Страничная адресация

Если с сегментной проблем не возникло, страничная освоится так же легко.

Прежде всего, нужно понять очень важную вещь: при использовании страничной адресации структуры из сегментной адресации (как то – таблицы дескрипторов, селекторы, регистры таблиц дескрипторов) никуда не деваются. Все остается на своих местах. Так в чем же тогда заключается страничная адресация? Где же она себя проявляет?

Важный момент: единственное место, где страничная адресация действительно вклинивается в процесс работы процессора – при переводе линейного адреса в физический.

Придется немного прокрутить пленку назад. Вспомним три понятия: логический, линейный и физический адрес. Логический адрес – это некий абстрактный, на деле ничего не значащий адрес, грубо говоря, CS:EIP – это и есть логический адрес, в самом деле, что он может нам сообщить? Ничего. Только то, что если мы вытащим из CS поле индекс, а затем по этому полю найдем в таблице дескрипторов соответствующий дескриптор, а затем к базе из этого дескриптора прибавим EIP – то вот только тогда получим линейный адрес. При использовании сегментной адресации физический адрес совпадает с линейным (физический – это адрес, который процессор выставляет уже на адресную шину). Т.е. при сегментной адресации процессор просто берет линейный адрес и без изменений выставляет на адресную шину. При использовании страничной адресации именно на этапе перевода линейного адреса в физический в действие вступают новые преобразования.

При использовании страничной адресации линейный адрес не совпадает с физическим, как в случае с сегментной адресацией. Т.е. мы имеем дело с виртуальной памятью. Процессор делит линейное адресное пространство на страницы фиксированного размера (длиной 4Кб, 2Мб или 4Мб), которые, в свою очередь, уже отображаются в физической памяти (или на диске). Когда программа (или задача) обращается к памяти через логический адрес, процессор переводит его в линейный и затем, используя механизму страничной адресации, переводит его в соответствующий физический адрес. Если страницы в данный момент нет в физической памяти, то возникает исключение #PF. Это, по сути, кульминационный момент: обработчик этого исключения (#PF) должен выполнить соответствующие манипуляции по устранению данной проблемы, т.е. подгрузить страницу с жесткого диска (или наоборот – скинуть ненужную страницу на диск).

По сути - #PF это не есть нечто ужасное и недопустимое (как #GP). Наоборот, без него вообще ничего бы и не получилось по сути дела.

Как только страница была благополучно водворена на место, то выполнение прерванной программы продолжается с той самой инструкции, которая вызвала #PF. А кстати, какая инструкция может вызвать #PF? Да в принципе любая, которая так или иначе обращается к памяти...

Страничная организация отличается от сегментной еще и тем, что все страницы имеют фиксированный размер (в сегментной размер сегментов абсолютно произволен). Также при сегментной адресации все сегменты обязательно должны присутствовать непосредственно в оперативной памяти, а при страничной возможна ситуация, когда кусок сегмента находится в памяти, а другой кусок фактически того же сегмента – на жестком диске (т.е. другими словами – часть страниц сегмента находится в памяти, а часть – валяется в то же время на диске).

Процессор всегда, где это возможно, облегчает себе жизнь. Вспомним про «теневую» часть сегментного регистра. Также и в случае со страничной адресацией – страницы, к которым процессор чаще всего обращается, кэшируются в процессоре, в области, которая называется буфер с ассоциативной выборкой (TLB - Table lookaside buffer). Т.е. не все такие страницы целиком, а только записи, которые необходимы для доступа к ним.

Итак, страничная организация непосредственно в процессоре управляется тремя флажками:

Флаг PG (paging): бит 31 в регистре CR0. Появился в 386 процессоре.

Флаг PSE (page size extensions): бит 4 в регистре CR4. Появился в Pentium.

PAE (physical address extension) flag: бит 5 в регистре CR4. Появился в pentium pro.

Теперь подробнее. Флаг PG разрешает страничную адресацию. Сразу после установки его в единицу страничная адресация включена.

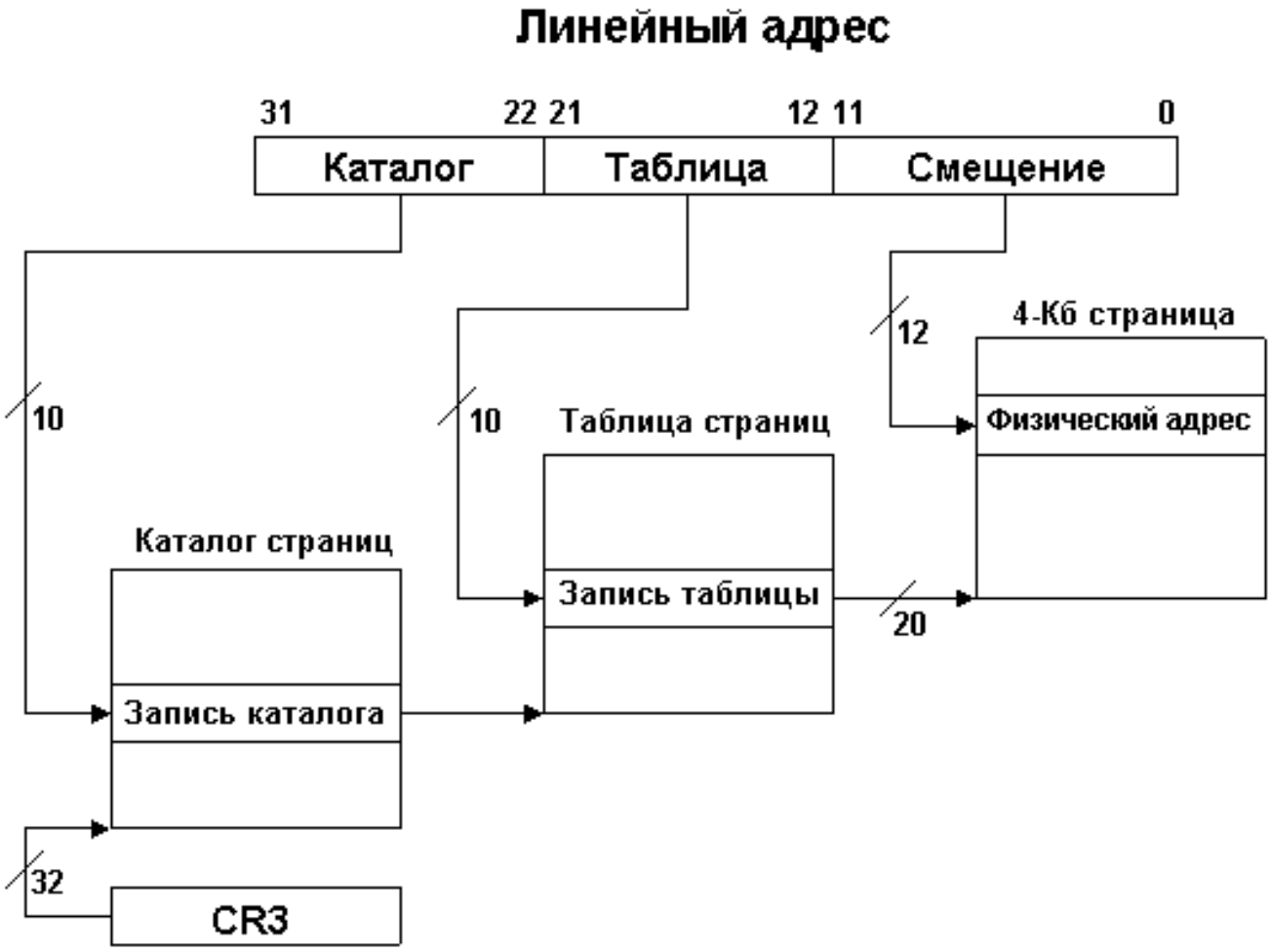
Флаг PSE, если его установить, позволяет использовать страницы больших размеров (4Мб и 2Мб). Если сброшен – страницы имеют размер 4Кб.

Флаг PAE позволяет расширить физический адрес до 36 бит (стандартно он 32-х битный). Данный флаг можно использовать только в режиме страничной адресации.

Кстати, использовать 36-разрядную адресацию можно как с помощью PAE-флага, так и с помощью PSE-флага, это два разных метода, мы их рассмотрим дальше.
Каталоги и таблицы страниц.
При трансляции линейного адреса в физический (при включенной страничной адресации) процессор использует 4 структуры данных:
1. Каталог страниц – массив 32-битных записей (PDE – page-directory entry), которые хранятся в 4Кб странице. Напоминает таблицу дескрипторов. В 4 Кб 32-битных записи помещается 1024 штуки, значит, всего 1024 PDE.
2. Таблица страниц – массив 32-битных записей (PTE – page-table entry), которые также все расположены в одной 4Кб странице. Т.е. PTE-шек тоже может быть всего 1024 штуки. Забегая вперед – для 2Мб и 4Мб страниц таблица страниц вообще никак не используется – все решают только PDE-шки.
3. Сама страница – 4Кб, 2Мб или 4Мб кусок памяти :)
4. Указатель на каталог страниц – массив 64-битных записей, каждая из которых указывает на каталог страниц. Эта структура данных используется процессором только при использовании 36-битной адресации.
Все эти таблицы позволяют обращаться к 4Кб или к 4Мб страницам при 32-х битной адресации и к 4Кб, 2Мб или 4Мб страницам при 36-битной адресации.
Смотрим на таблицу, какие флаги на что влияют:

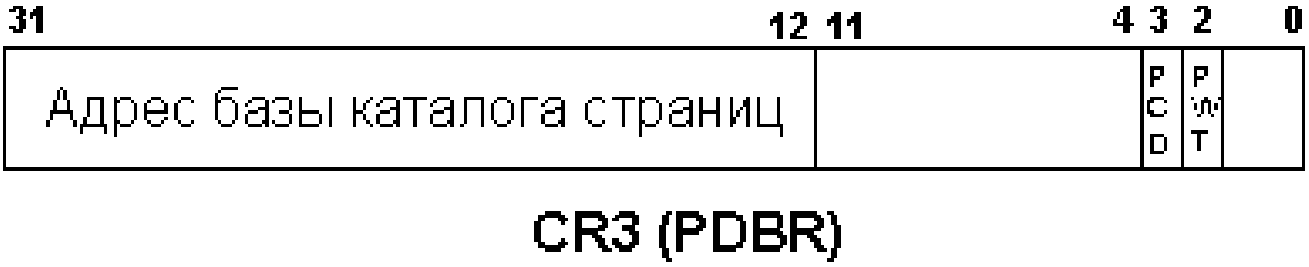
PG (CR0)	PAE (CR4)	PSE (CR4)	PS (PDE)	PSE-36 (CPUID)	Размер старницы	Размерность физического адреса (бит)
0	X	X	X	X	-	-
1	0	0	X	X	4 Кб	32
1	0	1	0	X	4 Кб	32
1	0	1	1	0	4 Мб	32
1	0	1	1	1	4 Мб	36
1	1	X	0	X	4 Кб	36
1	1	X	1	X	2 Мб	36

Непонятно тут может быть только одно – что за колонка PSE-36? Дело в том, что этот режим работы (PSE-36) появился только в Pentium 3, а доступен он или нет, можно узнать посредством CPUID. При использовании PSE-36 механизма доступны только страницы размером 4Мб. Фактически таким образом можно адресовать 64 Gb физического адресного пространства.
Конечно, на первый взгляд получается несколько туманно и путано, но на самом деле все довольно четко и логично.
В общем, расклад таков: при страничной организации существуют 3 способа адресации: 32-х разрядная, 36-разрядная с использованием флага PAE и 36-разрядная с использованием флага PSE.
Для начала рассмотрим самую простую и наглядную – 32-х разрядную адресацию. Все, о чем пойдет дальше речь – справедливо только для нее.
Линейная адресная трансляция (4Кб страницы)



Линейный адрес (тот самый, который получается в результате сложения базы сегмента со смещением при сегментной организации) при страничной организации с 4Кб страницами делится на три части:
Номер записи в каталоге страниц: биты 22-31.
Номер записи в таблице страниц: биты 12-21.
Смещение в самой 4Кб странице: биты 0-11.
Для справки: в регистре CR3 (см. рисунок) только биты 12-31 содержат смещение каталога страниц в памяти. Младшие 12 всегда равны нулю. Т.е. расположить каталог страниц ниже 4 Кб не удастся.
На самом деле, может создаться ошибочное мнение, что каталог страниц – один единственный, и указывает на него регистр CR3 (как в случае с глобальной таблицей дескрипторов и регистром GDTR). На самом же деле все обстоит совсем не так, каталогов страниц может быть несколько, даже у каждой задачи свой, а регистр CR3 меняется после переключения задачи.
Линейная адресная трансляция (4Мб страницы)
Для 4Мб страниц никакой таблицы страниц не существует, всё решает каталог:

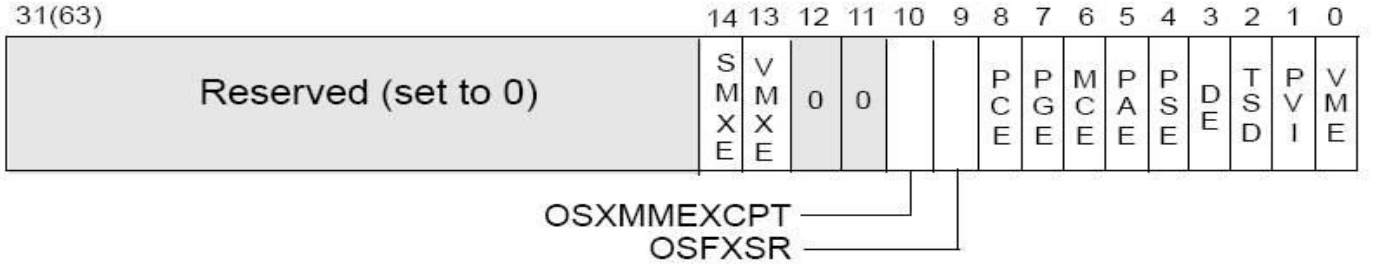
Фактически за счет размера страниц мы ничего не теряем и опять же можем адресовать 4Gb, как и в случае с 4Кб страницами. 4Мб размер страницы получается в случае установки флагов PSE и PS. Флаг PSE находится в CR4, а что за флаг PS? А флаг PS находится непосредственно в записи каталога страниц, о которой позже.
В случае 4Мб страниц линейный адрес делится на ДВЕ части:
Номер записи в каталоге страниц (биты 22-31)
Смещение в самой 4Мб-странице (биты 0-21)
РЕГИСТР CR3



Еще раз: младшие 12 бит все время равны нулю.

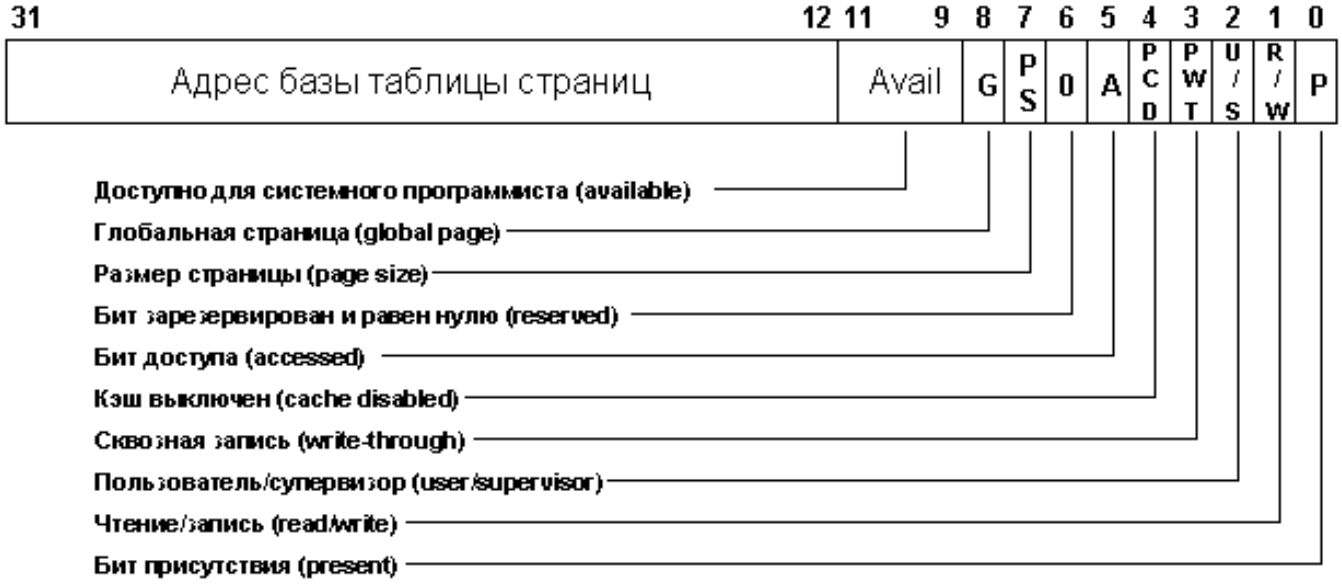
PCD-флаг (page level cache disabled) – контролирует кэширование каталога страниц. Если он установлен, то кэширование не происходит. Если сброшен – каталог страниц может кэшироваться. Этот флаг влияет только на внутренние кэши процессора. Процессор игнорирует этот флаг, если

страничная адресация отключена (флаг PG в CR0 сброшен) или если вообще отключен кэш (флаг CD в CR0).
Регистр CR3 (главным образом, хранящий начало каталога страниц текущей задачи) также называют PDBR (page directory base register). Если мы собираемся использовать страничную адресацию, мы должны загрузить этот регистр в процессе инициализации. Загрузить его можно либо непосредственно командой mov, либо он сам загрузиться при переключении задач.
Может возникнуть ситуация, когда каталог страниц в момент переключения отсутствует в физической памяти, поэтому ОС перед переключением задачи должна устранить это недоразумение. Каталог страниц должен находится в памяти столько времени, сколько активна задача, использующая этот каталог страниц.

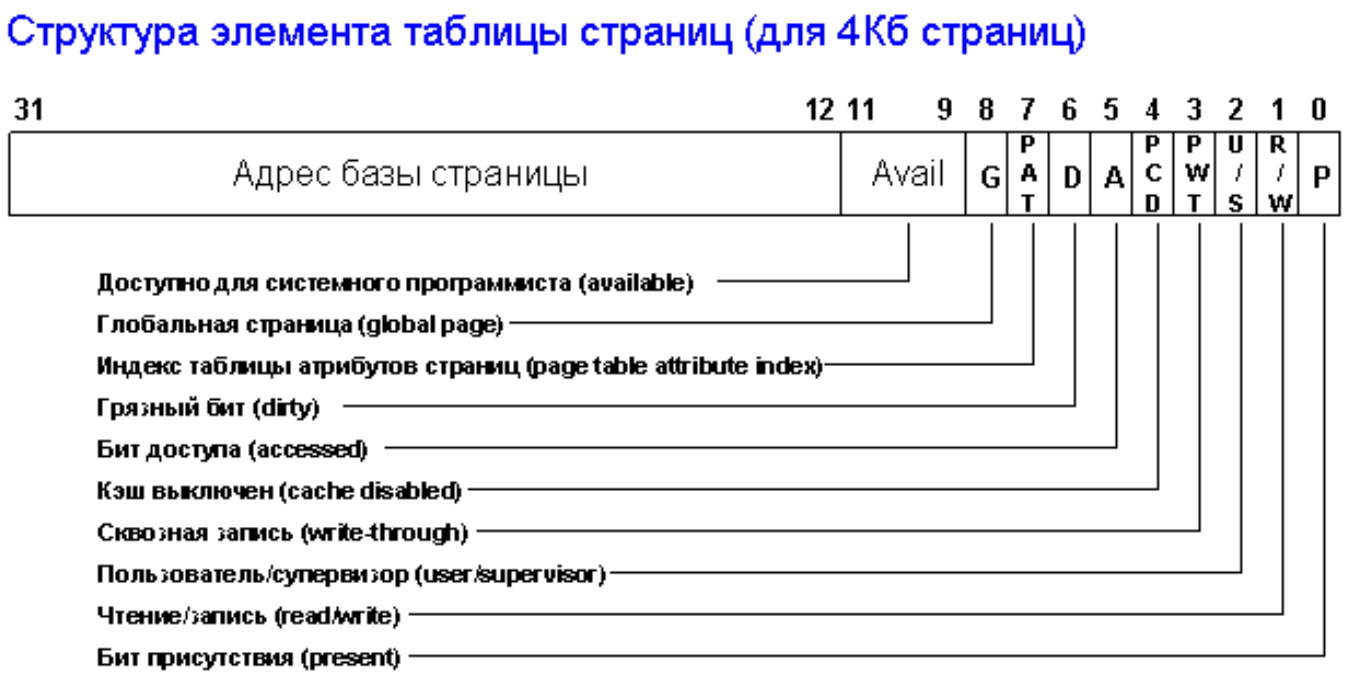


Элементы каталогов и таблиц страниц
Напомню, что при страничной организации линейный адрес (база из дескриптора + offset) не соответствует физическому (т.е. адресу, который процессор выставляет на адресную шину). Прежде, чем попасть на адресную шину он проходит ряд преобразований.
Итак, линейный адрес делится на три части в случае 4Кб страниц, и на две в случае 4Мб страниц.
Рассмотрим случай с 4Кб страницами.
Биты 22-31 линейного адреса – это номер элемента (записи) в каталоге страниц. Адрес начала каталога страниц в памяти содержит регистр CR3. Каждый элемент в каталоге страниц «весит» 32 бита. Рассмотрим его.

Структура элемента каталога страниц (для 4Кб страниц)



Поле «Адрес базы таблицы страниц» содержит адрес базы таблицы страниц. Т.е. не адрес элемента из таблицы страниц, а только адрес начала таблицы этих элементов. Сам элемент (его номер в таблице страниц) определяют биты 12-21 линейного адреса. Вот его структура:



P-флаг, бит 0: указывает, находится ли страница (или таблица страниц) в физической памяти. Если он равен 1 – то все в порядке, страница в памяти и можно продолжать формирование физического адреса, если же он сброшен – процессор вызывает исключение #PF (Page Fault), обработчик которого должен загрузить ее в память. Вообще-то процессор никогда не меняет этот флаг сам – этим должна заниматься ОС.
R/W-флаг, бит 1: определяет привилегии чтения/записи для страницы или группы страниц (в случае, когда элемент каталога страниц указывает на таблицу страниц). Если флаг сброшен – страница доступна только для чтения. Если установлен – для чтения и записи. Данный флаг тесно связан с флагами U/S и WP из CR0.
U/S-флаг, бит 2: определяет привилегии пользователя/супервизора для страницы или группы страниц. Если флаг сброшен – страница доступна только для супервизорского уровня привилегий, если установлен – для пользователя и супервизора. Данный флаг связан с флагами U/S и WP.
PWT-флаг, бит 3: контролирует кэширование страницы (write-through и write-back). В зависимости от того, установлен он или сброшен, доступны разные типы кэширования. Процессор игнорирует этот флаг, если установлен CD (cache disabled) флаг в CR0.
PCD-флаг, бит 4: контролирует кэширование страницы. Если установлен – кэширование страницы отключено, если сброшен – разрешено. Нужен для страниц, которые, по сути, могут только засорить кэш и не нужны для частого использования. Также, как и с PWT-флагом, значение PCD-флага игнорируется, если установлен CD-флаг в CR0.
A-флаг, бит 5: (аналог флага A в дескрипторе) показывает, было ли произведено обращение к странице с момента загрузки ее в память. По идее, ОС при загрузке страницы в память в первый раз должна сбросить этот бит в ноль. Затем, после ее загрузки в память, процессор отлавливает момент, когда к ней в первый раз произойдет обращение и сам устанавливает этот флаг в единицу. И все, больше он его не трогает. Флаг A (также как и D) нужны для управления и контроля за страницами и таблицами страниц.
D-флаг, бит 6: показывает была ли произведена запись в страницу. Этот флажок игнорируется в элементах каталога страниц, которые указывают на таблицы страниц, т.е. при 4Кб-страничной организации. ОС должна обнулить данный флаг как только страница загружена в память, затем процессор уже сам установит его, как только в страницу будет произведена первая запись и больше его трогать не будет. Т.е. он будет торчать до тех пор, пока его не скинет ОС. Вообще данный флаг по сути схож с флагом A.
PS-флаг, бит 7 (для 4Кб страниц): определяет размер страницы. Если сброшен – страница 4Кб и элемент каталога страниц указывает на таблицу страниц. Если установлен – страница 4Мб (при 32-х

битной адресации) или 2Мб (если используется EPA) и элемент каталога страниц указывает на саму страницу. Если элемент каталога страниц указывает на таблицу страниц, то все страницы, на которые ссылаются элементы таблицы страниц также являются 4Кб-ными.
G-флаг, бит 8: появился в Pentium Pro процессорах. Указывает, что описываемая страница является глобальной, если установлен. Если данный флаг установлен, а также установлен PGE-флаг в CR4, то соответствующие элементы каталога и таблицы страниц никогда не будут выкинуты из кэша. Страницы, содержащие код ядра ОС и т.п. желательно помечать глобальными, тогда их никто не посмеет выкинуть из кэша. Флаг G можно установить/сбросить только программно. Для элементов каталога, которые указывают на таблицу страниц, этот бит игнорируется, зато точно такой же бит есть и в соотв. элементе таблицы страниц, вот там процессор на него уже смотрит.
Зарезервированные и доступные для программ биты: существуют во всех IA-32 процах. Биты 9,10 и 11 доступны для использования программно. Если бит P (бит присутствия сегмента) сброшен, то доступны все 32 бита. Для элемента каталога страниц, который указывает на таблицу страниц, бит 6 должен всегда равняться нулю. Если PSE и PAE флаги в CR4 – установлены, то процессор будет кидать исключение #PF, если зарезервированные биты не сброшены в нули.
Защита
Самый главный вопрос, от которого и нужно отталкиваться - так что же от чего все таки нужно защищать?
Для того чтобы понять суть происходящего обязательно нужно взглянуть на дело со стороны. Представим картину: на столе стоит компьютер, из колонок играет музыка, на экране открыто несколько вкладок браузера, где-то в углу досматривается фильм (с выключенным звуком естественно, чтоб не мешать музыке), пара документов свернута в панель задач. И тут возникает вопрос: как все это может сосуществовать в одном ящике? Как процессор обрабатывает такой огромный поток данных и успевает все вовремя просчитать и не перепутать ни одного байта? Все это обеспечивается благодаря механизму защиты.
Как можно убедиться на практике, связать аппаратные механизмы защиты с программными разработчикам из Microsoft (по крайней мере до Win2000) удавалось крайне плохо: в 99% "синих окон смерти" повинны дыры и недоработки именно в реализации защитных механизмов. Отчасти это вызвано кривыми руками разработчиков ОС, отчасти - кривой реализацией аппаратной защиты (все любят спорить о глючности той или иной ОС, и никто никогда не скажет: "да этот процессор глючный, в нем много дыр и недоработок"). Просто следует иметь ввиду, что аппаратная часть обычно не лишена недостатков программной.
В процессорах с архитектурой IA-32 защитные механизмы реализованы как на сегментном, так и на страничном уровнях (т.е. после включения страничной адресации нам предоставляются некоторые дополнительные возможности защиты, которые, вообще говоря, без страничной адресации не имеет смысла применять). Процессор различает четыре уровня сегментной защиты (3-0) и два - страничной (здесь номеров нет, но условно их два). Чем уровень ниже (численно), тем сегмент круче. Так, весь код ядра ОС, дрова устройств и подобное помещают в сегменты с нулевым (самый крутой) уровнем привилегий.
Механизм защиты обеспечивает проверку каждого обращения к памяти (естественно, до цикла чтения/записи). Любое нарушение границ и других предписанных правил - генерация исключения (что может привести к любым последствиям - вплоть до убийства приложения-нарушителя).
Следует отметить, что все проверки производятся параллельно с процедурой преобразования адресов, поэтому задержка на проверку отсутствует. Условно проверки можно разделить на 6 разделов:
Проверка границ
Проверка типов
Проверка уровней привилегий
Ограничения на адресацию
Ограничения на использование инструкций
Ограничения на точки входа в процедуры
Как уже говорилось ранее, нарушение любой проверки влечет генерацию исключения и в зависимости от степени нарушения - соответствующие последствия для провинившегося. В ОС типа Win 9x любого нарушителя ждала одна и та же участь - расстрел на месте. В Win NT нарушитель в зависимости от степени нарушения может отделаться "предупреждением" либо небольшой задержкой на продолжение работы.

Очень важно понять: процессор не принимает решения самостоятельно, он просто констатирует уже свершившиеся факты, процессор не может убить программу (насильно завершить ее выполнение) - все решения принимает программная реализация. Принимает решение обработчик исключения #GP и других исключений. Все зависит только от их реализации.

ОБЩИЕ ПОЛОЖЕНИЯ

В это трудно поверить, но весь механизм защиты процессора Intel обеспечивается 3-мя флагами (+ еще 2 дополнительных при использовании страничной адресации) и 5-тью полями. Казалось бы, какую защиту могут обеспечить эти 8 сущностей? На самом же деле все довольно продумано и все чего надо они обеспечивают.

Флаги, обеспечивающие защитный механизм:

Тип дескриптора (S) - бит 12 во втором двойном слове дескриптора сегмента. Собственно, отвечает за тип сегмента: системный, кода или данных.

Флаг гранулярности (G) - бит 23 во втором двойном слове дескриптора сегмента. Вместе с полем "Лимит" дескриптора и флагом E (флаг направления роста) определяют размер сегмента.

Флаг направления роста сегмента (E) - бит 10 во втором двойном слове дескриптора сегмента. Работает на пару с флагом G и полем "Лимит".

Флаг пользователь/супервизор (U/S) - бит 2 элемента таблицы или каталога страниц. Определяет тип страницы: пользовательская или супервизорная. В прошлом выпуске упоминалось, что при использовании страничной адресации доступны некоторые дополнительные защитные механизмы. Этот флаг - один из них.

Флаг чтения/записи (R/W) - бит 1 элемента таблицы или каталога страниц. Определяют тип доступа к странице: только для чтения или для чтения/записи. Флаг R/W - второй (он же последний) дополнительный механизм защиты при использовании страничной адресации.

Поля, обеспечивающие защитный механизм: (просто запомните их)

Поле "Тип" - биты 8-11 во втором двойном слове дескриптора сегмента. Определяет тип сегмента кода, данных или системного сегмента. В принципе, работает на пару с флагом S (см. выше).

Поле "Лимит" - биты 0-15 первого двойного слова и биты 16-19 второго двойного слова дескриптора сегмента. Вмести с флагом G и E определят размер сегмента. С этим полем все ясно.

Поле "Уровень привилегий дескриптора" (DPL) - биты 13 и 14 во втором двойном слове дескриптора сегмента. Название поля немного сбивает с толку - на самом деле поле DPL задает уровень привилегий описываемого им сегмента, дескриптору уровень привилегий, по сути, ни к чему. Чем меньшее значение содержит это поле, тем описываемый сегмент круче (0 - самый крутой, 3 – наименее привилегированный). В чем именно «крутость»? Сегмент с нулевым уровнем привилегий может исполнять любые инструкции, обращаться к любым данным. По задумке, в сегментах с 0-ым уровнем должно располагаться ядро ОС, другие части ОС - в сегментах с 1-ым уровнем, драйвера - со 2-ым, пользовательские программы - с 3-им. На деле программисты из Microsoft при проектировании ОС Windows решили "упростить" модель до двухуровневой: все что не пользовательские программы - то в 0-ом уровне, пользовательские - в третьем. 1-й и 2-й уровни не используются. По сути, если бы Intel IA32 эксплуатировался только под осью Windows (что и происходит в 90% случаях), то для полей DPL, RPL и CPL с головой хватило бы одного бита вместо двух, и инженеры Intel-а немного перестарались.

Поле "Запрашиваемый уровень привилегий" (RPL) - биты 0 и 1 любого сегментного селектора.

Поле "Текущий уровень привилегий" (CPL) - биты 0 и 1 сегментного регистра CS. Фактически определяет уровень привилегий текущей исполняемой программы (процедуры), в общем - уровень привилегий исполняемого в данный кода.

Вся аппаратная защита такого монстра, как, скажем, Pentium 4, целиком и полностью стоит на описанных выше битах. Всего 35 бит, т.е. все аппаратные защитные структуры умещаются (почти) в одно двойное слово.

НАРУШЕНИЕ ГРАНИЦ

Поле "Лимит" дескриптора сегмента призвано уберечь программы (процедуры) от незаконного обращения к памяти за пределами, установленными этим полем. Эффективное (реальное) значение лимита зависит от флага гранулярности G.

Влияние бита G на поле "Лимит" изучено и рассмотрено со всех сторон в предыдущей лекции, но напомню: если бит G=0, то лимит сегмента совпадает со значением одноименного поля дескриптора; очевидно, что в данном случае сегмент может иметь длину от 0 до FFFFFh (1Мб). Если флаг G=1, то

реальный лимит сегмента равен содержимому поля "Лимит" дескриптора, умноженному на 4Кб (FFFh). Очевидно, сегмент в этом случае может занимать от 4Кб до 4Гб.
Если бит G=1, то младшие 12 бит адреса не проверяются на превышение лимита. Т.е. если создать дескриптор сегмента с G=1, и полем "Лимит" равным 0, то смещения от 0 до FFFh все равно доступны для обращения. Т.е. процессор игнорирует младшие 12 бит при проверке на превышение лимита. Что еще хорошо знать - умножение на 4Кб равносильно сдвигу на 12 влево
Итак, процессор сгенерирует исключение #GP в следующих случаях:
Обращение к байту по смещению БОЛЬШЕМУ, чем эффективный лимит
Обращение к слову по смещению БОЛЬШЕМУ, чем (эффективный лимит-1)
Обращение к двойному слову по смещению БОЛЬШЕМУ, чем (эффективный лимит-3)
Обращение к учетверенному слову по смещению БОЛЬШЕМУ, чем (эффективный лимит-7)
Существует еще один размытый момент, который мы сейчас обсудим во всех нюансах. Этот момент - растущие "вниз" сегменты данных. Тут сразу нужно сказать - сегмент никуда не растет, это образное выражение. Все дело в том, что у растущих "вниз" сегментов поле "Лимит" в дескрипторе определяет нижнюю границу сегмента, а верхняя граница всегда равна FFFFh (1Мб) если бит B=0 и FFFFFFFFh (4Гб) если бит B=1. Откуда взялся бит B? Вспоминаем структуру дескриптора, помните флаг D/B? Вот это он и есть. Теперь возникает вопрос: если у нас определена нижняя граница сегмента (поле "Лимит") и верхняя (1Мб или 4Гб), то зачем тогда поле база в дескрипторе? Поле "Лимит" хоть и определяет нижнюю границу, но все равно отсчитывается от базы. Т.е. фактически нижняя граница у растущих вниз сегментов равна: содержимое поля "Лимит" дескриптора + содержимое поля "База" дескриптора.
Еще раз нелишне повторить, что суть проверок лимитов заключается в том, чтобы не дать программам разгуляться по всей памяти, они не должны вылазить за отведенные им границы.
Процессор (помимо проверки лимитов сегментов) проверяет также лимиты таблиц дескрипторов, инфа о которых содержится в регистрах GDTR, IDTR, LDTR и TR. #GP будет сгенерировано, если произойдет попытка обращения к дескриптору, лежащему за пределами таблицы дескрипторов.
ПРОВЕРКА ТИПА СЕГМЕНТА
Это небольшой и простой раздел. Вспомним структуру дескриптора. Здесь нас интересуют два поля: бит S (бит 12 второго двойного слова) и поле «Тип» (биты 8-11 второго двойного слова).
Итак, процессор проверят эти два поля в следующих двух случаях:
При загрузке селектора в сегментный регистр
При обращении к сегменту
#GP возникнет если:
в регистр CS загрузить селектор на дескриптор, который описывает отличный от сегмента кода сегмент;
в регистры DS,ES,FS или GS загрузить селектор на дескриптор, который описывает «НЕЧИТАЕМЫЙ» сегмент кода;
в регистр SS загрузить селектор на дескриптор, описывающий отличный от «ЧИТАЕМЫЙ сегмент ДАННЫХ» сегмент;
в регистр LDTR загрузить селектор, не указывающий на LDT;
при записи в «ИСПОЛНЯЕМЫЙ» сегмент (т.е. писать в сегмент кода нельзя через CS, через DS - можно);
при записи в сегмент данных, который «ТОЛЬКО ДЛЯ ЧТЕНИЯ»;
при чтении из «НЕЧИТАБЕЛЬНОГО ИСПОЛНИМОГО» сегмента;
дескрипторы в таблице IDT должны быть шлюзами прерывания, ловушки или задачи;
Бит S и поле «Тип» в дескрипторе говорят процессору, чего с сегментом делать можно, а чего нельзя. Если кто-то попытается сделать то, чего делать нельзя, то обработчик #GP (который мы сами и пишем) должен предпринять необходимые действия.
НУЛЕВОЙ СЕЛЕКТОР
Если в CS или SS загрузить селектор, который указывает на нулевой дескриптор (т.е. 16-разрядное число, биты 15-3 которого равны 0), то #GP возникнет немедленно;
Если нулевой селектор загрузить в DS,ES,FS или GS то #GP возникнет при попытке обращения к данным через такой селектор (иначе – он там может хоть сто лет пролежать). Вообще нулевой селектор придуман в качестве подстраховки от ненужных обращений к сегментам.

УРОВНИ ПРИВИЛЕГИЙ

Иногда (в частности – в ОС Windows) их называют кольца защиты (Protection Rings). Как уже было сказано – всего уровней привилегий 4: 0 – самый крутой, 3 – наименее привилегированный. В мире ОС в нулевом кольце располагается код ядра ОС, драйвера, обработчики исключений (тот же #GP). Хотя по задумке инженеров Intel дрова и другие сервисные программы должны располагаться в 1 или 2 кольце. Однако ОС Windows используют только два кольца: 0 и 3. В третьем – пользовательские приложения и служебные сервисы ОС, в нулевом – все остальное (хотя это грубое разделение, но примерно так все и выглядит).

Здесь, во избежание дальнейших недоразумений, следует раз навсегда договориться: если я пишу «уровень привилегий задачи А больше уровня привилегий задачи В», то это значит, что численное значение привилегий задачи А меньше чем у В (хотя это и так очевидно, 0<3, 0 круче 3).

Уровни привилегий: это CPL (Current Privilege Level) и DPL (Descriptor Privilege Level). Название каждого говорит само за себя. Начнем по-порядку. Итак, CPL.

CPL – это текущий уровень привилегий, иначе говоря – уровень привилегий кода (задачи) исполняющейся в данный момент. Он хранится в битах 1-0 селектора. Т.е. под CPL отведено 2 бита – всего могут принять 4 значения. Где находиться сам селектор, хранящий CPL? В CS и нигде больше. Т.е. если в данный момент исполняется код обработчика #GP (и он решает, что сделать с программой, вызвавшей исключение) то CPL кода (этого самого обработчика #GP) = 0 (хотя и необязательно – все зависит от фантазии разработчика ОС, но логично предположить, что CPL обработчика #GP равен 0).

DPL – уровень привилегий СЕГМЕНТА. Храниться в дескрипторе каждого сегмента в таблице дескрипторов. Вообще вот посмотришь на предыдущий тип уровня привилегий – CPL. CPL – уровень привилегий кода, исполняющегося в данный момент. Но ведь код тоже находиться в каком-то сегменте! Так чем же они отличаются? Фразой «в данный момент». Вот их главное отличие. Ну и, помимо того, CPL относится только к сегментам кода (исполняющимся в данный момент), а DPL содержится в каждом дескрипторе, неважно что описывает этот дескриптор. Под него также отведено 2 бита. Процессор смотрит на поле DPL только при обращении к сегменту. Например, исполняется какая-то пользовательская программа. Вдруг процессор встречает инструкцию:

mov dword ptr DS:[00000000h],0 ; программа хочет записать ноль по адресу DS:00000000h

(а DS равно все равно чему, например, 1234h)

Тут процессор задумается: а можно ли этой подозрительной программе с CPL=3 записывать данные туда, куда она намеревается это сделать? Тут же из селектора 1234h извлекается индекс (старшие 13 бит - 246h) и в таблице дескрипторов (в данном случае – текущей LDT) находится нужный дескриптор, в котором процессор ищет в первую очередь поле DPL – допустим, там ноль. Т.е. уровень привилегий сегмента, в который хочет писать программа, явно выше, чем тот, который у программы. Тут же генерируется #GP, обработчик которого должным образом разрешит ситуацию. Может возникнуть вопрос: а если программа «успеет» записать нолик до тех пор, пока процессор производит необходимые проверки? Ответ: не успеет. #GP — это исключение типа ошибки (fault), оно всегда генерируется до исполнения инструкции.