# Передача управления в процессоре Pentium

## Прямой вызов процедуры из подчиненного сегмента.

Процессор должен поддерживать способ безопасного вызова модулей ОС, чтобы пользовательские программы могли получать доступ к службам ОС, например выполнять ввод-вывод с помощью соответствующих системных вызовов. Для реализации этой возможности существует несколько способов, и одним из них является размещение процедур ОС в подчиненном сегменте (С=1). **Подчиненный сегмент можно вызывать с помощью указания его селектора в командах CALL или JMP из кода программ с равным или более низким уровнем привилегий (CPL>DPL)**. Но нужно иметь в виду, что **вызываемый код будет в этом случае выполняться с привилегиями вызывающей программы**. Например, если код ОС, хранящийся в сегменте с уровнем привилегий 0, будет вызван из пользовательского приложения с уровнем привилегий 3, то процедура ОС будет наследовать привилегии пользовательской программы и возможности этой процедуры по доступу к системным данным будут весьма ограничены. Тем не менее выполнить действия над пользовательскими данными вызванная таким способом процедура ОС сможет.

## Косвенный вызов процедуры через шлюз.

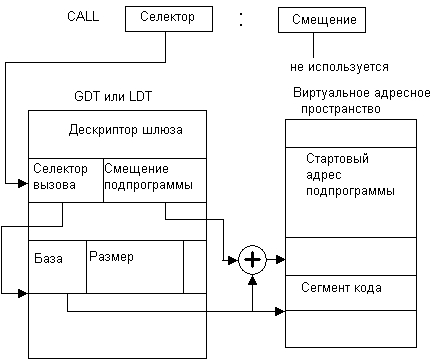
Очевидно, что оба рассмотренных выше способа вызова процедур не подходят для реализации системных вызовов. **Первый способ в принципе не позволяет вызвать из пользовательской программы с третьим уровнем привилегий процедуру операционной системы, находящуюся в неподчиненном сегменте и имеющую более высокий уровень привилегий. С помощью второго способа могут быть вызваны процедуры ОС, находящиеся в подчиненном сегменте, однако они будут выполняться с пользовательским уровнем привилегий и не смогут обрабатывать системные данные, что нужно для большинства системных вызовов.** Поэтому процессор Pentium предоставляет **еще один способ вызова подпрограмм — через шлюз (вентиль), позволяющий пользовательскому коду вызывать привилегированные процедуры, которые будут работать со своим высоким уровнем привилегий.** Шлюзы вызова обладают еще одним преимуществом — **появляется возможность контроля точек входа в вызываемые процедуры**. В обоих рассмотренных выше способах адрес точки входа в вызываемую процедуру определяется смещением, заданным в команде CALL вызывающей процедуры, то есть существует возможность задания некорректного значения смещения, в результате чего может произойти передача управления не на нужную команду или вообще в середину команды. Шлюзы вызова свободны от данного недостатка.

**Набор точек входа в привилегированные кодовые сегменты определяется заранее, и эти точки входа описываются с помощью специальных дескрипторов — дескрипторов шлюзов вызова процедур.** Дескрипторы этого типа принадлежат к системным дескрипторам, и хотя их структура отличается от структуры дескрипторов сегментов кода и данных (рис. 6.13), они также включены в таблицы LDT и GDT.



**Рис. 6.13.** Формат дескриптора шлюза вызова подпрограммы

Схема вызова процедуры через шлюз приведена на рис. 6.14. Селектор из поля команды CALL указывает на дескриптор шлюза в таблицах GDT или LDT. **Для того чтобы получить доступ к процедуре через шлюз, описываемый данным дескриптором, вызывающий код должен иметь не меньший уровень прав, чем дескриптор шлюза (то есть CPL<=DPL).** При этом вызываемый код может иметь любой уровень привилегий (в том числе и более высокий, чем у шлюза), который сохраняется при его выполнении. Это позволяет из пользовательской программы вызывать процедуры ОС, работающие с высоким уровнем привилегий. При определении адреса входа в вызываемом сегменте смещение из поля команды CALL не используется, а используется смещение из дескриптора шлюза, что не дает возможности задаче самой определять точку входа в защищенный кодовый сегмент.

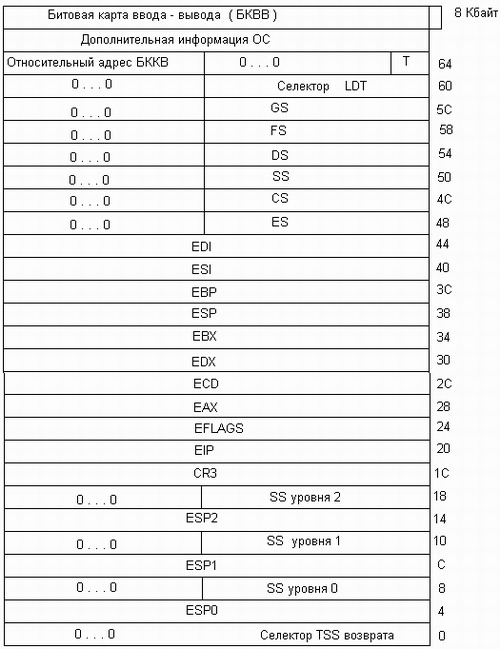


**Рис. 6.14**. Вызов подпрограммы через шлюз вызова

При вызове кодов, обладающих различными уровнями привилегий, возникает проблема передачи параметров между вызывающей и вызываемой процедурами. Для ее решения **в процессоре предусмотрено существование стеков разных уровней, по одному стеку на каждый уровень привилегий.** Используемый кодовым сегментом стек всегда соответствует текущему уровню привилегий кодового сегмента, то есть значению CPL. В сегменте контекста задачи TSS (более детально он описан ниже) хранятся значения селекторов стека SS для уровней привилегий О, 1 и 2. Если вызывается процедура, имеющая уровень привилегий, отличный от текущего, то при выполнении команды CALL создается новый стек. Для этого из сегмента TSS извлекается новое значение селектора стека, соответствующее новому уровню привилегий, которое загружается в регистр SS, из текущего стека в новый стек копируется столько 32-разрядных слов, сколько указано в поле счетчика слов дескриптора шлюза. **В новом стеке также запоминается селектор старого стека, который используется при возврате в вызывающую процедуру.**

## Вызов задачи

Механизм вызова при переключении между задачами отличается от механизма вызова процедур. В этом случае **операнд команды CALL должен указывать на дескриптор системного сегмента TSS (быть селектором дескриптора TSS).** Сегмент TSS хранит контекст задачи, то есть информацию, которая нужна для восстановления выполнения прерванной в произвольный момент времени задачи. Контекст задачи включает значения регистров процессора, указатели на открытые файлы и некоторые другие, зависящие от операционной системы, переменные. Скорость переключения контекста в значительной степени влияет на производительность многозадачной операционной системы.



**Рис. 6.15.** Структура сегмента TSS

Процессор Pentium производит аппаратное переключение контекстов задач, используя для этого сегменты специального типа TSS. Структура сегмента TSS задачи приведена на рис. 6.15. Как видно из рисунка, сегмент TSS имеет фиксированные поля, отведенные для содержимого регистров процессора, как универсальных, так и некоторых управляющих (например, LDTR и CR3). Для описания возможностей доступа задачи к портам ввода-вывода процессор использует в защищенном режиме поле IOPL (Input/Output Privilege Level) в своем регистре EFLAGS и карту битовых полей доступа к портам в сегменте TSS. **Для получения возможности безусловно выполнять команды ввода-вывода текущий код должен иметь уровень прав CPL не ниже, чем уровень привилегий операций ввода-вывода, задаваемый значением поля IOPL в регистре EFLAGS**. Если же это условие не соблюдается, то возможность доступа к порту с конкретным адресом определяется значением соответствующего бита в карте ввода-вывода сегмента TSS (карта состоит из 64 Кбит для описания доступа к 65 536 портам) — значение 0 разрешает операцию ввода-ввода с данным номером порта.

Кроме этого, сегмент TSS может включать дополнительную информацию, необходимую для работы задачи и зависящую от конкретной операционной системы (например, указатели открытых файлов или указатели на именованные конвейеры сетевого обмена).

**Информация сегмента TSS автоматически заменяется процессором при выполнении команды CALL, селектор которой указывает на дескриптор сегмента TSS в таблице GDT (дескрипторы этого типа могут быть расположены только в этой таблице, что предотвращает определение задачами своих собственных “подзадач”)**. Формат дескриптора сегмента TSS аналогичен формату дескриптора сегмента данных (за исключением, естественно, поля типа сегмента, в котором указывается, что это дескриптор сегмента TSS).

Как и в случае вызова процедуры, имеются **два способа вызова задачи — непосредственный вызов путем указания селектора дескриптора сегмента TSS нужной задачи в поле команды CALL и косвенный вызов через шлюз вызова задачи.**

Однако условие, разрешающее непосредственный вызов задачи, отличается от условия непосредственного вызова процедуры: **вызов возможен только в случае, если вызывающий код обладает уровнем привилегий, не меньшим, чем вызываемая задача (CPL<=DPL)**. Здесь применяется то же правило, что и при доступе к данным. Действительно, операционная система, работающая с высоким уровнем привилегий, должна иметь возможность запускать на выполнение пользовательские задачи, работающие с низким уровнем привилегий. В этом случае ОС не поручает ненадежному низкоуровневому коду выполнять некоторые свои функции, как это происходило бы при вызове низкоуровневых процедур, а просто выполняет переключение между пользовательскими процессами.

При вызове через шлюз (который может располагаться и в таблице LDT) вызывающему коду достаточно иметь права доступа к шлюзу, а шлюз может указывать на дескриптор TSS в таблице GDT с равным или более высоким уровнем привилегий. Поэтому **через шлюз вызова задачи можно выполнить переключение на более привилегированную задачу**.

Непосредственный вызов задачи показан на рис. 6.16. При переключении задач процессор выполняет следующие действия:

1. Выполняется команда CALL, селектор которой указывает на дескриптор сегмента типа TSS. Происходит проверка прав доступа, успешная при CPL<DPL.

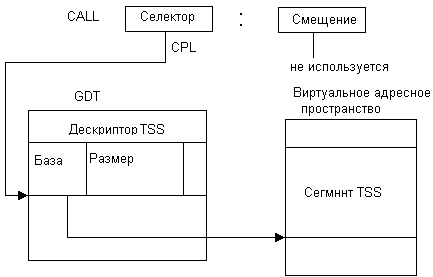
2. В TSS текущей задачи сохраняются значения регистров процессора. На текущий сегмент TSS указывает регистр процессора TR, содержащий селектор сегмента.

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

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

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

6. В поле селектора возврата нового сегмента TSS заносится селектор сегмента TSS снимаемой с выполнения задачи для организации возврата к ней в будущем.



**Рис. 6.16.** Непосредственный вызов задачи

Вызов задачи через шлюз происходит аналогично, добавляется только этап поиска дескриптора сегмента TSS по значению селектора в дескрипторе шлюза вызова (селектор в CALL определяет дескриптор шлюза задачи, дескриптор шлюза задачи содержит селектор TSS, TSS содержит CS и EIP новой задачи).

## Обработка прерываний

Таблица прерываний в реальном режиме состоит из 256 элементов, каждый из которых имеет длину в 4 байта и представляет собой дальний адрес (CS:IP) процедуры обработки прерываний. Таблица прерываний реального режима всегда находится в фиксированном месте физической памяти — с начального адреса 00000 по адрес 003FF.

В защищенном режиме таблица прерываний носит название IDT (Interrupt Descriptor Table) и может располагаться в любом месте физической памяти. Ее начало (32-разрядный физический адрес) и размер (16 бит) можно найти в регистре системных адресов IDTR. **Каждый из 256 элементов таблицы прерываний представляет собой 8-байтный дескриптор. В таблице прерываний могут находиться только дескрипторы определенного типа — дескрипторы шлюзов прерываний, шлюзов ловушек и шлюзов задач.**

Шлюзы задач уже рассматривались выше, они используются всегда для переключения с задачи на задачу. Шлюзы прерываний и ловушек специально вводятся для вызова процедур обработки прерываний. Если для вызова процедуры обработки прерывания используется шлюз задач, то происходит смена процесса, а по завершении обработки — возврат к прерванному процессу. **Обычно обслуживание прерываний со сменой процесса (и запоминанием его контекста) применяется для внешних прерываний, которые не связаны с текущим процессом**, например, когда принтер с помощью прерывания требует загрузить в его буфер новую порцию распечатываемых данных приостановленного процесса.

**Шлюзы прерываний и ловушек не вызывают смены контекста задачи**, следовательно, процедуры обработки прерываний в этом случае вызываются быстрее, чем при использовании шлюза задачи. Формат дескриптора шлюза прерывания и ловушки аналогичен формату дескриптора шлюза вызова, и обработка процессором этих шлюзов во многом аналогична вызову процедуры через шлюз вызова. Отличие состоит в том, что **при вызове процедуры через шлюз прерываний сбрасывается флаг IF и тем самым запрещаются вложенные прерывания**. При использовании шлюза ловушки сброса флага IF не происходит, но в стек при некоторых видах исключений дополнительно помещается код ошибки, вызвавшей исключение.

Итак, процессор Pentium предоставляет операционной системе широкий диапазон возможностей для организации обработки прерываний различного типа.

# Кэширование в процессоре Pentium

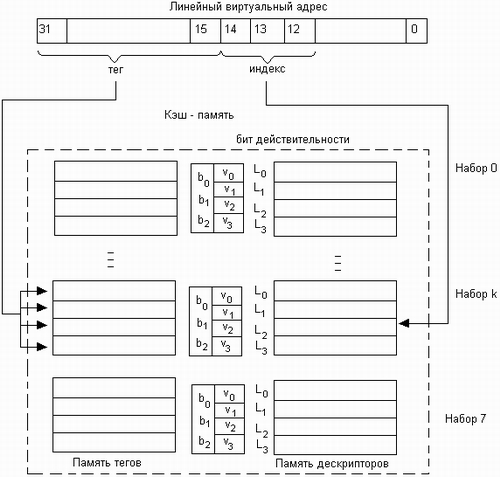
В процессоре Pentium кэширование используется в следующих случаях.

* **Кэширование дескрипторов сегментов в скрытых регистрах**. Для каждого сегментного регистра в процессоре имеется так называемый скрытый регистр дескриптора. В скрытый регистр при загрузке сегментного регистра помещается информация из дескриптора, на который указывает данный сегментный регистр. Информация из дескриптора сегмента используется для преобразования виртуального адреса в физический при чисто сегментной организации памяти либо для получения линейного виртуального адреса при страничном механизме. Доступ к скрытому регистру выполняется быстрее, чем поиск и извлечение информации из таблицы страниц, находящейся в оперативной памяти. Поэтому если очередное обращение будет относиться к одному из сегментов, дескриптор которого еще хранится в скрытом регистре (а вероятность этого велика), то преобразование адресов будет выполнено быстрее. Тем самым скрытые регистры играют роль кэша таблицы дескрипторов и ускоряют работу процессора.
* **Кэширование пар номеров виртуальных и физических страниц** в буфере ассоциативной трансляции **TLB (Translation Lookaside Buffer)** позволяет ускорять преобразование виртуальных адресов в физические при сегментно-страничной организации памяти. TLB представляет собой ассоциативную память небольшого объема, предназначенную для хранения интенсивно используемых дескрипторов страниц. В процессоре Pentium имеются отдельные TLB для инструкций и данных.
* **Кэширование данных и инструкций в кэш-памяти первого уровня**. Эта память, называемая также внутренней кэш-памятью, поскольку она размещена непосредственно на кристалле микропроцессора, имеет объем **16/32 Кбайт**. В процессоре Pentium кэш первого уровня разделен на память для хранения данных и память для хранения инструкций. Согласование данных выполняется только методом сквозной записи.
* **Кэширование данных и инструкций в кэш-памяти второго уровня.** Эта память называется также внешней кэш-памятью, поскольку она устанавливается в виде отдельной микросхемы на системной плате. Кэш-память второго уровня является общей для данных и инструкций и имеет объем **256/512 Кбайт.** Поиск в кэше второго уровня выполняется в случае, когда констатируется промах в кэше первого уровня. Для согласования данных в кэше второго уровня может использоваться как сквозная, так и обратная запись.

Рассмотрим более подробно принципы работы буфера ассоциативной трансляции и кэша первого уровня.

## Буфер ассоциативной трансляции

В буфере TLB кэшируются дескрипторы страниц из таблицы страниц (рис. 6.17). Для хранения дескриптора в кэше отводится одна строка. Каждая строка дополнена тегом, в котором содержится номер соответствующей виртуальной страницы. Строки объединены по четыре в группы, называемые наборами. Таблица TLB, используемая для преобразования адресов инструкций, имеет 32 строки и соответственно 8 наборов. Номер набора называют индексом (index). Таким образом, путем кэширования может быть получен физический адрес для доступа к 32 страницам памяти, содержащим инструкции.



**Рис. 6.17.** Буфер ассоциативной трансляции

После того как механизмом сегментации получен линейный адрес, он должен быть преобразован в физический адрес. Для этого прежде всего необходимо найти дескриптор страницы, к которой принадлежит данный адрес, и извлечь из него номер физической страницы. Обычная процедура предусматривает обращение к таблице разделов, а затем к таблице страниц. Однако физический адрес может быть получен гораздо быстрее благодаря тому, что в буфере TLB хранятся копии дескрипторов наиболее интенсивно используемых страниц. Поэтому перед тем, как начать сравнительно длительную процедуру преобразования адресов, делается попытка обнаружить нужный дескриптор страницы в быстрой ассоциативной памяти TLB. Затем на основании номера физической страницы, полученного из TLB, вычисляется физический адрес.

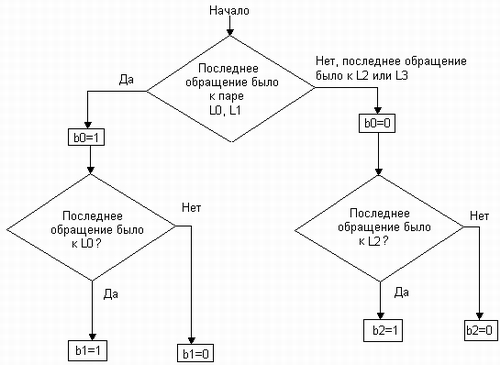
При поиске данных в TLB используется линейный виртуальный адрес. **Разряды 12-14 используются как индекс набора.** Далее проверяются биты действительности v всех строк выбранного набора. В начале работы кэш-памяти биты действительности всех строк сбрасываются в нуль. **Бит действительности принимает значение 1, когда в соответствующей строке содержится достоверная информация и сбрасывается в нуль, когда строка объявляется свободной, в результате работы алгоритма замещения**. Для всех действительных строк выполняется ассоциативная процедура сравнения тегов со старшими разрядами (15-31 разряд) линейного виртуального адреса. Если произошло кэш-попадание, то номер физической страницы быстро поступает в схему формирования физического адреса.

Если произошел промах и нужного дескриптора в TLB нет, то запускается многоэтапная процедура преобразования адреса, включающая обращения к таблицам разделов и страниц. Когда нужный дескриптор отыскивается в таблице страниц, он копируется в TLB. Номер набора, в который записывается кэшируемый дескриптор, определяется тремя младшими разрядами номера виртуальной страницы (разряды 12-14 линейного виртуального адреса).

Однако поскольку в наборе имеется четыре строки, необходимо определить, в какую именно надо поместить кэшируемые данные. Дескриптор записывается либо в первую попавшуюся свободную строку, либо, если все строки заняты, в строку, к которой дольше всего не обращались. Признаком занятости строки служит бит действительности v, имеющийся у каждой строки. Если v=0, значит, строка свободна для записи в нее нового содержимого. **Для определения строки, которая не использовалась дольше всех других в данном наборе, применяется упрощенный вариант алгоритма PseudoLRU (Pseudo Least Recently Used).** Этот алгоритм основан на анализе трех бит: b0,b1, bЗ, называемых битами обращения. Биты обращения приписываются набору и устанавливаются в соответствии с алгоритмом, приведенном на рис. 6.18. Здесь L0, LI, L2, L3 обозначают последовательные строки набора.

На замену выбирается одна из следующих строк:

* L0, если b0=0 и b1=0;
* L1, если b0=0 и b1=1;
* L2, если b0=1 и b2=0;
* L3, если b0=1 и b2=1.



**Рис. 6.18.** Алгоритм установки битов обращения

Можно легко показать, что данная процедура не всегда приводит к выбору действительно дольше всех не вызывавшейся строки. Пусть, например, обращения к строкам выполнялись в следующей хронологической последовательности: L0, L2, L3, L1, то есть ближайшее по времени обращение было к строке L1, дольше же всего не было обращений к строке LO. Биты обращения в данном случае примут следующие значения. Поскольку последнее по времени обращение было к строке из пары (LO, L1), значит, Ь0=1. А в паре (L2, L3) последнее обращение было к L3, следовательно, Ь2=0. Отсюда, по правилу, приведенному выше, на замену выбирается строка L2, вместо строки L0, к которой на самом деле дольше всего не было обращений.

Однако в большинстве случаев этот алгоритм дает результат, совпадающий с оптимальным. Например, для последовательности L0, L3, LI, L2 биты обращения имеют значения b0=0, b1=0, отсюда точное решение — L0. Даже в случае ошибки (вероятность которой составляет 33 %) решения, найденные по алгоритму PseudoLRU, близки к оптимальным. Так, в первом примере вместо строки L0, являющейся правильным решением, алгоритм дал ближайшую к ней по времени обращения строку L2.

Несмотря на то что алгоритм PseudoLRU дает в общем случае приближенные решения, он широко применяется при кэшировании, так как является быстрым и экономичным, что чрезвычайно важно для кэш-памяти.

Таким образом, в буфере TLB процессора Pentium используется комбинированный способ отображения кэшируемых данных на кэш-память: прямое отображение дескрипторов на наборы и случайное отображение на строки в пределах набора.

**Наличие TLB позволяет в подавляющем числе случаев заменить сравнительно долгую процедуру преобразования адресов, связанную с несколькими обращениями к оперативной памяти, быстрым поиском в ассоциативной памяти**.

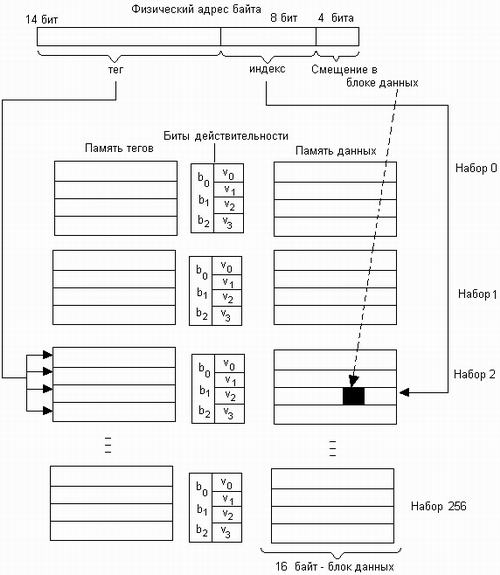
## Кэш первого уровня

Кэш первого уровня используется на этапе обработки запроса к основной памяти по физическому адресу.

Работа кэш-памяти первого уровня имеет много общего с работой буфера TLB. **В TLB единицей хранения является дескриптор, а в кэше первого уровня — байт данных**. **Обновление данных в кэше происходит блоками по 16 байт.** Таким образом, младшие 4 бита физического адреса байта могут интерпретироваться как смещение в блоке, а старшие разряды — как номер блока.

Для хранения блоков данных в кэше отводятся строки, также имеющие объем 16 байт. Строки объединены в наборы по четыре. При объеме кэша 16 Кбайт в него входят 256 (28) наборов.

При копировании данных в кэш номера блоков основной памяти прямо отображаются на номера наборов. Для этого в адресе основной памяти, относящегося к одному из байтов, входящих в блик, значение 8 битов, находящихся перед битами смещения, интерпретируется как номер набора в кэш-памяти (рис. 6.19). Остальные старшие биты адреса в дальнейшем используются в качестве тега.



**Рис. 6.19.** Кэш первого уровня процессора Pentium

Так же как в TLB, выбор строки в наборе осуществляется на основе анализа битов действительности и битов обращения по алгоритму PseudoLRU. Блок данных заносится в строку кэш-памяти вместе со своим тегом — старшими разрядами адреса основной памяти. Бит действительности строки устанавливается в 1.

При возникновении запроса на чтение из основной памяти вначале делается попытка найти данные в кэше (либо поиск в кэше совмещается с выполнением запроса к основной памяти). По индексу, извлеченному из адреса запроса, определяется набор, в котором могут находиться искомые данные. Затем для строк данного набора, содержимое которых действительно (установлены биты действительности), выполняется ассоциативный поиск: старшие разряды адреса из запроса сравниваются с тегами всех строк набора. Если для какой-нибудь строки фиксируется совпадение, это означает, что произошло кэш-попадание, и из соответствующей строки извлекается байт, смещение которого относительно начала строки определяется четырьмя младшими разрядами из адреса запроса.

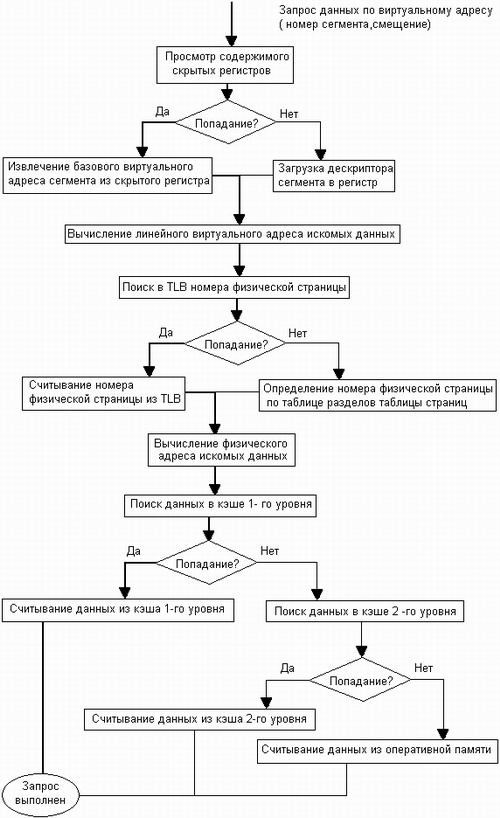
**Для согласования данных в кэше первого уровня используется метод сквозной записи, то есть при возникновении запроса на запись обновляется не только содержимое соответствующей ячейки основной памяти, но и его копия в кэш-памяти.** Заметим также, что запрос на запись при промахе не вызывает обновления кэша.

## Совместная работа кэшей разного уровня

Разные виды кэш-памяти вступают «в игру» на разных этапах обработки запроса к основной памяти. В зависимости от того, насколько удачно для запроса сложилась ситуация с попаданиями в кэш-память разного типа, время его выполнения может измениться в десятки раз. На рис. 6.20 показана схема выполнения запроса к памяти с сегментно-страничной организацией.

Рассмотрим операцию считывания операнда из оперативной памяти по его виртуальному адресу — номеру виртуального сегмента и смещения в этом сегменте. **Первое обращение к кэш-памяти происходит на этапе работы сегментного механизма, когда необходимо вычислить линейный виртуальный адрес, используя информацию дескриптора сегмента.** Все дескрипторы сегментов, входящих в виртуальное адресное пространство процесса, хранятся в оперативной памяти, в таблицах GDT и LDT. Однако реального обращения к оперативной памяти может и не быть, если нужный сегмент является одним из активных сегментов процесса — в этом случае его дескриптор находится в соответствующем скрытом регистре. Кэширование дескрипторов сегментов предоставляет первую возможность сокращения времени доступа к оперативной памяти.

Следующую возможность предоставляет **буфер ассоциативной трансляции TLB, в котором кэшируются дескрипторы страниц, что позволяет сэкономить время при вычислении физического адреса. Вероятность кэш-попадания в данном случае очень велика — в среднем она составляет 98 %, и только 2 % обращений требуют действительного чтения таблиц разделов и страниц из оперативной памяти.** При известном физическом адресе и известной степени везения искомый операнд может быть обнаружен в кэше первого уровня. Если же повезет немного меньше, то операнд найдется в кэше второго уровня.



**Рис. 6.20.** Использование кэширования на разных этапах обработки запроса (наиболее благоприятный путь выполнения запроса выделен утолщенной линией)

Таким образом, наличие разнообразных кэшей в процессоре Pentium позволяет во многих случаях существенно сократить время обработки запроса к оперативной памяти.

# Выводы

* Процессоры семейства Pentium обладают развитыми механизмами, необходимыми для организации мультипрограммного режима:
* набором привилегированных команд;
* средствами защиты сегментов кодов и данных, обеспечивающими четыре уровня привилегий;
* сегментным и сегментно-страничным механизмами виртуальной памяти;
* механизмом быстрого переключения процессов с сохранением контекста;
* встроенным кэшем оперативной памяти;
* векторной системой прерываний.
* Процессор Pentium при управлении памятью поддерживает два типа таблиц дескрипторов сегментов: глобальную таблицу дескрипторов GDT, описывающую сегменты операционной системы и разделяемые сегменты прикладных процессов, и локальные таблицы дескрипторов LDT, которые содержат дескрипторы сегментов отдельных пользовательских процессов.
* При страничном режиме работы виртуальное адресное пространство состоит из 16 Кбайт сегментов по 4 Гбайт каждый — всего 64 Тбайт, а при сегментно-страничном режиме работы все сегменты отображаются в общий диапазон адресов 4 Гбайт.
* Каждый сегмент виртуального адресного пространства описывается дескриптором, который содержит базовый адрес, размер сегмента, а также ряд признаков, в том числе уровень привилегий сегмента DPL, определяющий права доступа к нему.
* В процессоре Pentium существует несколько способов вызова процедур, а также специальные средства вызова задач, позволяющие автоматически сохранять и восстанавливать наиболее значимую часть контекста задачи.
* Процессор Pentium поддерживает векторную схему прерываний, с помощью которой может быть вызвано 256 процедур обработки прерываний. Прерывания могут быть инициированы внешним сигналом (аппаратные прерывания), некорректным выполнением инструкции (исключения), а также специальной инструкцией INT (программные прерывания).
* В процессоре Pentium активно применяется кэширование:
* кэширование дескрипторов сегментов в скрытых регистрах процессора;
* кэширование дескрипторов страниц в буфере ассоциативной трансляции TLB;
* кэширование данных и инструкций в кэш-памяти первого уровня;
* кэширование данных и инструкций в кэш-памяти второго уровня.

# Задачи и упражнения

1. Существует ли защищенный режим в большинстве современных процессоров или это специфический режим процессоров Pentium?

2. Значения каких системных регистров процессора должен использовать программный модуль ОС, чтобы произвести обращение к индивидуальной части памяти текущего процесса?

3. Представьте, что для задач всех уровней привилегий используется один общий стек. К каким последствиям это может привести?

4. Почему в сегменте состояния задачи TSS хранятся значения селекторов стека для уровней привилегий О, 1 и 2, но нет значения для селектора уровня 3?

5. В какой памяти — физической или виртуальной — задает положение сегмента при выключенном страничном механизме базовый адрес, хранимый в дескрипторе сегмента?

6. Зачем нужны шлюзы вызовов процедур и задач, если существует возможность непосредственного вызова?

7. Заполните следующую таблицу, в которой укажите возможность или невозможность непосредственного вызова процедуры со сменой кодового сегмента для различных сочетаний уровней привилегий вызывающего и вызываемого сегментов и типов сегментов.

|  |  |  |
| --- | --- | --- |
| **Соотношение уровней** | **Тип сегмента** | **Возможность доступа** |
| CPL < DPL | С= 1 |  |
| CPL < DPL | С=1 |  |
| CPL = DPL | С= 1 |  |
| CPL > DPL | C = 0 |  |
| CPL < DPL | C = 0 |  |
| CPL = DPL | C=0 |  |

8. Можно ли на базе процессора Pentium реализовать систему управления памятью с фиксированными разделами?

9. Можно ли выгружать страницы, которые хранят разделы таблицы страниц?

10. По каким соображениям в процессорах Pentium запрещено вызвать менее привилегированные процедуры, но разрешено вызывать менее привилегированные задачи?

11. В чем принципиальное отличие использования шлюза прерываний от использования шлюза задачи?

12. Поддерживает ли процессор Pentium приоритезацию запросов прерывания между несколькими внешними устройствами?