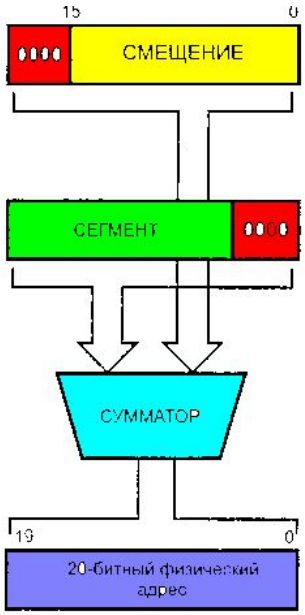
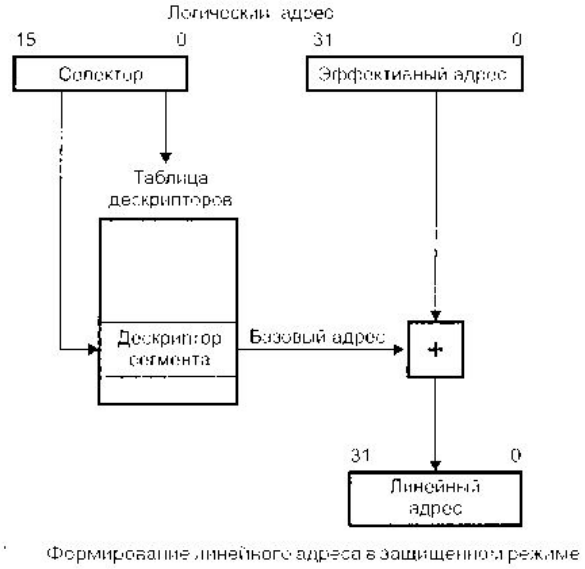
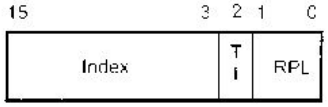
Модели памяти. Дескрипторы и таблицы (начало).

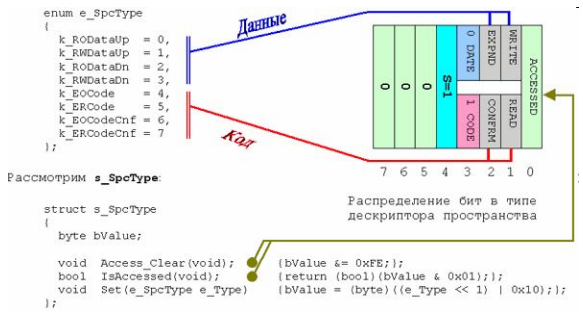


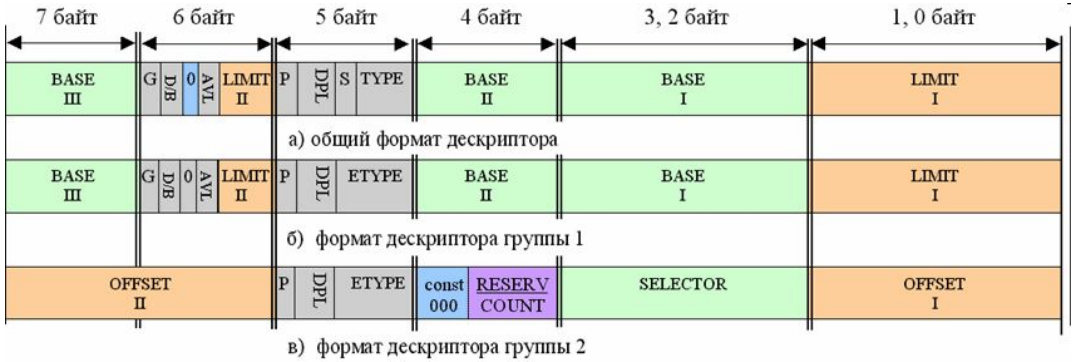


Формирование линейного адреса в защищенном режиме



Формат селектора



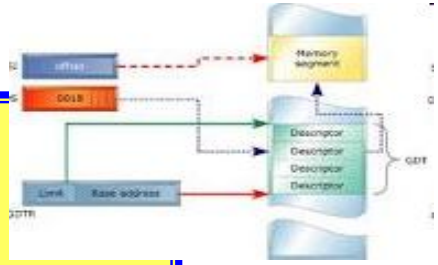


Outline

Организация памяти:

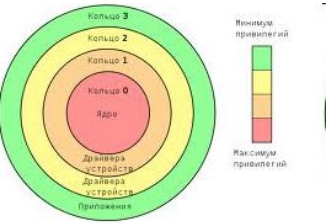
- Сплошная (плоская) модель;
- Сегментированная модель;
- Модель режима реального адреса.

Основные положения защищенного режима:

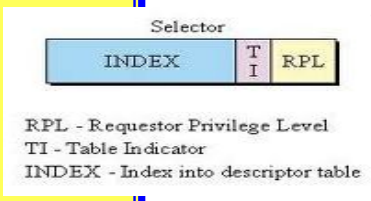


- дескриптор;
- формирование линейного адреса в защищенном режиме;
- особенности формирования физического адреса в защищенном режиме;

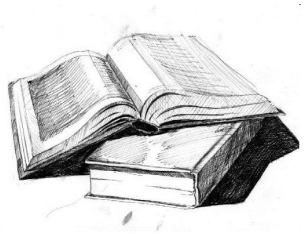
Дескрипторные таблицы:



- основные понятия;
- глобальная таблица дескрипторов;
- локальная таблица дескрипторов;
- таблица дескрипторов прерываний.



Резюме к лекции и список используемой литературы

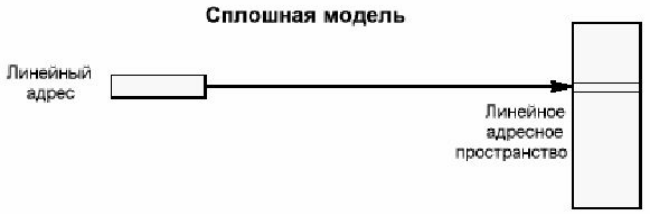


Модели памяти

При использовании средств процессора для управления памятью, программа может использовать одну из трех моделей доступа к памяти

- сплошная ("плоская") модель памяти
- сегментированная модель памяти
- модель режима реального адреса

При использовании **сплошной модели** (Flat Model) памяти программа оперирует единым непрерывным адресным пространством - линейным адресным пространством.
В нем содержатся и код, и стек, и данные программы, адресуемые смещением в пределах от 0 до 2^{32}-1.
Такое 32-битное смещение называется линейным адресом.



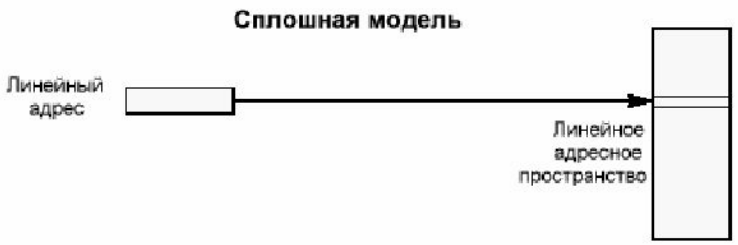
- **Идеология плоской модели**: вся память представляет собой единое линейной последовательностью байт

Ответственность за корректное использование памяти целиком ложится на прикладного программиста, т.е. позаботится о том, чтобы данные не затерли коды или на них не «наехал» растущий стек.

Важно отметить!
Что плоская модель – это классическая реализация фон-неймановской архитектуры – здесь хранятся и данные и коды.

Для 16-битных процессоров плоская модель памяти позволяет адресовать 64 кБ оперативной памяти; для 32-битных процессоров 4 ГБ, для 64-битных — 16 экзабайт.

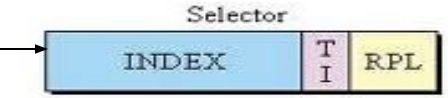
Сплошная ("плоская") модель памяти



Важно отметить!

Что плоская модель не может быть использована в реальном режиме – в ней не вся адресуемая память будет доступной.

Чтобы получить плоскую модель памяти достаточно все сегментные регистры загрузить селектором дескриптора, описывающим одну и ту же область памяти, но с разными свойствами для кода, стека и данных.

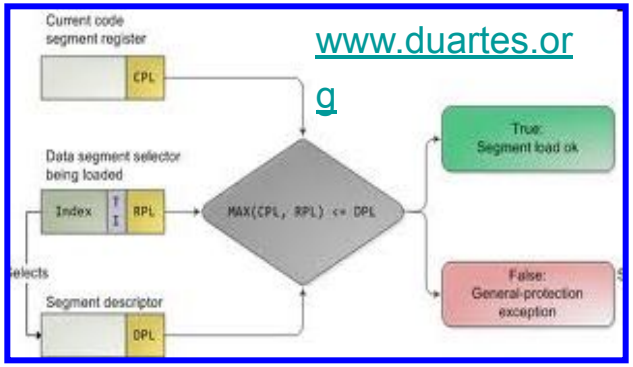


RPL - Requestor Privilege Level
TI - Table Indicator
INDEX - Index into descriptor table

Преимущества управления памятью с плоской моделью:

1) В одном из многозадачных встроенных приложений, где управление памятью не нужно и не желательно, модель обеспечивает простейший интерфейс для программирования, с прямым доступом ко всем местам в памяти и минимальной сложностью конструкции программы.

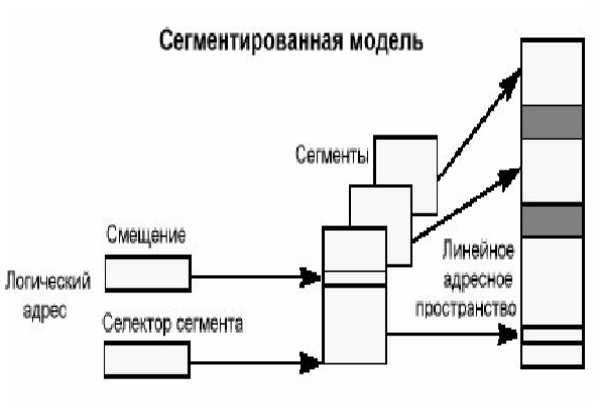
2) При многозадачности и распределении ресурсов плоская модель по-прежнему обеспечивает максимальную гибкость для реализации этого типа управления памятью.



Управление памятью все ещё (на 2011 год) реализуется на основе плоской модели, в целях содействия функциональности операционной системы, защиты ресурсов, многозадачности или увеличения объёма памяти за пределы ограничений, налагаемых физическим адресным пространством процессора.

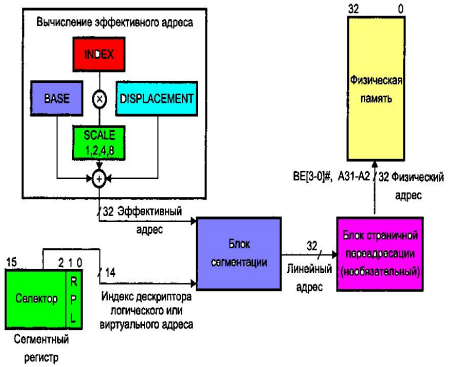
Сегментированная модель памяти

При использовании **сегментированной модели** (Segmented Model) для программы память представляется группой независимых адресных блоков, называемых *сегментами*.



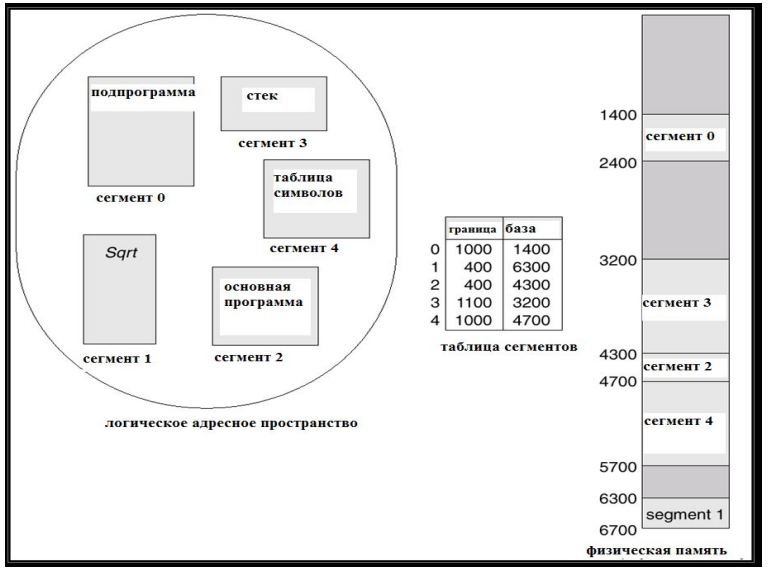
- Для адресации байта памяти программа должна использовать ***логический адрес***, состоящий из селектора сегмента и смещения.
- Селектор сегмента выбирает определенный сегмент, а смещение указывает на конкретный байт в адресном пространстве выбранного сегмента.
- Каждая задача в защищенном режиме может иметь до 16383 сегментов, размером до 4 Гбайт каждый, таким образом, общий объем памяти, адресуемой программой составляет 64 Тбайт.

Процессор при помощи блока сегментации отображает ***логический адрес*** в ***линейное адресное пространство***.

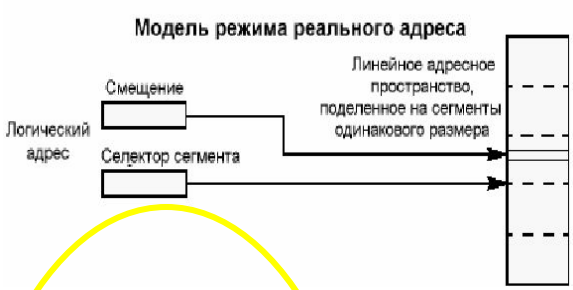


- Сегментация позволяет эффективно управлять пространством логических адресов.
- Сегменты используются для объединения областей памяти с общими атрибутами.

- Каждый сегмент имеет несколько связанных с ним атрибутов: размер, расположение, тип (стек, программа или данные) и характеристики защиты.



Модель режима реального адреса

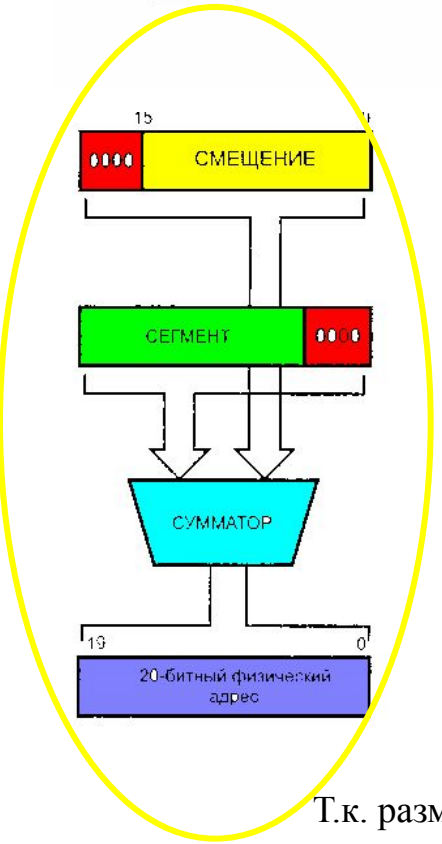


Архитектура может использовать **модель режима реального адреса** (Real-address Mode Model).

Эта модель является специфическим случаем сегментированной модели.

Линейное адресное пространство образуется из массива сегментов длиной по 64 Кбайт.

Размер такого линейного адресного пространства - 1 Мбайт.

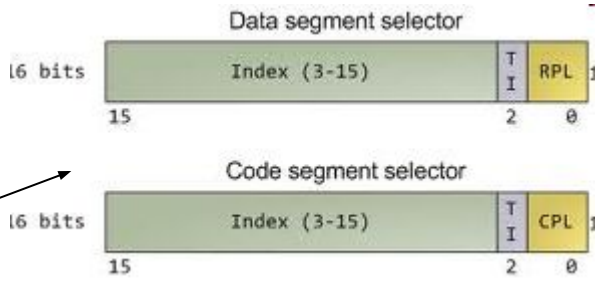


В этой модели селектор сегмента непосредственно используется для вычисления базового адреса в линейном адресном пространстве путем сдвига значения селектора на 4 бита влево (умножение на 16). Это значит, все сегменты начинаются с адреса, кратного 16.

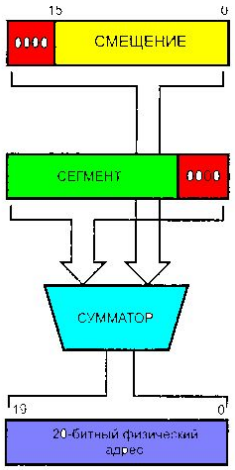
- 16-байтный блок памяти называют параграфом, поэтому говорят, что сегмент выравнивается по границе параграфа.

§

Т.к. размер сегмента намного превышает размер параграфа, то имеет место ***перекрытие сегментов***, т.е. один и тот же байт линейного адресного пространства может быть адресован ***различными логическими адресами*** (используя селекторы разных сегментов).



Модель памяти в реальном режиме



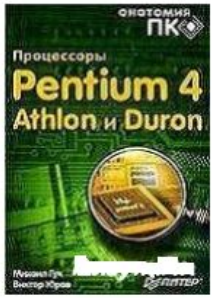
Исполнительный адрес, также называемый *эффективным адресом*, может быть константой, содержимым регистра, содержимым ячейки памяти или суммой нескольких величин (например, двух регистров и константы), но эта сумма является *16-разрядной* (перенос игнорируется). Таким образом, физический адрес *никогда* не перейдет границу 64-килобайтного сегмента, на начало которого указывает текущий сегментный указатель. Сегмент получается как бы свернутым в кольцо (wrapped): по мере увеличения суммируемых компонентов исполнительный адрес растет, но после достижения значения FFFFh снова обнуляется и начинает расти с начала. С одной стороны, это свойство обеспечивает некоторую защиту сегментов друг от друга (хотя некорректно написанная программа может легко перезагрузить указатель сегмента и повредить данные другого сегмента)

Поскольку свернутым в кольцо является всё пространство памяти: по мере увеличения исполнительного адреса и адреса сегмента физический адрес растет, но только до значения FFFFFh, после чего обнуляется и начинает расти с начала.

Модель памяти в реальном режиме

При вычислении физического адреса ( ) возможно возникновение переполнения, которое с 20-разрядной шиной адреса приводило к сворачиванию пространства в кольцо. Если, например, Seg = FFFFh и EA = FFFFh, физический адрес, вычисленный по формуле PA = 16 × Seg + EA, получается равным 10FFEF. Процессором 8086 он трактуется как 0FFEF — адрес, принадлежащий первому мегабайту. Однако на выходе A20 процессора в этом случае установится единичное значение, что соответствует адресу ячейки из второго мегабайта физической памяти. Для обеспечения полной программной совместимости с 8086 в схему IBM PC/AT был введен специальный вентиль *Gate A20*, принудительно обнуляющий бит A20 системной шины адреса. 32-разрядные процессоры имеют специальный вход A20M, по которому в реальном режиме принудительно обнуляется внешний бит адреса A20. Вентиль в PC управляется через программно-управляемый бит контроллера клавиатуры 8042 или более быстрым способом (Gate A20 Fast Control), определяемым чипсетом системной платы.

Основные положения защищенного режима



Назначение

Защищенный режим предназначен для обеспечения независимости выполнения нескольких задач, что подразумевает защиту ресурсов одной задачи от возможного воздействия другой (под задачами подразумеваются как приложения, так и задачи операционной системы).

Что защищаем?

Основным защищаемым ресурсом является память, в которой хранятся коды, данные и различные системные таблицы (например, таблица прерываний). Защищать требуется и совместно используемую аппаратуру, обращение к которой обычно происходит через операции ввода-вывода и прерывания. В защищенном режиме процессор аппаратно реализует многие функции защиты, необходимые для построения супервизора многозадачной ОС, в том числе механизм виртуальной памяти.

На чем основана защита?

Защита памяти основана на *сегментации*. *Сегмент* — это блок пространства памяти определенного назначения. К элементам сегмента возможно обращение с помощью различных инструкций процессора, использующих разные режимы адресации для формирования адреса в пределах сегмента.

Как работать в РМ?

В защищенном режиме сегменты тоже распределяются операционной системой, но прикладная программа может использовать только разрешенные для нее сегменты памяти, выбирая их с помощью *селекторов* из предварительно сформированных *таблиц дескрипторов сегментов*

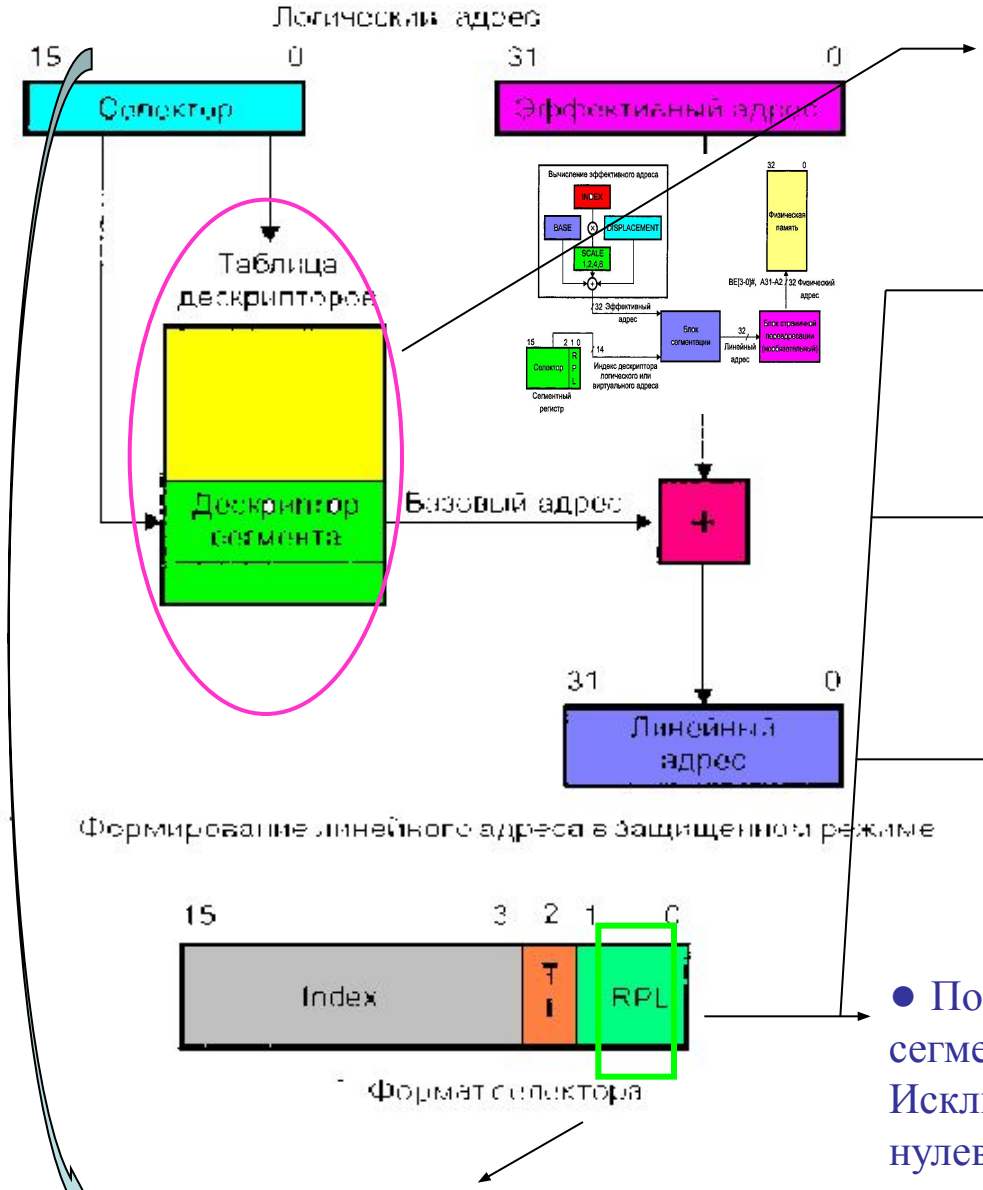
Дескрипторы

Дескрипторы представляют собой 8-байтные структуры данных, используемые для определения свойств программных элементов (сегментов, шлюзов и таблиц). Дескриптор определяет положение элемента в памяти, размер занимаемой им области (лимит), его назначение и характеристики защиты. Все дескрипторы хранятся в таблицах, обращение к которым поддерживается процессором аппаратно.

Защита памяти путем сегментации не позволяет:

- использовать сегменты не по назначению (например, пытаться трактовать область данных как коды инструкций);
- нарушать права доступа (пытаться модифицировать сегмент, предназначенный только для чтения, обращаться к сегменту, не имея достаточных привилегий и т. п.);
- адресоваться к элементам, выходящим за лимит сегмента;
- изменять содержимое таблиц дескрипторов (то есть параметров сегментов), не имея достаточных привилегий.

Формирование линейного адреса в защищенном режиме

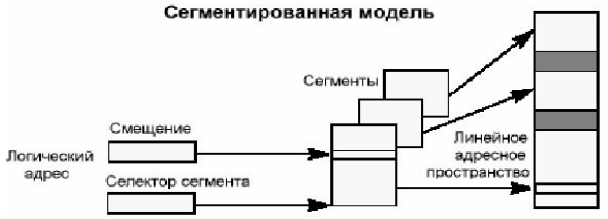


- Процессор может обратиться только к тем сегментам памяти, для которых имеются дескрипторы в таблицах.
- Дескрипторы выбираются с помощью 16-битных селекторов, программно загружаемых в сегментные регистры.
- Индекс совместно с индикатором таблицы TI позволяет выбрать дескриптор из локальной(TI =1) или глобальной(TI=0) таблицы дескрипторов.
- Для неиспользуемых сегментных регистров предназначен нулевой селектор сегмента , формально адресующийся к самому первому элементу глобальной таблицы.
- Попытка обращения к памяти по такому сегментному регистру вызовет исключение. Исключение возникает и при попытке загрузки нулевого селектора в регистре CS и SS.

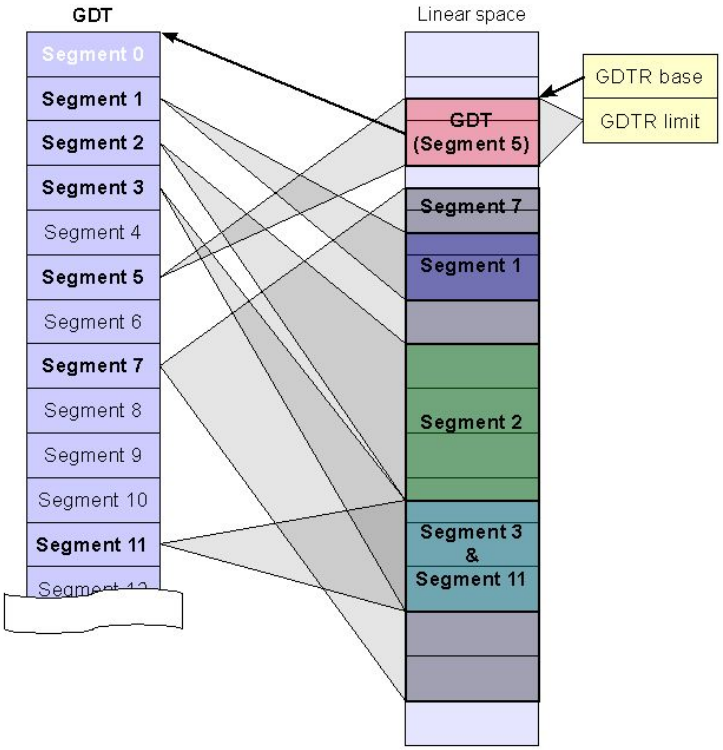
Поле RPL указывает требуемый уровень привилегий.

Особенности формирования физического адреса в Protected Mode

Как отмечалось выше, защищенный режим сохранил **сегментную модель памяти** (адрес ячейки памяти определяется суммой компонент: адрес начала сегмента и внутрисегментного смещения).



Но! в отличие от режима реального адреса, где адрес сегмента находился непосредственно в одном из сегментных регистров – алгоритм формирования *физического адреса* в защищенном режиме ***абсолютно иной***.

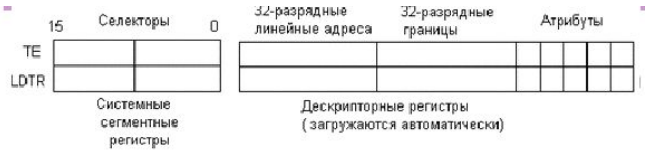


- Сегмент, в защищенном режиме, это не просто область памяти, ограниченная лишь максимально допустимым значением внутрисегментного смещения, как это было в реальном режиме, - это «объект», который имеет строго ***определенный размер.***

Минимальный размер сегмента может быть равен 1 байту, а максимальный – 1 Мб или 4 Гб (это зависит от, того, умножается размер сегмента на 4 килобайта или нет).

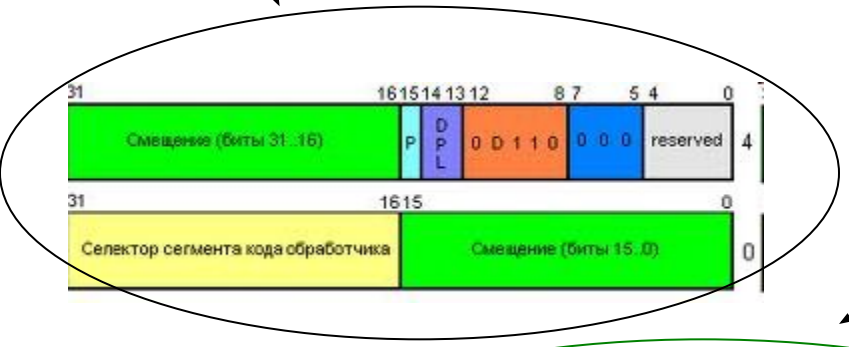
e-zine.excode.ru/online/3/syscall.html

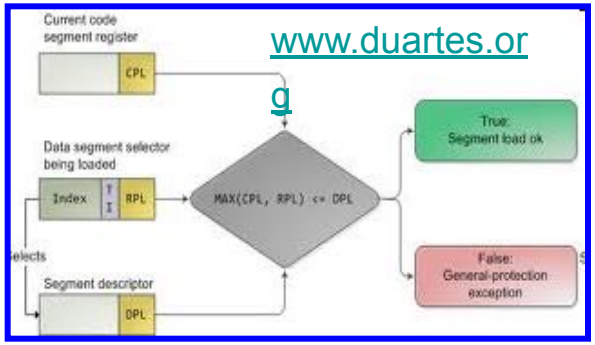
Особенности формирования физического адреса в Protected Mode



Этот сегмент:

- **не пересекается** с другими сегментами (хотя один и тот же сегмент можно описать дважды)
- и имеет ряд других атрибутов, по которым, в частности, происходит аппаратная защита памяти со стороны процессора.
- Каждый сегмент имеет **свой дескриптор** (описатель сегмента).
- Дескрипторы сегментов собираются в специализированных системных сегментах – дескрипторных таблицах.





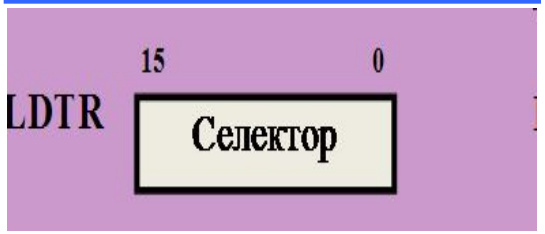
www.duartes.org

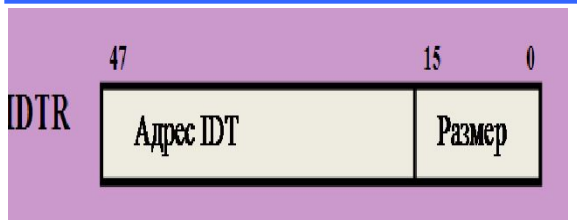
Дескрипторные таблицы

Дескрипторные таблицы — служебные структуры данных, содержащие дескрипторы сегментов

Дескрипторные таблицы

- глобальная таблица дескрипторов
- локальная таблица дескрипторов
- таблица дескрипторов прерываний





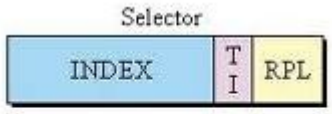
Существует три типа **дескрипторных таблиц** – *глобальная таблица дескрипторов* (одна в системе); *локальная таблица дескрипторов* (своя для каждой задачи); *таблица дескрипторов прерываний*

Каждый дескриптор (элемент) таблицы описывает свой сегмент памяти.

Сегменты памяти не пересекаются!

- Адрес начала каждой из таблиц (указатель на начало таблицы) хранится в специальных (программно доступных) регистрах процессора.
- Указатели на таблицу глобальных дескрипторов (GDTR) и таблицу прерываний (IDTR) имеют размер 48 байт, 32 из которых указывают линейный адрес начала таблицы, а остальные 16 – ее размер (предел).

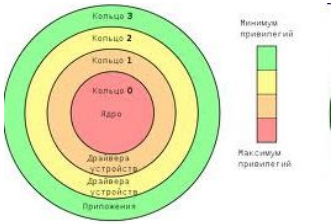
Дескрипторные таблицы

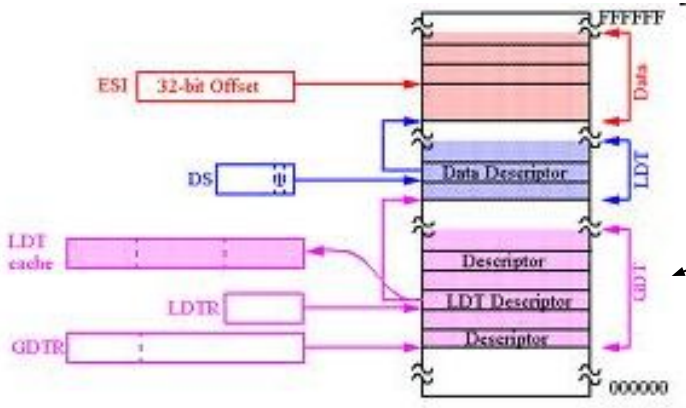


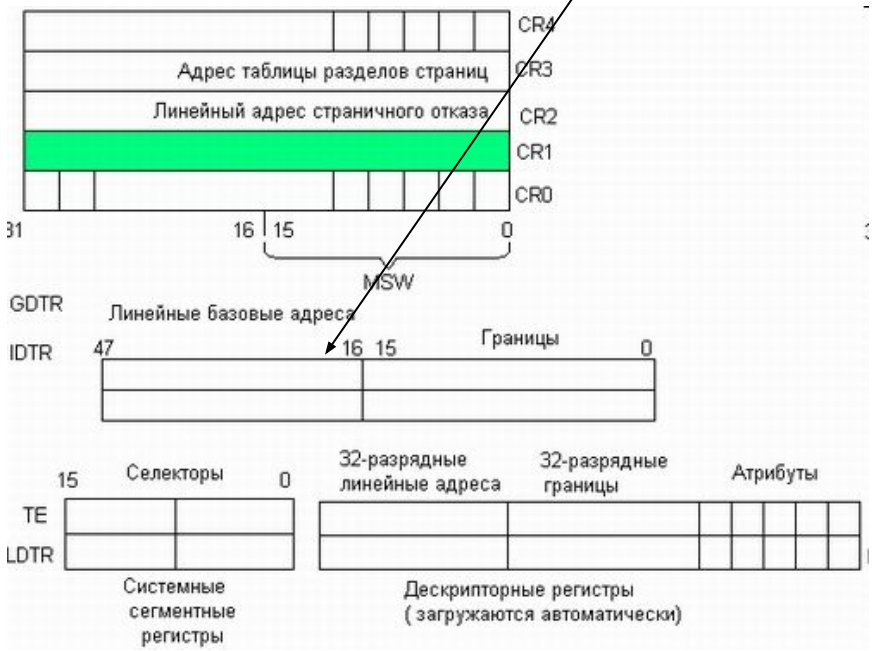
RPL - Requestor Privilege Level
TI - Table Indicator
INDEX - Index into descriptor table

- У регистра-указателя локальной дескрипторной таблицы (LDTR) программно доступно только 16-битное поле селектора (индекса для GDT), по которому из GDT автоматически загружаются программно недоступные и невидимые поля базового адреса и размера таблицы.
- Регистр LDTR указывает на дескриптор в GDT, описывающий локальную дескрипторную таблицу для текущей задачи.

Команды загрузки регистров-указателей таблиц (GDTR, LDTR, IDTR) являются привилегированными (выполняются только на нулевом кольце привилегий

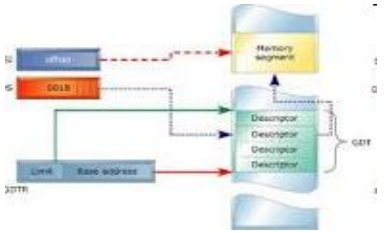






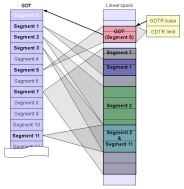
Глобальная таблица дескрипторов

Некоторые важные понятия о GTD (Global Descriptor Table) :

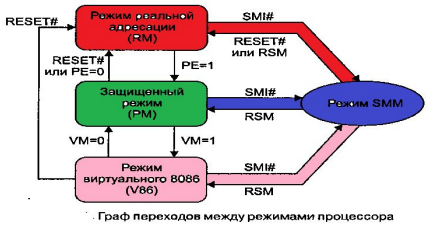


- Глобальная дескрипторная таблица ***одна***. Она общая для всех задач.

- Прежде, чем процессор перейдёт в PM, должна быть определена GDT, т.к. все сегменты и прочие системные объекты должны быть описаны в дескрипторной таблице;



- Процессору всё равно, где именно вы расположили эту таблицу, но, в любом случае, она будет находится в первом мегабайте адресного пространства, потому что только из режима реальных адресов можно перевести процессор в защищённый режим;



- Таблица GDT будет выровнена на границу 8 байт, так как дескрипторы, из которых она состоит, имеют 8-байтный размер. Выравнивание позволит процессору максимально быстро обращаться к дескрипторам, что, естественно, увеличивает производительность.

http://sasm.narod.ru/docs/pm/pm_in/chap_7.htm

Глобальная таблица дескрипторов

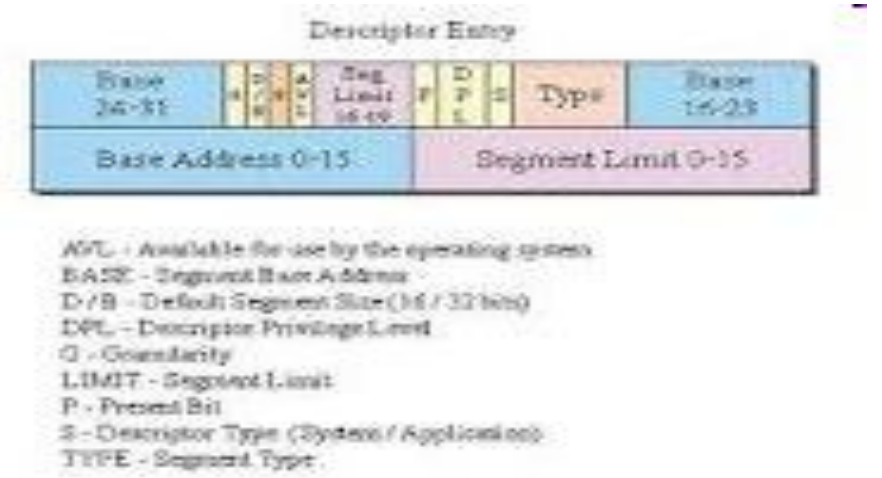
Некоторые важные понятия о GTD (Global Descriptor Table) :

● Число дескрипторов, определённых в GDT, может быть любым, от 0 до 8192.

Размер таблицы не может превышать 8192 дескрипторов, поскольку один дескриптор занимает 8 байт, а лимит в регистре GDTR - двухбайтный и хранит размер таблицы минус один (максимальное значение лимита - 65535), а 8192 x 8 = 65536.

● 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;

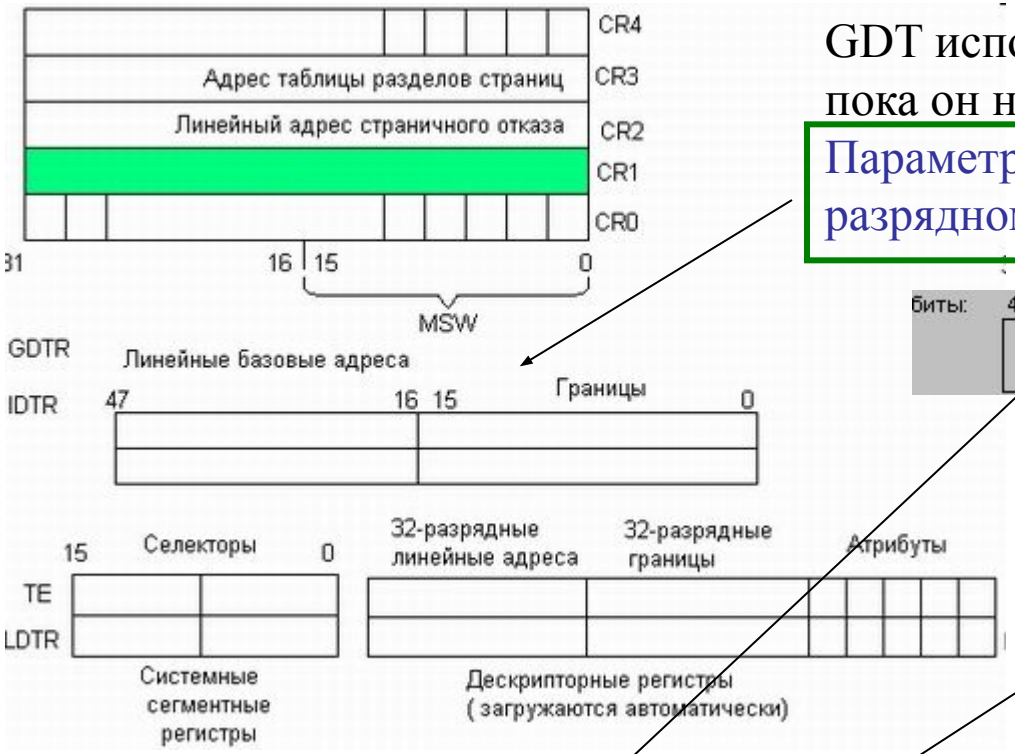
● Нулевой дескриптор можно использовать как шаблон, на основе которого программа может создавать новые дескрипторы.



Если всё же в программе встречается обращение к нулевому дескриптору, то процессор генерирует исключение и не позволит доступ к такому дескриптору.

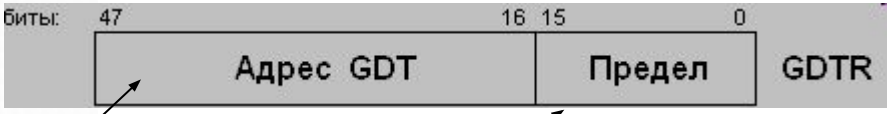
но на практике их удобнее создавать иными способами

Глобальная таблица дескрипторов



GDT используется процессором всё время, пока он находится в защищённом режиме.

Параметры GDT хранятся в специальном 48-разрядном регистре GDTR:



Формат регистра GDTR следующий:
биты:
[0..15]: 16-разрядный предел GDT
[15..47]: 32-разрядный адрес начала GDT

Адрес начала GDT - это тот адрес, по которому вы разместили GDT.

Предел таблицы GDT - это максимальное смещение относительно её начала.

Пример создания GDT

Например, вы создаёте GDT, состоящую из 3-х дескрипторов - для сегментов кода, стека и данных.

Общее число дескрипторов будет равно четырём, потому что первым по счёту будет идти нулевой дескриптор, а за ним уже остальные три:

Смещение от начала GDT	Назначение дескриптора
0	Нулевой
8	Сегмент кода
16	Сегмент стека
24	Сегмент данных

Размер GDT в данном случае будет равен 32 байтам, =>, предельное смещение в таблице будет равно 31 - это и есть предел GDT.

Алгоритм:
1. Вычисляем 32-разрядный адрес GDT
2. Вычисляем размер GDT
3. Сохраняем параметры GDT в 48-разрядную переменную
4. Загружаем значение в регистр GDTR

- Для загрузки значения в регистр GDTR используется команда LGDT.
- Операндом этой команды является 48-разрядное значение адреса в памяти, где размещается адрес и предел GDT.
- Сохранить содержимое GDTR командой SGDT, указав в операнде адрес 48-разрядной переменной в памяти.

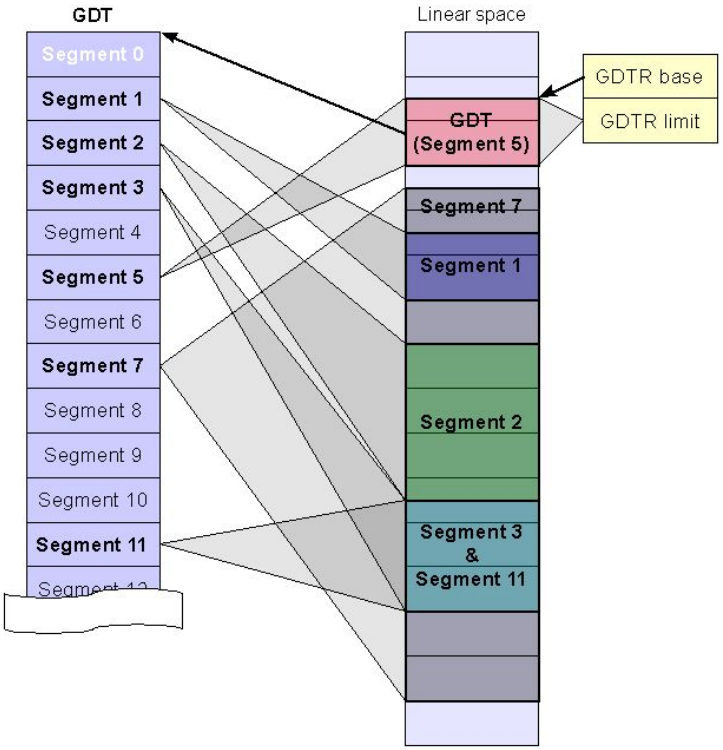
Пример создания GDT

Код:

```
; 1. Вычисляем 32-разрядный адрес GDT
        xor     eax,eax         ; EAX = 0; адрес будем вычислять в регистре EAX.
        mov     ax,ds  ; Подразумевается, что GDT находится в  текущем сегменте данных.
        shl     eax,4  ; EAX = адрес начала сегмента DS
        xor     edx,edx         ; EDX = 0.
        lea     dx,GDT          ; EDX = DX = смещение начала GDT  относительно DS.
        add     eax,edx         ; EAX = полный физический адрес GDT  в памяти.
; 2. Вычисляем размер GDT
        mov     cx,4    ; CX = число дескрипторов +
                        ;  + нулевой дескриптор.
        shl     cx,3    ; CX = CX * 8 - столько байт будут
                        ;  занимать в GDT эти дескрипторы.
        dec     cx      ; Предел меньше размера на 1
; 3. Сохраняем параметры GDT в 48-разрядную переменную
        mov     GDT_address,eax
        mov     GDT_limit,cx
; 4. Загружаем значение в регистр GDTR
        lgdt    GDT_params
;-------------------------------------------------------------------------
GDT_params      label   fword
GDT_limit       dw      ?
GDT_address             dd      ?
GDT:
        dd      ?       ; 0-й дескриптор
        dd      ?
        dd      ?       ; 1-й дескриптор (код)
        dd      ?
        dd      ?       ; 2-й дескриптор (стек)
        dd      ?
        dd      ?       ; 3-й дескриптор (данные)
        dd      ?
```

http://sasm.narod.ru/docs/pm/pm_in/chap_7.htm

Особенности работы с GTD



- Размер GDT желательно не менять в процессе выполнения программ в защищённом режиме.

- Если ваша программа будет динамически создавать новые дескрипторы, то размер GDT лучше всего заранее задать достаточно большим, например, 64 Кб (максимальный размер).

- Однако, следует учитывать, что при обращении процессора к несуществующим дескрипторам, его поведение непредсказуемо, хотя оно, скорее всего, закончится зависанием.

В следующей серии

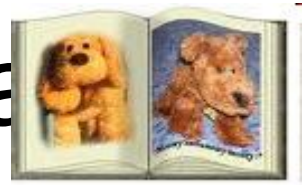
Дескрипторы (продолжение)

1. Дескрипторы адресных пространств:
   - Сегмент кода
   - Сегмент данных
   - Сегмент состояния задачи
2. Дескрипторы системных объектов:
   - Шлюз вызова*
   - Шлюз задачи
   - Шлюз прерывания
   - Шлюз ловушки
   - Таблицы локальных дескрипторов

…и многое другое…

Основные используемые Интернет-ресурсы:

http://cracklab.ru/faq/Int3
http://sasm.narod.ru/apps/eflags/app_b.htm
http://club155.ru/x86cmd/POPF
http://kit-e.ru/articles/cpu/2006_9_148.php
http://de.ifmo.ru/--books/electron/cpu-cod.htm
http://www.viva64.com/ru/k/0035/
http://www.kailib.ru/arhitevm?start=23
http://www.zcub.ru/blog/org_comp_system/registry-processora.php



Используемая литература

- Книга «Архитектура ЭВМ»,автор Мюллер
- Книга «Процессоры *Pentium4, Athlon* и *Duron*», авторы Михаил Гук, Виктор Юров
- Книга «Архитектура ЭВМ», автор Танненбаум
- Книга «*Assembler.* Учебник для ВУЗов», автор Юров