ЛАБОРАТОРНАЯ РАБОТА 6. ЗАЩИЩЕННЫЙ РЕЖИМ МИКРОПРОЦЕССОРОВ INTEL X86

Основные сведения

Понятие защиты

Микропроцессор 8086, ориентированный на однозадачные системы, как известно, не предлагает никаких средств, позволяющих контролировать поведение выполняющейся в данный момент программы, т.е. программа имеет доступ ко всем имеющимся в системе ресурсам, в том числе и принадлежащим операционной системе. Таким образом, целостность операционной системы может быть нечаянно или умышленно нарушена, и сама система ничем не может защитить себя.

Когда начали появляться микропроцессоры со встроенной поддержкой мультизадачности, вопрос обеспечения защиты вышел на первый план. Действительно, когда процессором одновременно выполняются несколько программ, почти всегда необходимо тщательно изолировать их друг от друга и от операционной системы. Тем самым исключается влияние отдельных программ на целостность системы, и аварийное завершение одной задачи не повлияет на устойчивую работу системы в целом.

Исходя из всего этого при проектировании процессора 80286 были заложены аппаратные средства защиты в основе которых лежит принцип разделения уровней привилегированности и механизм управления памятью.

Механизм защиты распознает четыре уровня привилегий, нумеруемых от 0 до 3. Чем больше номер, тем ниже уровень привилегированности. Поскольку основными системными объектами, которыми оперирует процессор, являются сегменты памяти, то с каждым сегментом кода, данных или стека ассоциируется некий уровень привилегий, и все, что находится внутри этого сегмента, имеет этот уровень привилегий.

Поскольку число программ, которые могут выполняться на более высоком уровне привилегий, уменьшается в направлении уровня 0, то уровни привилегий обычно изображают в виде колец защиты (рис. 2.1). Центральное кольцо защиты содержит

сегменты, в которых находится наиболее важное программное обеспечение, обычно ядро операционной системы. Прикладным программам, как правило, присваивается низший уровень привилегий и запрещается самостоятельная работа с системными средствами. Промежуточные уровни привилегий обычно присваиваются драйверам периферийных устройств, для того чтобы защитить операционную систему от сбоев в драйверах и одновременно защитить драйверы от прикладных программ.

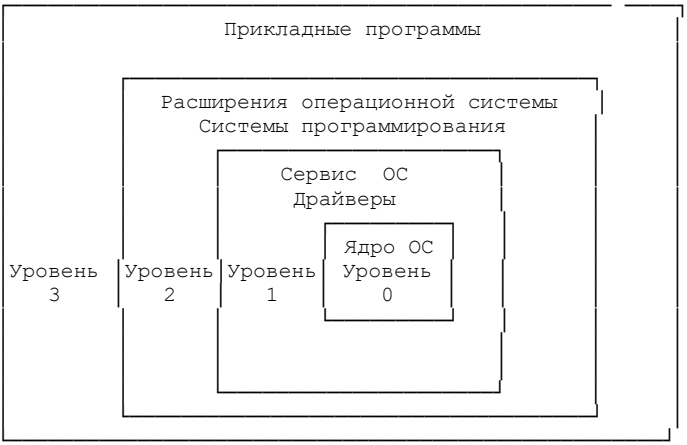


Рис. 2.1. Уровни привилегий

Хотя для отмены механизма защиты никакого управляющего регистра или бита режима не предусмотрено, того же эффекта можно достигнуть путем присвоения всем сегментам уровня привилегий 0. Отметим, что все средства работы с уровнями привилегий доступны только в защищенном режиме, являющемся основным режимом работы новых моделей процессоров. Если защищенный режим выключен, то процессор работает в режиме, который просто реализует возможности обладающего более высоким быстродействием и улучшенного процессора 8086.

Организация памяти в защищенном режиме

Основой организации памяти в защищенном режиме являются сегментация и подкачка страниц. Сегментация служит для того, чтобы дать каждой программе несколько независимых, защищенных адресных пространств. Подкачка страниц используется для создания среды, в которой большие адресные пространства моделируются на базе сравнительно небольшой области оперативной памяти и некоторого объема дисковой памяти. Подкачка страниц обеспечивает доступ к структурам данных, превышающих по размеру доступное пространство оперативной памяти, благодаря тому, что часть таких структур держится в оперативной памяти, а часть - на диске.

Аппаратное обеспечение сегментации транслирует сегментированный (логический) адрес в адрес непрерывного адресного пространства, который называется линейным адресом. Если разрешена подкачка страниц, то аппаратное обеспечение подкачки транслирует линейный адрес в физический. Если подкачка страниц запрещена, то в качестве физического адреса используется линейный адрес. Физический адрес подается на адресную шину процессора.

Каждый сегмент памяти в системе характеризуется 8-байтной структурой данных, называемой ДЕСКРИПТОРОМ СЕГМЕНТА, которая сообщает процессору размер и расположение в памяти сегмента, а также управляющую информацию и о состоянии сегмента. Дескрипторы обычно создаются компиляторами, компоновщиками, загрузчиками или операционной системой, но не прикладными программами.

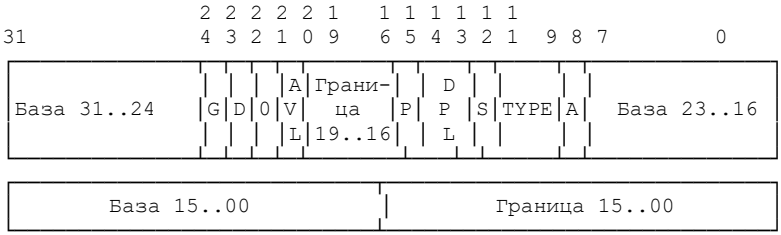
На рис 2.2 показан общий формат 32-битного дескриптора для процессора 80386. В процессоре 80286 старшее слово дескриптора не используется (зарезервировано и должно содержать нули).

Назначение полей дескриптора:

- базовый адрес сегмента определяет расположение сегмента в пределах 4-гигабайтного физического адресного пространства. Занимает байты 2, 3, 4 и 7 дескриптора. При задании нулевого смещения именно этот адрес будет сформирован процессором в качестве линейного;

- бит грануляции (G) включает масштабирование поля границы масштабным коэффициентом 4096. Если этот бит очи-

щен, то размер сегмента измеряется в байтах, а если бит G установлен - в страницах (блоках памяти размером 4Кбайт);



База - базовый адрес сегмента
Граница - граница сегмента
DPL - уровень привилегированности дескриптора
S - тип сегмента (0=системный; 1=прикладной)
G - грануляция
P - присутствие сегмента
TYPE - тип сегмента
A - бит обращения к сегменту
D - размер операции по умолчанию (распознается только в дескрипторах кодового сегмента: 0 = 16-битовый размер; 1 = 32-битовый размер)
AVL - доступно для использования системным программным обеспечением

Рис. 2.2. Дескриптор сегмента 32-битного процессора x86

- граница сегмента (предел) определяет размер сегмента. Процессор помещает рядом два поля границы сегмента, в совокупности образующих одно 20-разрядное значение. Размер сегмента определяется одним из двух следующих способов, в зависимости от установки бита грануляции:

1. Если бит грануляции очищен, то размер сегмента может принимать значения от 1 байта до 1 Мбайта, с приращениями в один байт.

2. Если бит грануляции установлен, то сегмент может иметь размер от 4 Кбайт до 4 Гбайт, с приращениями в 4 Кбайт.

Для большинства сегментов логический адрес может иметь значение смещения в диапазоне от 0 до значения грани-

цы. Прочие значения ведут к генерации особых ситуаций. Сегменты с расширением вниз (см. описание поля типа) изменяют смысл поля границы на противоположный: они позволяют адресацию любыми значениями смещения, кроме значений от 0 до значения границы. Такие сегменты обычно используются для хранения стеков;

- бит размера по умолчанию (D) указывает длину операндов и исполнительныхадресов сегмента по умолчанию. Если D=1, то предполагается режим 32-разрядных операндов и исполнительных адресов. Если этот бит очищен, то предполагается использование 16-разрядных операндов и адресов. Бит D обеспечивает совместимость с процессором 80286;

- бит присутствия сегмента (P). Если этот бит очищен, то процессор при загрузке в сегментный регистр селектора данного дескриптора генерирует особую ситуацию 11 ("сегмент не присутствует"). Это свойство используется для обнаружения попыток доступа к сегментам, которые стали недоступными. Операционная система может передавать содержимое некоторых сегментов на диск, если, например, физическая память заполнена. В этом случае, передаваемый на диск сегмент отмечается в своем дескрипторе, как временно не присутствующий. Когда программа обратится к не присутствующему сегменту, возникнет особая ситуация, и система может считать нужный сегмент с диска в память. Сегмент может быть помечен операционной системой как не присутствующий, если его по какой-либо причине необходимо "спрятать";

- поле уровня привилегий (DPL, Descriptor Privilege Level) определяет уровень привилегированности сегмента. Используется для управления доступом к сегменту при помощи механизма защиты;

- бит системного сегмента (S). В сегментах кода или данных данный бит всегда установлен в состояние 1, а конкретное назначение сегмента описывается полем TYPE. Если S=0, то дескриптор описывает системный объект, который может и не являться сегментом памяти. Системные дескрипторы будут описаны ниже;

- бит обращения к сегменту (A) устанавливается процессором при обращении к сегменту, определяемому данным дескриптором;

- поле типа сегмента (TYPE). Интерпретация этого поля зависит от того, относится ли данный дескриптор к прикладному, или же к системному сегменту. Системные сегменты имеют несколько иной формат дескриптора, рассматриваемый ниже. Поле Типа дескриптора памяти задает тип доступа, разрешенного к данному сегменту, а также направление расширения сегмента (табл. 2.1).

Таблица 2.1

Типы прикладных сегментов

Значение поля TYPE	Тип сегмента
000b	Сегмент данных, разрешено только чтение
001b	Сегмент данных, разрешено чтение и запись
010b	Сегмент стека, разрешено только чтение
011b	Сегмент стека, разрешено чтение и запись
100b	Сегмент кода, разрешено только выполнение
101b	Сегмент кода, разрешено выполнение и чтение
100b	Подчиненный (конформный) сегмент кода, разрешено только выполнение
100b	Подчиненный (конформный) сегмент кода, разрешено выполнение и чтение

Поле типа однозначно определяет правила доступа к сегментам. Например, в регистр CS нельзя загружать селекторы сегментов данных, в регистр SS может быть загружен только селектор стекового сегмента, никакая программа не сможет модифицировать данные в сегменте, для которого разрешено только считывание. В случае возникновения такого рода ситуаций генерируется особая ситуация общей защиты (прерывание 13).

Кодовые сегменты могут быть либо конформными, либо неконформными. Переход выполнения в более привилегированный конформный сегмент сохраняет текущий уровень привилегированности. Переход выполнения в неконформный сегмент с другим уровнем привилегированности приводит к генерации

особой ситуации общей защиты, если не использован шлюз задачи.

Байт 5 дескриптора (который содержит поля DPL, S, P и TYPE) называется байтом прав доступа.

В системе с процессорами x86 при работе может быть создано довольно много сегментов и описывающих их дескрипторов, поэтому для удобства все дескрипторы объединены вместе в ДЕСКРИПТОРНУЮ ТАБЛИЦУ. Существует три типа дескрипторных таблиц:

1. Глобальная дескрипторная таблица GDT. Она является основной общесистемной таблицей, и при работе ее могут использовать все программы. Первый дескриптор в таблице GDT не используется процессором. Селектор сегмента для данного "пустого дескриптора" при загрузке его в сегментный регистр не генерирует особой ситуации, однако особая ситуация генерируется всякий раз при попытке доступа к памяти с использованием такого дескриптора.

2. Локальная дескрипторная таблица LDT. В *мультизадачной* системе для каждой задачи можно построить свою локальную дескрипторную таблицу, которая определяет сегменты, доступные только этой конкретной задаче.

3. Таблица дескрипторов прерываний IDT. Она также является общесистемной таблицей и содержит дескрипторы специальных системных объектов - шлюзов (gate), которые определяют точки входа процедур обработки прерываний и особых ситуаций. Эта таблица служит заменой таблицы векторов прерываний процессора 8086.

Таблицы GDT и LDT имеют переменную длину и могут содержать до 8192 (2^13) дескрипторов.

Четыре регистра процессора (рис. 2.3) задают расположение структур данных, которые управляют организацией сегментированной памяти. Для загрузки и сохранения этих регистров имеются специальные команды.

В процессоре 80286 регистры GDTR и IDTR имеют длину 5 байт, а старший байт в них не используется, поэтому в них загружается 24-битовый линейный базовый адрес.

Регистр таблицы глобальных дескрипторов GDTR. Этот регистр содержит 32-битовый базовый адрес и 16-битовую гра-

ницу сегмента таблицы глобальных дескрипторов (GDT). При ссылке на данные в памяти для того, чтобы найти дескриптор сегмента в GDT или LDT используется селектор сегмента. Дескриптор сегмента содержит базовый адрес сегмента.

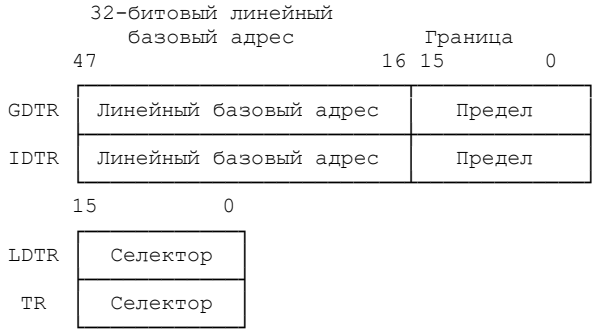


Рис. 2.3. Регистры управления памятью процессора x86

Регистр таблицы локальных дескрипторов LDTR и регистр задачи TR имеют длину 16 бит. Они выполняют функции сегментных регистров, т.к. определяют через таблицу GDT два важнейших сегмента, относящиеся к текущей задаче. Регистр LDTR содержит селектор сегмента локальной дескрипторной таблицы для текущей задачи, а регистр TR определяет текущий сегмент состояния задачи TSS. Использование этих двух регистров будет рассмотрено при описании мультизадачности.

Регистр таблицы дескрипторов прерываний IDTR. Он является точной копией регистра GDTR, только адресует в памяти таблицу дескрипторов прерываний. Когда происходит прерывание или особая ситуация, вектор прерывания используется в качестве индекса для получения из данной таблицы дескриптора шлюза. Последний содержит указатель, используемый для запуска обработчика прерывания.

Граничное значение (предел) выражается в байтах. Как и для сегментов, оно складывается с базовым адресом для получения адреса последнего допустимого байта. Поскольку дескрипторы всегда имеют размер восемь байтов, то граница всегда должна быть на единицу меньше целочисленного произведения восьми, т.е. 8*N*-1.

Рассмотрим, как процессор использует дескрипторы после их определения. Для доступа к дескриптору служит содержимое сегментного регистра, которое называется СЕЛЕКТОРОМ. Длина сегментного регистра, в целях совместимости с предыдущими процессорами, равна 16 бит, однако его содержимое интерпретируется иначе. Формат селектора сегмента показан на рис. 2.4.

TI (Table Indicator) - индикатор таблицы (0=GDT, 1=LDT);

RPL (Requested Privilege Level) - запрошенный уровень привилегий

Рис. 2.4. Селектор сегмента

Бит-индикатор таблицы (TI) задает используемую дескрипторную таблицу. Если этот бит очищен, то дескриптор выбирается из таблицы GDT; если же он установлен - то из текущей таблицы LDT.

Индекс выбирает один из 8192 дескрипторов в таблице дескрипторов. Индекс умножается на восемь (число байтов в дескрипторе сегмента) и складывается с 32-битным базовым адресом дескрипторной таблицы, который берется из регистра GDTR, либо из регистра LDTR, в зависимости от состояния бита TI.

Запрошенный уровень привилегий (RPL): если это поле содержит уровень привилегий с большим значением, чем программа (т.е. привилегированность меньше), то оно переопределяет уровень привилегий программы. Если программа использует менее привилегированный селектор сегмента, то доступ к памяти происходит с меньшим уровнем привилегий.

С каждым сегментным регистром, а также с регистрами LDTR и TR ассоциируется "теневой" или кэш-регистр (рис. 2.5).

Кэш-регистры невидимы и явно недоступны программам.

Когда программа загружает селектор в сегментный регистр, процессор автоматически считывает соответствующий дескриптор в кэш-регистр. После этого процессор уже не обращается к таблицам дескрипторов до тех пор, пока не произойдет загрузка другого селектора сегмента. Частые изменения сегментных регистров в защищенном режиме ухудшают быстродействие программы, поскольку при этом тратится время на обращение к дескрипторным таблицам.

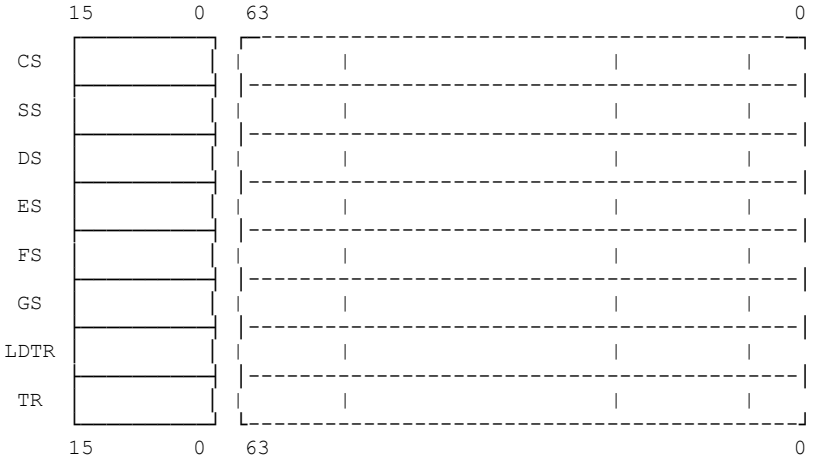


Рис 2.5. Теневые регистры

Для поддержки защищенного режима в регистр флагов были введены новые поля:

- IOPL (In/Out Privilege Level): уровень привилегий ввода-вывода. Показывает минимальный уровень привилегий выполняющейся задачи, на котором разрешено производить операции ввода/вывода;

- NT (Nested Task): вложенная задача. С помощью этого флага организуется цепь вложенных задач, аналогично вложению подпрограмм. Команда IRET проверяет состояние этого флага и при NT=1 происходит переключение задачи, а при NT=0 - обычный возврат из прерывания;

- RF (Resume Flag): флаг возобновления. Действует совместно с регистрами отладки. Временно запрещает особые ситу-

ации отладки, для возможности осуществить рестарт команды без немедленного формирования еще одной особой ситуации отладки;

- VM (Virtual Mode): виртуальный режим. При V=1 процессор переходит в режим эмуляции 8086 из защищенного режима.

Прикладными программами эти флаги должны игнорироваться, а попытки модификации их состояния из прикладных программ недопустимы. В большинстве систем попытка изменения системного флага из прикладной программы приводит к возникновению особой ситуации).

Системные дескрипторы

В дополнение к дескрипторам сегментов кода и данных имеются дескрипторы для системных сегментов и шлюзов, используемые для управления задачами, особыми ситуациями и прерываниями.

В байте прав доступа такого дескриптора 3-битовое поле TYPE объединено с битом A, образуя 4-битовое поле TYPE, определяющее тип системного дескриптора (табл. 2.2).

Формат всех системных дескрипторов 32-битного процессора x86 представлен на рис 2.6. В процессоре 80286 старшее слово не используется и всегда равно нулю; в остальном системные дескрипторы 80286 отличаются от 32-битных дескрипторов только значениями битов поля TYPE (биты 8...11).

Шлюз вызова

Шлюзы вызовов используются для передачи управления между различными уровнями привилегированности. Их использование необходимо только в системах, где имеется более одного уровня привилегированности.

Шлюз вызова имеет две основные функции:

1. Определение точки входа в процедуру.

2. Задание уровня привилегированности, требуемого для входа в процедуру.

Таблица 2.2

Типы системных дескрипторов

TYPE	Тип объекта
0000b	Зарезервирован
0001b	Доступный TSS процессора 80286
0011b	Таблица LDT
0010b	Занятый TSS процессора 80286
0100b	Шлюз вызова
0101b	Шлюз задачи
0110b	Шлюз прерывания процессора 80286
0111b	Шлюз ловушки процессора 80286
1000b	Зарезервирован
1001b	Доступный TSS процессора 80386
1010b	Зарезервирован
1011b	Занятый TSS процессора 80386
1100b	Шлюз вызова процессора 80386
1101b	Зарезервирован
1110b	Шлюз прерывания процессора 80386
1111b	Шлюз ловушки процессора 80386

Дескрипторы шлюзов вызова используются командами CALL и JMP аналогичным способом, что и дескрипторы кодовых сегментов. Они могут находиться в GDT или в LDT, но не в таблице дескрипторов прерываний (IDT).

Поля селектора и смещения в шлюзе образуют указатель на точку входа в процедуру. Шлюз вызова гарантирует, что любые передачи управления другим сегментам будут происходить в правильные точки входа процедуры. Операнд команды передачи управления не является селектором сегмента и смещением в сегменте точки входа желаемой процедуры. Вместо этого селектор сегмента указывает на шлюзовой селектор, а смещение не используется.

Уровнями привилегированности, проверяемыми при передаче управления через шлюз вызова, являются:

1. CPL (текущий уровень привилегированности).

2. RPL (уровень привилегированности источника запроса) селектора сегмента, который был использован для задания шлюза вызова.

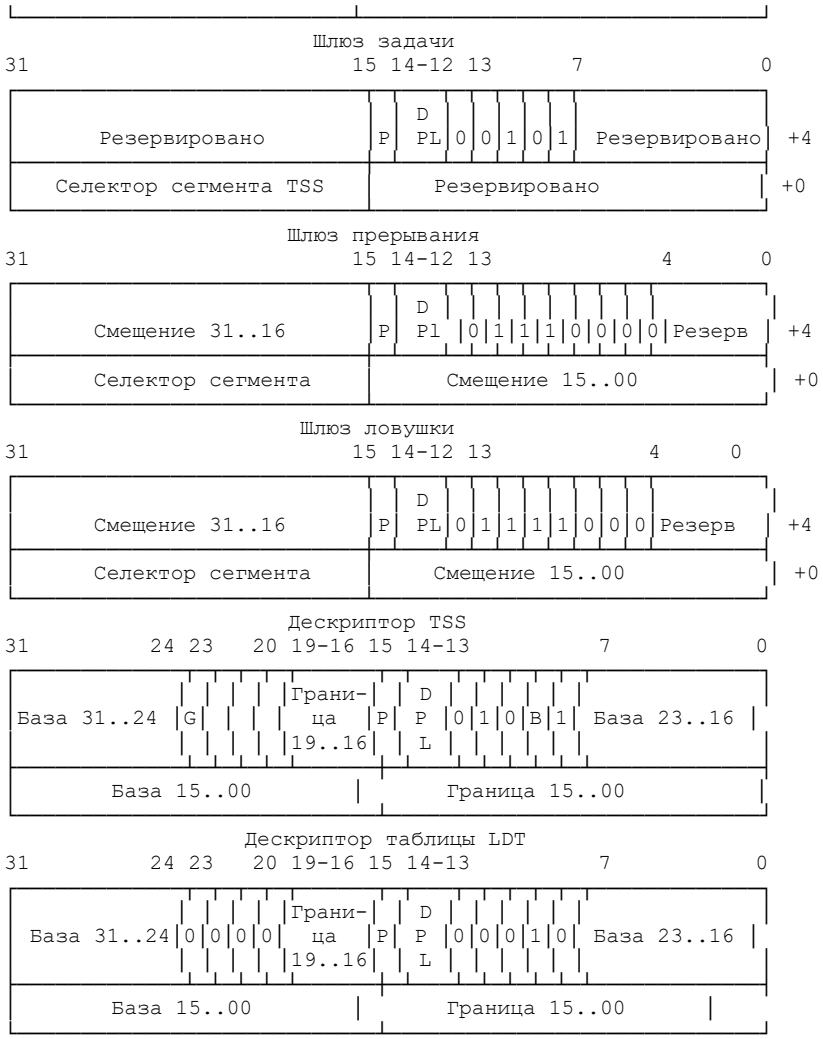


Рис. 2.6. Системные дескрипторы

4. DPL дескриптора сегмента кодового сегмента назначения перехода.

Поле DPL дескриптора шлюза определяет, из каких уровней привилегированности может использоваться данный шлюз.

Шлюзы могут использоваться для передачи управления как на более привилегированные уровни, так и на тот же уровень привилегированности (хотя в последнем случае их использование не оправдано). Для передачи управления на более привилегированные уровни шлюзы могут использоваться только командами CALL. Команда JMP может использовать шлюз только для передачи управления кодовому сегменту с тем же уровнем привилегированности или конформному кодовому сегменту с тем же или более высоким уровнем привилегированности.

Для команды CALL при обращении к неконформному сегменту должны удовлетворяться следующие два правила привилегированности (в противном случае генерируется особая ситуация общей защиты):

1) MAX(CPL,RPL) <= DPL шлюза;

2) DPL кодового сегмента назначения < CPL.

Таблица дескрипторов прерываний

Таблица дескрипторов прерываний (IDT) ассоциирует каждый вектор прерывания или особой ситуации с дескриптором процедуры или задачи, которая обслуживает соответствующее событие. Подобно таблицам GDT и LDT, IDT представляет собой массив 8-байтовых дескрипторов. В отличие т глобальной дескрипторной таблицы, первый элемент таблицы IDT содержит действительный дескриптор.

Для формирования индекса в IDT процессор умножает вектор на масштабный коэффициент 8, т.е. число байтов дескриптора. Поскольку существует всего 256 векторов, IDT не может содержать более 256 дескрипторов.

IDT может находиться в любой области физической памяти. Процессор находит IDT при помощи регистра IDTR (рис. 2.7). Этот регистр содержит как 32-разрядный базовый адрес, так и 16-разрядную границу IDT. Команды LIDT и SIDT выполняют загрузку и сохранение содержимого регистра IDTR. Если вектор ссылается на дескриптор вне заданной границы, процессор входит в режим отключения. В этом режиме процессор прекращает выполнение команд до приема немаскируемого прерывания или поступления сигнала RESET. Процессор генерирует

специальный цикл шины, указывающий на то, что он вошел в режим отключения.

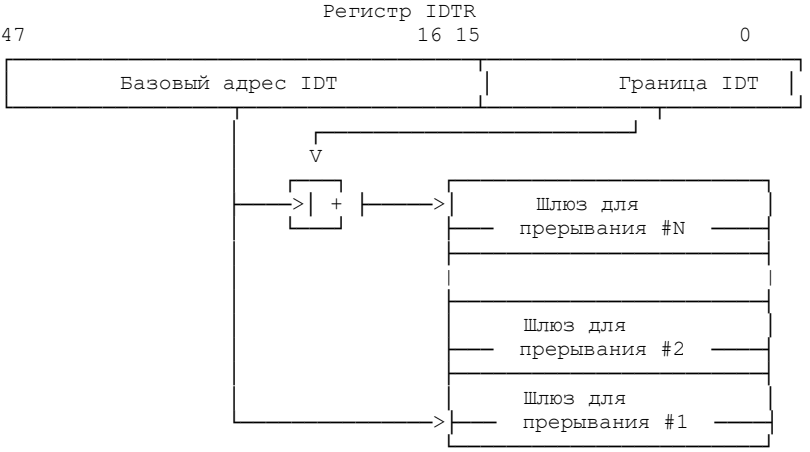


Рис. 2.7. Регистр IDTR определяет положение IDT в памяти

Дескрипторы IDT

Таблица дескрипторов прерываний может содержать любые из трех видов системных дескрипторов (рис. 2.6): шлюзы задачи, шлюзы прерывания и шлюзы ловушки.

Если вектор прерывания индексирует шлюз прерывания или шлюз ловушки, то вызов обработчика происходит аналогично вызову при помощи CALL шлюза вызова. Если же вектор индексирует шлюз задачи, это приводит к переключению задачи аналогично вызову при помощи CALL шлюза задачи.

При возникновении особой ситуации или прерывания процессор сохраняет текущее состояние в стеке (рис. 2.8).

При возникновении некоторых особых ситуаций в защищенном режиме процессором помещается на вершину стека (после вызова процедуры прерывания) код ошибки (рис. 2.8а).

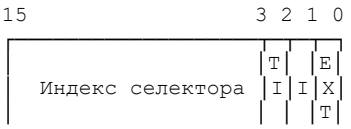


Рис. 2.8а. Формат кода ошибки

Процессор устанавливает бит EXT, если особая ситуация вызвана событием, внешним по отношению к программе.

Процессор устанавливает I-бит (IDT-bit), если индексная часть кода ошибки ссылается к дескриптору шлюза в IDT.

Если бит IDT не установлен, то бит TI указывает на то, ссылается ли код ошибки к GDT (бит TI очищен), или же к LDT (бит TI установлен). Остальные 14 битов - это старшие биты селектора сегмента. В некоторых случаях код ошибки является пустым (т.е. все биты его младшего слова очищены).

В 32-битном процессоре x86 код ошибки помещается в стек в виде двойного слова, старшая половина которого резервируется.

Полная информация о используемых прерываниях и особых ситуациях приведена в приложении 2.

Шлюз ловушки. Процессор осуществляет межсегментную передачу управления, используя селектор и смещение, содержащиеся в шлюзе ловушки. Селектор не может адресовать никакой другой сегмент, кроме сегмента кода. При достижении команды IRET из стека восстанавливается старое состояние и выполнение прерванной задачи возобновляется.

Шлюз прерывания. Различие между шлюзом прерывания и шлюзом ловушки состоит в их воздействии на флаг IF. Прерывание, использующее шлюз прерывания, очищает флаг IF, предотвращая тем самым влияние на текущий обработчик прочих возможных аппаратных прерываний. Последующая команда IRET восстанавливает флаг IF в состояние, которое он имел в сохраненном в стеке регистре EFLAGS. Прерывание, использующее шлюз ловушки, не изменяет флаг IF. Прерывания, использующие шлюзы прерывания или ловушки, вызывают очистку флага TF после того, как его текущее значение сохранено в стеке как часть регистра EFLAGS. Этим процессор предотвращает воздействие трассировки команд на реакцию прерывания. Последующая команда IRET восстанавливает флаг TF из сохраненного регистра EFLAGS.

Шлюз задачи. Шлюз задачи в IDT косвенно ссылается на задачу. Селектор сегмента в шлюзе задачи адресует дескриптор TSS в GDT. Когда прерывание или особая ситуация вызывает шлюз задачи в IDT, происходит переключение задач. Обработка прерывания в отдельной задаче имеет следующие преимущества:

1. Автоматически выполняется полное сохранение всего контекста прерванной задачи.

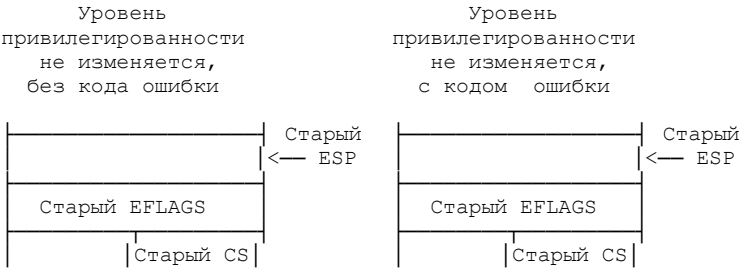
2. Обработчик прерывания может быть изолирован от прочих задач за счет выделяемого ему отдельного адресного пространства.

Переключение задачи, вызванное прерыванием, работает аналогично другим переключениям задач. Задача прерывания возвращается к прерванной задаче, выполняя команду IRET.

Правило привилегированности, управляющее процедурами прерывания, аналогично правилу, действующему при вызове процедуры: процессор не разрешает прерыванию передавать управление менее привилегированному сегменту кода. Попытка нарушить это правило приводит к возникновению особой ситуации общей защиты. В связи с непредсказуемостью прерываний и особых ситуаций требуется гарантировать соблюдение правил защиты по привилегиям. Это достигается одним из 2 способов:

1. Обработчики помещаются в кодовый сегмент с уровнем привилегированности 0. Такие обработчики будут выполняться всегда, независимо от CPL программы;

2. Обработчики помещаются в подчиненных сегментах кода.



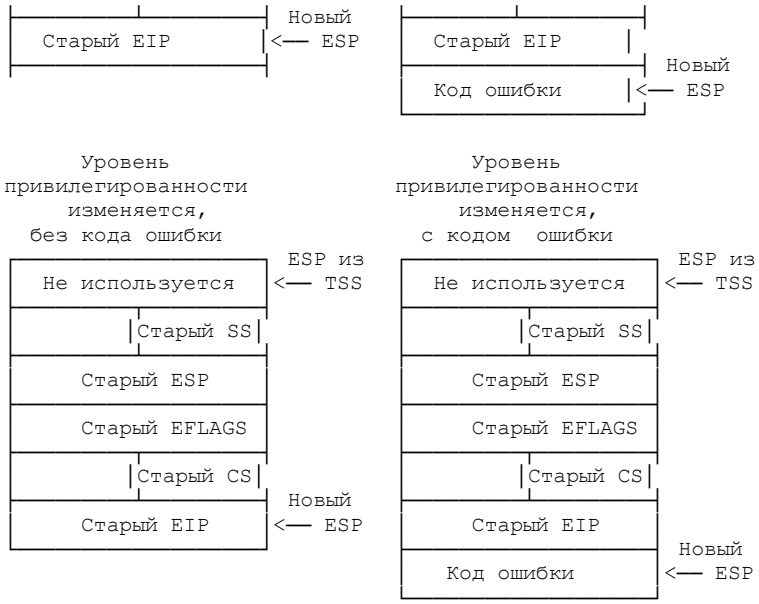


Рис. 2.8. Сохранение состояния в стеке

Переключение в защищенный режим

При включении питания устанавливается реальный режим работы процессора. Для того, чтобы переключить процессор в защищенный режим, в общем случае необходимо выполнить следующие действия:

- подготовить в оперативной памяти глобальную дескрипторную таблицу GDT. В ней должны быть созданы дескрипторы для сегментов, которые будут нужны при работе программы. Создаются как минимум два дескриптора - сегмента кода и данных;

- если предполагается возврат в реальный режим, то необходимо знать полный адрес, с которого будет продолжено выполнение после выхода из защищенного режима (в форме сегмент: смещение). Также необходимо запомнить содержимое сегментных регистров;

- если программа в защищенном режиме будет работать с прерываниями, то необходимо создать таблицу дескрипторов

прерываний IDT, в которой описать шлюзы обработчиков прерываний и особых ситуаций;

- запретить все прерывания;

- открыть адресную линию A20. В реальном режиме эта линия, управляемая процессором клавиатуры 8042, может быть закрыта (всегда значение 0), и расширенная память за пределами первого мегабайта становится недоступной. Чтобы программа в защищенном режиме могла обращаться к расширенной памяти необходимо открыть прохождение сигнала через линию A20. Для этого необходимо записать в порт A 8042 (порт 60h) байт 0DFh, что послужит командой для открывания A20. Для закрывания A20 нужно записать в тот же порт байт 0DDh (в современных компьютерах не нужно);

- загрузить регистр GDTR;

- если создана таблица IDT, то загрузить регистр IDTR;

- переключиться в защищенный режим, установив бит 0 в регистре CR0;

- выполнить команду межсегментного перехода для очистки очереди команд и загрузки регистра CS селектором кодового сегмента.

После выполнения перечисленных пунктов программа может приступать к работе в защищенном режиме.

Переключение в реальный режим

Для того чтобы переключиться из защищенного режима в реальный программа должна выполнить следующие действия:

- передать управление сегменту с пределом 64 Кбайт. Это загрузит регистр CS границей, требуемой для реального режима;

- загрузить в сегментные регистры SS, DS, ES, FS и GS селектор дескриптора, содержащего подходящие для реального режима значения: предел = 0FFFFh, бит G = 0 (байтная гранулярность), бит E = 0 (расширение вверх), бит W = 1 (записываемый), бит P = 1 (присутствующий);

- запретить все прерывания;

- сбросить бит 0 в регистре CR0. После этого выполнение продолжается в реальном режиме;

- выполнить команду межсегментного перехода на программу реального режима для очистки очереди команд и загрузки регистра CS сегментным адресом;

- загрузить в регистр LIDT базу и предел таблицы прерываний реального режима;

- разрешить прерывания;

- загрузить сегментные регистры значениями, необходимыми для работы в реальном режиме;

- далее программа может продолжать работу в реальном режиме.

В процессоре 80286 не было предусмотрено программное переключение в реальный режим и при написании программ пользовались следующей особенностью систем PC AT: если записать в ячейку 0Fh CMOS-памяти значение 05h, а затем выполнить сброс процессора или перевод его в состояние отключения, то процессор перейдет в реальный режим и управление будет передано по адресу, содержащемуся в ячейке 0040:0067h. При этом будет восстановлено состояние контроллеров прерываний. Сброс процессора можно инициировать путем вывода байта 0FEh в порт 64h (процессор клавиатуры 8042). Таким способом, например, выходит в реальный режим программа Multi.asm, приведенная в лабораторной работе 3.

Проиллюстрируем все вышесказанное на примере программы, выполняющей переход в защищенный режим, простейшие действия с экраном и возврат в реальный режим. Рассмотрим работу программы:

- формируем в памяти таблицы GDT и IDT. GDT содержит дескриптор, описывающий саму таблицу GDT; дескриптор, описывающий сегмент программы как сегмент кода; дескриптор, описывающий сегмент программы как сегмент данных; дескрипторы, описывающие таблицу IDT в реальном и в защищенном режимах; дескриптор, описывающий сегмент текстового видеобуфера. Таблица IDT содержит 16 шлюзов ловушек, ссылающихся на обработчики исключительных ситуаций, 22 пустых дескриптора и 1 шлюз прерывания, который ссылается на процедуру вывода строки на экран. Такая структура таблицы IDT приведена в качестве примера использования прерываний;

- запрещаем прерывания;

- открываем адресную линию A20;

- формируем в памяти адреса для косвенных межсегментных переходов, которые нужно будет выполнить, сразу после переключения режимов работы процессора, для загрузки CS новым содержимым и очистки очереди команд;

- загружаем регистры GDTR и IDTR;

- переключаемся в защищенный режим, устанавливая бит 0 регистра CR0;

- выполняем косвенный межсегментный переход по адресу, сформированному в памяти (Protect_Jump). В результате в регистр CS заносится селектор дескриптора кодового сегмента (CS_CODE), и управление передается на метку Protect;

- заносим в сегментные регистры DS, ES и SS селектор дескриптора сегмента данных;

- вызываем рабочую процедуру, выполняющую очистку экрана и вывод сообщения, причем сообщение выводится при помощи вызова через шлюз прерывания функции вывода строки (присутствующей в том же сегменте);

- запрещаем прерывания;

- переключаемся в реальный режим, очищая бит 0 регистра CR0;

- выполняем косвенный межсегментный переход по адресу, сформированному в памяти (Real_Jump). В результате в регистр CS заносится сегментный адрес программы, и управление передается на метку Real;

- загружаем регистр IDTR значением, необходимым для адресации таблицы прерываний реального режима (База = 0000:0000, граница = 3FFh);

- восстанавливаем сегментные регистры;

- закрываем адресную линию A20;

- разрешаем прерывания;

- передаем управление в DOS.

```
;Программа транслируется в COM-файл:
; TASM demo.asm
; Tlink demo.obj /t
; Demo.asm
.386p                    ; Разрешение трансляции
                         ; всех инструкций 80386

Gdt_Descriptor    STRUC; Шаблон дескриптора GDT
 Seg_Limit   dw   0     ; Длина сегмента
 Base_Lo_Word      dw   0; Младшие 16 бит базового
                         ; адреса
 Base_Hi_Byte      db   0 ; Биты 16..23 базового адреса
 Acces_Rights      db 0; Байт прав доступа
                   db 0
 Base_Top_Byte     db 0; Биты 24..31 базового
                        ; адреса
Gdt_Descriptor    ENDS

Idt_Descriptor    STRUC   ; Шаблон дескриптора IDT
 Int_Offset  dw   0       ; Точка входа в процедуру
                          ;  обработки прерывания
 Int_Selector      dw   0 ; Селектор сегмента в GDT
             db   0          ;
 Access      db   0          ; Права доступа
             dw   0          ;
Idt_Descriptor    ENDS

Code_Seg_Access  Equ  10011011b; Байт прав  доступа
                      ; дескриптора сегмента кода
Data_Seg_Access  Equ  10010011b; Байт прав  доступа
                 ; дескриптора сегмента данных
Disable_Bit20    Equ  11011101b; Код команды 8042
                 ; для закрывания линии A20
Enable_Bit20     Equ  11011111b; Код команды 8042
                      ; для открывания линии A20
Port_A            Equ  060h       ; Порт A 8042
Status_port  Equ  064h       ; Порт состояния 8042
Cmos_Port    Equ  070h  ; Адрес порта CMOS-памяти

; Макро для записи базового адреса сегмента
; в дескриптор
```

```
FILLDESCR MACRO   Seg_Addr,Offset_Addr,Descr
         xor      edx,edx              ; EDX := 0
         xor      ecx,ecx              ; ECX := 0
         mov      dx,Seg_Addr; Сегментная часть
         mov      cx,offset Offset_Addr; Смещение
         call    Form_32Bit_Address; CX:DX :=
                                    ; линейный адрес
; Занесение базового адреса в дескриптор
         mov      &Descr.Base_Lo_Word,dx;
         mov      &Descr.Base_Hi_Byte,cl
         mov      &Descr.Base_Top_Byte,ch
        ENDM

CSEG     SEGMENT  Para USE16 public 'code'
         ASSUME  cs:Cseg,ds:Cseg
         ORG         100h
Start:         jmp         Main

; Глобальная дескрипторная таблица GDT
 EVEN
Gdt   label word
; Дескриптор, описывающий саму таблицу GDT
 Gdt_Desc   EQU $-gdt; Селектор дескриптора
 Gdt1 Gdt_Descriptor <gdt_leng,,,data_seg_access,>

; Дескриптор, описывающий сегмент Cseg как кодовый
 Cs_Code    EQU $-gdt; Селектор дескриптора
 Gdt2 Gdt_Descriptor<cseg_leng,,,code_seg_access,>

;** Дескриптор, описывающий Cseg как сегмент данных ; с
пределом 0FFFEh. Он будет использоваться также ; в роли
стекового
 Cs_Data    EQU $-gdt; Селектор дескриптора
 Gdt3 Gdt_Descriptor<cseg_leng,,,data_seg_access,>

; Дескриптор, описывающий таблицу IDT
 Idt_Pointer Gdt_Descriptor<idt_leng-1,,,\
 data_seg_access>

; Дескриптор, описывающий таблицу IDT реального
; режима
 Idt_Real   Gdt_Descriptor<3FFh,,,data_seg_access>
```

```
; Дескриптор, описывающий сегмент видеопамяти
 Video_Desc EQU $-gdt; Селектор дескриптора
 GdtB800 Gdt_Descriptor<1000h,8000h,0bh,\
 data_seg_access>

Gdt_Leng    EQU $-gdt        ; Длина таблицы GDT

;Таблица дескрипторов прерываний IDT.
EVEN
Idt   label   word
ex0     Idt_Descriptor<offset ex0_proc,cs_code,0,10000111b,0>
ex1     Idt_Descriptor<offset ex1_proc,cs_code,0,10000111b,0>
ex2     Idt_Descriptor<offset ex2_proc,cs_code,0,10000110b,0>
ex3     Idt_Descriptor<offset ex3_proc,cs_code,0,10000111b,0>
ex4     Idt_Descriptor<offset ex4_proc,cs_code,0,10000111b,0>
ex5     Idt_Descriptor<offset ex5_proc,cs_code,0,10000111b,0>
ex6     Idt_Descriptor<offset ex6_proc,cs_code,0,10000111b,0>
ex7     Idt_Descriptor<offset ex7_proc,cs_code,0,10000111b,0>
ex8     Idt_Descriptor<offset ex8_proc,cs_code,0,10000111b,0>
ex9     Idt_Descriptor<offset ex9_proc,cs_code,0,10000111b,0>
ex10    Idt_Descriptor<offset ex10_proc,cs_code,0,10000111b,0>
ex11    Idt_Descriptor<offset ex11_proc,cs_code,0,10000111b,0>
ex12    Idt_Descriptor<offset ex12_proc,cs_code,0,10000111b,0>
ex13    Idt_Descriptor<offset ex13_proc,cs_code,0,10000111b,0>
ex14    Idt_Descriptor<offset ex14_proc,cs_code,0,10000111b,0>
ex15    Idt_Descriptor<offset ex15_proc,cs_code,0,10000111b,0>
ex16    Idt_Descriptor<offset ex16_proc,cs_code,0,10000111b,0>
        Idt_Descriptor 22 dup(<>)
Int39  Idt_Descriptor<offset int10_proc,cs_code,0,\
 10000110b,0>
 Idt_Leng   EQU $-Idt; Длина таблицы IDT

Mess            db  'Protected Mode$'
Len             dw  14d
Gate_Failure         db "Error open A20$"

Main:    FillDescr  cs,Gdt,Gdt1; Формирование
; 32-разрядного адреса из CS:GDT и запись его
; в дескриптор с номером Gdt_Desc
         FillDescr  cs,0,gdt2; Дескриптор Cs_Code
; указывает на CSEG как на кодовый сегмент.
```

```
        FillDescr  cs,0,gdt3; Дескриптор Cs_Data
; указывает на CSEG как на сегмент данных
      FillDescr  cs,Idt,Idt_Pointer; Дескриптор
; Idt_Pointer указывает на IDT.
        cli      ; Запрет прерываний
        mov      al,8fh; Запрет немаскируемых
        out      cmos_port,al; прерываний
        jmp      short $+2
        mov      al,5
        out cmos_port+1,al

        mov      ah,Enable_Bit20; Открываем
        call    Gate_A20     ; адресную линию A20
        or    al,al          ; Если произошла
        jz    A20_Opened; ошибка, то выдать
        mov   dx,offset Gate_Failure ; сообщение
        mov   ah,9             ; на экран,
        int   21h
        sti  ; разрешить прерывания
        int         20h; вернуться в DOS
A20_Opened:

        lea   di,Real_CS; Сохранение сегмента
        mov   word ptr cs:[di],cs ; кода для
                    ; перехода в реальный режим
        lgdt    Gdt1        ; Загрузка GDTR
        lidt    Idt_Pointer       ; Загрузка IDTR
        mov     eax,cr0; Переходим в защищенный
        or    eax,1         ; режим, устанавливая
        mov   cr0,eax       ; бит 0 в регистре CR0

        db 0EAh             ; Дальний переход
        dw offset Protect; с непосредственным
        dw Cs_Code        ; операндом
        ; Работа в защищенном режиме
Protect:   mov       ax,Cs_Data
        mov   ss,ax         ; Регистры DS, ES и SS
        mov   ds,ax         ; содержат  селектор
        mov   es,ax         ; сегмента Cs_Data
        call  My_Proc; Вызов рабочей процедуры
        cli
        mov   eax,cr0       ; Переходим в реальный
```

```
        and   eax,0FFFEh  ; режим, сбрасывая бит 0
        mov   cr0,eax      ; регистра CR0

        db 0EAh             ; Дальний переход с
        dw offset Real      ; непосредственным
Real_CS   dw ?              ; операндом

; Работа в реальном режиме.
Real:   lidt   Idt_Real   ; Загружаем регистр IDTR
                     ; для работы в реальном
                      ; режиме
        mov   dx,cs         ; Восстанавливаем
        mov   ds,dx         ; сегментные
        mov   ss,dx         ; регистры
        mov   ah,Disable_Bit20; Закрытие адресной
        call    Gate_A20          ; линии A20
        sti           ; Разрешение прерываний
        int         20h     ; Выход в DOS

ex0_proc:  iret             ; Обработчики особых
ex1_proc:  iret             ; ситуаций
ex2_proc:  iret             ; Здесь установлены
ex3_proc:  iret             ; заглушки вместо
ex4_proc:  iret             ; обработчиков
ex5_proc:  iret
ex6_proc:  iret
ex7_proc:  iret
ex8_proc:  iret
ex9_proc:  iret
ex10_proc: iret
ex11_proc: iret
ex12_proc: iret
ex13_proc: iret
ex14_proc: iret
ex15_proc: iret
ex16_proc: iret

;************************************************
;Управление прохождением сигнала A20
;ВХОД: (AH)=0DDH  установить A20 всегда равным нулю
;      (AH)=0DFh  открыть адресный разряд A20
```

```
;ВЫХОД: (AL)=0       8042 принял команду
;      (AH)=2       сбой
;************************************************
Gate_A20 PROC
         cli                  ; Запрет прерываний
         call  Empty_8042
         jnz   Gate_1
         mov   al,0d1h ; Выдаем команду  8042 для
         out Status_Port,al; записи в выходной порт
         call    Empty_8042
         jnz   Gate_1
         mov   al,ah     ; Записываем в порт A 8042
         out   Port_A,al           ; код команды
         call    Empty_8042
Gate_1:    ret
Gate_A20 ENDP
;************************************************
;Ждать пока буфер 8042 не опустеет
;Вход: нет
;Выход:(AL)=0     буфер пуст
;      (AL)=2     не пуст
;************************************************
Empty_8042  PROC
         push    cx
         xor   cx,cx     ; CX = 0 (256 повторений)
Empty_1:   in   al,Status_Port    ; Порт 8042
         and   al,00000010b      ; Бит 2 очищен ?
         loopnz  Empty_1
         pop         cx
         ret
Empty_8042  ENDP
;************************************************
; Формирование 32-разрядного адреса
; Вход: CX:DX - адрес в формате <сегмент: смещение>
; Выход: CX:DX - 32-разрядный линейный адрес
Form_32Bit_Address   PROC
         shl         edx,4

         add         edx,ecx
         mov         ecx,edx
         shr         ecx,16
         ret
```

```
Form_32Bit_Address   ENDP

;************************************************
;   Процедура вывода строки на экран, работает
; в качестве обработчика прерывания.
; Вход : DS:BX - адрес сообщения
;      DL - строка экрана
;      DH - колонка экрана
;************************************************
Int10_Proc   Proc  Near   ; Обработчик прерывания
         pusha               ; INT 39d
         xor   cx,cx         ; Очистка CX
         mov   cl,dh         ; CL = колонка
         sal   cl,1          ; CL = CL*2
         xor   dh,dh         ; DX = строка
       imul dx,160d; Умножаем на число байт в строке
       add dx,cx; Прибавляем смещение в строке
      ; Результат: DX = смещение в видеопамяти
         push    Video_Desc
         pop   es    ; ES = сегмент видеопамяти
         mov   di,dx; DI = смещение в этом сегменте
m:       mov   ax,[bx]; AL = очередной символ строки
         cmp   al,'$'     ; Конец строки ?
         jz    Ex       ; Да - выход
         mov   cx,es:[di] ; Получить атрибут в CH
         mov   ah,ch      ; AX = символ с атрибутом
         stosw      ; Записать символ в видеопамять
         inc bx     ; Перейти к следующему символу
         jmp         short   m
ex:      popa
         iret           ; Возврат из прерывания
Int10_Proc Endp

;************************************************
;Процедура выполняющая какие-либо действия
; в защищенном режиме
;************************************************
MY_PROC    PROC
         pusha
         push  es
         push  Video_Desc; В регистр ES заносим
         pop   es; селектор сегмента видеопамяти
```

```
        mov   dh,0fh  ; Очищаем экран
        call  Paint_Screen
        mov   ax,Cs_Data
        mov   ds,ax     ; DS - сегмент данных
        lea   bx,Mess     ; Адрес сообщения
        mov   dx,200Bh     ; Координаты вывода
        int   39d     ; Вывод строки на экран
        pop         es
        popa
        ret
MY_PROC ENDP
;************************************************
; Процедура очищает экран и устанавливает цвета
; в соответствии с заданным атрибутом.
; Вход : ES - селектор дескриптора текстового
; видеобуфера, DH - атрибут.
;************************************************
PAINT_SCREEN  PROC
        push    cx si di es
        mov   cx,80*25; Размер видеопамяти (слов)
        xor   si,si   ; SI и DI установим на
        xor   di,di         ; начало видеопамяти
Paint1:    lodsw; Увеличиваем смещение
                 ; в видеопамяти
        mov   ah,dh; Байт атрибута символа
        mov   al,20h; Код символа "ПРОБЕЛ"
        stosw; Записываем символ с атрибутом
                ; в видеопамять
        loop  Paint1; Повторить для каждого
                      ; символа на экране
        pop         es di si cx
        RET
PAINT_SCREEN ENDP
Cseg_Leng  Equ  $; Длина сегмента Cseg
Cseg    Ends
End     Start
```

ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ

Модифицировать программу Demo в соответствии с заданием, соответствующим порядковому номеру студента в списке группы.

1. Написать обработчик прерывания INT 27d, выполняющий вывод содержимого 32-разрядного регистра на экран в шестнадцатеричном виде и вывести с его помощью содержимое нескольких регистров друг под другом. Координаты вывода указываются при вызове данного прерывания.

2. Написать обработчик исключительной ситуации 0 (ошибка деления), выводящий на экран сообщение. Проверить работу программы для двух случаев: а) нормальная работа; б) при выполнении ошибочной команды деления.

3. Разделить сегменты кода и данных (так, чтобы их дескрипторы указывали не на одну и ту же область памяти). Для стекового сегмента организовать отдельный дескриптор.

4. Написать обработчик прерывания INT 29d, выдающий звуковой сигнал в защищенном режиме.

5. Написать обработчик особой ситуации 11 (сегмент не присутствует), выводящий соответствующее сообщение на экран. Для проверки его работы создать в GDT дескриптор не присутствующего сегмента и обратиться к нему.

6. Написать обработчик прерывания INT 18d, выводящий на экран байт прав доступа заданного дескриптора в двоичной форме. Селектор дескриптора задается в качестве параметра при вызове этой функции. Для получения байта прав доступа можно воспользоваться командой LAR.

7. Создать в таблице GDT дескриптор, описывающий таблицу GDT как сегмент данных. Написать процедуру, которая, используя этот дескриптор, будет выдавать базовый адрес любого сегмента, описанного в GDT. Селектор сегмента передается процедуре в качестве параметра в регистре AX. Процедура возвращает 32-разрядный линейный адрес в регистре EBX. Для проверки работы процедуры, после возврата в реальный режим вывести содержимое EBX на экран, используя стандартные функции DOS.

8. Написать процедуру, меняющую в защищенном режиме предел сегмента в дескрипторе GDT. Для этого нужно внести в GDT дескриптор, описывающий GDT как сегмент данных, и, в

дальнейшем, модифицировать содержимое GDT, пользуясь этим дескриптором. При вызове процедуры значение селектора и нового предела задается в регистрах. Для проверки работы считать новое значение предела командой LSL, и после выхода в реальный режим вывести его на экран, используя стандартные функции DOS.

9. Написать обработчик ошибки неверного кода операции, выводящий информацию на экран. Для проверки вызвать генерацию данной особой ситуации.

10. Написать обработчик ошибки переполнения, выводящий информацию на экран. Для проверки вызвать генерацию данной особой ситуации.

11. Написать обработчик ошибки превышения диапазона, выводящий информацию на экран. Для проверки вызвать генерацию данной особой ситуации.

12. Преобразовать программу DEMO из формата .COM в формат .EXE.

13. Написать обработчик прерывания 30d, который в защищенном режиме выводит размер сегмента, описываемого дескриптором GDT на экран. Селектор дескриптора передается в регистре DX.

14. Написать обработчик прерывания 40d, который в защищенном режиме выводит содержимое дескриптора GDT на экран. Селектор дескриптора передается в регистре CX.

15. Написать обработчик особой ситуации общей защиты, выводящий информацию из кода ошибки на экран. Для проверки вызвать генерацию данной особой ситуации.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить основные сведения по работе.

2. В соответствии с заданием модифицировать программу Demo.

3. Выполнить ввод, трансляцию, построение кода программы и получить результаты ее работы.

СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

1. Цель работы.
2. Текст задания и общая схема решения задачи.
3. Текст программы или подпрограммы.
4. Результаты выполнения программы.
5. Выводы по работе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое дескриптор сегмента, какие поля он имеет?
2. Назовите типы дескрипторных таблиц и различие между ними.
3. Каково назначение полей селектора сегмента?
4. Какие бывают типы системных дескрипторов и их назначение?
5. Опишите механизм передачи управления при возникновении особой ситуации или прерывания.
6. Поясните, как определяется максимальный объем виртуальной памяти в процессоре 80286 (1 Гбайт) и 80386 (64 Тбайт).

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Брэй Б. Микропроцессоры Intel: 8086/8088…80486, Pentium: пер. с англ. СПб: BHV-Петербург, 2005. 1328 с
2. Зубков С.В. Ассемблер для DOS, Windows и UNIX. М.: ДМК-Пресс, 2013. 638 с.
3. Таненбаум Э. Архитектура компьютера: 5-е изд. пер. с англ. СПб.: Питер, 2013. 848 с.
4. Юров В.И. Assembler: учебник для вузов. СПб.: Питер, 2011. 640 с.
5. Юров В.И. Assembler: практикум: учеб. Пособие для вузов. СПб.: Питер, 2007. 400 с.
6. Андреева А.А. и др. Программирование на языке ассемблера в операционной системе Windows: лаб. практикум. Чебоксары: Изд-во Чуваш. ун-та, 2006. 104 с.
7. Андреева А.А. и др. Программирование микропроцессоров семейства Intel 80x86: лаб. практикум. Чебоксары: Изд-во Чуваш. ун-та, 1996. 144 с.

ПРИЛОЖЕНИЯ

1. НЕКОТОРЫЕ КОМАНДЫ, СВЯЗАННЫЕ С ЗАЩИЩЕННЫМ РЕЖИМОМ

При описании команд использованы следующие обозначения:

- r8 : один из регистров AL, CL, DL, BL, AH, CH, DH, BH;
- r16: один из регистров AX, CX, DX, BX, SP, BP, SI, DI;
- r32: один из регистров EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI;
- imm8, imm16, imm32: непосредственная одно-, двух- или четырехбайтная величина соответственно;
- r/m8, r/m16, r/m32, r/m64: одно-, двух-, четырех- или восьмибайтный регистр или операнд в памяти;
- m8, m16, m32: байт, слово или двойное слово в памяти.

При описании особых ситуаций, в некоторых командах использовано выражение "стандартные". К стандартным особым ситуация здесь относится следующий набор:

- реальный режим: прерывание 13, если любая часть операнда находится вне пространства эффективных адресов от 0 до 0FFFFh.
- защищенный режим: особая ситуация 13 (общая защита), если результат находится в сегменте, запрещенном для записи, или если недопустимый эффективный адрес в сегментах CS, DS, ES, FS или GS; особая ситуация 12 (ошибка стека), если недопустимый адрес в сегменте SS; особая ситуация 14 (страничная ошибка) при страничной ошибке.
- виртуальный режим: то же, что и в реальном режиме; особая ситуация 12 при страничной ошибке.

1 **Команды LGDT** и **LIDT.**

LGDT - загрузить регистр глобальной дескрипторной таблицы;

LIDT - загрузить регистр дескрипторной таблицы прерываний;

Синтаксис: LGDT m48

LIDT m48

Действие: обе команды действуют совершенно одинаково и служат для загрузки регистров GDTR и IDTR из шестибайтного операнда в памяти. Если используется 16-битный операнд, то регистр загружается 16-битным пределом и 24-битной базой, старшие 8 бит не используются. Этот вариант является единственным для процессора 80286. В случае использования 32-битного операнда загружается 16-битный предел и 32-битная база.

Использование: эти команды используются обычно операционной системой; они могут применяться только на нулевом уровне привилегий.

Изменяемые флаги: нет.

Особые ситуации:

- реальный режим: прерывание 13, если часть операнда лежит вне предела адресов 0..0FFFFh; прерывание 6, если операнд - регистр;

- защищенный режим: особая ситуация 13, если текущий уровень привилегий не равен 0; 6 - если операнд - регистр; особая ситуация 13 при недопустимом эффективном адресе операнда в памяти в сегментах CS, DS, ES, GS или FS; особая ситуация 12 при недопустимом адресе в сегменте SS; особая ситуация 14 при страничной ошибке;

- виртуальный режим: то же, что и в реальном; особая ситуация 14 при страничной ошибке.

Пример:

```
LGDT  GlobalTable
```

2. **Команды SGDT и SIDT.**

SGDT - сохранить регистр глобальной дескрипторной таблицы;

SIDT - сохранить регистр дескрипторной таблицы прерываний;

Синтаксис: SGDT m48
SIDT m48

Действие: команды копируют содержимое соответствующего регистра дескрипторной таблицы в шесть байтов памяти, указанных операндом. Поле предела регистра помещается в самое младшее слово. Если атрибут размера операнда равен 16

бит, то следующим трем байтам присваивается значение поля базы, а четвертый байт не определен. Если размер 32 бита, то поле базы присваивается всем четырем байтам.

Изменяемые флаги: нет.

Особые ситуации: стандартные.

Примеры:

```
SIDT  [bx+4]
SGDT  [ecx*2]+100h
SGDT Qword Ptr Table
```

3. **Команды LLDT и LTR** - загрузка регистров LDTR и TR.

Синтаксис: LLDT r/m16
LTR r/m16

Действие: команда LLDT загружает регистр локальной дескрипторной таблицы значением двухбайтного операнда (в памяти или в регистре). Операнд должен обязательно содержать селектор таблицы GDT, причем элемент GDT должен быть дескриптором локальной дескрипторной таблицы. Команда LTR аналогичным образом загружает регистр задачи (TR) селектором сегмента TSS, причем загруженный TSS помечается как занятый.

Использование: команды не используются в прикладных программах, поскольку являются системными. Атрибут размера операнда не воздействует на эти команды.

Изменяемые флаги: нет.

Особые ситуации:

- реальный режим: прерывание 6, поскольку команды в реальном режиме не распознаются.

- защищенный режим: особая ситуация 13, если текущий уровень привилегий не равен 0, если операнд не является селектором GDT, если запись в GDT - не дескриптор локальной дескрипторной таблицы (для LLDT) или не дескриптор сегмента TSS (для LTR), при недопустимом эффективном адресе операнда в памяти в сегментах CS, DS, ES, GS или FS; особая ситуация 12 при недопустимом адресе в сегменте SS; особая ситуация 14 при страничной ошибке;

- виртуальный режим: то же, что и в реальном режиме.

Пример:

```
LLDT [si]
```

4. **Команды SLDT и STR** - сохранение регистров LDTR и TR.

Синтаксис: SLDT r/m16
STR r/m16

Действие: команда SLDT сохраняет регистр локальной дескрипторной таблицы (LDTR) в двухбайтном регистре или слове памяти. Команда STR аналогичным образом сохраняет регистр TR.

Использование: команды не используются в прикладных программах, поскольку являются системными.

Изменяемые флаги: нет.

Особые ситуации:

- реальный режим: прерывание 6, поскольку команда в реальном режиме не распознается.

- защищенный режим: особая ситуация 13, если результат находится в сегменте, запрещенном для записи; при недопустимом эффективном адресе операнда в памяти в сегментах CS, DS, ES, GS или FS; особая ситуация 12 при недопустимом адресе в сегменте SS; особая ситуация 14 при страничной ошибке;

- виртуальный режим: то же, что и в реальном режиме.

Примеры:

```
SLDT CX
STR [bx]
```

5. **Команда LAR** - загрузить байт прав доступа дескриптора.

Синтаксис: LAR r16, r/m16
LAR r32, r/m32

Действие: если дескриптор доступен для чтения, то команда принимает второе двойное слово дескриптора, маскирует его значением 00FxFF00H, записывает результат в заданный 32-разрядный регистр назначения и устанавливает флаг ZF (x означает, что соответствующие четыре бита записанного значения не определены).

Если указан 16-битный размер операнда, то 16-битный регистр назначения загружается младшими 16 битами старшего двойного слова дескриптора, маскированными FF00, и устанавливается флаг ZF.

Изменяемые флаги: ZF.

Особые ситуации:

- реальный режим: прерывание 6, поскольку команда в реальном режиме не распознается;

- защищенный режим: особая ситуация 13, если результат находится в сегменте, запрещенном для записи; при недопустимом эффективном адресе операнда в памяти в сегментах CS, DS, ES, GS или FS; особая ситуация 12 при недопустимом адресе в сегменте SS; особая ситуация 14 при страничной ошибке;

- виртуальный режим: то же, что и в реальном режиме.

Пример:

```
LAR EAX,ECX
```

6. **Команда LSL** - загрузить границу сегмента.

Синтаксис: LSL r16, r/m16
LSL r32, r/m32

Действие: команда загружает границу сегмента из дескриптора в операнд-приемник (первый операнд). Если размер операнда равен 16 бит, то старшие 16 бит двойного слова границы теряются.

Команда действительна только для дескрипторов, описывающих сегменты (данных, кода, состояния задачи и таблиц дескрипторов); дескрипторы шлюзов таким способом недоступны. При успешном выполнении команды устанавливается флаг ZF.

Изменяемые флаги: ZF.

Особые ситуации:

- реальный режим: прерывание 6, поскольку команда в реальном режиме не распознается;

- защищенный режим: особая ситуация 13, сли результат находится в сегменте, запрещенном для записи; при недопустимом эффективном адресе операнда в памяти в сегментах CS, DS, ES, GS или FS; особая ситуация 12 при недопустимом адресе в сегменте SS; особая ситуация 14 при страничной ошибке;

- виртуальный режим: то же, что и в реальном режиме.

Пример:

```
LAR AX,BX
```

7. **Команда ARPL** - коррекция поля RPL селектора.

Синтаксис: ARPL r/m16, r16

Действие: первый операнд команды - 16-битная переменная в памяти или двухбайтный регистр, который содержит значение селектора. Второй операнд - всегда 16-битный регистр. Если поле RPL (запрашиваемый уровень привилегий, младшие два бита) первого операнда меньше, чем поле RPL второго операнда, то устанавливается флаг ZF, и поле RPL первого операнда увеличивается до значения поля RPL второго операнда. В противном случае сбрасывается флаг ZF, и первый операнд не изменяется.

Изменяемые флаги: ZF.

Особые ситуации:

- реальный режим: прерывание 6, поскольку команда в реальном режиме не распознается.

- защищенный режим: особая ситуация 13, если результат находится в сегменте, запрещенном для записи; при недопустимом эффективном адресе операнда в памяти в сегментах CS, DS, ES, GS или FS; особая ситуация 12 при недопустимом адресе в сегменте SS; особая ситуация 14 при страничной ошибке;

- виртуальный режим: то же, что и в реальном режиме.

Пример:

```
ARPL  AX, BX
```

8. **Команды VERR и VERW** – проверка сегмента на возможность чтения или записи.

VERR (Verify Read) - проверить сегмент на чтение;

VERW (Verify Write) - проверить сегмент на запись.

Синтаксис: VERR r/m16
VERW r/m16

Действие: двухбайтный операнд этих командсодержит значение селектора проверяемого сегмента. Команды проверяют, доступен ли сегмент для чтения (VERR) или записи (VERW). Если сегмент доступен, то устанавливается флаг ZF, в

противном случае флаг ZF очищается. Для того чтобы флаг ZF установился необходимо выполнение следующих условий:

- селектор должен попадать в пределы таблиц GDT или LDT;

- селектор должен индексировать дескриптор сегмента кода или данных, а не системный дескриптор.

Использование: значение проверяемого селектора не приводит к возникновению особых ситуаций защиты, предоставляя возможность программному обеспечению выявлять проблемы доступа к сегменту.

Изменяемые флаги: ZF.

Особые ситуации:

- реальный режим: всегда вырабатывается прерывание 6, поскольку команды не распознаются в реальном режиме;

- защищенный режим: особая ситуация 13 при недопустимом эффективном адресе операнда в памяти в сегментах CS, DS, FS, GS или FS; особая ситуация 12 при недопустимом адресе в сегменте SS; особая ситуация 14 при страничной ошибке;

- виртуальный режим: то же, что и в реальном; особая ситуация 14 при страничной ошибке.

Примеры:

```
MOV AX, Selector
VERR AX
```

2. ПРЕРЫВАНИЯ И ОСОБЫЕ СИТУАЦИИ В МИКРОПРОЦЕССОРАХ x86

Таблица П1

Номер	Название	Тип	Код ошибки	Появилось в процессоре
0	Ошибка деления	Ошибка	Нет	8086
1	Отладка		Нет	8086
2	Немаскируемое прерывание	Прерывание	Нет	8086
3	Точка останова	Ловушка	Нет	8086
4	Переполнение	Ловушка	Нет	8086

5	Проверка границ	Ошибка	Нет	80186
6	Неверный код операции	Ошибка	Нет	80286
7	Сопроцессор недоступен	Ошибка	Нет	80286
8	Двойная ошибка	Сбой	Есть (0)	80286
9	**Резерв**			
10	Неверный TSS	Ошибка	Есть	80286
11	Сегмент не присутствует	Ошибка	Есть	80286
12	Ошибка при работе со стеком	Ошибка	Есть	80286
13	Общая защита	Ошибка	Есть	80286
14	Страничная ошибка	Ошибка	Есть	80386
15	Зарезервировано			
16	Ошибка сопроцессора	Ошибка	Нет	80286
17	Контроль выравнивания	Ошибка	Есть (всегда0)	80486
18	Контроль машины			PENTIUM

Прерывания 10-12, 14 определены только для защищенного режима.

Прерывание 0 - ошибка деления. Возникает в командах DIV или IDIV, если частное не помещается в отведенный регистр.

Прерывание 1 - отладочные особые ситуации. Является ли данная особая ситуация сбоем или ловушкой, зависит от условия, как показано ниже:

- сбой в контрольной точке адреса команды;
- ловушка в контрольной точке адреса данных;
- ловушка шага выполнения;
- ловушка контрольной точки переключения задачи.

Прерывание 2 - немаскируемое прерывание. Вызывается сигналом на входе NMI процессора.

Прерывание 3 - точка останова. Генерируется командой INT3. Сохраненное содержимое регистров CS и EIP указывает на байт, следующий за контрольной точкой.

Прерывание 4 - переполнение. Происходит, когда процессор выполняет команду INTO при установленном флаге OF.

Прерывание 5 - проверка границ. Генерируется, когда при выполнении команды BOUND обнаруживается, что операнд превышает заданные границы.

Прерывание 6 **- неверный код операции**. Генерируется, когда исполнительный модуль обнаруживает неверный код операции. Эта особая ситуация происходит также при неверном типе операнда для данного кода операции. Примером может служить команда межсегментного перехода JMP, использующая регистровый операнд. Третье условие генерации данной особой ситуации состоит в использовании префикса LOCK с командой, для которой захват шины невозможен.

Прерывание 7 – сопроцессор недоступен. Генерируется в одном из следующих случаев:

1. Процессор выполняет команду ESC при установленном бите EM регистра CR0.

2. Процессор выполняет команду WAIT или ESC при установленном бите TS регистра CR0.

Таким образом, прерывание 7 происходит, когда программист хочет, чтобы команда ESC обрабатывалась программно (EM=1), либо когда встречена команда WAIT или ESC, а контекст сопроцессора отличен от контекста для текущей задачи.

Для процессоров 80286 и 80386 бит MP регистра CR0 используется с битом TS для определения того, должна ли команда WAIT генерировать особую ситуацию. Для программ, выполняемых на процессоре 80486, бит MP должен быть установлен всегда.

Прерывание 8 - двойная ошибка.

Обычно когда процессор обнаруживает особую ситуацию при попытке вызвать обработчик особой ситуации, происшедшей ранее, эти два прерывания могут обрабатываться последо-

вательно. Однако, если процессор не в состоянии этого сделать, он генерирует особую ситуацию двойной ошибки. Процессор всегда помещает код ошибки в стек обработчика двойной ошибки, однако этот код всегда равен 0. Рестарт вызвавшей сбой команды невозможен. Если при попытке вызвать обработчик двойной ошибки произойдет еще одна особая ситуация, то процессор входит в режим останова системы. Никакие дальнейшие команды не выполняются до приема немаскируемого прерывания или сигнала сброса системы RESET. Процессор генерирует специальный цикл шины, указывающий на вхождение в режим останова.

Прерывание 10 - неверный TSS. Генерируется при попытке переключения задачи на сегмент с неверным TSS. TSS является неверным в случаях, описанных в табл. П2. Код ошибки помещается в стек обработчика особой ситуации, что помогает идентифицировать причину сбоя.

Чтобы гарантировать доступность TSS для обработки особой ситуации, обработчик прерывания типа "неверный TSS" должен вызываться как задача посредством шлюза задачи.

Прерывание 11 - сегмент не присутствует. Генерируется, когда процессор обнаруживает, что бит присутствия в дескрипторе очищен. Данная ошибка позволяет выполнение рестарта. Обработчик особой ситуации загружает сегмент и прерванная программа продолжает выполнение.

Прерывание 12 - особая ситуация при работе со стеком. Ошибка в стеке генерируется в следующих случаях:

1. В результате нарушения границы операцией, ссылающейся к регистру SS. Сюда входят стековые команды, такие как POP, PUSH, ENTER и LEAVE, а также ссылки к памяти, неявно использующие стек (например, MOV DX,[BP]).

2. При попытке загрузить регистр SS дескриптором, отмеченным признаком "сегмент не присутствует".

Таблица П2

Условия неверного TSS

Индекс кода ошибки	Описание
Сегмент TSS	Граница сегмента TSS меньше 67H
Сегмент LDT	Неверная LDT или LDT не присутствует

Сегмент стека	Селектор сегмента стека превышает границу таблицы дескрипторов
Сегмент стека	Сегмент стека не доступен для записи
Сегмент стека	DPL сегмента стека не совместим с CPL
Сегмент стека	RPL селектора сегмента стека не совместим с CPL
Сегмент кода	Селектор сегмента кода превышает границу таблицы дескрипторов
Сегмент кода	Сегмент кода не является выполняемым
Сегмент кода	DPL неподчиненного сегмента кода не равен CPL
Сегмент кода	DPL подчиненного сегмента кода больше CPL
Сегмент данных	Селектор сегмента данных превышает границу таблицы дескрипторов
Сегмент данных	Сегмент данных не доступен для чтения

Прерывание 13 - общая защита. Генерируется для всех нарушений защиты, не вызывающих прочих особых ситуаций. Такими нарушениями могут быть:

- превышение границы сегмента при использовании сегментов CS, DS, ES, FS или GS;
- превышение границы сегмента при ссылке к таблице дескрипторов;
- передача управления сегменту, не являющемуся кодовым;
- попытка записи в сегмент данных, доступный только для чтения, или в кодовый сегмент;
- чтение из кодового сегмента, доступного только для выполнения;
- загрузка регистра SS селектором сегмента, доступного только для чтения;
- загрузка регистра SS, DS, ES, FS или GS селектором системного сегмента;
- загрузка регистра DS, ES, FS или GS селектором кодового сегмента, доступного только для выполнения;
- загрузка регистра SS селектором выполняемого сегмента;
- доступ к памяти при помощи регистров DS, ES, FS или GS, когда в них содержится пустой селектор;
- переключение на занятую задачу;
- нарушение правил привилегий;
- превышение длины команды 15 байт;

- загрузка регистра CR0 при установленном бите PG (трансляция страниц разрешена) и очищенном бите PE (защита запрещена).

В реальном режиме это прерывание генерируется, если команда или операнд пересекают границу сегмента (0FFFFh) или длина команды превышает 15 байт.

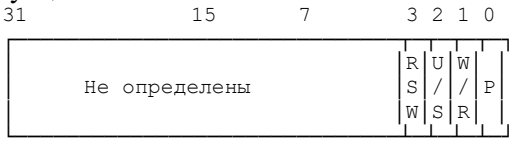
Прерывание 14 – страничная ошибка. Происходит, когда подкачка страниц разрешена (бит PG регистра CR0 установлен) и процессор обнаруживает одно из следующих условий во время трансляции линейного адреса в физический:

- элемент каталога страниц или таблицы страниц, необходимый для трансляции адреса, имеет очищенный бит Присутствия, т.е. нужная таблица страниц или страница, содержащая операнд, не присутствует в физической памяти;

- процедура не имеет достаточного уровня привилегий для доступа к указанной странице.

Обработчик страничной ошибки получает информацию об ее причине из двух источников:

1. Код ошибки в стеке. Код ошибки для страничной особой ситуации имеет формат, отличный от формата для других особых ситуаций:



Назначение битов 0..3 кода ошибки приведено в табл. П3.

2. Содержимое регистра CR2. Процессор загружает в регистр CR2 32-разрядный линейный адрес, вызвавший ошибку.

Таблица 3

Назначение битов 0..3 кода страничной ошибки

Поле	Значение	Описание
RSW	1	Сбой произошел из-за обнаружения единицы в одном из элементов таблицы или каталога страниц (ТОЛЬКО В ПРОЦЕССОРЕ PENTIUM)
	0	В остальных случаях
U/S	0	Сбой, произошел при работе процессора в в режиме супервизора

	1	Сбой, произошел при работе процессора в в режиме пользователя
W/R	0	Ошибка произошла при чтении
	1	Ошибка произошла при записи
P	0	Сбой был вызван отсутствием страницы
	1	Сбой был вызван нарушением защиты на уровне страниц

Прерывание 16 - ошибка сопроцессора. Эта особая ситуация сигнализирует об ошибке, возникшей во время выполнения команды сопроцессором. Прерывание 16 может произойти только если бит NE регистра CR0 установлен.

Прерывание 17 - контроль выравнивания. Генерируется при попытке доступа к не выравненным операндам. Для разрешения контроля выравнивания должны выполняться следующие условия:

- бит AM регистра CR0 должен быть установлен;
- флаг AC должен быть установлен;
- CPL должен быть равен 3 (уровень пользователя).

Требования к выравниванию по типам данных приведены в табл. П4.

Таблица 4

Требования к выравниванию данных

Тип данных	Адрес должен быть кратен
Слово	2
Двойное слово	4
Короткое вещественное (32 бита)	4
Длинное вещественное (64 бита)	8
Временное вещественное (80 бит)	8
Селектор	2
48-разрядный сегментированный указатель	4
32-разрядный плоский указатель	4
32-разрядный сегментированный указатель	2
48-разрядный псевдодескриптор	4
Область хранения FSTENV/FLDENV	4 или 2, в зависимости от размера операнда
Область хранения FSAVE/FRSTOR	4 или 2, в зависимости от размера операнда
Битовая строка	4

Нарушения контроля выравнивания генерируются только в режиме пользователя (уровень привилегий 3). Обращения к памяти, которые по умолчанию относятся к уровню привилегий 0, например загрузка дескриптора сегмента, не формируют нарушений контроля выравнивания.

Аппаратные прерывания и особые случаи распознаются на границах команд. При этом процессор производит следующую последовательность проверок:

1. Проверка особого случая отладки (по флажку TF или контрольной точке по данным, установленной в регистрах отладки). Формирование этого особого случая, если удовлетворяются условия его возникновения.

2. Проверка внешнего сигнала на входе немаскируемого прерывания NMI и при его активном состоянии формирование прерывания с номером 2.

3. Проверка внешнего сигнала на входе маскируемого прерывания INTR и при его активном состоянии (и флажке IF = 1) выполнение двух циклов шины подтверждения прерывания, ввод номера прерывания и обработка этого прерывания.

4. Проверка особого случая отладки (контрольная точка по команде, установленная в регистрах отладки).

5. Проверка нарушений доступа к сегменту, которые могут возникнуть при выборке следующей команды, и при необходимости формирование особого случая 11 или 13.

6. Проверка страничного нарушения, которое может появиться при выборке следующей команды. При обнаружении страничного нарушения формирование особого случая 14.

7. Проверка нарушений, которые могут появиться при декодировании следующей команды (недействительный код операции, превышение длины команды или нарушение защиты по привилегиям).

8. Проверка каждого обращения к памяти, инициируемого командой: при наличии нарушения сегмента, препятствующего передаче операнда, формирование особого случая 11, 12 или 13; при наличии страничного нарушения - особого случая 14.