax, NR_CTRL
      mov     [keyb_status], ax
      jmp     keyb_putq_exit
@@kb8:
      cmp     ax, 0038h       ; L_ALT down
      jnz     @@kb9
      mov     ax, [keyb_status]
      or      ax, L_ALT
      mov     [keyb_status], ax
      jmp     keyb_putq_exit
@@kb9:
      cmp     ax, 00b8h       ; L_ALT up
      jnz     @@kb10
      mov     ax, [keyb_status]
```

```
        and     ax, NL_ALT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb10:
        cmp     ax, 0e038h      ; R_ALT down
        jnz     @@kb11
        mov     ax, [keyb_status]
        or      ax, R_ALT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb11:
        cmp     ax, 0e0b8h      ; R_ALT up
        jnz     @@kb12
        mov     ax, [keyb_status]
        and     ax, NR_ALT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb12:
        cmp     ax, 003ah       ; CAPS_LOCK up
        jnz     @@kb13
        mov     ax, [keyb_status]
        xor     ax, CAPS_LOCK
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb13:
        cmp     ax, 00bah       ; CAPS_LOCK down
        jnz     @@kb14
        jmp     keyb_putq_exit
@@kb14:
        cmp     ax, 0046h       ; SCR_LOCK up
        jnz     @@kb15
        mov     ax, [keyb_status]
        xor     ax, SCR_LOCK
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb15:
        cmp     ax, 00c6h       ; SCR_LOCK down
        jnz     @@kb16
        jmp     keyb_putq_exit
```

```
@@kb16:
      cmp    ax, 0045h       ; NUM_LOCK up
      jnz    @@kb17
      mov    ax, [keyb_status]
      xor    ax, NUM_LOCK
      mov    [keyb_status], ax
      jmp    keyb_putq_exit
@@kb17:
      cmp    ax, 00c5h       ; NUM_LOCK down
      jnz    @@kb18
      jmp    keyb_putq_exit
@@kb18:
      cmp    ax, 0e052h      ; INSERT up
      jnz    @@kb19
      mov    ax, [keyb_status]
      xor    ax, INSERT
      mov    [keyb_status], ax
      jmp    keyb_putq_exit
@@kb19:
      cmp    ax, 0e0d2h      ; INSERT down
      jnz    @@kb20
      jmp    keyb_putq_exit
@@kb20:

      test   ax, 0080h       ; фильтруем отжатия клавиш
      jnz    keyb_putq_exit
      mov    al, 0ffh        ; устанавиваем признак
      mov    [key_flag], al  ; готовности для чтения
                             ; символа из "буфера"
keyb_putq_exit:
      pop    ax
      ret
ENDP   Keyb_PutQ

; ---------------------------------------------------
; Программное прерывание, предназначенное для чтения
; символа из буфера клавиатуры. По своим функциям
; напоминает прерывание INT 16h реального режима.
; В AX возвращается скан-код нажатой клавиши,
; в BX – состояние переключающих клавиш.
; ---------------------------------------------------

PROC   Int_30h_Entry   NEAR

      push   dx              ; запрещаем прерывания и
      cli                    ; сбрасываем признак
```

```
        mov     al, 0           ; готовности скан-кода
        mov     [key_flag], al  ; в буфере клавиатуры

; Ожидаем прихода прерывания от клавиатуры.
; Процедура клавиатурного прерывания установит
; признак в переменной key_flag.

keyb_int_wait:
        sti             ; разрешаем прерывания
        nop             ; ждем прерывание
        nop
        cli                     ; запрещаем прерывания
        mov     al, [key_flag]  ; и опрашиваем флаг
        cmp     al, 0           ; готовности скан-кода
        jz      keyb_int_wait
        mov     al, 0           ; сбрасываем флаг
        mov     [key_flag], al  ; готовности
        mov     ax, [key_code]    ; записываем скан-код
        mov     bx, [keyb_status] ; и состояние переключающих
                                ; клавиш

        sti                     ; разрешаем прерывания
        pop     dx
        iret
ENDP    Int_30h_Entry

; ------------------------------------------
;       TIMER section
; ------------------------------------------

DATASEG
        timer_cnt dw    0
CODESEG
PROC    Timer_int       NEAR
        cli
        push    ax
; Увеличиваем содержимое счетчика времени
        mov     ax, [timer_cnt]
        inc     ax
        mov     [timer_cnt], ax
; Примерно раз в секунду выдаем звуковой сигнал
        test    ax, 0fh
        jnz     timer_exit
        call    beep
timer_exit:
; Посылаем команду конца прерывания
```

```
        mov     al,EOI
        out     MASTER8259A,al
        pop     ax
        sti
        iret
ENDP    Timer_int

; -------------------------------------------------
; Процедуры обслуживания видеоконтроллера
; -------------------------------------------------

DATASEG
        columns db      80d
        rows    db      25d
        rl_crt  dw      COLOR_SEG
        vir_crt dw      CRT_DESCR
        curr_line       dw      0d
        text_buf        db      "       "
CODESEG
; ----------------------------------------
; Определение адреса видеопамяти
; ----------------------------------------
PROC    set_crt_base    NEAR
        mov     ax,40
        mov     es,ax
        mov     bx,[WORD es:4a]
        mov     [columns],bl
        mov     bl,[BYTE es:84]
        inc     bl
        mov     [rows],bl
        mov     bx,[WORD es:PORT_6845]
        cmp     bx,COLOR_PORT
        je      color_crt
        mov     [rl_crt],MONO_SEG
        mov     [vir_crt],MDA_DESCR
color_crt:
        ret
ENDP    set_crt_base

; ------------------------------------
; Запись строки в видеопамять
; ------------------------------------
PROC    writexy         NEAR
        push    si
        push    di
        mov     dl,[columns]
```

```
        mul     dl
        add     ax,bx
        shl     ax,1
        mov     di,ax
        mov     ah,dh
write_loop:
        lodsb
        stosw
        loop    write_loop
        pop     di
        pop     si
        ret
ENDP    writexy

; ---------------------------------------
; Стирание экрана (в реальном режиме)
; ---------------------------------------
PROC    clrscr          NEAR
        xor     cx,cx
        mov     dl,[columns]
        mov     dh,[rows]
        mov     ax,0600
        int     10
        mov ah,02       ;Установим курсор
        mov bh,00       ;Страница
        mov dh,1        ;Строка
        mov dl,30       ;Столбец
        int 10h

        ret
ENDP    clrscr

DATASEG
```

hello_msg db «Эта программа выводит в защищенном режиме скан-коды клавиш и состояние переключающихся клавиш. При нажатии клавиши 'Esc' возникает исключение и отображается информация об этом исключении».

```
CODESEG
; ------------------------------------
; Вывод начального сообщения
; в защищенном режиме
; ------------------------------------

PROC    write_hello_msg NEAR
        mov     ax,[vir_crt]
        mov     es,ax
        mov     si,OFFSET hello_msg
```

```
        mov     bx,0
        mov     ax,[curr_line]
        inc     [curr_line]
        mov     cx,SIZE hello_msg
        mov     dh,30h
        call    writexy
        call    beep
        ret
ENDP    write_hello_msg

; ---------------------------------------------
; Процедура выводит на экран содержимое AX
;       (x,y) = (bh, bl)
; ---------------------------------------------

PROC Print_Word near
        push ax
        push bx
        push dx

        push ax
        mov cl,8
        rol ax,cl
        call Byte_to_hex
        mov     [text_buf], dh
        mov     [text_buf+1], dl

        pop ax
        call Byte_to_hex
        mov     [text_buf+2], dh
        mov     [text_buf+3], dl

        mov     si, OFFSET text_buf
        mov     dh, 70h
        mov     cx, 4
        mov     al, bh
        mov     ah, 0

        mov     bh, 0
        call    writexy

        pop dx
        pop bx
        pop ax
        ret
ENDP Print_Word

DATASEG
```

```
tabl db '0123456789ABCDEF'
CODESEG
; ----------------------------------------
; Преобразование байта в шестнадцатеричный
; символьный формат
; al – входной байт
; dx – выходное слово
; ----------------------------------------

PROC Byte_to_hex near

      push    cx
      push    bx
      mov     bx, OFFSET tabl

      push    ax
      and     al,0fh
      xlat
      mov     dl,al

      pop     ax
      mov     cl,4
      shr     al,cl
      xlat
      mov     dh,al

      pop bx
      pop cx
      ret

ENDP Byte_to_hex

DATASEG

reg_title      db " CS   IP   AX   BX   CX   DX   SP   BP   SI   DI  "
;                  ....  ....  ....  ....  ....  ....  ....  ....
sreg_title     db " DS   ES   SS   FLAGS                             "
;                  ....  ....  ....  ....  ....  ....  ....  ....

CODESEG

; ----------------------------------------------
; Вывод на экран содержимого регистров процессора
; ----------------------------------------------

PROC    rdump NEAR

      pushf
      pusha
```

```
        mov     di, es

        mov     ax,[vir_crt]
        mov     es,ax
        mov     si,OFFSET reg_title
        mov     bx,1                    ; (X,Y) = (AX,BX)
        mov     ax,6
        mov     cx,SIZE reg_title
        mov     dh,1fh                  ; черный на голубом фоне
        call    writexy

; Выводим содержимое всех регистров

        mov     ax,cs           ; cs
        mov     bx, 0702h
        call    Print_Word

        mov     bp, sp

        mov     ax, [bp+18d]    ; ip
        mov     bx, 0708h
        call    Print_Word

        mov     bx, 070eh
        mov     ax,[bp+14d]     ; ax
        call    Print_Word

        mov     bx, 0714h
        mov     ax,[bp+8d]      ; bx
        call    Print_Word

        mov     bx, 071ah
        mov     ax,[bp+12d]     ; cx
        call    Print_Word

        mov     bx, 0720h
        mov     ax,[bp+10d]     ; dx
        call    Print_Word

        mov     ax,bp
        add     ax,20d          ; sp
        mov     bx, 0726h
        call    Print_Word

        mov     ax,[bp+4d]      ; bp
        mov     bx, 072ch
        call    Print_Word

        mov     bx, 0732h
        mov     ax,[bp+2]       ; si
```

```
        call    Print_Word

        mov     bx, 0738h
        mov     ax, [bp]        ; di
        call    Print_Word

        mov     si,OFFSET sreg_title
        mov     bx,1
        mov     ax,8
        mov     cx,SIZE sreg_title
        mov     dh,1fh
        call    writexy

        mov     bx, 0902h
        mov     ax, ds          ; ds
        call    Print_Word

        mov     bx, 0908h
        mov     ax, di          ; es
        call    Print_Word

        mov     bx, 090eh
        mov     ax,ss           ; ss
        call    Print_Word

        mov     bx, 0914h
        mov     ax, [bp+16d]    ; flags
        call    Print_Word

; Восстанавливаем содержимое регистров

        popa
        popf
        ret
ENDP    rdump

CSEG_SIZE       = ($ - start)

DATASEG

DSEG_SIZE       = ($ - DSEG_BEG)

wrong   db      ?
        END     start
```

ЗАДАНИЕ НА ВТОРУЮ ЧАСТЬ ЛАБОРАТОРНОЙ РАБОТЫ

Изучить принцип работы с прерываниями в защищенном режиме. Проверить выполнение одного из исключений, приведенных в программе и сравнить с исключениями, приведенными в таблице 1.

Таблица А1 – Формат регистра CR0 процессора i80386

Номер бита	Назначение
0 – PE	Включение защищенного режима работы процессора
1 – MP	Присутствие сопроцессора
2 - EM	Эмуляция сопроцессора
3 – TS	Переключение задачи
4 – ET	Тип сопроцессора – i80287 или i80387
5 –14	Зарезервировано
15 – PG	Включение механизма трансляции страниц

Таблица А2 – Формат регистра CR0 процессора i80486

Номер бита	Назначение
0 – PE	Включение защищенного режима работы процессора
1 – MP	Присутствие сопроцессора
2 – EM	Эмуляция сопроцессора
3 – TS	Переключение задачи
4 – ET	Тип сопроцессора – i80287 или i80387
5 – NE	Числовая ошибка. Разрешает обработку ошибок при операциях с плавающей точкой
6–15	Зарезервировано
16 – WP	Защита записи. При установке этого бита страницы пользователя защищены от записи в режиме супервизора
17	Зарезервировано
18 – AM	Бит маски выравнивания. Этот бит разрешает или запрещает контроль выравнивания операндов команд в памяти. Контроль выравнивания разрешен только для программ, работающих в третьем кольце, при условии что установлен бит AM
19 – 28	Зарезервировано
29 – NW	Разрешение сквозной записи. Используется в механизме управления кэшированием
30 – CD	Запрещение кэширования. Если этот бит установлен, внутреннее кэширование запрещено
31 – PG	Включение механизма трансляции страниц

Системные команды процессоров i80386/i80486. Системные команды предназначены для использования, главным образом, в модулях операционных систем (в модулях ядра операционной системы, в драйверах и т.д.). Некоторые

из перечисленных ниже команд полезны и при разработке прикладных программ, работающих в защищенном режиме. Мы приведем только краткий перечень основных системных команд.

Как правило, системные команды могут использовать только те программы, которые выполняются в нулевом привилегированном кольце.

ARPL – Коррекция поля привилегий инициатора запроса в селекторе. Эта команда используется системными модулями для проверки уровня запрашиваемых привилегий в передаваемых им в качестве параметров селекторов. Прикладная программа не должна запрашивать привилегии, превышающие ее собственные.

Первый операнд команды – 16-разрядный регистр или слово памяти, содержащие значение проверяемого селектора. Второй операнд – регистр, в который записано содержимое CS прикладной программы.

Если команда не изменяла уровень привилегий, в регистре FLAGS (EFLAGS для процессоров i80386 и i80486) устанавливается флаг нуля. В противном случае этот флаг сбрасывается.

Пример использования команды:

```
mov    dx, cs
mov    ax, TESTED_SELECTOR
arpl   dx, ax
```

CLTS – Сброс флага TS переключения задачи в регистре CR0

Каждый раз при переключении задачи флаг TS устанавливается в 1. Команда CLTS позволяет сбросить этот флаг.

LAR – Загрузка байта прав доступа

В процессорах i80386 и i80486 команда LAR использует в качестве первого операнда 32-разрядный регистр. Кроме байта прав доступа в этот регистр заносятся биты типа сегмента (9-11), DPL (14), бит присутствия (15), бит дробности (23).

LGDT – Загрузка регистра GDTR

Команда выполняет инициализацию регистра GDTR, указывающего расположение в памяти и размер глобальной таблицы дескрипторов.

LIDT – Загрузка регистра IDTR

Команда выполняет инициализацию регистра IDTR, указывающего расположение в памяти и размер дескрипторной таблицы прерываний.

LLDT – Загрузка регистра LDTR

Команда выполняет инициализацию регистра LDTR, указывающего расположение в памяти и размер локальной таблицы дескрипторов.

LMSW – Загрузка слова состояния процессора

С помощью этой команды можно выполнить загрузку младшего слова регистра CR0 из регистра – операнда команды.

Эта команда может использоваться для переключения процессора в защищенный режим. Обратного переключения эта команда не обеспечивает (даже для процессоров i80386 и i80486).

LSL – Загрузка предела сегмента

Команда имеет два операнда. Граница сегмента, селектор которого используется в качестве второго операнда (задается в регистре), загружается в регистр, указанный в качестве первого операнда.

LTR – Загрузка регистра задачи

Команда предназначена для загрузки регистра TR – регистра задачи. Загрузка этого регистра не приводит к переключению задачи.

MOV – Загрузка системных регистров

Для процессоров i80386 и i80486 в качестве операндов обычной команды MOV допустимо (на нулевом уровне привилегий) указывать системные регистры – CR0, CR2, CR3, DR0, DR1, DR2, DR3, DR6, DR7, TR6, TR7. Команда MOV может быть использована процессорами i80386 и i80486 для возврата процессора из защищенного режима в реальный.

SGDT – Запись в память содержимого регистра GDTR

Команда позволяет узнать текущее содержимое регистра глобальной дескрипторной таблицы GDTR, обычное ее используют в системных отладчиках.

SIDT – Записать в память содержимое регистра IDTR

Команда позволяет узнать текущее содержимое регистра глобальной дескрипторной таблицы прерываний IDTR, используется в системных отладчиках.

SLDT – Записать в память содержимое регистра LDTR

Команда позволяет узнать текущее содержимое регистра локальной дескрипторной таблицы LDTR, используется в системных отладчиках.

SMSW – Записать слова состояния процессора

Команда записывает в память или 16-битовый регистр младшее слово регистра CR0 и может быть использована в системных отладчиках.

STR – Запись регистра задачи

Команда записывает текущее содержимое регистра задачи TR в 16-разрядную ячейку памяти или 16-разрядный регистр. Может использоваться в системных отладчиках.

VERR – Проверить сегмент на возможность чтения

VERW – Проверить сегмент на возможность записи

С помощью этих двух команд можно проверить доступность выбранного селектором сегмента на чтение и запись, соответственно. Если операция чтения или записи доступна, флаг нуля ZF устанавливается в единицу, в противном случае он сбрасывается в ноль.

Основное назначение этой команды – предотвратить возникновение исключения по защите памяти при попытке обращения к сегменту. Прежде чем выполнять обращение, программа может проверить доступность сегмента и сделать соответствующие выводы.

Вопросы для самоконтроля

1 В чем отличие адресации памяти в реальном и в защищенном режимах работы микропроцессора?

2 Какова структура и назначение GDT (Global Descriptor Table)?

3 Какова структура и назначение LDT (Local Descriptor Table)?

4 Зачем нужно запрещать все прерывания при переходе в защищенный режим работы микропроцессора?

5 Каким образом можно осуществить переход из реального режима в защищенный режим работы микропроцессора и обратно?

6 Какие условия (операционная система, наличие/отсутствие драйверов расширенной памяти) необходимы для удачного перехода в защищенный режим и почему?

7 Какова структура и назначение IDT (Interrupt Descriptor Table)?

8 На какие два типа делятся прерывания в защищенном режиме?

9 Что такое «повторная запускаемость» и в чем может быть ее польза?

10 Какая проблема может возникнуть при обработке аппаратных прерываний в защищенном режиме и как с ней бороться?

Список литературы

1 Юров В. Assembler. – Санк-Петербург : Питер, 2001. – 624 с. : ил.

2 Фролов А. В., Фролов Г. В. Защищенный режим процессоров Intel 80286/80386/80486. – Москва : ДИАЛОГ-МИФИ, 1993.

Стукало Виктор Александрович

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ
Методические указания к выполнению лабораторных работ
для студентов очной и заочной формы обучения направления 09.03.04

Редактор Н.Н. Погребняк

Подписано к печати	Формат 60×84 1/16	Бумага 65 г/м^2
Печать цифровая	Усл. печ. л. 4,5	Уч. изд. л. 4,5
Заказ	Тираж 10	Не для продажи

БИЦ Курганского государственного университета.
640020, г. Курган, ул. Советская, 63/4.
Курганский государственный университет.