# Режимы

Компьютеры на базе процессоров Intel работают в 3 режимах:

1. Реальный – 16р режим с 20р шиной адреса, intel 8086. 2^20=позволяют адресовать до 1 Mбайт=1024Кбайт

Работали под управлением DOS (disk operating system), то есть с внешней дисковой памятью. DOS – однозадачная ОС – в оперативной памяти только одна программа, которая выполняется от начала до конца.

Компьютер начинает работать в реальном режиме (чтобы выполнять меньше команд при загрузке)

- Максимально возможный размер сегмента в реальном режиме - 64КБ

- Минимальная адресная единица памяти – байт.

1. Защищенный (protected) – 32р режим с 32р регистрами и шиной адреса (4 ГБ). 4х уровневая система привилегий. Поддерживали 2 независимые схемы управления виртуальной (те фактически несуществующей) памятью – сегментами по запросу и страницами по запросу. Существует аппаратная схема управления памятью. (третья) сегменты, поделенные на страницы – взяли лучшее из 2 схем. Но поддерживаются только первые 2 и они независимые.

Существует специальный режим защищенного режима – v86 (virtual). Как задачи в режиме v86 запускаются ОС реального режима. Запускается виртуальная 86 машина, и в этой среде может выполняться одна программа реального режима.

Многозадачный. Каждая запущенная виртуальная машина является v86 со всеми вытекающими (1 задача, 1Мб, 16р операнды).

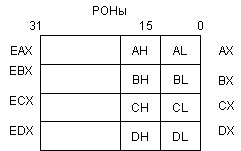
Почему «защищенный»: Windows – система разделения времени. Адресное пространство каждого процесса должно быть защищено, как и адресное пространство ОС.

1. Long (длинный) – 64р регистры, операнды. Многопроцессность. Только страничная виртуальная память. Поддерживает compatibility – режим совместимости, в котором могут выполняться 32р. Как работает обратная совместимость – рассмотреть регистры. Режима v86 нет!

?но адреса меньше 64-х разрядов (это связано с аппаратными ограничениями).

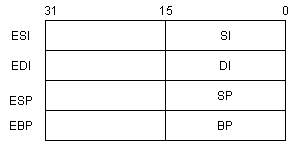
# Группы регистров

1. Регистры общего назначения (32)



Аппаратно доступна младшая часть, в которой доступны младшая и старшая части (?Док-во того, что аппаратно поддерживается реальный режим)

1. Индексные и указательные регистры (32)



1. Сегментные регистры (16)

В защищенном режиме 6 сегментных регистра, в реальном режиме 4.

CS, DS, SS, ES, FS, GS

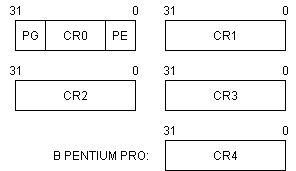
1. Регистры системных адресов (регистры управления памятью)

При работе в защищенном режиме микропроцессора адресное пространство делится на глобальное – общее для всех задач; локальное – отдельное для каждой задачи.

* 1. GDTR(32, в РФ-6байт) (Global Descriptor Table Register) - регистр таблицы глобальных дескрипторов. Содержит линейный физический адрес начала таблицы дескрипторов – адрес байта начала таблицы глобальных дескрипторов (GDT)
  2. IDTR(32) (Interrupt Descriptor Table Register) – регистр таблицы дескрипторов прерываний. Содержит линейный физический адрес начала таблицы дескрипторов прерываний. – адрес байта начала таблицы глобальных дескрипторов (IDT)
  3. LDTR(16) (Local Descriptor Table Register) - регистр локальной таблицы дескрипторов дескрипторов. 16 бит -> не может содержать линейный физический адрес. содержащего так называемый селектор дескриптора локальной дескрипторной таблицы LDT Этот селектор является указателем в таблице GDT, который и описывает сегмент, содержащий локальную дескрипторную таблицу LDT;
  4. TR (16) (Task Register), который подобно регистру ldtr, содержит селектор, т. е. указатель на дескриптор в таблице GDT. Для переключения задач.

Линейный адрес - физический адрес оперативной памяти

1. Управляющие регистры (32)



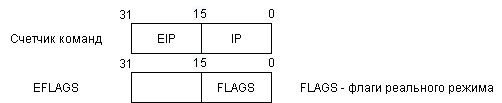
* CR0 – регистр слова состояния.

Содержит 6\_\_(7) флагов

* + 0 PE (protection enable) - определяет, в каком режиме работает комп. Если установлен, значит в защищенном, нет – в реальном
  + 31 PG (paging enable). Если установлен, значит выполняется страничное преобразование. НО это имеет смысл только в защищенном режиме, те PG=1 имеет смысл только при PE=1
* CR1 – не исп.
* CR2 - регистр линейного адреса ошибки обращения к странице (страничная неудача) – исключение, которое возникает, когда процессор обращается к странице, отсутствующей в физической памяти. На какой команде это произошло, ее адрес будет занесен.
* CR3 - регистр начального адреса каталога таблиц страниц. ЕГО НАЛИЧИЕ – доказательство того, что управление таб. по запросу независимо от сегмента (те 2 схема, а не 3) (чисто страничное преобразование). А флаг PG не является доказательством.

В LONG только PAE Physical Address Extension (расширение физического адреса). Включается же PAE установкой пятого бита регистра CR4 в единицу. С помощью PAE 32битный указатель позволяет адресовать до шестидесяти четырех (2^36) гигабайт физической памяти.

1. 32-р: Тестовые, отладочные регистры, EIP, EFLAGS



19-VIF флаг виртуального прерывания. *используется совместно с флагом VIP и позволяет обеспечить нормальное выполнение старого ПО, использующего команды управления внешними маскируемыми прерываниями (векторы от 32 до 255), в современной мультипроцессорной и мультизадачной программно-аппаратной среде.*

20-VIP Флаг ожидания виртуального прерывания

17 VM – Virtual 8086 mode (386+)

В режиме long – RFLAGS, но все не используются.

# Переход в защищенный режим.

Установить флаг PE не достаточно чтобы прейти в защищенный режим - нужно провести некоторые приготовления. Программам нужны некоторые ресурсы, чтобы они могли исполняться - это может быть память, процессорное время, регистры процессора, порты ввода/вывода.

Нас будет интересовать память(адресация). В реальном режиме были только сегмент и смещение. Адрес = сегмент\*16 + смещение - получаем линейный физический адрес. Доступное адресное пространство - 1 МБ.

В защищенном же режиме есть поддержка виртуальной памяти, в том числе и на аппаратном уровне.

## Управление памятью, сегментами в защищенном режиме.

Для того, чтобы использовать память, нам ее нужно сначала выделить и описать. В системе есть специальные таблицы для управления памятью.

* GDT - глобальная таблица дескрипторов
* IDT - таблица дескрипторов прерываний
* LDT - локальная таблица дескрипторов

Регистры системных адресов - чтобы поддерживать эти таблицы.

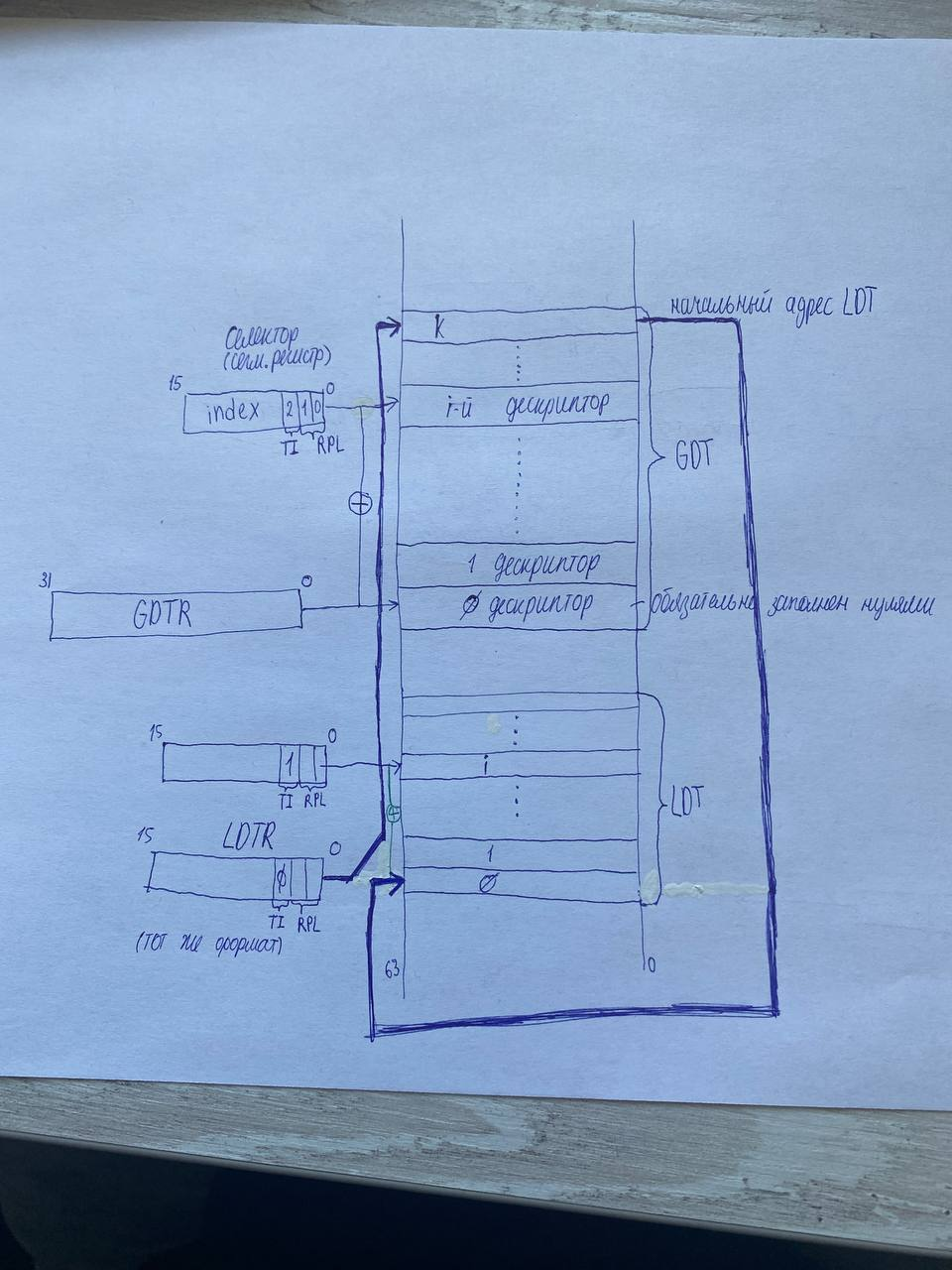
* GDTR
* IDTR
* LDTR
* TR

Напрямую эти регистры недоступны - есть специальные команды чтобы загружать или выгружать их, эти команды привилегированные. (Lgdt)

# Системные таблицы

## Глобальная таблица дескрипторов

В SMP архитектуре (наши компы) равноправные процессоры, которые работают с общей памятью. Один – главный, обрабатывает прерывание от системного таймера, но не руководит другими процессорами.

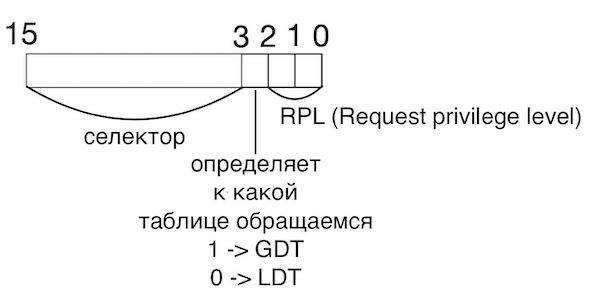


Так как одна память, то и GDT в системе одна. Она находится в памяти ядра системы. На начало таблицы указывает GDTR (32).

GDT состоит из 8-байтных записей - дескрипторов. Первая запись всегда должна быть заполнена нулями и не используется. Далее следуют дескрипторы сегментов.

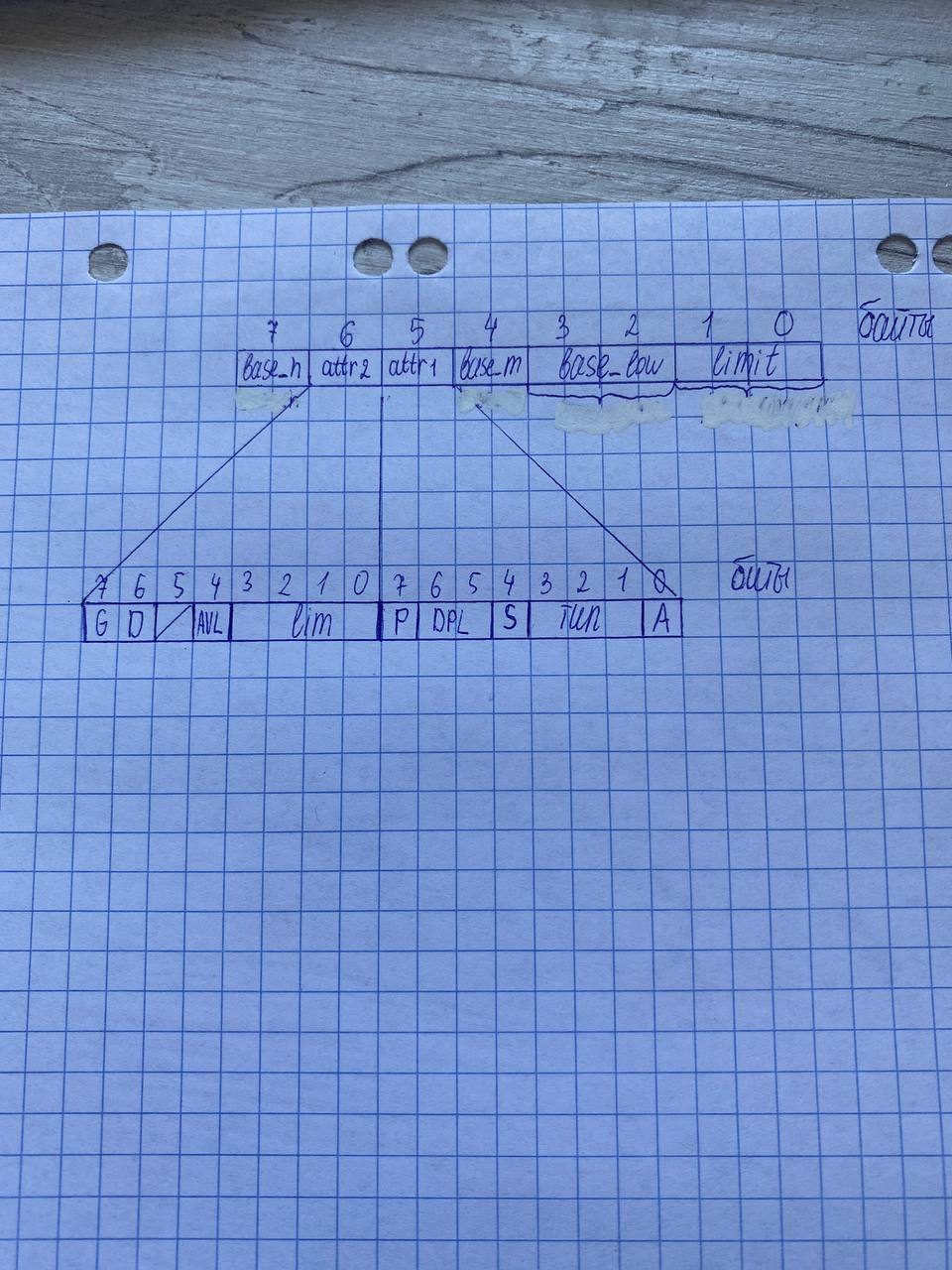
## Формат селектора.

Сегментный регистр – селектор (16) – идентификатор сегмента - внутри себя содержат номер записи в таблице(индекс), по нему получаем запись в ней - то есть дескриптор.



* 0 и 1 бит – RPL (request privilege level) отвечают за уровни привилегий - их всего 4, используются процессором при проверке возможности доступа к сегментам.
* 2 бит TI table indicator - определяет к какой таблице идет обращение (1- к локальной или 0 - к глобальной)
* 3-15 - индекс. *Так как размер дескриптора 8 байт, то минимальный селектор должен тоже быть 8 - мы через первые три бита домножаем селектор на 8(добавляем три разряда в двоичной). 01 000 -> 8 (первый селектор), 10 000 -> 16 (второй селектор)*

## Формат дескриптора



В РР сегменты определяются базовыми адресами, задаваемыми в явной форме, В ЗР - дескриптором (8-байтовым полем)

Формат дескриптора для GDT:

* Байты 2-3 (base\_low), 4 (base\_middle), 7 (base\_heigh): база сегмента - начальный линейный адрес сегмента в адресном пространстве процессора. (имеет длину 32 бита, номер байта, может располагаться в любом месте адресного пространства 4Гбайт). Если страничная адресация выключена, он совпадает с физическим (как во 2 ЛР), включена - могут и не совпадать.

База - адрес, с которого начинается данный сегмент. Повторюсь: адрес в виртуальном адресном пространстве. Вообще, все упоминаемые здесь и далее адреса упоминаются в контексте виртуальности; к физическим адресам мы доступа не имеем.

* Байты 0-1 (limit): младшие 16 бит границы сегмента -номер последнего байта сегмента).
* Байт 6 (atrr\_2)
  + 0-3 (lim): оставшиеся старшие 4 бита границы сегмента (итого 20 бит). *Поскольку у регистров доступны младшие части, это показывает, что старшие компьютеры поддерживают реальный режим аппаратно - основанная идея Intel - обратная совместимость. Возникает вопрос - сколько разрядов в шине адреса в реальном режиме? 20. Это максимально возможный объем который мы можем адресовать в реальном режиме - 1 МБ памяти. 0,1 - limit (только 16 разрядов), а шина 20-разрядная - нам не хватает 4 разрядов. Таким образом мы имеем базовый линейный адрес - можем адресовать начало сегмента.*
  + 6 бит (D default): разрядность операндов и адресов по умолчанию
    - 0-16
    - 1-32

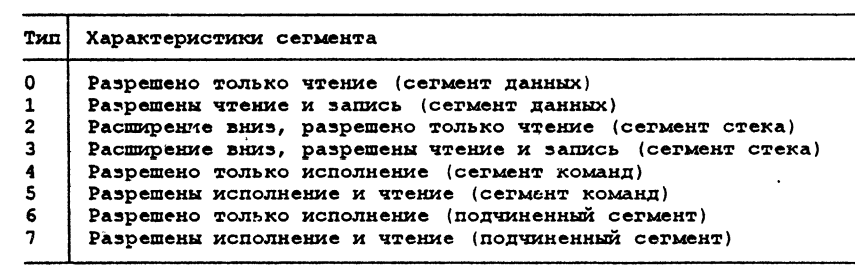
Можно изменить на противоположный префиксом замены размера 66h(операнда)

и 67h (адреса). D=0 не запрещает использовать 32 регистры: компилятор сам добавит префикс

* + 7 бит G (бит дробности (гранулярности)): единицы, в которых задается граница.
    - 0-в байтах (и тогда сегмент <=1 Mбайт),
    - 1-в блоках по 4 Кбайт (страницах) (до 4 Гбайт)

гр. сег.=гр.в.дескр.\*4К+4095 - до конца последнего 4-Кбайтного блока).

* + *5 L - флаг, который ранее был зарезервирован, теперь служит признаком 64-разрядности сегмента. Если он установлен, флаг D/B должен быть сброшен.*
  + *4 AVL - неиспользуемый бит. Может использоваться по усмотрению ОС.*
* Байт 5: attr\_1:
  + 0 бит (A accessed): устанавливается процессором, когда в какой-либо сегментный регистр загружается селектор данного сегмента (было обращение)
  + 1-3 биты: тип сегмента.
    - 3 бит – бит предназначения: 0 – сегмент данных/стека, 1-кода
    - 2 бит:
      * Для кода [бит подчинения: 0 – код подчинен (связан с каким –то другим сегментом),1 – обычный]. Подчиненные, или согласованные сегменты обычно используются для хранения подпрограмм общего пользования; для них не действуют общие правила защиты программ друг от друга.
      * Для стека и данных [0-данные, 1-стек]
    - 1 бит:
      * Для кода [0–чтение из сегмента запрещено (не относится к выборке команд) - считывание из памяти и загрузка в регистры процессора, mov, 1 – разрешено]
      * Для данных [0- модификации запрещены, 1-модификации разрешены]
  + 4 бит (S system): идентификатор сегмента (0-системный сегмент, 1-сегмент памяти).
  + 5-6 биты (DPL descriptor privilege level): уровень привилегий этого дескриптора: от 0 (УП ядра системы) до 3 (УП приложений). (как в селекторе RPL request PL (программно), CPL current PL (аппаратно))
  + 7 (P present): бит присутствия, представлен ли сегмент в памяти (выгружен ли из внешней в оперативную).



## Локальные таблицы дескрипторов.

LDT описывает виртуальное адресное пространство процессов-> их столько, сколько процессов выполняется системой. В памяти занимают сегмент (64 кб). Сегменты описывает GDT->LDTR-селектор к дескриптору сегмента, в котором находится таблица локальных дескрипторов LDT.

Используются процессами - они могут там хранить свои сегменты и обращаться к ним вместо того чтобы хранить это в глобальной таблице.

Начинается с нулевого дескриптора.

См. рисунок выше

Формат дескрипторов абсолютно такой же.

*В LDTR хранится селектор дескриптора глобальной таблицы, по этому смещению хранится дескриптор сегмента LDT. То есть у нас локальная таблица дескрипторов находится в сегменте, который уже должен быть выделен в GDT.*

*GDT размещается в защищенной области памяти, к которой имеет доступ только ядро операционной системы.*

*Регистр LDTR может загружаться при переключении между задачами и у каждой задачи может быть своя локальная таблица дескрипторов.*

*Таким образом мы берем из GDT базовый линейный адрес и получаем начало таблицы LDT.*

*Например, мы загрузили в DS селектор из LDT, то процессору нужно сначала найти сегмент где хранится LDT, получить ее базовый линейный адрес и уже тогда к нему прибавить смещение из селектора. Получим снова базовый линейный адрес, к которому прибавляем смещение - получаем соответствующий линейный адрес.*

P6 есть, но нет флага, отключающего сегментное преобразование. ЭТО БАЗА. В защищенном режиме используется модель памяти flat.

# Семинар 2

GDT позволяет адресовать память. В архитектуре ФН все адресуется, команды и данные в оперативной памяти, доступ по адресу, который нужно сформировать.

В ЗР (под 9) 2 схемы управления памятью:

* сегментами по запросу
* страницами по запросу.

Они предполагают выполнение преобразований.

* Сегментированный адрес – не прямая аналогия с DOS. В DOS это связано с увеличением до 20р шины данных, и адрес вычисляется как сдвинутый сегментный регистр + смещение.
* В ЗР – полноценно: есть начальный адрес сегмента, к которому добавляется смещение (смещение берется из команды). Любая программа считает, что она начинается с 0 адреса, следовательно, смещение берется из команды. Это счетчик команд, или индексный регистр, или указательный регистр, или операнд из команды. Это прибавляется к адресу сегмента, и мы получаем линейный физический адрес.

Intel поддерживает LDT, но LDTR 16 разрядный – не может содержать линейный физический адрес. Но по аналогии с сегментными регистрами – селекторы к дескрипторам GDT, и они уже описывают сегменты памяти. LDT описывает виртуальное адресное пространство, следовательно, их столько, сколько процессов. (Но мы процессы не создаем в ЛР)

GDT и IDT – системные таблицы.

*Сегменты 16-разрядные, а ЗР 32-разрядный, следовательно, не выходят за 1 МБ. В limit для всех дескрипторов записано ffff=2^16=64Кбайт – разрядность регистров в РР, следовательно, смещение в РР не может превышать 2^16. FFFFF=2^20=1Мбайт (мое с семинара, но что-то странное)*

# Теневые регистры

С каждым сегментным регистром сопоставлен теневой регистр, в котором при обращении к сегментному регистру записывается информация из дескриптора. Цель – исключить обращение к GDT, которая находится в оперативной памяти. Оперативная память отстает по времени от проц., затратное действие (цикл обращения к памяти требует определенного количества тактов).

Обращение к ОП осуществляется на каждой команде, а то и несколько раз + надо сформировать (преобразовать) адрес особенно если адресация косвенная. Чтобы цикл обращения к GDT для получения физического адреса, или данных, или следующей команды, информация с дескриптора записывается в теневой регистр, и после того, как было обращение к сегментному регистру.

Теневой регистр находится непосредственно в процессоре. Это не исключает обращение для считывания команд, записи и тд.

(в ЛР тк в РР смещение <=ffff, ни один сегмент не выходит за границы (видимо, про ненужную загрузку ffff во 2 ЛР))

# Определение объема доступного физического адресного пространства

Max=4ГБ в ЗР. Чтобы его адресовать, надо объявить.

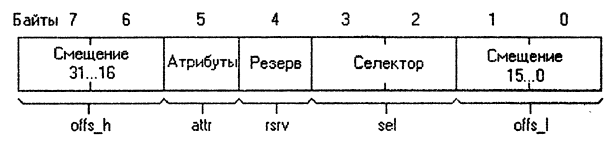
Потому что в 1 мб лежит ROM BIOS и мы его перезаписываем и происходит ошибка прав доступа. По-моему она говорила ROM Bios

1Мб пропускаем (там наша программа или *В первом мегабайте хранится операционная система ROM BIOS* - спорно), со второго сохраняем байт/слово, записываем туда сигнаттуру, сравниваем со своей, если совпали-память и инкремент счетчика. Сегмент данных, limit=fffff, G=1, чтение и запись.

В GDT дб описаны: 16р кода (тк стартуем в 16р), 16р данных (в РР описываем там таблицы), 32р кода (тк перешли), 32р данных (для определения памяти), 32р стек (для прерываний в ЗР)(хотя при возвращении надо бы вернуться к новому). В ЗР 2 обработчика прерываний-от клавиатуры и системного таймера.

# Прерывания в ЗР

Формат дескриптора (шлюза) для IDT (байты)



* Байты 0-1 (offs\_1), 6-7 (offs\_h): 32-битное смещение обработчика
* Байты 2-3 (sel): селектор(сег. команд)(итого полный 3-хсловный адрес обработчика селектор: смещение)
* Байт 4 зарезервирован
* Байт 5: байт атрибутов - как в дескрипторах памяти за исключением типа:

Типы: назначение:

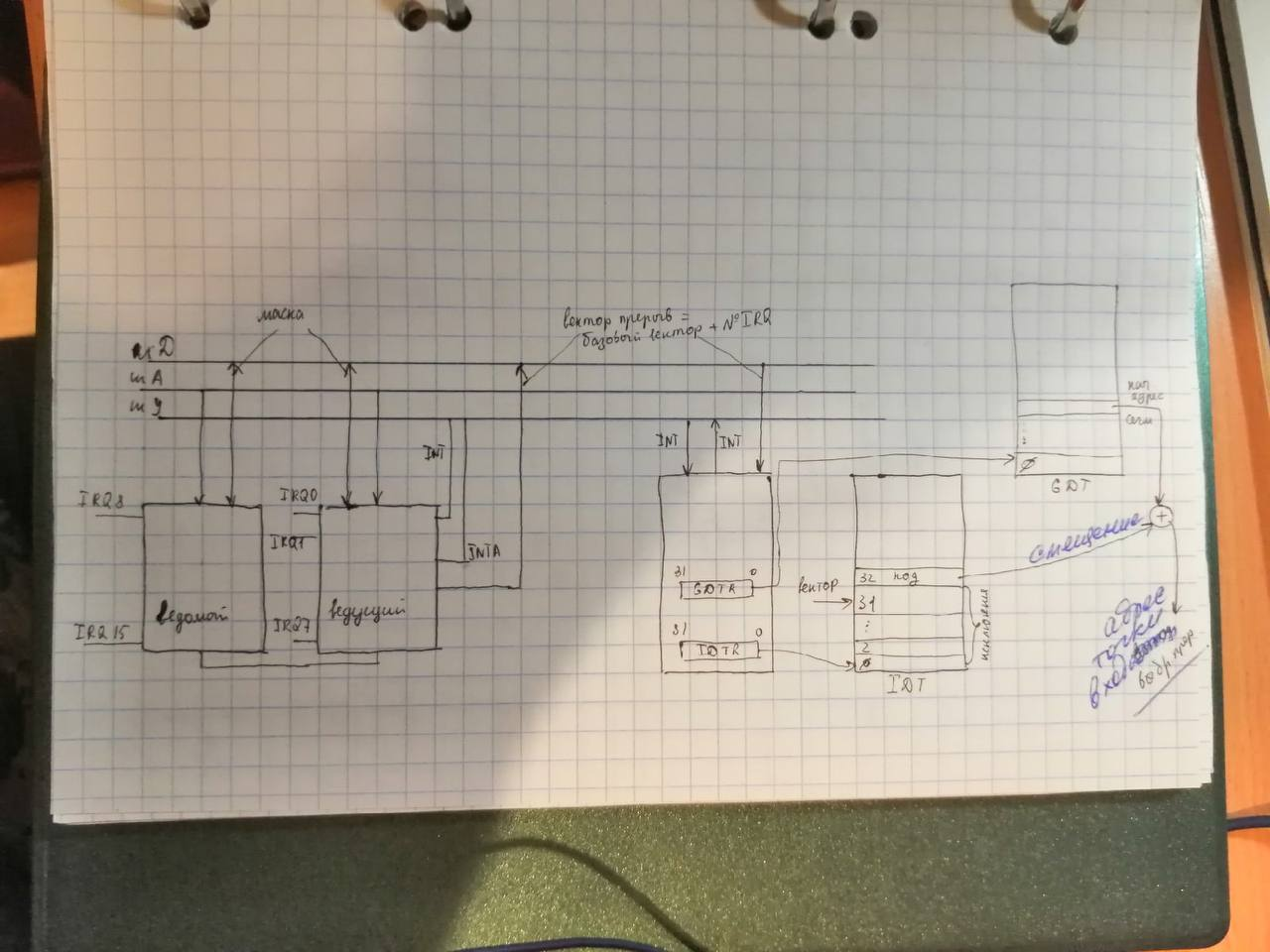
* + 0-не определен
  + 1-свободный сегмент состояния задачи TSS 80286
  + 2-LDT
  + 3-занятый сегмент состояния задачи TSS 80286
  + 4-шлюз вызова Call Gate 80286
  + 5-шлюз задачи Task Gate
  + 6-шлюз прерываний Interrupt Gate 80286
  + 7-шлюз ловушки Trap Gate 80286
  + 8-не определен
  + 9- свободный сегмент состояния задачи TSS 80386+
  + Ah-не определен
  + Bh- занятый сегмент состояния задачи TSS 80386+
  + Ch-шлюз вызова Call Gate 80386+
  + Dh- не определен
  + Eh-шлюз прерываний Interrupt Gate 80386+
  + Fh- шлюз ловушки Trap Gate 80386+

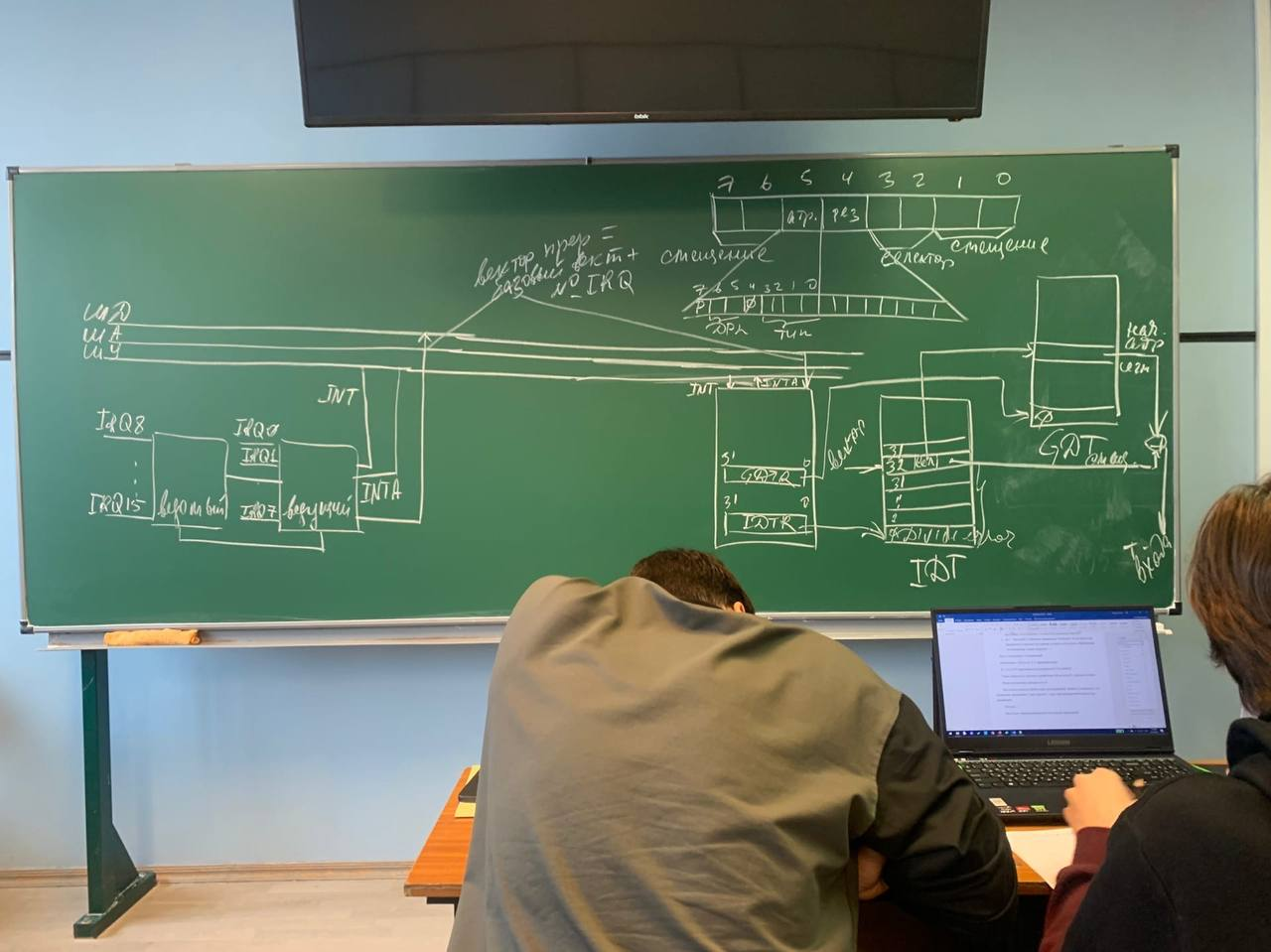
Адресация исключений, аппаратных прерываний и системных вызовов (программных прерываний в ЗР). Исключения и системные вызовы-TRAP. Когда говорят interrupt без ничего – аппаратное прерывание

через шлюзы прерываний (interrupt) обрабатываются аппаратные прерывания, ловушек (TRAP)-программные прерывания (системные вызовы) и исключения

*Может принимать 16 значений, но в IDT допустимо 5: 5(задачи), 6(прерываний 286), 7(ловушки 286), Eh(прерываний 3/486), Fh(ловушки 3/486) через шлюзы прерываний обрабатываются аппаратные пр., ловушек-программные пр. и искл. (это по РФ)*

* 4-пустой, 5-6-DPL, 7 P (1)





(шедевральный рисунок Рязановой)

(говняный рисунок Иры с божественными подписями Рязановой)

(точка входа)

(к рисункам выше)

IDTR-32 разрядный. Содержит начальный линейный адрес IDT (все эти регистры в каждом ядре). Нулевой регистр не пустой, как в GDT, а деление на 0.

IDT:

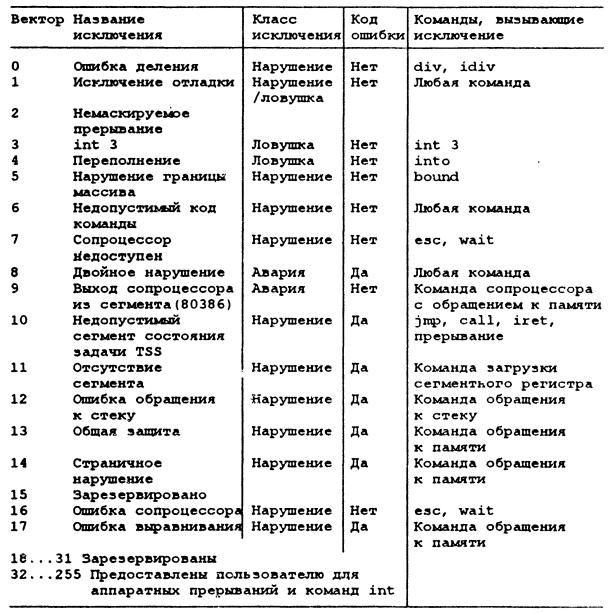
Первые 32 элемента таблицы - под исключения (синхронные события в процессе работы программы) (внутренние прерывания процессора) (в 386 всего 19 исключений (0-19), остальные (20-31) зарезервированы, а *в 486 – и того меньше (реально-18, остальные-зарезервированы, на рисунке-для 486)*)

32-255 – определяются пользователем (user defined)

* 0-divide error (ошибка деления на 0)
* 8-double fault (если выполнить исключение или маскируемое/немаскрируемое прерывание и возникла ошибка (паника…), завершается работа компьютера)
* 11-segment not present (сегмент отсутствует – надо выполнить определенные действия, чтобы сделать сегмент доступным. Касается управления памятью (нашей программе-не очень))
* 13-general protection (общая защита, должно быть обработано специальным образом. На все исключения-заглушки (double fault-не искл.), а на 13-специальная заглушка (у РФ отражено в структуре таблицы дескрипторов прерываний-без dup))

(нарушение общей защиты (нарушение, код ошибки-та команда), происходит: за пределами сегмента, запрет чтения, за гр. таблицы дескр., int с отс. Номером)

* 14-page fault (fault переводится как исключение, но по-русски здесь прерывание) (страничное прерывание) – обращение к команде/данным, отсутствующим в программе – система должна загрузить нужную страницу. В CR2 адрес, на котором произошло прерывание)



Смещение=номер исключения\*8

Нам надо адресовать 2 обработчика – таймера и клавиатуры (аппаратные прерывания) через программирование контроллера прерываний. Сейчас прерывания как MSI (message signal interrupts)

Схема:

(Если прерывание не замаскировано)

В РР процессор использует вектор и таблицу векторов прерываний. У ведущего контроллера базовый вектор=8 (8+0=8h) и номер используется для получения смещения к адресу в таблице векторов прерываний. В DOS адрес наз. вектор (4 байт)

В ЗР для адресации прерывания имеется специальная таблица IDT. Если первые 32 исключения и мы возьмем 8, то попадем на double fault. Чтобы адресовать прерывания, надо перепрограммировать контроллер на: ведущий-на базовый вектор 32, и этот номер использовать для обращения к таблице. Базового адреса нет, есть смещение и селектор. Отдельно адресуются ведущий и ведомый. От контроллера они могут получить маску??

## Вопросы ко 2 ЛР

* какую программу вы написали

В защищенном режиме написали управляющую программу с 0 уровнем привилегий. Данная программа выполняет две функции ОС: переход в защищенный режим и выделение сегментов памяти, описанных в глобальной таблице дескрипторов и размер, которых мы подсчитываем в программе

Программа 0 уровня привилегий - фактически часть ОС или даже своя ОС.

где в двух местах говорится что у нас 0 уровня привилегии

DPL и Rpl

Где находится rpl ?

В селекторе.

* что пришлось создать в этой программе раз это такая программа

Создать две системные таблицы – глобальную таблицу дескрипторов сегментов ( GDT ) для описания сегментов физической памяти, с которыми будет работать запущенная программа и таблицу дескрипторов прерываний ( IDT), в которой заполняются дескрипторы прерываний, которые необходимы для выполнения поставленной задачи.

* где явно установлен соответствующий уровень привилегий

То, что мы используем системные таблицы.

GDT размещается в защищенной области памяти, к которой имеет доступ только ядро операционной системы. Она находится в памяти ядра системы.

МБ еще команды lidt, lgdt

привилегии (4 кольца защиты) != приоритет (приоритет процессов потоков, который назначается и потом может быть пересчитан)

* какие сегменты вы описали в глобальной таблице и почему (для чего)

1. 16-разрядный сегмент кода, граница в байтах (G=0)(для реального режима)
2. 32-разрядный (D=1) сегмент данных, размер: 4 Гб (для определения объема выделенной памяти)
3. 32-разрядный (D=1) сегмент кода, граница в байтах (G=0) ;(для защищенного режима)
4. 32-разрядный (D=1) сегмент данных, граница в байтах (G=0) (от нас там данные всякие, рязанова хотела 16-р)
5. 32-разрядный (D=1) сегмент стека, граница в байтах (G=0);(для прерываний в ЗР)
6. видеобуфер (0,0)

* охарактеризовать дескриптор сегмента дополнительной памяти, который описали

gdt\_data4gb descr <0FFFFh,0,0,92h,0CFh,0>

* + limit=fffff=2^32=4Гб
  + (G=1 (7 бит 6 байта attr2) граница в блоках по 4 Кбайт размер: 4 Гб
  + (D=1 (6 бит 6 байта attr2)32-разрядный по умолчанию
  + attr\_1=92h=10010010b:
    - 0 бит A -не интересует

1-3 тип:

* + - 3 бит=0 – сегмент данных/стека
    - 2 бит=0 сегмент данных
    - 1 бит=1 разрешены чтение и запись,
    - 4 бит S=1 является сегментом памяти
    - 5-6 DPL=0 - 0 уровень привилегий
    - 7 бит S=1 присутствует в памяти,
* почему таблица дескрипторов прерываний имеет такую структуру

Первые 32 элемента таблицы - под исключения-внутренние прерывания процессора (в 386 всего 19 исключений (0-19), остальные (20-31) зарезервированы, а *в 486 – и того меньше (реально-18, остальные-зарезервированы)*)

исключение 13 - нарушение общей защиты (нарушение, код ошибки-та команда) происходит: за пределами сегмента, запрет чтения, за гр. таблицы дескр., int с отс. Номером

32-255 – определяются пользователем (user defined)

Смещение=номер исключения\*8

Затем 16 векторов аппаратных прерываний (асинхронные),

* что написали для исключений

заглушки

* как адресуются аппаратные прерывания в ЗР (ударный)

Контроллер прерывания получает сигнал о прерывании и формирует вектор прерывания, который содержит селектор к ТДП.

Взяв значение из регистра IDTR значение базового адреса ТДП. В нём по селектору находим дескриптор который уже и содержит селектор, смещение и атрибуты

По селектору выбираем дескриптор из ТГД, берём оттуда базовый адрес сегмента и прибавляем его к смещению

Получаем линейный адрес точки входа

В ЗР для адресации прерывания имеется специальная таблица IDT. Если первые 32 исключения и мы возьмем 8, то попадем на double fault. Чтобы адресовать прерывания, надо перепрограммировать контроллер на: ведущий-на базовый вектор 32, и этот номер использовать для обращения к таблице. Базового адреса нет, есть смещение и селектор. Отдельно адресуются ведущий и ведомый.

В РР процессор использует вектор и таблицу векторов прерываний. У ведущего контроллера базовый вектор=8 (8+0=8h) и номер используется для получения смещения к адресу в таблице векторов прерываний. В DOS адрес наз. вектор (4 байт)

*Таким образом, обработчики для аппаратных прерываний должны начинаться минимум с 32-го (в винде принято начинать обработку прерываний со смещения 50h, но мы на это забьём, чтобы не пилить ещё дофига лишних обработчиков).*

* когда вывязывается обработчик от клавы? таймера?

Показать обработчик клавиатуры

Что этот обработчик анализирует? Скан коды, приходящие с клавиатуры.

Когда вызывается? Вызывается при нажатии или отжатии кнопки на клавиатуре.

Что мы можем использовать в своем обработчике прерываний от клавиатуры?Только порты, так как мы работаем с голым железом

Когда вызывается наш обработчик прерывания от системного таймера?

По тику (18.2 раза в секунду)

* какие действия необходимо выполнить для корректного возвращения в РР
  + запретить маскрируемые прерывания cli
  + сбросить влаг PE (0) в CR0-слово состояния машины
  + загрузить в используемые сегментные регистры адреса соответствующих сегментов к регистру CS недопустимо прямое програмнео обращение, поэтому юзаем переход
  + перепрограммировать ведущий контроллер на 8 обратно
  + восстановить маски ведущего и ведомого (из сохраненных)
  + загрузить idtr
  + закрыть линию A20
  + разрешить маскрируемые прерывания sti

что пишем в теневые регистры и почему.

после перехода в защ. режим прога не должна работать, т.к. в регстре CS ещё нет селектора сегмента команд и процесор не может обращаться к этому сгементу. в действительности это не совсем так. в процесоре для каждого из сегментных регистров имется так называемый теневой регистр дескриптор, который имеет формат дескриптора. теневые регистры недоступны программисту. они автоматически загружаются процесором из таблицы дескрипторов каждый раз, когда процесор инициализуерует соотвующий сегментрный регистр. таким образом в защ. режиме пр-мист имеет дело с селекторами, т.е. номерами дескрипторов, а процессор с самими дескрипторами, хранящимися в теневых регистрах. именно содержимое теневого регистра(в первую очередь линейный адрес сегмента) определяет область памяти, к которой обращается процесор при выполнении конкретной команды. после перехода в защ. режим прежде всего следует загрузить в используемые сегментные регистры селекторы соответствующих сегментов. это позволит процесору правильно заполнить все поля теневых регистров из таблиц дескрипторов. к регистру CS недопустимо прямое програмнео обращение, поэтому юзаем переход. при работе в реальном режиме некоторые поля теневых регистров должны быть заполнены определённым образом. граница ffffh, бит дробности 0, доступ для зписи разрешен. границы всех сегментов должны быть точно равны ffffh.перед переходом в реальный режим необходимо исправить все дескрипторы всех наших сегментов. линия а20 для обращения к расширеной памяти.

* что такое теневые регистры, для чего они включены в процессор, какую информацию содержат

С каждым сегментным регистром сопоставлен теневой регистр, в котором при обращении к сегментному регистру записывается информация из дескриптора. Цель – исключить обращение к GDT, которая находится в оперативной памяти. Оперативная память отстает по времени от проц., затратное действие (цикл обращения к памяти требует определенного количества тактов).

Обращение к ОП осуществляется на каждой команде, а то и несколько раз + надо сформировать (преобразовать) адрес. Чтобы цикл обращения к GDT для получения физического адреса, или данных, или следующей команды, информация с дескриптора записывается в теневой регистр, и после того, как было обращение к сегментному регистру.

Теневой регистр находится непосредственно в процессоре. Это не исключает обращение для считывания команд, записи и тд.

таким образом в защ. режиме пр-мист имеет дело с селекторами, т.е. номерами дескрипторов, а процессор с самими дескрипторами, хранящимися в теневых регистрах. именно содержимое теневого регистра (в первую очередь линейный адрес сегмента) определяет область памяти, к которой обращается процесор при выполнении конкретной команды

Процессоры начиная с 80386 начали поддерживать защищённый режим - для этого в них появились теневые регистры, вся адресация памяти теперь проходит через них. В РЕАЛЬНОМ режиме для обеспечения обратной совместимости при загрузке в CS,DS,SS адреса сегмента происходит неявное создание дескриптора сегмента. MOV AX,DataSeg MOV DS,AX Где-то на этом участке происходит определение размера сегмента ДатаСег, и создание его дескриптора (8 байт). После этого полученные 8 байт сразу же загружаются в теневой регистр (быдлоговоря, DS\_shad). В ЗАЩИЩЕННОМ режиме "основной" сегментный регистр для адресации используется не целиком: его младшие биты (0,1,2) содержат в себе флаги (за описаниями к Р. и Ф.). В качестве смещения используются биты 3..7. Другими словами, если в регистре лежит значение 8d (00001000b), то смещение == 00001b. При этом смещение всегда обязательно начинается с 1 - по смещению 0 лежит пустой дескриптор - его адрес используется для определения, а где же в памяти лежит сама GDT (Global Descriptor Table).

* Немного про линию А20

В защищенном режиме определяют порт линии А20 - 21-ая адресная линия. Компьютер стартует в реальном режиме. Связано это со стремлением хранить в энергонезависимой микросхеме ... (предполагаю что тут "как можно меньший объем информации"). ОС - программа, и пока компьютер выключен, она хранится во внешней памяти.

Когда компьютер стартует, линия А20 принудительно обнулена. При переходе в защищенный режим нужно открыть линию А20, то есть снять заземление.

* Зачем мы открываем линию А20?

Открываем, чтобы получить все адреса. Иначе у нас недоступны любые адреса, имеющие в 20 бите 1 - она обращается в ноль (линия заземлена, любой сигнал ноль)

открытие линии А20

Перед переходом в защищенный режим (или после перехода в нею) следует открыть линию А20, т.е. адресную линию, на которой устанавливается единичный уровень сигнала, если происходит обращение к мегабайтам адресною пространства с номерами l, 3, 5 и т.д. (первый мегабайт имеет номер 0). В реальном режиме линия А20 заблокирована, и есели значение адреса выходит за пределы FFFFFh, выполняется его циклическое оборачивание (линейный адрес lOOOOOh превращается в OOOOOh, адресс lOOOOlh в 00001h и т.д.). открытие (разблокирование) линии А20 выключает механизм циклическою оборачивания адреса, что позволяет адресоваться к расширенной nамяти.

Управление блокированием линии А20 осуществляется через порт 64h, куда сначала едедует послать команду Dlh управления линией А20, а затем - код открытия (DFh).За вентиль линии A20 отвечает 1-й разряд порта; остальные разряды изменять нельзя.

in al, 92h

or al, 2

out 92h, al

* Что будет, если при переходе в защищенный режим не откроем линию А20?

Нам не доступны адреса, в которых 20-ая адресная линия 1.

Если не включить линию А20, то 20ый бит всегда будет равен нулю. Если мы захотим обратиться к адресам, в которых этот бит равен единице, то мы не сможем получить к ним доступ. Мы получаем «битую память»»

* Можно ли её не закрывать при переходе в реальный режим? что произойдет, если при возвращении в РР забудем закрыть линию а20

Если в реальном режиме открыть линию А20, то в реальном режиме станет доступно еще 64 Кбайта памяти - HMA, high memory area.

закрытие линии A20 (если не закроем, то сможем адресовать еще 64кб памяти

in al, 70h

and al, 7Fh

out 70h, al

*Ничего страшного не произойдет, если мы так сделаем.*

* 64 мб это какая память?

память кот. мы выделили под вирт. машину. (Нам недоступна вся оперативная память.)

* В сегменте кода можно выполнять read - write?

Можно,

* а в стек можно делать r-w?

да

* что делает 13 dup? Выделяет, но как? Почему не выделяем подряд 13 дескрипторов, почему так написали? Дублируем
* могли бы 32 строки не выделять под исключения? Что нужно обеспечить, если мы отказываемся от этого?

Ну и запретить аппаратные прерывания видимо

перепрограммировать контроллер. Базовый вектор меняется с8 на 32 и обратно.

* Для этого что они перепрограммируют? в результате формируемый вектор прерываний является. Чем является базовый вектор прерываний?

смещение в табл. дескр. прер.

* «Чему кратно значение селектора?»

«8»

* «Что такое индекс?»

«Смещение в таблице дескрипторов.»

* «В реальном режиме есть привилегии?»

«Нет»

* «Как вы считаете объем оперативной памяти?»

Почему мы не можем использовать первый мегабайт памяти?

В первом мегабайте хранится операционная система ROM BIOS. Это зона ROM - read only memory. Если попытаться что-то в него записать, то возникнет ошибка.

-«У нас есть сегмент, объемом в 4 гигабайта. Считываем значение из ячейки памяти. Сохраняем его. Записываем в ячейку свое значение. Считываем. Если записанное и считанное значения совпадают, то это память.

Возвращаем в ячейку первоначальное значение. Если значение не совпадают, то пустота. Мы действуем до первого прокола»

* «Какой формат дескриптора?»
* «Какие характеристики имеет сегмент «памяти»?»
* «Почему вы в реальном режиме используете привилегированную команду lgdt?»

«Мы можем это делать, т.к. используем директиву .386»

ряд вопросов по коду, такие как

«Покажите где вы входите и выходите из защищенного режима», «Покажите какие сегменты вы описали»,

«Что вы делаете до входа и после входа в защищенный режим».

* Почему вдруг ваша программа стала выполнять функции ОС?

Потому что наша программа работает в защищенном режиме с нулевым уровнем привилегий.

* Есть у вас системные вызовы?

Нет.

* Если у вас нет системных вызовов, то какие средства вы используете для того, чтобы вводить и выводить символы?

Для этого мы используем бесконечный цикл. Для того, чтобы вывести символ мы напрямую обращаемся к видеопамяти, т.к. у нас нет ДОСовской команды вывода страницы.

* Почему существует деление на простые команды и привилегированные?

Привилегированные команды могут выполняться только с нулевым уровнем привилегий.

Это нужно для того, чтобы какая-нибудь программа не смогла нарушать работу системы, выполнив привилегированную команду.

* Что использую в обработчике прерываний от клавиатуры?

Порт клавиатуры

* В чём особенность реального режима?

особенность реального режима - ограничение объёма адресуемой оперативной памяти величиной 1мб.

перевод в защ. реж =

* увеличие адресуемого пространства до 4гб.
* возможность работать в виртуальном адресном пространстве.
* организация многозадачного режима с параллельным выполнением нескольких программ(процессов)
* страничная организация памяти, повыщающая уровень защиты задач друг от друга.
* в защищённом режиме процесор выполняет процедуру перрывания не так как в реальном. при поступлении сигнала прерывания процессор не обращается к таблице векторов прерываний в первом кб памяти, как в реальном режиме, а извлекает адрес программы обработки прерывания из таблицы дескрипторов прерываний, построенной схоже с ТГД.
* Привилегированные команды.

тридцати двухразрядные микропроцессоры отличаются расширенным набором команд, часть которых относится к привилегированным. для того чтобы разрешить транслятору обрабатывать эти команды в текс включена директива .386Р

Просто кусок информации

На процессорном уровне поддерживаются уровни привелегий 0..3; ОС запускается с уровнем 0, приложения с 3. Для каждого сегмента памяти устанавливается уровень привелегий, необходимый для доступа. Существует три режима работы процессора: реальный, защищенный и виртуальный-86. Процессор всегда запускается в реальном режиме и выполняет код биоса. После этого он путём плясок с бубном и скакания на [s]майдане[/s] костылях переходит в защищённый.

Реальный режим: однозадачный, 16битные регистры, поддерживает до 1 мегабайта памяти. НЕ ПОДДЕРЖИВАЕТ разделение доступа (^ уровни 0..3) и виртуальную адресацию памяти. Защищенный - умеет в наоборот: 32битные регистры, многозадачность, до 4 ГБ виртуальной памяти. ПОДДЕРЖИВАЕТ уровни доступа (кольца защиты) и виртуальную адресацию. В защищенном режиме добавлены несколько дополнительных регистров. Сегментные: FS, GS; управляющие: CR0, CR1, CR2, CR3. Все они, соответственно, 32-разрядные.

Управляющие регистры CR2 и CR3 используются для страничного преобразования, благодаря которому работает техномагия виртуальных адресов. В CR0 лежит несколько флагов, которые управляют поведением процессора - например, последний (31 (не забываем что нумерация у нас с нуля)) бит Protected отвечает за режим, в котором находится процессор - защищённый если ==1 и реальный если ==0. CR1 зарезервирован для будущих поколений процессоров.

Сегменты памяти описываются дескрипторами по 8 байт.

База - адрес, с которого начинается данный сегмент. Повторюсь: адрес в виртуальном адресном пространстве. Вообще, все упоминаемые здесь и далее адреса упоминаются в контексте виртуальности; к физическим адресам мы доступа не имеем.

Кроме того, используется также таблица дескрипторов прерываний, содержащая в себе следующие структуры: int\_descr struc offs\_l dw 0 ;смещение в сегменте, нижняя часть sel dw 0 ;селектор сегмента с кодом прерывания counter db 0 ;счётчик, не используется в программе attr db 0 ;атрибуты offs\_h dw 0 ;смещение в сегменте, верхняя часть int\_descr ends Обработчики 0..16 зарезервированы под прерывания и исключения системы; 17..31 - под "будущие поколения процессоров"; остальные могут быть использованы пользователем. Таким образом, обработчики для аппаратных прерываний должны начинаться минимум с 32-го (в винде принято начинать обработку прерываний со смещения 50h, но мы на это забьём, чтобы не пилить ещё дофига лишних обработчиков).

* Нужны ли 0FFFFh? Работает ли без этого?

Они не нужны так как не нужно т к не выходим за 1МБ ffff

* С помощью какой функции выводили строку в реальном режиме

Для вывода сообщения используется 9 функция 21 прерывания. (INT 21H: сервис DOS)

* Что делали с базовым вектором при переходе в защищенный режим и обратно?

В реальном режиме наш базовый вектор 8, в защищенном 32, при переходе в другой режим мы перепрограммируем контроллер прерываний.

* Каким образом мы перепрограммируем контроллер прерываний? Почему надо перепрограммировать контроллер прерываний?

Мы смещаем базовый вектор контроллера прерываний на 32. Потому что теперь прерывания имеют базовый вектор не 8, а 32.

перепрограммирование ведущего контроллера, т.к. в ЗР первые 32 вектора зарезервированны для обработки исключений, аппаратным прерываниям нужно назначить другие векторы 32=20h Для смены базового веткора требуется полностью выполнить процедуру инициализации контр., которая состоит из ряда команд инициализации СКИ

mov al, 11h ;СКИ1: два контроллера в компьютере, будет СКИ3

out 20h, al

mov al, 32 ;СКИ2: базовый вектор (был 8)

out 21h, al

mov al, 4 ;СКИ3: ведомый подключен к уровню 2

out 21h, al

mov al, 1 ;СКИ4:8086, требуется EOI программно

out 21h, al

В чем особенность 32разрядного сегмента данных?

4 гига

для чего? Для того, чтобы посчитать память

"Основной ответ - размер 4 гб Показываешь ей 5 эфок И говоришь про бит гранулярнсти и дигит"

"Ну есть особенность, что там, например, offset чего-то будет не 2, а 4 байта. И с циклами будет работать не CX, а ECX".

"У него установлен максимальный лимит

* В каком режиме мы в сегментные регистры записываем селекторы и для чего?

В защищенном.

* Что происходит при загрузке селектора в сегментный регистр?

теневой регистр, связанный с сегментным регистром, обновляются значением линейного адреса сегмента

* Что мы делаем с масками прерываний?

Маски - перед переходом в защищенный режим у контроллера ведущего и ведомого есть свои маски. Мы их сохраняем, чтоб потом восстановить. Мы их сохраняем, потому что мы ставим свои - теряем информацию о том что было. Потом их надо вернуть.

# На черный день советы от старосты четвертой группы

Главный вопрос: Опишите переход в реальный режим

1. Устанавливаем флаг перех. в рр
2. Запрещаем маскируемые прерывания
3. Переходим в реальный
4. Через команду far jmp, заданную прямо кодом, обновляем значение в теневом регистре, связанном с CS
5. Обновляем остальные теневые регистры значениями
6. Возвращаем маски контроллерам прерываний, возращаем значение базового вектора прерывания (8, не 32)
7. Возвращаем базовый линейный адрес таблице векторов прерываний
8. Разрешаем немаскируемые
9. Разрешаем маскируемые
10. Печатаем сообщение с помощью функций Dos
11. Выходим через функцию DOS (ред.) Через что взаимодействуем с клавиатурой? — через порты (ред.) Что за память на экране? — доступная программе dosbox память

* Почему делим? Потому что в мегабайтах (ред.)
* Зачем сегмент стека в защищенном? — для использования прерываний (ред.)
* Что происходит при загрузке селектора в сегментный регистр? — теневой регистр, связанный с сегментным регистром, обновляются значением линейного адреса сегмента

A20 и теневые регистры, доп вопросики