**Министерство образования Республики Беларусь**

**Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»**

**КАФЕДРА ИНФОРМАТИКИ**

**Лабораторный практикум**

**по дисциплине курса «Архитектура вычислительных систем»**

**Часть II**

**Для студентов специальности**

**«Информатика»**

**МИНСК 2013**

# **ЦИКЛ 3. ЗАЩИЩЕННЫЙ РЕЖИМ 32-РАЗРЯДНЫХ ПРОЦЕССОРОВ**.

## 3.1. ЦЕЛЬ ЦИКЛА

Изучить особенности защищенного режима процессора. Получить практические навыки по программированию переключения процессора из реального в защищенный режим и обратно (8 часов)

## 3.2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

### *3.2.1. Обзор режимов 32-разрядных микропроцессоров (IA-32)*

32-разрядные процессоры Intel 80386, 80486 и Pentium с точки зрения рассматриваемых в данном разделе вопросов имеют аналогичные средства, поэтому для краткости в тексте используется термин "процессор i386", хотя вся информация этого раздела в равной степени относится к трем моделям процессоров фирмы Intel.

Процессор i386 имеет два режима работы - реальный (real mode, R-Mode) и защищенный (protected mode, P-Mode).

**Реальный режим.** Реальный режим - это режим, в котором процессор работает как быстрый процессор 8086, но позволяет пользоваться большинством своих технологий (ММХ / SSE / SSE2, 32-разрядные регистры общего назначения, регистры управления и отладки и пр.). После аппаратного сброса процессор переходит в этот режим и начинает выполнять программную инициализацию из BIOS-a. Реальный режим в современных процессорах предназначен для запуска компьютера и подразумевается, что операционная система будет работать в защищённом режиме (поэтому оптимизация по производительности для процессоров IA-32 производится для защищённого режима).

В реальном режиме не доступны основные достоинства процессора - виртуальная память, мультизадачность, уровни привилегий, работа с кэшами, буферами TLB, буфером ветвлений и некоторыми другими технологиями, обеспечивающими высокую производительность.

**Защищённый режим**. Как утверждает Intel, это "родной" (native) режим 32-разрядного процессора. В защищённый режим процессор надо переводить специальными операциями над системными регистрами и войти в этот режим процессор может только из реального режима. При работе в защищённом режиме процессор контролирует практически все действия программ и позволяет разделить операционную систему, драйвера и прикладные программы разными уровнями привилегий. Благодаря этому ОС может ограничить области памяти, выделяемой программам, запретить или разрешить ввод/вывод по любым адресам, управлять прерываниями и многое другое. При попытке программы выйти за допустимый диапазон адресов памяти, выделенной ей, либо при обращении к "запрещённым" для неё портам процессор будет генерировать исключения - специальный тип прерываний. Грамотно оперируя исключениями, операционная система может контролировать действия программ, организовать систему виртуальной памяти, мультизадачность и другие программные технологии.

В защищённом режиме максимально доступны все ресурсы процессора. Например, в R-Mode максимальный диапазон адресов памяти ограничен одним мегабайтом, а в защищённом режиме он расширен до 4 Гб для процессоров 386 и 486 и 64 Гб для Pentium-ов.

### *3.2.2. Адресация памяти в защищенном режиме.*

Физическое адресное пространство процессора i386 составляет 4 Гбайта, что определяется 32-разрядной шиной адреса. Физическая память является линейной с адресами от 00000000 до FFFFFFFF в шестнадцатеричном представлении. Процессор i80386 в защищённом режиме использует трёхступенчатую схему преобразования адреса.

Программы используют *логический адрес (виртуальный адрс),* состоящий из селектора и смещения. Это первая ступень в схеме преобразования ад­ресов. Смещение хранится в соответствующем поле команды, а номер сегмента - в одном из шести сегментных регистров процессора (CS, SS, DS, ES, FS или GS), каждый из которых является 16-битным. Компонента смещения является 32-азрядной, т. к. до­пустимый размер сегмента значительно превышает 64 Кбайта. Селектор – это индекс, с помощью которого процессор извлекает из специальной таблицы базовый адрес сегмента. В реальном режиме мы имеем дело с сегментным адресом и смещением, а в защищенном – с селектором и смещением.

Получение из логического адреса 32-разрядного *линейного адреса* с помощью механизма сегментации является второй ступенью в схеме преобразования ад­ресов.

Далее с помощью страничного механизма линейный адрес преобразуеться в 32-разрядный физический адрес. Это третья ступень в схеме преобразования адресов. Страничный механизм включается установкой специального бита PG в регистре CR0 при помощи привилегированной команды.При отключенном страничном механизме линейный адрес является физическим адресом сегмента.

Сначала рассмотрим механизм преобразования логического адреса в линейный при отключенном механизме управления страницами.

Рассмотрим структуру селектора. На самом деле не все 16 бит селектора используются для индексации по таблице базовых адресов. Формат селектора адреса приведен на рис.4.

|  |  |  |  |
| --- | --- | --- | --- |
| 15 | 3 | 2 | 1-0 |
| Индекс | | TI | RPL |
| Рис.4. Полный формат селектора адреса | | | |

В качестве индекса выступают старшие 13 бит. Два младших бита используються системой защиты памяти. На рисунке они обозначены как RPL (Requested Privilege Level). Это поле является запрошенным программой уровнем привелегий и будет обсужден позже. Поле TI (Table Indicator) состоит из одного бита. Если этот бит равен нулю, для преобразования адреса используется так называемая глобальная дескрипторная таблица GDT (Global Descriptor Table), в противном случае – локальная дескрипторная таблица LDT (Local Descriptor Table). Глобальная дескрипторная таблица предназначена для описания сегментов операционной системы и сегментов межзадачного взаимодействия, то есть сегментов, которые в принципе могут использоваться всеми процессами, а локальная дескрипторная таблица - для сегментов отдельных задач. Таблица GDT одна, а таблиц LDT должно быть столько, сколько в системе выполняется задач, но не более чем 8K (213). При этом активной в каждый момент времени может быть только одна из таблиц LDT. Разрядность поля индекса определяет максимальное число глобальных и локальных сегментов задачи - по 8K (213) сегментов каждого типа, всего 16 K. С учетом максимального размера сегмента - 4 Гбайта - каждая задача при чисто сегментной организации виртуальной памяти работает в виртуальном адресном пространстве в 64 Тбайта.

Процесс преобразования логического адреса в линейный приведено на рис.5.

32

GDT/LDT

Логический адрес

Селектор

Смещение

136

Линейный адрес

32

322

Рис. 5. Преобразование логического адреса в линейный

Рассмотрим сначала использование таблицы GDT. Значение из поля индекса селектора используется в качестве индекса в табли­це GDT для выборки 32-разрядного базового адреса. Этот базовый адрес складывается со второй компонентой логического адреса - смещением. В результате получается 32-разрядный линейный адрес.

В дескрипторных таблицах хранятся так называемые дескрипторы сегментов. Дескриптор представляет собой 8 байтную структуру, которая содержит базовый адрес описываемого сегмента, предел сегмента и права доступа к сегменту. Формат дескриптора приведен на рис. 6.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |  |
| Предел 0-15 | | | | | | | | | | | | | | | |  |
|  | | | | | | | | | | | | | | | | Мл. |
| 31 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 16 | слово |
| База 0-15 | | | | | | | | | | | | | | | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 |  | 9 | 8 | 7 |  |  |  |  | |  | |  | 0 |  |
| P | DPL | | S | Type | | | A | База 23-16 | | | | | | | | | |  |
|  | | | | | | | | | | | | | | | | | | Ст. |
| 31 |  |  |  |  |  |  | 24 | 23 | 22 | 21 | 20 | | 19 | |  |  | 16 | Слово |
| База 24-31 | | | | | | | | G | X | 0 | AV | | Предел 16-19 | | | | |  |
| Рис. 6. Дескрипторы для процессора i80386 | | | | | | | | | | | | | | | | | | |

Рассмотрим элементы дескриптора.

Адрес сегмента - также называется базовым адресом, - 32-разрядный адрес области памяти, с которой начинается сегмент.

Предел сегмента - предельное значение смещения в сегменте; также можно рассматривать предел как размер сегмента минус один элемент размера - байт или страницу, смотря в чём измеряется сегмент.

Поля А, Тип, S, DPL и Р образуют так называемое поле доступа. Биты А и Р предназначены для организации виртуальной памяти. Бит Р называеться битом присутствия сегмента в памяти. Для тех сегментов, которые находятся в физической памяти, этот бит должен быть установлен в 1. Если программе понадобится память, то она сохранит содержимое какого-либо сегмента на диск и сбросит бит P. Если любая программа в дальнейшем обратится к этому сегменту, то процессор сгенерирует исключение неприсутствующего сегмента и запустит обработчик этой ситуации, который должен будет подгрузить содержимое сегмента с диска и установить бит P. После этого управление снова передаётся команде, обратившейся к этому сегменту (производится повторное выполнение команды, вызвавшей сбой) и работа программы будет продолжена. Бит P устанавливается и сбрасывается программами, сам процессор его только считывает. Бит A (Acessed) - бит доступа в сегмент. Этот бит показывает, был ли произведен доступ к сегменту, описываемому этим дескриптором, или нет. Если процессор обращался к сегменту для чтения или записи данных или для выполнения кода, размещённых в нём, то бит A будет установлен (равен 1), иначе - сброшен (0). С помощью бита A операционная система может определить, использовался ли за последнее время этот сегмент или нет и предпринять какие-либо действия. Бит A процессором только устанавливается, сбрасывать его должна операционная система. При создании нового дескриптора подразумевается, что бит A будет равен 0 (т.е. обращений к этому сегменту ещё не было).

Бит **S** (System) - определяет системный объект. Если этот бит установлен, то дескриптор определяет сегмент кода или данных, а если сброшен, то системный объект (например, сегмент состояния задачи, локальную дескрипторную таблицу, шлюз).

Поле DPL (Descriptor Privilege Level) - Уровень привилегий, который имеет объект, описываемый данным дескриптором. Это двухбитовое поле, в него при создании дескриптора записывают значения от 0 до 3, определяющее уровень привилегий.

Поле Type - трёхбитовое поле, это поле интерпретируется по-разному, в зависимости от типа сегмента (см. Рис.7.).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
| P | DPL | | 1 | 1 | C | R | A | Поле доступа сегмента кода |
|  | | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
| P | DPL | | 1 | 0 | D | W | A | Поле доступа сегмента даных |
|  | | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
| P | DPL | | 0 | Type | | | | Поле доступа системного сегмента |
| Рис.7. Формат поля доступа дескриптора | | | | | | | | |

Для несистемных сегментов старший бит поля Type (3-й бит в байте прав доступа) имеет следующее значение: если он равен 1, то это сегмент кода, иначе это сегмент данных.

Поле доступа сегмента кода имеет битовые поля C и R. Поле С указывет, согласованный это сегмент (бит = 1) или нет (бит = 0). Этот бит будет рассмотрен позже. Поле R указывает, разрешено чтение сегмента кода (бит = 1) или - только выполнение (бит = 1).

Поле доступа сегмента кода имеет битовые поля D и W. Поле W называется битом разрешения записи в сегмент. Если этот бит установлен в 1, то наряду с чтением возможна и запись в данный сегмент. Поле D задает направление расширения сегмента. Обычный сегмент данных расшираеться в область старших адресов (расширение вверх). Если же в сегменте расположен стек, расширение происходит в обратном направлении – в область младших адресов (расширение вниз). Для сегментов, в которых организуются стеки, необходимо устанавливать поле D равным 1.

Бит G (Granularity) - Гранулярность сегмента, т.е. единицы измерения его размера. Если бит G=0, то сегмент имеет байтную гранулярность, иначе - страничную (одна страница - это 4Кб).

Бит X указывает разрядность выполняемых команд. Если этот бит установлен в 1, используются 32-разрядные команды, если сброшен в 0 – 16-разрядные.

Бит AVL используеться системным программным обеспечением.

Расположение дескрипторных таблиц определяется регистрами процессора GDTR, LDTR. Регистры GDTR - 6-байтный, он содержат 32 бита линейного базового адреса дескрипторной таблицы и 16 бит предела таблицы. Формат регистра GDTR приведен на рис. 8.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 47 | 16 | 15 | 0 |  |
| Адрес GDT | | Предел | | GDTR |
| Рис.8. Формат регистра GDTR. | | | | |

Перед переходом в защищенный режим программа должна создать в оперативной памяти таблицу GDT и загрузить регистр GDTR при помощи команды LGDTR src, где src указывает на область памяти с адресом и пределом таблицы GDT.

В случае, когда селектор задачи указывает на таблицу LDT, виртуальный адрес преобразуется в физический аналогичным образом, но для доступа к самой таблице LDT добавляется еще один этап, так как в процессоре регистр LDTR указывает на размещение таблицы LDT не прямо, а косвенно. Сам регистр LDTR имеет размер 16 бит и содержит селектор дескриптора таблицы GDT, который описывает расположение этой таблицы в физической памяти. Поэтому при доступе к элементу физической памяти через таблицу LDT происходит двукратное преобразование виртуального адреса в физический, причем оба раза по описанной выше схеме. Сначала по значению селектора LDTR определяется физический адрес дескриптора из таблицы GDT, описывающего начало расположения таблицы LDT в физической памяти, а затем с помощью селектора задачи вычисляется смещение в таблице LDT и определяется физический адрес нужного дескриптора. Далее процесс аналогичен преобразованию виртуального адреса с помощью таблицы GDT.

Следует заметить, что нулевой дескриптор, т.е. определённый в самом начале GDT, процессор не использует, обращение к такому дескриптору могло бы быть, когда поле Index селектора равно 0. Если всё же в программе встречается обращение к нулевому дескриптору, то процессор генерирует исключение и не позволит доступ к такому дескриптору. Первый дескриптор GDT описывает саму таблицу GDT.

Рассмотрим теперь страничный механизм. Преобразование линейного адреса в физический иллюстрируется рис. 9. Процесс вычисления адреса страницы часто называют трансляцией страниц. Старшие 10 бит линейного адреса используются как индекс в таблице, называе­мой каталогом таблиц страниц. Расположение каталога таблиц страниц в физиче­ской памяти определяется содержимым системного регистра процессора CR3.

Каталог таблиц страниц содержит дескрипторы таблиц страниц, определяющие физический адрес таблиц страниц. В каталоге всего может быть 1024 дескрипто­ра. Самих же каталогов может быть сколько угодно, но в каждый момент време­ни используется только один - тот, на который указывает регистр CR3.

Следующие 10 бит линейного адреса предназначены для индексации таблицы страниц, выбранной с помощью старших 10 бит адреса. Таблица страниц содер­жит 1024 дескриптора, определяющих физические адреса страниц памяти. Раз­мер одной страницы составляет 4 Кбайта, т. е. 4096 байт.

Младшие 12 бит линейного адреса указывают смещение к адресуемому байту внутри страницы.

Таблица

страниц

32

122

Каталог

таблиц страниц

Линейный адрес

3222

Таблица

Смещение

Страница

10

1000

CR3

Физический адрес

Рис.9. Преобразование линейного адреса в физический

### *3.2.3. Защита по привилегиям.*

Основой защищённого режима являются уровни привилегий (кольца защиты). Уровень привилегий - это степень использования ресурсов процессора. Всего таких уровней четыре и они имеют номера от 0 до 3. Уровень номер 0 - самый привелигерованный. Когда программа работает на этом уровне привилегий, ей "можно всё". Обычно такими привилегиями обладает ядро операционной системы. Уровень 1 - менее привелигерованный и запреты, установленные на уровне 0 действуют для уровня 1. Уровень 2 - ещё менее привелигерованный, а 3-й - имеет самый низкий приоритет. Несложные системы могут могут использовать не все кольца, а только некоторые или даже одно. Например, в схеме “супервизор-пользователь” все программы операционной системы расположены в кольце 0, а пользовательские программы – в кольце 3.

Для описания механизма защиты пользуются следующими понятиями:

*Уровень привилегий дескриптора (Descriptor Privilege Level: DPL*) - уровень привилегий, на который помещен описываемый дескриптором объект. Поле DPL хранится в байте прав доступа дескриптора.

*Текущий уровень привилегий (Current Privilege Level: CPL*) - уровень привилегий выполняемого сегмента кода. Это значение соответствует DPL сегмента кода (кроме подчиняемых сегментов кода). Значение CPL хранится в поле RPL селектора сегмента кода, который помещен в регистр CS.

*Запрашиваемый уровень привилегий (Requested Privilege Level: RPL) -* используется для временного понижения своего уровня привилегий при обращении к памяти. RPL заносится в младшие биты селектора.

*Уровень привилегий ввода-вывода (Input/Output Privilege Level: IOPL)* - указывает какой уровень привилегирован для работы с портами ввода-вывода. Это значение хранится в регистре EFLAGS и может быть различным для разных задач.

В защищеном режиме реализованы: защита от выполнения привилегированных команд, защита доступа к данным и защита сегментов кода. Рассмотрим их.

В процессоре есть команды, которые могут кардинально изменить состояние всей системы. Такие команды выполняются только на нулевом уровне привилегий, а на всех других уровнях вызывают нарушение общей защиты (исключение №13). К этим командам относятся:

* **HLT** - останов процессора;
* **CLTS** - сброс флажка Task Switched (исп. при управлении мультизадачностью);
* **LIDT, LGDT, LLDT** - загрузка регистров дескрипторных таблиц;
* **LTR** - загрузка регистра задачи;
* **LMSW** - загрузка младшего слова регистра CR0;
* **MOV CRx,reg32** - работа с управляющими регистрами;
* **MOV DRx,reg32** - работа с регистрами отладки;

а также команды работы со специфическими регистрами (TRx - для 386,486; MCRs - для Pentium и P6; MTRRs - для P6). Следует отметить, что команда **POPFD** также чувствительна к уровню привилегий. Она не изменяет состояние управляющих флажков IOPL, IF и др., если выполняется на уровне привилегий, отличном от нулевого. Кроме безусловно привилегированных команд есть команды чувствительные к уровню привилегий ввода-вывода. Это команды работы с портами (**IN, INS, OUT, OUTS**) и команды сброса/установки флажка разрешения прерываний (**CLI, STI**). Эти команды выполняются только в том случае, если **CPL<=IOPL**. Если это условие не выполняется, то для команд ввода-вывода производится дополнительная сверка с картой разрешения портов ввода-вывода. Если код не имеет привилегий на выполнение команды, то возникает нарушение общей защиты(исключение №13).

Второй аспект защиты - защита доступа к данным. Код имеет право обратиться к данным, которые находятся на том же или на более низком уровне привилегий. При этом учитывается не только CPL, но и RPL. Данные доступны, если дескриптор сегмента данных имеет DPL>=max(CPL,RPL). Такой контроль производится при загрузке селекторов в сегментные регистры (DS,ES,FS,GS). В сегментный регистр можно загрузить только селектор доступного с текущего уровня привилегий сегмента данных или, если сегментный регистр не будет использоваться, пустой селектор. Попытка нарушить правило привилегий или загрузить селектор системного дескриптора или дескриптора сегмента кода только для выполнения в сегментный регистр данных приведет к нарушению общей защиты (исключение №13). Кроме того, в командах изменения данных в памяти производится проверка на возможность записи в сегмент.

Особое правило привилегий для сегментов стека. Стек должен находиться строго на том же уровне привилегий, что и код программы (**DPL=CPL**). При этом сегмент стека обязательно должен быть присутствующим (P=1) и для него должны быть доступны операции и чтения, и записи.

Вызов подпрограмм без смены кодового сегмента в защищенном режиме процессора i386 производится аналогично вызову в реальном режиме с помощью команд JMP и CALL. Для вызова подпрограммы, код которой находится в другом сегменте (который может принадлежать библиотеке, другой задаче или операционной системе), процессор i386 предоставляет 2 варианта вызова, причем оба используют защиту с помощью прав доступа.

Первый способ состоит в непосредственном указании в поле команды JMP или CALL селектора сегмента, содержащего код вызываемой подпрограммы, а также смещение в этом сегменте адреса начала подпрограммы. Разрешение вызова происходит в зависимости от значения поля C в дескрипторе сегмента вызываемого кода. При C=0 вызываемый сегмент не считается подчиненным (согласованным), и вызов разрешается, только если уровень прав вызывающего кода не меньше уровня прав вызываемого сегмента, т.е. при условии CPL <= DPL. При C=1 вызываемый сегмент считается подчиненным и допускает вызов из кода с любым уровнем прав доступа, но при выполнении подпрограмма наделяется уровнем прав вызвавшего кода.

Очевидно, что первый способ непригоден для вызова функций операционной системы, имеющей обычно нулевой уровень прав, из пользовательской программы, работающей, как правило, на третьем уровне. Поэтому процессор i386 предоставляет другой способ вызова подпрограмм, основанный на том, что заранее определяется набор точек входа в привилегированные кодовые сегменты, и эти точки входа описываются с помощью специальных дескрипторов - дескрипторов шлюзов вызова подпрограмм. Этот дескриптор принадлежит к системным дескрипторам, и его структура отличается от структуры дескрипторов сегментов кода и данных (рисунок 10). Селектор из поля команды CALL используется для указания на дескриптор шлюза вызова подпрограммы в таблицах GDT или LDT. Для использования этого дескриптора вызывающий код должен иметь не меньший уровень прав, чем дескриптор, но если дескриптор шлюза доступен, то он может указывать на дескриптор сегмента вызываемого кода, имеющий более высокий уровень, чем имеет шлюз, и вызов при этом произойдет. При определении адреса входа в вызываемом сегменте смещение из поля команды CALL не используется, а используется смещение из дескриптора шлюза, что не дает возможности задаче самой определять точку входа в защищенный кодовый сегмент.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |  |
| Предел 0-15 | | | | | | | | | | | | | | | |  |
|  | | | | | | | | | | | | | | | | Мл. |
| 31 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 16 | слово |
| Селектор | | | | | | | | | | | | | | | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 |  | 9 | 8 | 7 |  |  |  | 4 | | |  | |  | 0 |  |
| P | DPL | | 0 | 0 | 1 | 0 | 0 |  | | | | | Счетчик слов | | | | | |  |
|  | | | | | | | | | | | | | | | | | | | Ст. |
| 31 |  |  |  |  |  |  |  |  |  |  |  | | |  | |  |  | 16 | Слово |
| Смещение 16-31 | | | | | | | | | | | | | | | | | | |  |
| Рис10. Дескриптор шлюза вызова для процессора i80386 | | | | | | | | | | | | | | | | | | | |

Каждый уровень привелегий использует свой стек. При переключении уровней привилегий происходит переключение стека. При этом из текущего стека в стек уровня доступа вызываемого сегмента копируется столько 32-разрядных слов, сколько указано в поле счетчика слов дескриптора шлюза. Перед этим во внутреннем стеке сохраняется указатель вершины внешнего стека. После копирования параметров во внутренний стек заносится адрес возврата. Привилегированная процедура должна заканчиваться инструкцией **RETF n**, где n - число байт, занимаемых параметрами в стеке.

Шлюзы можно адресовать только в команде FAR CALL, т.е. "насовсем" сменить уровень привилегий таким способом нельзя, всегда предполагается возврат на более низкий уровень привилегий.

### *3.2.4. Переключение в защищенный режим и обратно.*

Перед тем как переключить процессор в защищенный режим, надо выполнить некоторые подготовительные действия, а именно:

* Подготовить в оперативной памяти глобальную дескрипторную таблицу. В этой таблице должны быть созданы дескрипторы для всех сегментов, которые будут нужны программе сразу после того, как она переключется в защищенный режим. Впоследствии, Находясь в защищенном режиме, программа может модифицировать имеющиеся дескрипторы или добавлять новые, загрузив заново регистр GDTR.
* Запретить все маскируемые и немаскируемые прерывания.
  + Запомнить в оперативной памяти содержимое всех сегментных регистров, которые необходимо сохранить для возврата в реальный режим, в частности указатель стека реального режима.
* Загрузить регистр GDTR.

Для того, чтобы запретить немаскируемые прерывания, следует в порт с адресом 70h записать байт, в котором старший бит установлен в 1.

Для загрузки регистра GDTR используеться команда LGDTR src, где src указывает на область памяти с адресом и пределом таблицы GDT. Образ регистра GDTR в оперативной памяти может быть представлен следующей структурой:

GDTR label fword

GDT\_Limit DW ?

GDT\_Addr DD ?

Для переключения процессора из реального режима в защищенный надо установить бит PE системного регистра CR0 в 1. Для переключения можно использовать, например, такую последовательность команд:

Mov AX, CR0

Or AX, 1

Mov CR0, AX

Первой командой после перехода в защищённый режим должна быть команда дальнего перехода (far jump), в которой будет указан селектор дескриптора сегмента кода и смещение в этом сегменте. При работе в защищённом режиме процессор может использовать в сегментных регистрах только селекторы существующих дескрипторов, любые другие значения (например, сегментный адрес) использовать нельзя - процессор сгенерирует исключение общей защиты. Тем не менее, при переходе в защищённый режим регистр CS будет содержать сегментный адрес, который использовался в режиме реальных адресов, поэтому выполнение следующей команды, какой бы она ни была, должно было бы привести к генерации процессором исключения. На самом деле этого не происходит, так как эта команда не выбирается из памяти - она уже находится в конвейере процессора (даже в таком процессоре, как i386, есть конвейер) и поэтому вы можете выполнить эту команду.

Команда дальнего перехода обязательно очистит конвейер процессора и заставит его обратится к таблице GDT, выбрать оттуда дескриптор, селектор которого указан в адресе команды и начать выборку команд со смещения, также указанного в этом адресе. Это критический момент в работе программы. Если в GDT, селекторе, смещении или самой команде будет обнаружена ошибка, то процессор сгенерирует исключение, а так как систему прерываний мы для него пока не определяли, то он попросту зависнет либо произойдёт сброс - это уже зависит от "железа". Если вы не выполните первой команду дальнего перехода, а другую, которая не изменит содержимое регистра CS (а это - все остальные команды), то процессор произведёт выборку в конвейер новой команды, используя текущие значения CS:IP, а так как в CS содержится не селектор (процессор уже в защищённом режиме!), то произойдёт исключение и зависание.

Команду jmp far можно закодировать, например, следующим образом:

db 0eah ; jmp far Code\_selector:P\_Mode\_entry

dw P\_Mode\_entry

dw Code\_selector

Здесь Code\_selector – селектор дескриптора сегмента кода, а P\_Mode\_entry – смещение f

Для переключение из защищенного режима в реальный режим программа должна выполнить следующие действия:

* Передать управление сегменту кода с пределом 64Kбайта.
* Загрузить в регистры SS, DS, ES, FS и GS селекторы дескрипторов, подготовленных для адресации памяти в реальном режиме и содержащих соответствующие реальному режиму значения. Дескрипторы должны иметь следующие параметры:
  + Предел = 64 Кб (FFFFh)
  + Байтная гранулярность (G = 0)
  + Расширяется вверх (E = 0)
  + Записываемый (W = 1)
  + Присутствующий (P = 1)
  + Базовый адрес = любое значение

Сегментные регистры должны быть загружены ненулевыми селекторами. Те сегментные регистры, в которые не будут загружены описанные выше значения, будут использоваться с атрибутами, установленными в защищённом режиме.

* Запретить все маскируемые и немаскируемые прерывания.
* Сбросить бит PE регистра CR0, переключив процессор в реальный режим.
* Выполнить команду дальнего перехода для очистки внутренней очереди команд процессора.
* Настроить систему прерываний для работы в реальном режиме. (если она была изменена при входе в защищенный режим)
* Разрешить прерывания.
* Загрузить в сегментные регистры значения, необходимые для работы в реальном режиме.

Для разрешения всех прерываний можно воспользоваться следующим кодом:

Mov AL, 0dh ;разрешаем немаскируемые прерывания

Out 70h, AL

In AL, 21h ;разрешаем маскируемые прерывания

And AL, 0

Out 21h, AL

STI

## 3.3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем преимущество защищенного режима по сравнению с реальным?

2. Какую структуру имеет селектор адреса?

3. Поясните процесс преобразования логического (виртуального адреса) в линейный.

4. Поясните процесс преобразования линейного адреса в физический.

5. Какая информация храниться в дескрипторах дескрипторных таблиц?

6. Какой максимальный размер сегмента в защищенном режиме?

7. Сколько в системе может быть глобальных и локальных дескрипторных таблиц?

8. Какие подготовительные действия нужно выполнить перед переключением в защищенный режим?

10. Какие подготовительные действия нужно выполнить перед возвратом в реальный режим?

11. Почему первой командой после переключения процессора в защищенный режим должна быть команда дальнего перехода?

12. Как запретить немаскируемые прерывания?

## 3.4. ЛАБОРАТОРНОЕ ЗАДАНИЕ

Написать программу, переключающую процессор в защищенный режим, выводящую на экране сообщение и затем возвращающую процессор в реальный режим (8 часов).

# **ЦИКЛ 4. МУЛЬТИЗАДАЧНОСТЬ В ЗАЩИЩЕННОМ РЕЖИМЕ**

## 4.1. ЦЕЛЬ ЦИКЛА

Изучить принципы и средства реализации мультизадачности в защищенном режиме процессора. Получить практические навыки по программированию и использованию этих средств (8 часов).

## 4.2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Под мультизадачностью подразумевают способность компьютера выполнять несколько задач одновременно. На самом деле процессор некоторое время выполняет один командный поток, затем быстро переключается на второй и выполняет его, переключается на третий и т.д. При этом при каждом переключении сохраняется контекст прерываемого потока, так что потом процессор сможет "безболезненно" продолжить выполнение прерванного потока команд. Благодаря высокому быстродействию создается иллюзия того, что все задачи выполняются одновременно (параллельно).

Для управления мультизадачностью нет специальных команд. Задачи переключаются командами FAR CALL, FAR JMP, INT, IRET. Однако при этом участвуют специальные дескрипторы: дескриптор сегмента состояния задачи (Task State Segment) и дескриптор шлюза задачи. Когда управление передается на один из таких дескрипторов, происходит переключение задачи. При переключении задачи процессор сохраняет (восстанавливает) свой контекст в  [сегменте состояния задачи](http://avsdov.newmail.ru/modes/gdt-idt.htm) (TSS). Селектор TSS выполняемой задачи хранится в регистре задачи (Task Register). При переключении задачи процессор может сменить LDT, что позволяет назначить каждой задаче свое адресное пространство, недоступное для других задач. Можно также перегрузить CR3, что позволяет применить для изолирования задач механизм страничного преобразования.

Дескриптор TSS относиться к системным дескрипторам Поле Type дескриптора TSS может содержать код 1001, если это доступный TSS, или 1011, если это занятый TSS, т.е. если задача активна в настоящий момент.

На рис. 11 представлен формат сегмента TSS для процессора i80386. Из рису­нка видно, что в TSS предусмотрены поля для хранения сегментных регистров GS, FS, DS, SS, CS, ES. Имеется поле для хранения содержимого регистра LDTR, указываю­щего на локальную таблицу дескрипторов, распределённую данной задаче. Для хранения содержимого 32-разрядных регистров используются поля TSS, обозначенные на рисунке как EDI, ESI, EBP, ESP, EBX, EDX, ЕСХ, ЕАХ, EFLAGS, EIP.

Поле CR3 хранит содержимое системного регистра CR3. Этот регистр являет­ся указателем на каталог таблиц страниц. Таким образом, каждая задача может иметь свой собственный каталог таблиц страниц, что позволяет выполнить изо­ляцию задач не только на уровне сегментов, но и на уровне страниц.

|  |  |  |  |
| --- | --- | --- | --- |
| Битовая карта ввода/вывода (БКВВ) | | |  |
| Дополнительная информация ОС | | |  |
| Относительный адрес БККВ | 0 | T | 64 |
| 0 | LDTR | | 60 |
| 0 | GS | | 5С |
| 0 | FS | | 58 |
| 0 | DS | | 54 |
| 0 | SS | | 50 |
| 0 | CS | | 4С |
| 0 | ES | | 48 |
| EDI | | | 44 |
| ESI | | | 40 |
| EBP | | | 3С |
| ESP | | | 38 |
| EBX | | | 34 |
| EDX | | | 30 |
| ECX | | | 2С |
| EAX | | | 28 |
| EFLAGS | | | 24 |
| EIP | | | 20 |
| CR3 | | | 1С |
| 0 | SS2 | | 18 |
| ESP2 | | | 14 |
| 0 | SS1 | | 10 |
| ESP1 | | | С |
| 0 | SS0 | | 8 |
| ESP0 | | | 4 |
| 0 | LINK | | 0 |
| Рис. 11. Сегмент TSS процессора i80386 | | | |

TSS процессора i80386 содержит указатели на стеки для второго, первого и нулевого приоритетных колец. Это поля SS2:ESP2, SS1:ESP1, SS0:ESP0.

Поле LINK используется для ссылки на TSS, вызвавшей задачи при вложен­ном вызове задач, аналогично тому, как это было в процессоре i80286.

Бит Т используется для отладки. Если он установлен в 1, при переключении на задачу возникает отладочное исключение, которое может быть использовано системным отладчиком.

Для обеспечения безопасной работы системы необходимо ограничить доступ программам пользователя ко всем или по крайней мере к некоторым портам ввода/вывода. Злонамеренная программа, имеющая доступ к портам контроллера прямого доступа к памяти, может выполнить с помощью этого контроллера чтение или запись информации по любым физическим адресам. Процессор i80286 хранит в регистре флагов уровень привилегий I0PL, на котором разреше­но выполнять команды ввода/вывода. С помощью этого механизма можно зап­ретить непривилегированным программам выполнять команды ввода/вывода.

Однако такой способ защиты не слишком удобен. Некоторые порты ввода/вы­вода не только безопасны для использования, но и весьма полезны для обычных программ (например, порт системного динамика или принтера).

Битовая карта ввода/вывода процессора i80386 позволяет для каждой задачи определить порты, которые эта задача может использовать. То есть операционная система имеет возможность санкционировать любую задачу для использова­ния любого набора адресов портов ввода/вывода. Если задача попытается обра­титься к несанкционированному порту ввода/вывода, произойдёт исключение.

Сегмент TSS содержит поле, обозначенное на рис. 2.1. как база карты ввода/вывода. Оно служит для указания расположения битовой карты ввода/вывода задачи, использующей данный TSS. Поле базы карты ввода/вывода указывает 16-разрядное смещение начала битовой карты ввода/вывода относительно TSS. Предел TSS должен определяться с учётом карты. Каждый бит в карте ввода/вывода соответствует адресу байта порта ввода/вывода (карта состоит из 64 Кбит для описания доступа к 65536 портам). После битовой карты должен располагаться байт OFFh.

При выполнении 16- или 32-разрядных операций ввода/вывода процессор проверяет все биты (2 или 4 бита), соответствующие адресу порта. Если проверяемый бит установлен в 1, происходит исключение.

Для привилегированных программ, если уровень привилегий меньше или ра­вен уровню I0PL, процессор не выполняет проверку битовой карты ввода/выво­да. Чтобы полностью запретить задаче обращаться к портам ввода/вывода, дос­таточно установить базу карты ввода/вывода большей или равной пределу TSS. В этом случае любая команда ввода/вывода приведёт к генерации исключения.

Лишь значение первых 68h байт сегмента состояния задачи строго определены. Именно это число является минимальным размером TSS. Операционная система может по своему усмотрению устанавливать размер TSS и включать в сегмент TSS дополнительную информацию, необходимую для работы задачи и зависящую от конкретной операционной системы (например, контекст сопроцессора, указатели открытых файлов или указатели на именованные конвейеры сетевого обмена). Включенная в этот сегмент информация автоматически заменяется процессором при выполнении команды CALL или JMP, селектор которой указывает на дескриптор сегмента TSS в таблице GDT (дескрипторы этого типа могут быть расположены только в этой таблице).

При переключении задачи с помощью прерывания или особого случая происходит автоматический возврат к прерванной задаче. Однако, организуя вложение задач, необходимо помнить, что, в отличие от процедур, при переключении задачи в стек ничего не включается. Дескриптор TSS задачи, выполняемой в данный момент, помечается как занятый. При переключении на другую задачу с вложением (по INT или FAR CALL) дескриптор TSS остается помеченным. Переключиться на занятую задачу нельзя (возникает нарушение общей защиты - исключение №13).

Для переключения задач также действуют правила привилегий. По команде JMP или CALL можно переключиться на задачу, TSS которой менее привилегирован:

DPLTSS>=max(CPL,RPL).

Для особых случаев и прерываний это правило не действует. Если обработчик прерывания выполнен в виде отдельной задачи, то он может быть вызван независимо от значения CPL.

Не совсем удобно адресовать именно TSS для переключения задачи, т.к., во-первых, TSS могут быть размещены только в GDT (а в IDT или LDT - нет), а, во-вторых, если пользоваться только TSS, то каждую задачу мы "намертво" привязываем к определенному уровню привилегий (DPLTSS), с которого она доступна для переключения. Этих недостатков лишены шлюзы задачи. Формат дескриптора шлюза задачи приведен на рис. 12.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |  |
| Не используется | | | | | | | | | | | | | | | |  |
|  | | | | | | | | | | | | | | | | Мл. |
| 31 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 16 | слово |
| Селектор TSS | | | | | | | | | | | | | | | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 |  | 9 | 8 | 7 |  |  |  |  | |  | |  | 0 |  |
| P | DPL | | 0 | 0 | 1 | 0 | 1 | Не используется | | | | | | | | | |  |
|  | | | | | | | | | | | | | | | | | | Ст. |
| 31 |  |  |  |  |  |  |  |  |  |  |  | |  | |  |  | 16 | Слово |
| Не используется | | | | | | | | | | | | | | | | | |  |
| Рис12. Дескриптор шлюза задачи для процессора i80386 | | | | | | | | | | | | | | | | | | |

Шлюз задачи содержит селектор TSS. Шлюзы задач можно размещать и в IDT, что позволяет выполнять обработчики прерываний в виде отдельных задач, и в LDT, что позволяет более гибко управлять переключением задач: для второй задачи первая может быть видна с одного уровня привилегий, а для третьей - с другого. Последняя возможность обеспечивается особым правилом привилегий: при переключении задачи через шлюз учитывается только DPLшлюза, а DPLTSS не играет роли, поэтому одной задаче может соответствовать множество шлюзов с различными DPL.

При переключении задач процессор выполняет следующие действия:

1) Выполняется команда CALL, селектор которой указывает на дескриптор сегмента типа TSS.

2) В TSS текущей задачи сохраняются значения регистров процессора. На текущий сегмент TSS указывает регистр процессора TR, содержащий селектор сегмента.

3) В TR загружается селектор сегмента TSS задачи, на которую переключается процессор.

4) Из нового TSS в регистр LDTR переносится значение селектора таблицы LDT в таблице GDT задачи.

5) Восстанавливаются значения регистров процессора (из соответствующих полей нового сегмента TSS).

6) В поле селектора возврата заносится селектор сегмента TSS снимаемой с выполнения задачи для организации возврата к прерванной задаче в будущем.

Вызов задачи через шлюз происходит аналогично, добавляется только этап поиска дескриптора сегмента TSS по значению селектора дескриптора шлюза вызова.

### *4.3. Контрольные вопросы*

1. Какая обязательная информация сохраняется в сегменте состояния задачи?

2. Какая дополнительная информация может быть сохранена в сегменте состояния задачи?

3. Для чего используется битовая карта ввод/вывода?

4. Какие существуют в защищенном режиме способы переключения задач? Сравните их.

5. Какие действия выполняет процессор при переключении задач?

### *4.4. Лабораторное задание*

Написать программу, реализующую мультизадачность в защищенном режиме. Программа должна переключить процессор в защищенный режим, а затем запустить на выполнение 2-3 задачи, которые должны выполняться параллельно. Каждая задача выводит на экран свое сообщение. Задача выводит на экран часть сообщения, затем происходит переключение на другую задачу и т.д. Когда все задачи отработают программа должна вернуть процессор в реальный режим (6 часов).