**Логическое адресное пространство**

Для адресации операндов в физическом адресном пространстве программы используют логическую адресацию. Процессор автоматически транслирует логические адреса в физические, выдаваемые затем на системную шину.

Архитектура компьютера различает физическое адресное пространство (ФАП) и логическое адресное пространство (ЛАП). **Физическое адресное пространство** представляет собой простой одномерный массив байтов, доступ к которому реализуется аппаратурой памяти по адресу, присутствующему на шине адреса микропроцессорной системы. **Логическое адресное пространство** организуется самим программистом исходя из конкретных потребностей. Трансляцию логических адресов в физические осуществляет блок управления памятью MMU.

В архитектуре современных микропроцессоров ЛАП представляется в виде набора элементарных структур: байтов, сегментов и страниц. В микропроцессорах используются следующие варианты организации **логического адресного пространства**:

* **плоское (линейное) ЛАП**: состоит из массива байтов, не имеющего определенной структуры; трансляция адреса не требуется, так как логический адрес совпадает с физическим;
* **сегментированное ЛАП**: состоит из сегментов - непрерывных областей памяти, содержащих в общем случае переменное число байтов; логический адрес содержит 2 части: идентификатор сегмента и смещение внутри сегмента; трансляцию адреса проводит блок сегментации MMU;
* **страничное ЛАП**: состоит из страниц - непрерывных областей памяти, каждая из которых содержит фиксированное число байтов. Логический адрес состоит из номера (идентификатора) страницы и смещения внутри страницы; трансляция логического адреса в физический проводится блоком страничного преобразования MMU;
* **сегментно-страничное ЛАП**: состоит из **сегментов**, которые, в свою очередь, состоят из страниц; логический адрес состоит из идентификатора сегмента и смещения внутри сегмента. Блок сегментного преобразования MMU проводит трансляцию логического адреса в номер страницы и смещение в ней, которые затем транслируются в физический адрес блоком страничного преобразования MMU.

Таким образом, основой получения физического адреса памяти служит логический адрес. В какой-то степени логическое адресное пространство, с которым имеет дело программист, можно сравнить со структурой книги, где аналогом сегмента выступает рассказ, страница книги соответствует странице ЛАП, а искомая информация - это некоторое слово. При этом если память организована как линейная, то номер искомого слова задается в явном виде и просто отсчитывается от начала книги. При сегментном представлении памяти искомое слово определяется его номером в заданном рассказе. Страничное представление памяти предполагает задание информации о слове в виде номера страницы в книге и номера слова на указанной странице. При сегментно-страничном представлении логический адрес слова задается номером слова в определенном рассказе. В этом случае по оглавлению книги определяется номер страницы, с которой начинается указанный рассказ. Затем, зная количество слов на странице и положение слова в рассказе, можно вычислить страницу книги и положение искомого слова на этой странице.

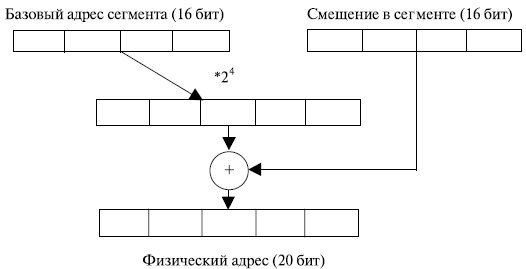
**Формирование физического адреса в универсальном микропроцессоре при различных режимах работы**

Микропроцессор способен работать в двух режимах: реальном и защищенном.

При работе в **реальном режиме** возможности процессора ограничены: емкость адресуемой памяти составляет 1 Мбайт, отсутствует страничная организация памяти, **сегменты** имеют фиксированную длину 216 байт.

Этот режим обычно используется на начальном этапе загрузки компьютера для перехода в **защищенный режим**.

В **реальном режиме** сегментные регистры процессора содержат старшие 16 бит физического адреса начала **сегмента**. Сдвинутый на 4 разряда влево **селектор** дает 20-разрядный базовый адрес сегмента. Физический адрес получается путем сложения этого адреса с 16-разрядным значением смещения в сегменте, формируемого по заданному режиму адресации для операнда или извлекаемому из регистра EIP для команды ([рис. 3.1](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=1#image.3.1)). По полученному адресу происходит выборка информации из памяти.



**Рис. 3.1.** Схема получения физического адреса

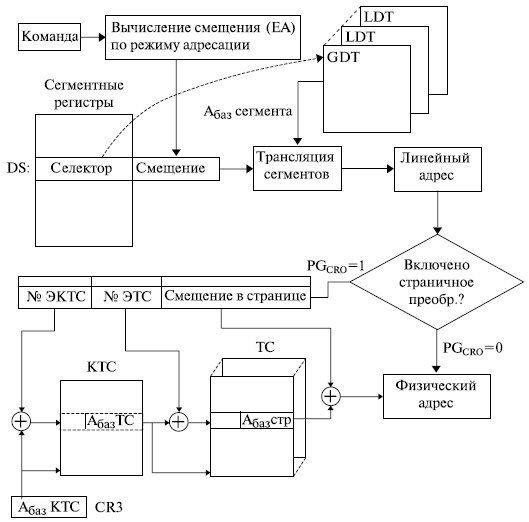
Наиболее полно возможности микропроцессора по адресации памяти реализуются при работе в **защищенном режиме**. Объем адресуемой памяти увеличивается до 4 Гбайт, появляется возможность страничного режима адресации. **Сегменты** могут иметь переменную длину от 1 байта до 4 Гбайт.

Общая схема формирования физического адреса микропроцессором, работающим в **защищенном режиме**, представлена на [рис. 3.2](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=1#image.3.2).

Как уже отмечалось, основой формирования физического адреса служит логический адрес. Он состоит из двух частей: **селектора** и **смещения в сегменте**.

**Селектор** содержится в сегментном регистре микропроцессора и позволяет найти описание сегмента **(дескриптор)** в специальной таблице дескрипторов. **Дескрипторы** сегментов хранятся в специальных системных объектах глобальной (GDT) и локальных (LDT) таблицах дескрипторов. **Дескриптор** играет очень важную роль в функционировании микропроцессора, от формирования физического адреса при различной организации адресного пространства и до организации мультипрограммного режима работы. Поэтому рассмотрим его структуру более подробно.

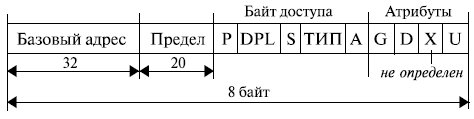
**Сегменты** микропроцессора, работающего в **защищенном режиме**, характеризуются большим количеством параметров. Поэтому в универсальных 32-разрядных микропроцессорах информация о сегменте хранится в



**Рис. 3.2.** Формирование физического адреса при сегментно-страничной организации памяти

специальной 8-байтной структуре данных, называемой **дескриптором**, а за сегментными регистрами закреплена основная функция - определение местоположения дескриптора.

Структура **дескриптора сегмента** представлена на [рис. 3.3](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=1#image.3.3).



**Рис. 3.3.** Структура дескриптора сегмента

Мы будем рассматривать именно структуру, а не формат дескриптора, так как при переходе от микропроцессора i286 к 32-разрядному МП расположение отдельных полей дескриптора потеряло свою стройность и частично стало иметь вид "заплаток", поставленных с целью механического увеличения разрядности этих полей.

32-разрядное поле базового адреса позволяет определить начальный адрес сегмента в любой точке адресного пространства в 232 байт (4 Гбайт).

**Поле предела** (limit) указывает длину сегмента (точнее, длину сегмента минус 1: если в этом поле записан 0, то это означает, что сегмент имеет длину 1) в адресуемых единицах, то есть максимальный размер сегмента равен 220 элементов.

Величина элемента определяется одним из атрибутов дескриптора битом G ( Granularity - гранулярность, или дробность):

G=\begin{cases}
   \mbox{0-длина в байтах}\\
   \mbox{1-длина в страницах}
     \end{cases}

Таким образом, сегмент может иметь размер с точностью до 1 байта в диапазоне от 1 байта до 1 Мбайт (при G = 0 ). При объеме страницы в 212 = 4 Кбайт можно задать объем сегмента до 4 Гбайт (при G = l ):

V_{\mbox{сегм макс}}=2^{20}_{стр}*2^{12}_{байт}=2^{32}байт

Так как в архитектуре IA-32 сегмент может начинаться в произвольной точке адресного пространства и иметь произвольную длину, сегменты в памяти могут частично или полностью перекрываться.

**Бит размерности** ( Default size ) определяет длину адресов и операндов, используемых в команде по умолчанию:

D=\begin{cases}
   \mbox{0-16 разрядов}\\
   \mbox{1-32 разряда}
     \end{cases}

Конечно, этот бит предназначен не для обычного пользователя, а для системного программиста, применяющего его, например, для отметки сегментов для сбора"мусора" или сегментов, базовые адреса которых нельзя модифицировать. Этот бит доступен только программам, работающим на высшем уровне привилегий. Микропроцессор в своей работе его не меняет и не использует.

**Байт доступа** определяет основные правила обращения с сегментом.

**Бит присутствия** P (Present) показывает возможность доступа к сегменту. Операционная система (ОС) отмечает сегмент, передаваемый из оперативной во внешнюю память, как временно отсутствующий, уставливая в его дескрипторе P = 0. При P = 1 сегмент находится в физической памяти. Когда выбирается дескриптор с P = 0 (сегмент отсутствует в ОЗУ), поля базового адреса и предела игнорируются. Это естественно: например, как может идти речь о базовом адресе сегмента, если самого сегмента вообще нет в оперативной памяти? В этой ситуации процессор отвергает все последующие попытки использовать **дескриптор** в командах, и определяемое **дескриптором** адресное пространство как бы"пропадает".

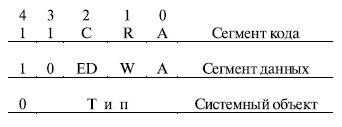
Возникает особый случай неприсутствия сегмента. При этом операционная система копирует запрошенный сегмент с диска в память (при этом, возможно, удаляя другой сегмент), загружает в **дескриптор** базовый адрес сегмента, устанавливает P = 1 и осуществляет рестарт той команды, которая обратилась к отсутствовавшему в ОЗУ сегменту.

**Двухразрядное** **поле** DPL (Descriptor Privilege Level) указывает один из четырех возможных (от 0 до 3) **уровней привилегий дескриптора**, определяющий возможность доступа к сегменту со стороны тех или иных программ (уровень 0 соответствует самому высокому уровню привилегий).

**Бит обращения** A (Accessed) устанавливается в"1" при любом обращении к сегменту. Используется операционной системой для того, чтобы отслеживать сегменты, к которым дольше всего не было обращений.

Пусть, например, 1 раз в секунду операционная система в дескрипторах всех сегментов сбрасывает бит А. Если по прошествии некоторого времени необходимо загрузить в оперативную память новый сегмент, места для которого недостаточно, операционная система определяет "кандидатов" на то, чтобы очистить часть оперативной памяти, среди тех сегментов, в **дескрипторах** которых бит А до этого момента не был установлен в"1", то есть к которым не было обращения за последнее время.

**Поле типа** в байте доступа определяет назначение и особенности использования сегмента. Если бит S ( System - бит 4 байта доступа) равен 1, то данный **дескриптор** описывает реальный сегмент памяти. Если S = 0, то этот дескриптор описывает специальный системный объект, который может и не быть сегментом памяти, например, шлюз вызова, используемый при переключении задач, или дескриптор локальной таблицы дескрипторов LDT. Назначение битов <3...0> байта доступа определяется типом сегмента ([рис. 3.4](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=1#image.3.4)).



**Рис. 3.4.** Формат поля типа байта доступа

**В сегменте кода**: бит подчинения, или согласования, C (Conforming) определяет дополнительные правила обращения, которые обеспечивают защиту сегментов программ. При C = 1 данный сегмент является подчиненным сегментом кода. В этом случае он намеренно лишается защиты по привилегиям. Такое средство удобно для организации, например, подпрограмм, которые должны быть доступны всем выполняющимся в системе задачам. При C = 0 - это обычный сегмент кода; бит считывания R (Readable) устанавливает, можно ли обращаться к сегменту только на исполнение или на исполнение и считывание, например, констант как данных с помощью префикса замены сегмента. При R = 0 допускается только выборка из сегмента команд для их выполнения. При R = 1 разрешено также чтение данных из сегмента.

Запись в сегмент кода запрещена. При любой попытке записи возникает программное прерывание.

**В сегменте данных**:

* ED (Expand Down) - бит направления расширения. При ED = 1 этот сегмент является сегментом стека и смещение в сегменте должно быть больше размера сегмента. При ED = 0 - это сегмент собственно данных (смещение должно быть меньше или равно размеру сегмента);
* бит разрешения записи W(Writeable). При W = 1 разрешено изменение сегмента. При W = 0 запись в сегмент запрещена, при попытке записи в сегмент возникает программное прерывание.

В случае обращения за операндом **смещение в сегменте** формируется микропроцессором по режиму адресации операнда, заданному в команде. Смещение в сегменте кода извлекается из **регистра - указателя команд** EIP.

Сумма извлеченного из дескриптора начального адреса сегмента и сформированного смещения в сегменте дает **линейный адрес** (ЛА).

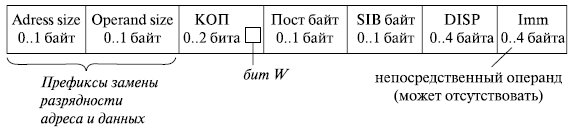
Если в микропроцессоре используется только сегментное представление адресного пространства, то полученный линейный адрес является также и физическим. Если помимо сегментного используется и страничный механизм организации памяти, то **линейный адрес** представляется в виде двух полей: старшие разряды содержат номер виртуальной страницы, а младшие смещение в странице. Преобразование номера виртуальной страницы в номер физической проводится с помощью специальных системных таблиц: **каталога таблиц страниц** (КТС) и **таблиц страниц** (ТС). Положение каталога таблиц страниц в памяти определяется системным регистром CR3. Физический адрес вычисляется как сумма полученного из таблицы страниц адреса физической страницы и смещения в странице, полученного из линейного адреса.

Рассмотрим теперь все этапы преобразования логического адреса в физический более подробно.

**Структура кода команды и формирование смещения в сегменте**

Как отмечалось выше, смещение в сегменте кода команд извлекается из регистра EIP и поэтому не требует дополнительных пояснений.

Механизм же формирования **смещения в сегменте** данных проводится на основе режима адресации операнда и требует отдельного изучения. Рассмотрим сначала структуру кода команды универсального 32-разрядного микропроцессора. Команды в архитектуре IA-32 имеют большое разнообразие форматов, которые зависят от типа операции, режимов адресации операндов, длины используемых непосредственных операндов и смещений и ряда других факторов. Они имеют длину от 1 до 15 байт. Все это существенно затрудняет их декодирование в МП с данной архитектурой. На [рис. 3.5](http://www.intuit.ru/studies/courses/604/460/print_lecture/5870) представлен формат двухоперандной команды общего вида.



**Рис. 3.5.** Формат команды 32-разрядного микропроцессора

Команда может начинаться с нескольких необязательных байт (префиксов), которые определяют особенности выполнения команды. Префиксы размеров длины адреса и операндов позволяют изменить их значения, установленные по умолчанию **битом размерности** D в дескрипторе сегмента. Для операндов совместно с битом w, содержащимся в коде команды, префикс размера позволяет определить операнд длиной 8, 16 или 32 разряда. Префикс размера адреса определяет 16- или 32-разрядное смещение в сегменте ([табл. 3.1](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=2#table.3.1)).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 3.1. Использование префиксов переопределения размеров адресов и операндов | | | | | | | | |
| Бит размерности D в дескрипторе сегмента | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Префикс размерности Операнда\* | - | - | + | + | - | - | + | + |
| Префикс размерности адреса\* | - | + | - | + | - | + | - | + |
| Разрядность операнда (бит) \*\* | 16/8 | 16/8 | 32/8 | 32/8 | 32/8 | 32/8 | 16/8 | 16/8 |
| Разрядность адреса (бит) | 16 | 32 | 16 | 32 | 32 | 16 | 32 | 16 |

Примечания:

\* + - префикс присутствует;

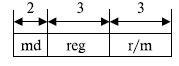
- - префикс отсутствует.

\*\* w = 1/0

В коде команды могут использоваться также дополнительные байты для префикса замены сегментного регистра, установленного по умолчанию, префикса повторения операции или префикса, предотвращающего прерывания операции перемещения данных.

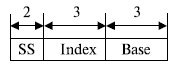
Поле КОП содержит код выполняемой команды, а также бит w размерности используемых операндов. Для команд, применяющих непосредственный операнд, код операции может также занимать также часть постбайта.

Постбайт ([рис. 3.6](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=2#image.3.6)) определяет местоположение операндов. Основная часть команд микропроцессора с архитектурой IA-32 позволяет работать только с одним операндом, находящимся в оперативной памяти. Его режим адресации кодируется полями md и r/m постбайта. Второй операнд либо извлекается из **регистров общего назначения** микропроцессора (его номер указывается в поле reg постбайта), либо кодируется в поле Imm самой команды (непосредственный операнд).



**Рис. 3.6.** Формат постбайта

Байт масштабируемого индекса базы (SIB) служит для представления сложных структур памяти. На его наличие указывает код 100 в поле r/m постбайта. SIB-байт имеет следующую структуру ([рис. 3.7](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=2#image.3.7)):



**Рис. 3.7.** Формат SIB-байта

Здесь SS - поле масштаба, Index задает номер одного из РОН, используемого в качестве индексного регистра (регистр ESP не может быть регистром индекса). Поле Base в комбинации с полем mod постбайта указывает регистр базы и смещение для индексированных операндов.

Применение SIB-байта позволяет формировать смещение в сегменте, иногда называемое эффективным адресом (ЭА), для операндов следующим образом:

|  |  |
| --- | --- |
| \mbox{(смещение в сегменте)} = [base] + [index]^{ss} + disp | ( 3.1) |

где [base] - значение базового регистра, [index] - значение индексного регистра, ss - величина масштабного множителя, disp - значение смещения, закодированного в самой команде. В качестве базы или индекса может быть использован любой **регистр общего назначения** микропроцессора. Величина индекса может быть умножена на масштабный коэффициент (1, 2, 4 или 8), что дает возможность ссылки на элемент массива или записи соответствующей длины.

Смещение disp кодируется как величина со знаком в дополнительном коде. Его длина определяется значением бита D в **дескрипторе** сегмента, битом w в первом байте команды и наличием или отсутствием префикса разрядности адреса согласно [табл. 3.1.](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=2#table.3.1)

Этот механизм отражает основные усовершенствования в способах адресации операндов для 32-разрядной архитектуры IA-32 по сравнению с архитектурой x86. Различные комбинации слагаемых в выражении (3.1) дают следующие способы адресации памяти:

* прямая (только смещение),
* косвенная (только база),
* базовая относительная (база + смещение),
* индексная (индекс с масштабом),
* индексная со смещением (индекс с масштабом + смещение),
* базовая индексная (база + индекс с масштабом)
* относительная базовая индексная (база + индекс с масштабом + смещение).

**Главные особенности формата команд МП** с архитектурой IA-32 по сравнению с 16-разрядным микропроцессором:

* возможность использования любого из **регистров общего назначения** в любом из режимов адресации;
* возможность использования 32-разрядных непосредственных операндов и смещений при относительных режимах адресации наряду с имевшимися ранее 8- и 16-разрядными;
* добавление еще одного режима адресации - относительного базового индексного с масштабированием.

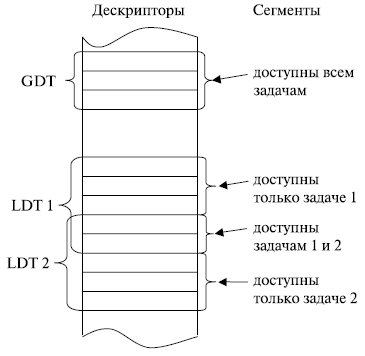
**Сегментная организация памяти в защищенном режиме**

В основе сегментной модели памяти лежит разделение ее на независимые адресные пространства переменной длины - **сегменты**. Для программы адресное пространство разделено на блоки смежных адресов, называемых сегментами, а программа может обращаться только к данным, находящимся в этих сегментах. Внутри сегментов применяется линейная адресация, то есть программа может обращаться к байту 0, байту 1 и т. д. Такая адресация осуществляется относительно начала сегмента, и физический адрес, ассоциируемый, например, с программным адресом 0, по существу, скрыт от программиста. Этот подход к управлению памятью опирается на тот факт, что программы обычно логически разделяются на области (сегменты) кода, данных и стека. При этом упрощается изоляция программ друг от друга в мультипрограммном режиме работы. Каждый **сегмент** имеет свое целевое назначение. Каждая задача имеет непосредственный доступ к трем основным сегментам: кода, данных и стека, определяемых сегментными регистрами CS, DS SS соответственно, и к трем дополнительным сегментам данных, определяемых сегментными регистрами ES, FS, GS. Описания этих сегментов содержатся в их **дескрипторах**. Любая программа, независимо от уровня ее привилегий, не может обращаться к сегменту до тех пор, пока он не описан с помощью дескриптора, а сам дескриптор не помещен в таблицу дескрипторов.

Дескрипторы хранятся либо в **глобальной таблице дескрипторов** (Global Descriptor Table - GDT), либо в локальных таблицах дескрипторов **(Local Descriptor Table - LDT)**. В **GDT** содержатся дескрипторы сегментов, которые доступны всем активным задачам, имеющимся в системе на данный момент. GDT может содержать любые дескрипторы сегментов, за исключением дескрипторов прерываний и ловушек. Обычно GDT включает дескрипторы сегментов кодов и данных операционной системы, сегментов состояния задач и дескрипторы сегментов, содержащих локальные таблицы дескрипторов. Микропроцессорная система имеет единственную глобальную таблицу дескрипторов.

**Локальная таблица дескрипторов** LDT используется для хранения дескрипторов, доступных только данной задаче. Их количество определяется количеством активных задач в системе.

С точки зрения расположения в памяти, локальные таблицы дескрипторов представляют собой обычные сегменты. Они могут накладываться друг на друга, частично пересекаться. Это приводит к тому, что отдельные сегменты, описанные дескрипторами в своих LDT, могут раздел ться несколькими задачами ([рис. 3.8](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=3#image.3.8)).



**Рис. 3.8.** Описание сегментов в таблицах дескрипторов

Для нахождения **дескриптора** в таблице дескрипторов используется **селектор**, который содержится в одном из сегментных регистров. **Селектор** представляет собой 16-разрядое слово, которое разбито на 3 поля ([рис. 3.9](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=3#image.3.9)):

* TI ( Table Indicator - **индикатор таблицы** ) показывает, к какой таблице идет обращение: TI = 0 - дескриптор находится в глобальной таблице дескрипторов GDT, TI = 1 - в локальной таблице LDT;
* Index: **поле индекса** - номер дескриптора в соответствующей таблице дескрипторов;
* RPL ( Request privilege level - **уровень привилегий запроса** ). При обращении сравнивается с полем DPL в байте доступа дескриптора.

Обращение разрешается, если уровень привилегий запроса не ниже, чем уровень привилегий дескриптора.

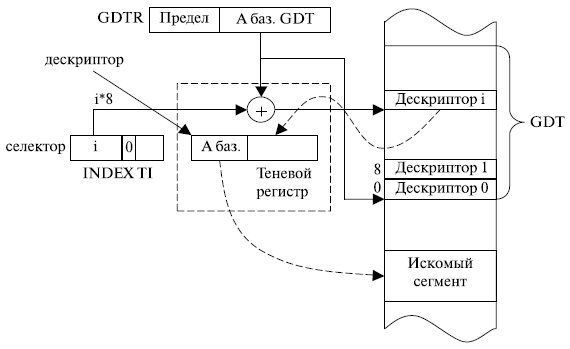
Формат селектора

**Рис. 3.9.** Формат селектора

Максимальное количество дескрипторов, находящихся в таблице дескрипторов, определяется длиной поля Index селектора и равно 213. Так как каждый дескриптор имеет длину 8 байт, максимальный объем любой таблицы дескрипторов составляет 216 байт. Каждая из таблиц дескрипторов имеет регистр ( GDTR для глобальной таблицы и LDTR для локальной), определяющий ее положение в памяти. Регистр GDTR содержит 48 разрядов, из которых 32 задают базовый адрес **глобальной таблицы дескрипторов**, а 16 указывают ее объем в байтах (границу таблицы). Для определения положения **дескриптора** относительно начала таблицы его номер (поле Index селектора) умножается на 8, то есть реально сдвигается на три разряда влево, так как длина дескриптора составляет 8 байт. Если **селектор** обращается к **дескриптору**, содержащемуся в таблице GDT (при TI = 0 в селекторе), то полученное смещение сравнивается с хранящейся в GDTR границей таблицы. Если нарушения границы нет, то смещение прибавляется к содержащемуся в GDTR базовому адресу, в результате чего образуется физический адрес выбираемого дескриптора ([рис. 3.10](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=3#image.3.10)).

Нулевой дескриптор в GDT является пустым, не используемым. Селектор с нулевым значением разрядов 2….15 называется нуль-индикатором. Он обеспечивает обращение к нулевому дескриптору GDT. Так как этот дескриптор не используется, то при обращении к нему происходит прерывание. Одно из возможных применений пустых селекторов заключается в следующем. Перед инициированием задачи операционная система может загрузить в регистры DS и ES пустые селекторы. Если в последующем не инициализировать эти регистры, то адресация памяти через них вызовет особый случай (прерывание). Загрузка в LDTR пустого селектора, для которого поле Index = 0, допустима. Такая операция сообщает процессору о том, что в задаче не будет использоваться локальная дескрипторная таблица. Это характерно для небольших однопользовательских систем.

Для обращения к **локальной таблице дескрипторов** предназначен 16-разрядный регистр LDTR. Он содержит **селектор**, определяющий размещение в GDT дескриптора используемой локальной таблицы дескрипторов.



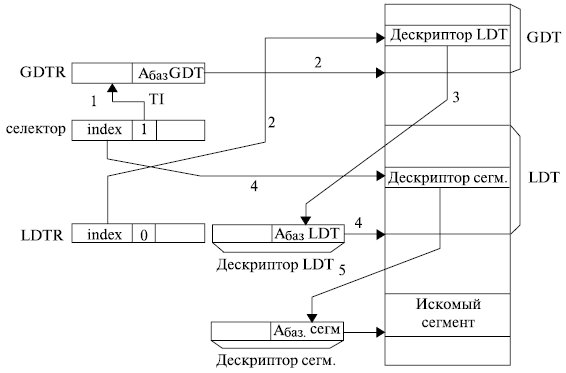
**Рис. 3.10.** Получение дескриптора, находящегося в глобальной таблице дескрипторов GDT

Такая структура упрощает работу с таблицами LDT. Благодаря описанию LDT с помощью селектора эти таблицы превращаются в обычные сегменты памяти и, в частности, могут размещаться в любых областях памяти, участвовать в свопинге и т. п. Внутри процессора с регистром LDTR ассоциируется так называемый "теневой регистр", в котором и хранится дескриптор LDT текущей задачи. Это ускоряет в последующем обращение к локальной таблице дескрипторов текущей задачи. При переключении с одной задачи на другую для замены используемой LDT достаточно загрузить в регистр LDTR селектор новой LDT, а процессор уже автоматически загрузит в теневой регистр дескриптор новой LDT при первом обращении к нему.

Если в селекторе **индикатор таблицы** TI = 1, то дескриптор сегмента выбирается из **локальной таблицы дескрипторов**. Процесс определения адреса сегмента в этом случае представлен на [рис. 3.11](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=3#image.3.11).

Он более сложен по сравнению с получением дескриптора из глобальной таблицы дескрипторов и проходит следующие этапы:

1. Анализируем, к какой из двух возможных таблиц дескрипторов идет обращение. Если в селекторе TI = 1, то обращение идет к локальной таблице дескрипторов.
2. Находим дескриптор локальной таблицы дескрипторов в глобальной таблице дескрипторов.



**Рис. 3.11.** Получение дескриптора, находящегося в локальной таблице дескрипторов LDT

1. Считываем дескриптор LDT в "теневой" регистр регистра LDTR микропроцессора. После считывания дескриптора в "теневой" регистр дальнейшее обращение к сегменту аналогично обращению к GDT, где вместо GDT R используются поля базового адреса и предела из дескриптора LDTR, находящегося в "теневом" регистре.
2. По адресу локальной таблицы дескрипторов, находящемуся в ее дескрипторе, и номеру дескриптора из обрабатываемого селектора находим дескриптор сегмента в локальной таблице дескрипторов.
3. Записываем дескриптор сегмента в "теневой" регистр сегментного регистра микропроцессора для ускорения последующих обращений к искомому сегменту.

Таким образом, при обращении к сегменту через таблицу LDT появляется дополнительный уровень вложенности, снижающий быстродействие микропроцессора. "Теневые" регистры микропроцессора частично обеспечивают решение этой проблемы.

Поле адреса дескриптора, полученного из локальной или глобальной таблицы дескрипторов, определяет начало искомого сегмента. При суммировании полученного базового адреса сегмента и смещения в сегменте получается линейный адрес искомой ячейки памяти.

В случае если режим страничной адресации выключен (в регистре CR0 бит PG = 0), полученный линейный адрес равен физическому адресу искомого операнда или команды.

Рассмотрим подробнее процесс получения адреса операнда на примере команды

MOV EAX, [ECX+ESI+20h].

В этой команде нет специальных указаний об использовании сегмента, поэтому она обращается к текущему сегменту данных, селектор которого по умолчанию находится в сегментном регистре DS. Пусть (DS) = 0000000000011.0.XXb.

Формирование физического адреса операнда включает следующие действия (для сегментированного ЛАП):

1. Образовать эффективный адрес (вычислить смещение в сегменте):

EA = (ECX)+(ESI)+20h.

1. Выбрать 3-й дескриптор ( Index = 3 ) из GDT (TI = 0).

Для этого:

* + считать базовый адрес глобальной таблицы дескрипторов ( А_{базGDT}) из GDTR ;
  + вычислить А_{баз GDT}+ (Index)*8;
  + обратиться по полученному адресу в память и считать нужный дескриптор.

1. Получить **линейный адрес**: ЛА = EA + A_{баз дескр.3}, где A_{баз дескр.3}-базовый адрес сегмента из считанного дескриптора с номером 3.
2. Так как при сегментной организации адресного пространства **линейный адрес** равен физическому, следует обратиться к памяти по сформированному адресу и передать двойное **слово** в EAX.

При TI = 1 потребовалось бы еще одно обращение к памяти для счтывания дескриптора LDT из GDT.

КОНЕЦ СЛАЙДОВ

Чтобы сократить число обращений к памяти (а такой процесс должен проходить и при считывании кода каждой команды), в микропроцессорах с архитектурой IA-32 применяется так называемое кэширование дескрипторов. Кэширование опирается на тот факт, что обращение к памяти производятся гораздо чаще, чем изменение используемых **сегментов** и переключение задач. Поэтому с каждым регистром, содержащим **селекторы** тех или иных сегментов (сегментные регистры, а также регистры **локальной таблицы дескрипторов** LDTR и регистр задач TR), ассоциируются "теневые", или кэш-регистры.

При первом считывании **дескриптора**, определяемого данным **селектором**, процессор автоматически считывает (кэширует) нужный дескриптор в соответствующий "теневой" регистр. Поскольку теперь дескриптор находится внутри МП, для получения линейного адреса памяти потребуется только сформировать эффективный адрес и просуммировать его с базовым адресом сегмента из нужного "теневого" регистра.

Так как программа обычно редко модифицирует регистры с **селекторами**, в **защищенном режиме** она будет выполняться примерно с такой же скоростью, как и в реальном режиме.

Помимо локальной и глобальной таблиц дескрипторов в микропроцессорной системе используется также **дескрипторная таблица прерываний** ( IDT ). Она содержит дескрипторы специальных системных объектов, которые определяют точки входа в процедуры обработки прерываний.

IDT служит заменой таблицы векторов прерываний 16-разрядного микропроцессора. Обращение к ней проводится только аппаратными средствами МП при возникновении аппаратных прерываний или особых случаев при выполнении программы. Программы самостоятельно не могут обратиться к IDT, так как единственный бит **индикатора таблицы** в **селекторе** сегмента идентифицирует только GDT или LDT.

До перевода процессора в **защищенный режим** необходимо создать таблицы GDT и IDT и соответственно инициализировать регистры GDTR и IDT R. Таблицы GDT и IDT определяются при загрузке в соответствующие регистры GDTR и IDT R базового адреса и предела. Это действие осуществляется только один раз в ходе подготовки к переходу в **защищенный режим**, и в дальнейшем содержимое GDTR и IDTR не изменяется. Это значит, что местонахождение таблиц GDT и IDT в известном смысле фиксировано, и они не могут участвовать в свопинге.

**Страничная организация памяти**

Страничная организация памяти применяется только в **защищенном режиме**, если в регистре управления CR0 бит PG = 1.

Основное применение страничного преобразования адреса связано с реализацией виртуальной памяти, которая позволяет программисту использовать большее пространство памяти, чем физическая основная память.

Принцип **виртуальной памяти** предполагает, что пользователь при подготовке своей программы имеет дело не с физической ОП, действительно работающей в составе компьютера и имеющей некоторую фиксированную емкость, а с виртуальной (кажущейся) одноуровневой памятью, емкость которой равна всему адресному пространству, определяемому размером адресной шины (Lша) компьютера:

V_{вирт}>>V_{физ}\\ 
V_{вирт}=2^{Lша}

Для 32-разрядного микропроцессора:

V_{вирт}=2^{32}=4 Гбайт

Программист имеет в своем распоряжении адресное пространство, ограниченное лишь разрядностью адресной шины, независимо от реальной емкости оперативной памяти компьютера и объемов памяти, которые используются другими программами, параллельно обрабатываемыми в мультипрограммной ЭВМ.

**Виртуальная память**, обеспечивая возможность программисту обращаться к очень большому объему непрерывного адресного пространства, предоставляемого в его монопольное распоряжение, обладает обычными свойствами: побайтовая адресация, время доступа, сравнимое со временем доступа к оперативной памяти.

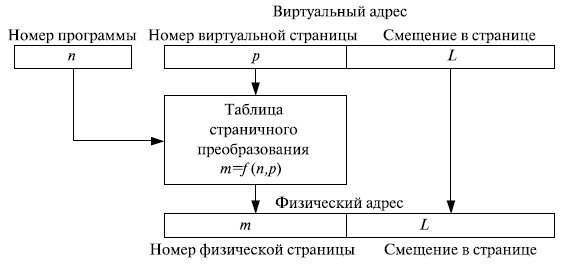
На всех этапах подготовки программ, включая загрузку в память, программа представляется в виртуальных адресах, и лишь при выполнении машинной команды виртуальные адреса преобразуются в физические. Для каждой программы, выполняемой в мультипрограммном режиме, создается своя виртуальная память. Каждая программа использует одни и те же виртуальные адреса от нулевого до максимально большого в данной архитектуре.

Для преобразования виртуальных адресов в физические физическая и виртуальная память разбиваются на блоки фиксированной длины, называемые **страницами**. Объемы виртуальной и физической страниц совпадают. Страницы виртуальной и физической памяти нумеруются. Отсутствующие в физической памяти страницы обычно хранятся во внешней памяти. Фиксированный размер всех страниц позволяет загрузить любую нужную виртуальную страницу в любую физическую.

Как отмечалось выше, при страничном представлении памяти виртуальный (логический) адрес представляет собой номер виртуальной страницы и смещение внутри этой страницы. В свою очередь, физический адрес - это номер физической страницы и смещение в ней.

Правила перевода номеров виртуальных страниц в номера физических страниц обычно задаются в виде таблицы страничного преобразования. Такие таблицы формируются системой управления памятью и модифицируются каждый раз при перераспределении памяти. Операционная система постоянно отслеживает состояние виртуальных страниц той или иной программы и определяет, находится ли она в оперативной памяти, и если находится, то в каком конкретно месте. Прикладные программы не касаются процесса страничного преобразования адреса и могут использовать все адресное пространство. Процессор автоматически формирует особый случай неприсутствия, когда программа обращается к странице, отсутствующей в физической памяти. При обработке этого особого случая ОС загружает затребованную страницу из внешней памяти, при необходимости отправляя некоторую другую страницу на диск (процесс свопинга).

Перевод виртуальных адресов в физические проиллюстрирован на [рис. 3.12](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=4#image.3.12).



**Рис. 3.12.** Принцип преобразования виртуального страничного адреса в физический

Рассмотрим пример преобразования адреса виртуальной страницы в адрес физической страницы. Пусть компьютер использует адресное пространство, предполагающее разбиение на страницы объемом Vстр = 1I, и имеет оперативную память VОЗУ = 3 страницы. Пусть на компьютере одновременно выполняются четыре программы, имеющие следующее количество страниц: VA = 2, VB = 1, VC = 3, VD = 2. Переключение между программами происходит через время кванта tk = 1. Время выполнения каждой страницы любой программы составляет t = 2tk. Полагаем, что страницы программ загружаются в оперативную память по мере необходимости и по возможности в свободные области ОЗУ. Если вся память занята, то новая страница замещает ту, к которой дольше всего не было обращений.

При таких условиях таблица загрузки оперативной памяти и таблицы страничного преобразования для каждой программы будут иметь вид, представленный в [табл. 3.2.](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=4#table.3.2)

В таблице распределения оперативной памяти выделены номера активных в данном такте страниц. В таблицах страничного преобразования прочерками отмечены ситуации, когда данная виртуальная страница отсутствует в оперативной памяти.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 3.2. Пример страничного распределения памяти в мультипрограммной ЭВМ | | | | | | | | | | | | | | | | |
| Страница | Такты | | | | | | | | | | | | | | | |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| Динамическое распределение оперативной памяти | | | | | | | | | | | | | | | |
| ОЗУ 0 | АО | АО | АО | DO | DO | DO | CO | CO | CO | C1 | C1 | C1 | C1 | C1 | C1 | C1 |
| 1 |  | B0 | B0 | B0 | A0 | A0 | A0 | D0 | D0 | D0 | D1 | D1 | D1 | D1 | D1 | D1 |
| 2 |  |  | C0 | C0 | C0 | B0 | B0 | B0 | A1 | A1 | A1 | A1 | A1 | A1 | C2 | C2 |
|  | Таблица страничного преобразования для программы А | | | | | | | | | | | | | | | |
| A 0 | 0 | 0 | - | - | 1 | 1 | 1 | - | - | - | - | - | - | - | - | - |
| 1 | - | - | - | - | - | - | - | - | 2 | 2 | 2 | 2 | 2 | 2 | - | - |
|  | Таблица страничного преобразования для программы В | | | | | | | | | | | | | | | |
| B 0 | - | 1 | 1 | 1 | - | 2 | 2 | 2 | - | - | - | - | - | - | - | - |
|  | Таблица страничного преобразования для программы С | | | | | | | | | | | | | | | |
| С 0 | - | - | 2 | 2 | 2 | - | 0 | 0 | 0 | - | - | - | - | - | - | - |
| 1 | - | - | - | - | - | - | - | - | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 2 | 2 |
|  | Таблица страничного преобразования для программы D | | | | | | | | | | | | | | | |
| D 0 | - | - | - | 0 | 0 | 0 | - | 1 | 1 | 1 | - | - | - | - | - | - |
| 1 | - | - | - | - | - | - | - | - | - | - | 1 | 1 | 1 | 1 | 1 | 1 |

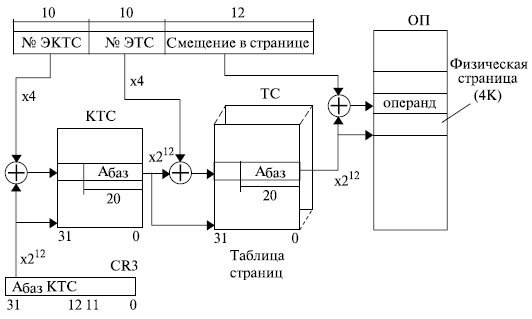
Если каждая страница имеет объем 1000 адресуемых ячеек, то, например, в такте 9 обращение по виртуальному адресу 1100 программы A (виртуальная страница 1, смещение в странице равно 100) приведет к обращению по физическому адресу 2100 (физическая страница 2, смещение в физической странице такое же, как и в виртуальной, то есть 100).

Рассмотрим теперь применение этих общих принципов страничного преобразования адреса в микропроцессоре с архитектурой IA-32 при объеме страницы в 4 Кбайт.

Основой страничного преобразования служит 32-разрядный **линейный адрес**, полученный на этапе сегментного преобразования **логического адреса**. Страничное преобразование выполняется при значении бита PG = 1 в управляющем регистре CR0.

В этом случае старшие 20 разрядов **линейного адреса** фактически представляют собой номер виртуальной страницы. Однако при прямом одноступенчатом преобразовании этого номера в номер физической страницы необходима таблица из 220 элементов длиной 4 байта каждый (20-разрядный номер страницы плюс некоторая дополнительная информация), т. е. 4 Мбайт. В мультипрограммной среде такая таблица может потребоваться для каждой задачи. Эта таблица должна постоянно храниться в оперативной памяти, чтобы существенно не увеличивать время формирования физического адреса. Для этих целей потребуется постоянное резервирование существенной части емкости ОЗУ, что на этапе появления первых ЭВМ на основе МП с архитектурой IA-32 было практически невозможно.

Вместо этого микропроцессор использует двухступенчатое страничное преобразование адреса. Корневая страница, называемая **каталогом таблиц страниц** (КТС), содержит 1024 32-разрядных **элемента каталога таблиц страниц** (ЭКТС - PDE page directory entry ). Каждый из них адресует подчиненную **таблицу страниц** (ТС), то есть всего допускается до 1024 подчиненных таблиц страниц. Каждая из таблиц страниц содержит 1024 32-разрядных **элемента таблицы страниц** (ЭТС - PTE page table entry ), каждый из которых и адресует физическую страницу. Таким образом, общее количество адресуемых физических страниц равно 220, то есть все виртуальное адресное пространство (4 Кбайт \* 220 элементов = 232 байт). Каждая таблица занимает 1024 \* 4 = 4 Кбайт, то есть ровно 1 страницу. Общий объем таблиц, используемых для страничного преобразования, не уменьшился, а даже несколько возрос за счет использования каталога таблиц страниц. Однако, во-первых, практически всегда в системе этот размер можно существенно уменьшить за счет того, что некоторые **линейные адреса** никогда не будут сформированы (а эту информацию дают **таблицы дескрипторов** сегментов), и для них не нужно создавать таблицу страниц. А во-вторых, в оперативной памяти должны постоянно находиться лишь **каталог таблиц страниц и таблица страниц** выполняемой в настоящее время программы. Остальные **таблицы страниц** могут временно храниться во внешней памяти.



**Рис. 3.13.** Страничное преобразование линейного адреса в физический

Таким образом, преобразование линейного адреса в физический имеет вид, представленный на [рис. 3.13](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=4#image.3.13).

Старшие 20 разрядов **линейного адреса** разбиваются на два 10-разрядных поля: поле **номера элемента каталога таблиц страниц** и поле **номера элемента таблицы страниц**. Так как и **каталог таблиц страниц**, и каждая **таблица страниц** занимают ровно 1 страницу и выровнены по границе страницы, то младшие 12 разрядов их базового адреса равны нулю, и для определения их физического адреса достаточно 20-разрядного поля.

Для **каталога таблиц страниц** его 20-разрядный адрес находится в регистре управления CR3. КТС постоянно находится в памяти и не участвует в свопинге.

Старшие 20 разрядов физического адреса **таблицы страниц** извлекаются из ЭТС. Структуры элемента КТС и элемента ТС схожи ([рис. 3.14](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=4#image.3.14)).

Структура элементов каталога таблиц страниц и таблицы страниц

**Рис. 3.14.** Структура элементов каталога таблиц страниц и таблицы страниц

Старшие 20 разрядов элемента дают базовый адрес **таблицы страниц** (в ЭКТС) или физической страницы (в ЭТС). Биты P, A, R/W и U/S имеют определенное сходство с аналогичными атрибутами дескриптора сегмента, другие биты имеют специфическое назначение.

**Бит присутствия** P показывает, отображается ли адрес страничного кадра (таблицы страниц или страницы памяти) на страницу в физической памяти. При P = 1 страница присутствует в ОЗУ. При P = 0 страницы в памяти нет, и обращение к этой странице вызывает прерывание типа "страничное нарушение".

**Бит доступа** А устанавливается микропроцессором в состояние А = 1 при обращении к данному страничному кадру для записи или чтения информации.

**Бит модификации** D ( Dirty - "грязный") устанавливается процессором равным 1 в элементе ЭТС при записи на данную страницу. Для элементов каталога таблиц страниц значение бита D является неопределенным. При загрузке страницы в память операционная система сбрасывает бит D. Если при необходимости выгрузки страницы во внешнюю память оказывается, что для нее D = 0, это означает, что к странице в памяти не было обращений на запись, во внешней памяти есть ее точная копия, и реально передавать страницу из памяти на диск не нужно. Тем самым экономится время при свопинге.

Бит чтения-записи R/W и бит U/S ( user/supervisor - пользователь/супервизор) определяют права доступа к таблице страниц или к странице для программ с различными уровнями привилегий. Для страниц существует только 2 уровня привилегий: уровень супервизора ( U/S = 0 ), соответствующий значению DPL сегмента 0, 1, 2, и уровень пользователя ( U/S = 1 ), соответствующий DPL = 3. Если к странице осуществляется запрос с уровнем привилегий 3 (программы пользователя), то при значении U/S = 0 ему запрещается доступ к соответствующей таблице или странице. Если U/S = 1, то при значении R/W = 0 разрешается только чтение таблицы или страницы, а при R/W = 1 - и чтение, и запись.

При запросах с большими привилегиями (системные программные уровни 0, 1, 2) допускается запись и чтение таблиц и страниц при любых значениях U/S, R/W ([табл. 3.3](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=4#table.3.3)).

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 3.3. Допустимые действия со страницами на различных уровнях привилегий | | | |
| **U/S** | **R/W** | **Допустимо для уровня 3** | **Допустимо для уровней 0, 1, 2** |
| 0 | Х | Ничего | Чтение/запись |
| 1 | 0 | Чтение | Чтение/запись |
| 1 | 1 | Чтение/запись | Чтение/запись |

Биты PWT и PCD используются для управления работой кэш-памяти при страничной адресации. Бит PCD - запрещение кэширования страницы. При PCD = 1 кэширование запрещено. Бит PWT - бит обратной записи страниц. Определяет метод обновления внешней кэш-памяти (кэш 2-го уровня). При PWT= 1 - обновление проводится методом сквозной записи (как для внутреннего кэша), при PWT = 0 - методом обратной записи.

Биты 9…11 в ЭКТС и ЭТС зарезервированы за операционной системой. Процессор никогда не использует и не изменяет эти биты. Разработчики ОС могут привлечь эти биты для хранения информации о "старении" страниц, чтобы определять страницы, подлежащие замене из внешней памяти, и для других целей.

Старшие 10 разрядов **линейного адреса** совместно с содержимым регистра управления CR3 определяют необходимый **элемент каталога таблиц страниц**. Следующие 10 разрядов линейного адреса содержат номер элемента в выбранной таблице страниц.

Так как и ЭКТС, и ЭТС имеют длину 4 байта, для получения смещения начала элемента относительно начала соответствующей таблицы необходимо его номер умножить на 4.

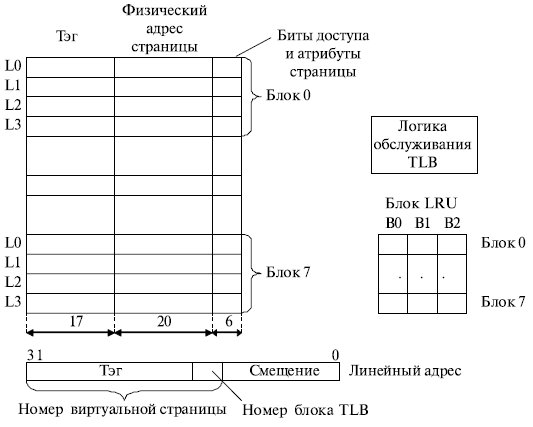
Последние 12 разрядов **линейного адреса** содержат смещение в странице. Таким образом, сумма смещения в странице и базового адреса страницы, извлеченного из ЭТС, дает физический адрес искомого байта.

**Буфер ассоциативной трансляции страничного адреса**

Страничная организация памяти требует двух дополнительных обращений к памяти: для считывания ЭКТС и ЭТС. Чтобы не ухудшать производительность процессора, в схемы управления страничным преобразованием адреса встроен **буфер ассоциативной трансляции страничного адреса** ( Translation Lookaside Buffer - TLB ).

Когда программа формирует **линейный адрес**, который отображен на находящийся в TLB элемент PTE, преобразование выполняется без дополнительных обращений к памяти.

TLB представляет собой память с ассоциативной выборкой, которая содержит 20-разрядные базовые адреса 32 страниц. Каждый из базовых



**Рис. 3.15.** Структура буфера TLB ассоциативной трансляции страничного адреса

адресов имеет свой признак (тег), в качестве которого используются старшие разряды **линейного адреса**.

Программы не могут управлять кэшированием элементов PTE. Диспетчер памяти MMU кэширует каждый используемый элемент PTE до заполнения буфера. При заполненном TLB процессор может найти страничную информацию для 128 Кбайт физической памяти (32 страницы по 4 Кбайт). При такой емкости TLB доля кэш-попаданий составляет в среднем 98 %.

TLB состоит из модуля основной памяти, блока LRU, используемого при ее обновлении, и логики обслуживания ([рис. 3.15](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=5#image.3.15)).

Основной модуль памяти содержит 8 блоков, каждый из них содержит информацию о 4 страницах, для которых ранее производилось преобразование страничного адреса. Таким образом, основной модуль содержит 32 строки, позволяющие непосредственно, без обращения к КТС и таблице страниц, определить базовый физический адрес для одной из 32 страниц, параметры которой загружены в TLB.

Каждая строка TLB содержит информацию, необходимую для ее выбора (тег, биты, определяющие доступ к странице), и информацию о выбираемой странице (базовый адрес, атрибуты). Ее структура представлена на [рис. 3.16](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=5#image.3.16).

Формат строки модуля основной памяти TLB

**Рис. 3.16.** Формат строки модуля основной памяти TLB

Поля базового адреса физической страницы и атрибуты D, U/S, R/W, PWT, PCD аналогичны соответствующим полям в ЭТС. Поле тега содержит старшие разряды линейного адреса, для которого номер виртуальной страницы преобразовывался в номер физической страницы. Бит V определяет достоверность хранимой в данной строке информации ( V = 0 -незаполненная строка, V = 1 - достоверная информация).

После формирования **линейного адреса** 3 младших разряда поля номера виртуальной страницы (биты 14….12 линейного адреса) определяют номер одного из 8 блоков TLB. Старшие 17 разрядов (биты 31…15) ЛА сравниваются одновременно ассоциативным образом с 17 битами тегов, содержащихся в каждой из 4 строк выбранного блока, с учетом бита достоверности V каждой строки. Если для некоторой строки сравнение прошло успешно, значит, эта строка содержит информацию по искомой физической странице. То есть для нее преобразование виртуального адреса в физический уже выполнялось, и результат этого преобразования содержится в найденной строке. Указанный в найденной строке базовый адрес обеспечивает выборку нужной физической страницы.

Если совпадения не было, то базовый адрес нужной страницы отсутствует в TLB, и преобразование страничного адреса проводится обычным путем с обращением к КТС и ТС. Полученная из таблицы страниц информация о физической странице вместе с 17 старшими разрядами **линейного адреса** (тег) заносится в строку одного из блоков TLB, номер которого задается битами 14…12 линейного адреса. Выбор одной из 4 строк адресованного блока, в которую заносится новое содержимое (тег, базовый адрес и др.), определяется принятым механизмом замещения.

Для TLB принят механизм замещения наиболее долго неиспользуемой строки ( least recently used - LRU ). При этом выбор замещаемой строки в блоке определяется битами B0, B1, B2 (биты LRU ), которые хранятся в дополнительном модуле памяти TLB. Этот модуль содержит 8 строк по 3 разряда каждая. Строка модуля соответствует одному из блоков основной памяти TLB. Логика обслуживания TLB переопределяет биты строки LRU по мере обращения к соответствующим строкам блока TLB.

В любой момент сочетание разрядов B0…B2 указывает, к какой из строк данного блока дольше всего не было обращения. Именно эта строка и замещается при необходимости записи новой строки, если все 4 строки блока уже заполнены. Заполненность строки определяется значением бита ее достоверности V.

При инициализации все биты B0, B1, B2 для всех блоков сбрасываются в "0". В ходе работы биты B0, B1, B2 принимают значения в соответствии с алгоритмом LRU следующим образом. Если последнее обращение в текущем блоке производилось к строкам L0 или L1, устанавливается B0 = 1. Если же оно осуществлялось к L3 или L4, то B0 = 0. При проверке пары строк L0:L1 в случае последнего обращения к L0 устанавливается B1 = 1, в противном случае B1 = 0. При проверке пары строк L2:L3 в случае последнего обращения к L2 устанавливается B2 = 1, в противном случае B2 = 0 ([табл. 3.4](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=5#table.3.4)).

|  |  |
| --- | --- |
| Таблица 3.4. Порядок изменения бит в строке LRU | |
| **Бит** **LRU** | **Последнее обращение** |
| B0 = 1  B0 = 0 | L0 или L1  L2 или L3 |
| B1 = 1  B1 = 0 | L0  L1 |
| B2 = 1  B2 = 0 | L2  L3 |

При поиске подлежащей замене строки в блоке, если все 4 строки достоверны, вначале проверяется, к какой из пар строк L0:L1 или L2:L3 производилось последнее обращение. Затем производится анализ внутри пары, отобранной на предыдущем этапе. Таким образом, замена строки в блоке TLB в случае, когда они все достоверны, проводится в соответствии с [табл. 3.5.](http://www.intuit.ru/studies/courses/604/460/lecture/5870?page=5#table.3.5)

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 3.5. Порядок замены строк в блоке TLB | | | |
| **B0** | **B1** | **B2** | **Заменяемая строка** |
| 0 | 0 | X | L0 |
| 0 | 1 | X | L1 |
| 1 | X | 0 | L2 |
| 1 | X | 1 | L3 |

В принципе, для того чтобы задать номер одной из 4 строк, достаточно 2-разрядного кода. Но при этом механизм, определяющий строку, к которой дольше всего не было обращений, может оказаться достаточно сложным. Здесь же путем добавления всего лишь одного бита получаем прозрачный механизм как по установке этих разрядов, так и по определению строк, подлежащих замене. Хранение **элементов таблиц страниц** в TLB таит в себе опасность, связанную с модификацией ЭТС "на лету", то есть в ходе выполнения программы. Предположим, что в программе предусматривается вносить изменения в **каталог таблиц страниц** или в **таблицы страниц**. После подготовки необходимых таблиц и разрешения страничного преобразования процессор будет загружать ЭТС в TLB до заполнения. Затем по мере необходимости процессор заменяет старые элементы новыми. Пусть теперь программа изменила некоторый ЭТС, хранящийся в памяти. Если измененный элемент уже находится в TLB, процессор будет пользоваться его старым значением, так как он не может узнать о том, что этот элемент в памяти был модифицирован.

Для преодоления этой коллизии программисту необходимо после всей подготовки к изменению ЭТС сразу же перезагрузить регистр CR3, например, с помощью такой пары команд:

MOV EAX, CR3

MOV CR3, EAX

При любой перезагрузке регистра CR3 все биты достоверности в TLB сбрасываются ( V = 0 ), и МП будет вынужден проводить новые преобразования страничного адреса для всех страниц, загружая при этом в TLB и модифицированный ЭТС.

Аналогичные операции выполняются при отсутствии страницы в оперативной памяти ( P = 0 ), но уже операционной системой. Страничное нарушение при отсутствии страницы в оперативной памяти вызывает следующие действия:

1. операционная система копирует запрошенную страницу с диска в оперативную память;
2. ОС загружает адрес страничного кадра в элемент таблицы страниц или каталога таблиц страниц и устанавливает P = 1. При этом могут быть установлены и другие биты, например, R/W ;
3. так как в TLB может оставаться копия старого ЭКТС или ЭТС, операционная система очищает его;
4. осуществляется рестарт команды, вызвавшей страничное нарушение.

Расширение объемов обрабатываемой информации, особенно мультимедийной, вместе с увеличением функциональных возможностей микропроцессоров и микропроцессорных систем в целом, привело к существенному изменению в параметрах страниц. Современные микропроцессоры, сохраняя главную особенность страничной организации постоянство объема страницы, обеспечивают возможность использования широкого диапазона их размеров. Так, в микропроцессоре Itanium на аппаратном уровне поддерживаются страницы емкостью 4/8/16/64/256 Кбайт, 1/4/16/64/256 Мбайт.

**Краткие итоги**. В лекции рассмотрено физическое и логическое представление памяти. Показан общий механизм формирования физического адреса при сегментно-страничном представлении адресного пространства. Рассмотрен порядок формирования адреса блоком сегментации и блоком страничного преобразования микропроцессора. Показаны способы, обеспечивающие сокращение времени такого преобразования.