Мультипрограммный режим работы микропроцессора

Многозадачностью (мультипрограммным режимом работы) называют такой способ организации работы системы, при которой в ее памяти одновременно содержатся программы и данные для выполнения нескольких процессов обработки информации (задач). В этом режиме должна обеспечиваться взаимная защита программ и данных, относящихся к различным задачам, а также возможность перехода от выполнения одной задачи к другой (переключение задач).

Под **задачей (процессом)** понимается последовательность взаимосвязанных действий, ведущих к достижению некоторой цели. В вычислительной технике под задачей понимается конкретная сущность, которая тесно связана с архитектурой процессора и обладает своим виртуальным адресным пространством и состоянием. В более простом смысле будем полагать, что **задача** - это *программа*, которая выполняется или ожидает выполнения, пока выполняется другая *программа*.

Процесс может находиться в следующих состояниях:

- порождение - подготавливаются условия для первого исполнения на процессоре;
- активное состояние (счет) - программа исполняется на процессоре;
- ожидание - программа не исполняется по причине занятости какого-либо ресурса;
- готовность - программа не исполняется, но для исполнения предоставлены все необходимые в текущий момент ресурсы, кроме *центрального процессора*;
- окончание - нормальное или аварийное завершение программы, после которого процессор и другие ресурсы ей не предоставляются.

Применительно к компьютерам в *определение* задачи обычно включаются ресурсы, необходимые для достижения цели. Понятие "*ресурс*" строго не определено. Будем считать, что всякий потребляемый *объект* (независимо от формы его существования), обладающий некоторой практической ценностью для потребителя, является **ресурсом**:

объем оперативной памяти, время счета на процессоре, дисковое *пространство* и т. д.

Основные черты мультипрограммного режима:

- оперативной памяти находятся несколько программ в состояниях *активности, ожидания* или готовности;
- время работы процессора разделяется между программами, находящимися в памяти в состоянии готовности;
- параллельно с работой процессора происходит подготовка и обмен с несколькими внешними устройствами (ВУ).

Мультипрограммирование предназначено для повышения пропускной способности вычислительной системы путем более равномерной и полной загрузки всего его оборудования, в первую *очередь* - процессора.

При этом скорость работы самого процессора и номинальная *производительность* ЭВМ не зависят от использования *мультипрограммирования*.

Мультипрограммный режим имеет в ЭВМ аппаратную и программную поддержку. Аппаратура, используемая при организации мультипрограммного режима, включает контроллеры *ОЗУ* и внешних устройств, которые могут работать параллельно с процессором, систему прерывания, *аппаратные средства* защиты программ и данных и т. д.

Программная составляющая содержит мультизадачную операционную систему, драйверы внешних устройств, обработчики прерываний и другие средства. *Операционная система*, реализуя **мультипрограммный режим**, должна распределять (в том числе динамически) между параллельно выполняемыми программами **ресурсы** системы (время процессора, оперативную и *внешнюю*

память, устройства ввода-вывода) с целью увеличения пропускной способности и с учетом ограничений на **ресурсы** и требований *по* срочности выполнения отдельных программ.

Эффективность работы мультипрограммной ЭВМ можно оценить количеством задач, выполненных в единицу времени (*пропускная способность*), и временем выполнения отдельной программы.

Важное *значение* при *анализе работы* ЭВМ имеет *определение* степени использования ее ресурсов. Для этого широко применяются следующие показатели (рис. 6.1):

k_q коэффициент загрузки устройства:

$$k_q = \frac{T_q}{T}$$

где T_q - время занятости устройства q за общее время T работы ЭВМ;

L_q *средняя длина очереди* запросов к устройству q:

$$L_q = \frac{\sum_{j=1}^{n} L_{q_i} * \Delta t_i}{T}$$

где L_{qi} - *длина* очереди к устройству q на интервале времени Δt_i и $\sum_{j=1}^{n} \Delta t_i = T$

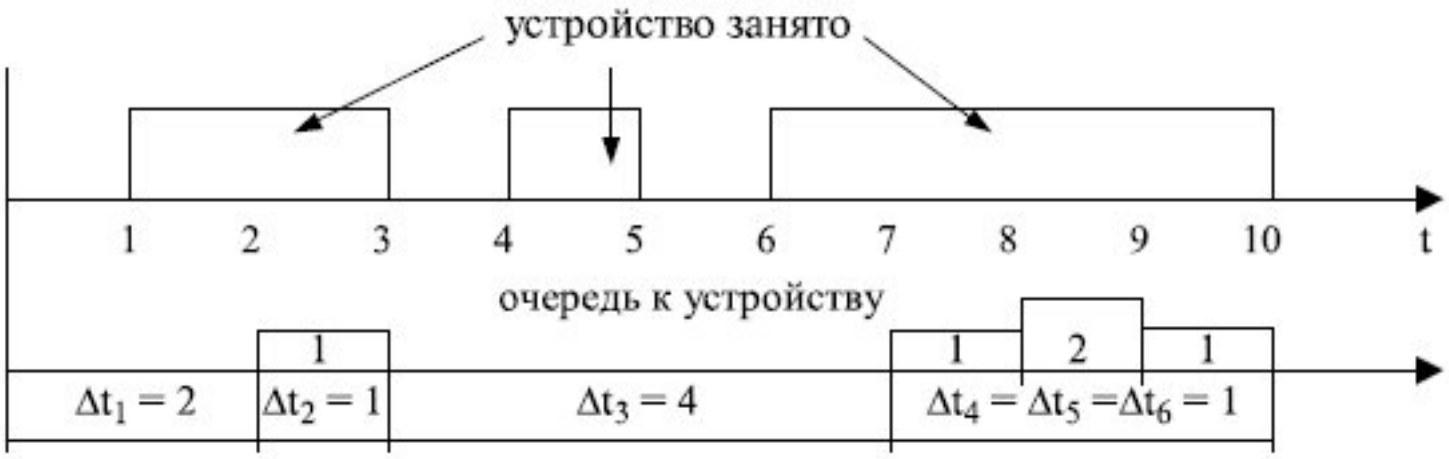

Рис. 6.1. Оценка показателей работы мультипрограммной ЭВМ

Для представленного на рис. 6.1 случая:

$$L_q = \frac{0 \times 1 + 1 \times 1 + 0 \times 4 + 1 \times 1 + 1 \times 2 + 1 \times 1}{10}$$

Помимо средней длины очереди важна также и динамика изменения ее длины.

По значениям k_q, L_q и изменениям во времени значения Lq можно определить наиболее дефицитный **ресурс** в системе, ее "узкое *место*".

Устранение этих "узких мест" может проводиться или за счет *увеличения производительности* соответствующего **ресурса**, или выбором такой смеси задач, которая обеспечивала бы более равномерное использование всех ресурсов (например, одни задачи более активно используют *процессор* (счетные задачи), другие - *жесткий диск* (работа с базами данных), третьи - устройства ввода-вывода (*вывод* большого объема графической информации)).

Рассмотрим пример выполнения четырех программ в мультипрограммном режиме при *коэффициенте мультипрограммирования* равном 2.

Коэффициент мультипрограммирования (КМ) - это количество программ, обрабатываемых одновременно в **мультипрограммном режиме**.

Полагаем, что ЭВМ имеет 3 устройства, которые могут работать параллельно: центральный *процессор* (*CPU*), устройство ввода (*IN*) и устройство вывода (OUT), а программы проходят следующий *цикл работы*:

счет1 - ввод - счет2 - *вывод*. Времена выполнения соответствующих блоков программ заданы в табл. 6.1.

Будем считать, что программы имеют относительный приоритет. То есть, во-первых, если на какой-либо *ресурс* при его освобождении одновременно претендует несколько программ, то он предоставляется программе с наименьшим номером. Во-вторых, *программа*, занявшая *ресурс*, не освобождает его до истечения требуемого времени, даже если в этот период на *ресурс* станет претендовать *программа*, имеющая больший приоритет.

Таблица 6.1. Характеристики программ

Программа	CPU1	IN	CPU2	OUT
1	2	1	4	2
2	2	2	1	3
3	4	3	3	1
4	2	2	2	2

Более приоритетной считаем программу с меньшим номером.

Очередь программ к процессору обозначим *Ready*, а общую *очередь* к внешним устройствам - Wait.

Полученная *диаграмма* выполнения программ представлена на рис. 6.2.

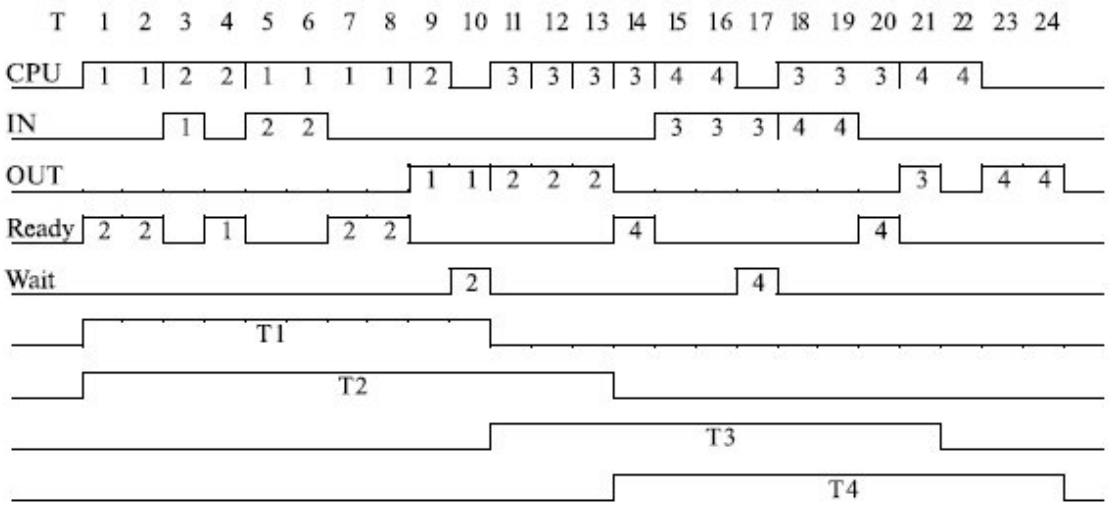

Рис. 6.2. Диаграмма выполнения программ в ЭВМ при Км = 2

Построив аналогичную диаграмму работы ЭВМ для Км = 3, получим результаты, представленные в табл. 6.2 (Км = 1 соответствует однопрограммной работе ЭВМ, и результаты для этого случая могут быть получены расчетными методами).

Анализ показывает, что с увеличением **коэффициента мультипрограммирования** *пропускная способность* ЭВМ будет увеличиваться, стремясь к некоторому пределу, определяемому характеристиками **ресурсов** ЭВМ. В то же время каждая *программа* будет выполняться в общем случае более длительное время из-за необходимости ожидания освобождения **ресурсов**, занятых другими программами. При увеличении **коэффициента мультипрограммирования** изменение значений *показателей эффективности* зависит от того, в каком состоянии находится система: перегрузки или недогрузки. Если какие-либо **ресурсы** ЭВМ задействованы достаточно интенсивно, то добавление новой программы, активно использующей эти ресурсы, будет малоэффективным.

Таблица 6.2. Характеристики работы ЭВМ при различных коэффициентах *мультипрограммирования*

Характеристика	**Км = 1**	**Км = 2**	**Км = 3**
Время выполнения программы Т1	9	10	10
-"- Т2	8	13	13
-"- Т3	11	11	19
-"- Т4	8	11	12
Время выполнения всех программ	36	24	22
Пропускная способность (П)	0.11	0.17	0.18
k_{cpu}	0.56	0.83	0.91
k_{in}	0.22	0.33	0.36
k_{out}	0.22	0.33	0.36

Микропроцессор для поддержки **мультипрограммного режима** работы использует определенный набор аппаратных средств.

В компьютере, работающем в **мультипрограммном режиме**, выделяется область памяти, доступная только операционной системе, в которой хранится вся *информация*, необходимая для *рестарта* задачи (контекстная *память*, или *кадр* состояния). При *переключении контекста компьютер* просто переходит от одного раздела к другому. В качестве такого раздела в универсальных микропроцессорах выступает **сегмент состояния задачи** TSS, который является небольшим сегментом данных с разрешенными операциями считывания и записи, - *доступ* к нему не разрешается никаким программам, даже на самом высоком **уровне привилегий**. К сегментам TSS может обращаться только сам *процессор*.

Для поддержки работы с сегментом состояния задачи служит 16-разрядный **регистр задачи** TR, в который заносится **селектор**дескриптора TSS, и связанный с TR программно недоступный 64-разрядный "теневой" *регистр*, в который загружается **дескриптор** TSS. Дескрипторы сегментов состояния задач хранятся только в **глобальной таблице дескрипторов** *GDT*. Для переключения задач используется **шлюз задачи**.

Сегмент состояния задачи состоит из двух частей (рис. 6.3). Обязательная часть TSS объемом 104 байта содержит информацию, необходимую для *рестарта* данной задачи после ее вызова на *исполнение*, а также некоторую другую информацию. Дополнительная часть может содержать какую-либо информацию о задаче, используемую операционной системой (имя задачи, комментарии и т. д.), и **битовую карту ввода/вывода (БКВВ)**, определяющую устройства ввода/вывода, к которым разрешено обращение данной задачи при определенных ситуациях.

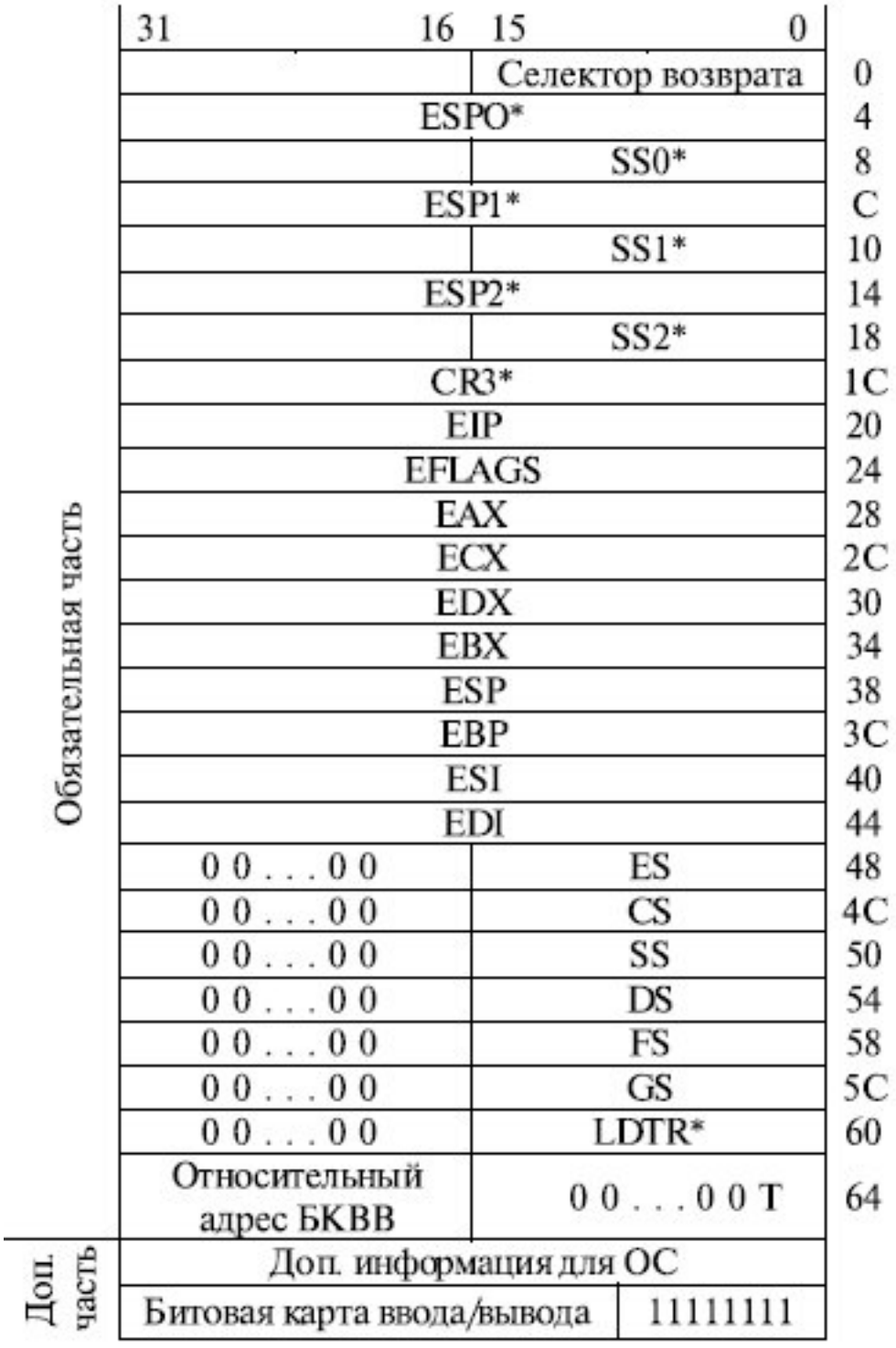

Рис. 6.3. Структура сегмента состояния задачи

Содержимое ряда полей TSS не изменяется при решении задачи (на рис. 6.3 они помечены звездочкой): селектор *LDT* данной задачи (то есть *регистр* LDTR), *регистр управления* CR3 (базовый *адрес* **каталога таблиц страниц**), поля *SSi* и ESPi, которые определяют начальные адреса стеков при переключении к задачам с более высоким **уровнем привилегий**, что обеспечивает их более надежную защиту.

При переключении задачи *процессор* может перейти к другой **локальной таблице дескрипторов** *LDT* и перезагрузить базовый *регистр* каталога страниц CR3. Это позволяет назначить каждой задаче свое *отображение логических адресов* на физические, что служит дополнительным средством защиты, так как задачи можно изолировать и предотвратить их взаимодействие друг с другом.

Поле селектора возврата обеспечивает *связь* данной задачи с вызвавшей ее программой. **Селектор возврата** - это **селектор** TSSпредыдущей задачи, при выполнении которой произошел вызов данной задачи (TR предыдущей задачи, если предполагается возврат к ней).

Бит ловушки T используется при отладке программного обеспечения аналогично биту TF в регистре флагов: если T = 1, то при переключении на данную задачу возникает *прерывание*.

В TSS отсутствуют поля для хранения *регистров управления* CR0 и CR2. Это означает, что их содержимое не меняется при переключении задач. Следовательно, *страничное преобразование* и условия работы с *FPU* являются глобальными для всех задач. Для каждой задачи может быть свой **каталог таблиц страниц**, но *страничное преобразование* может быть разрешено или запрещено только для *микропроцессорной системы* в целом.

Объем дополнительной части TSS зависит от количества служебной информации ОС, определяемой характером решаемой задачи, и размеров применяемой битовой карты ввода/вывода. Дополнительная часть TSS может вообще отсутствовать.

Относительный *адрес* БКВВ определяет положение битовой карты в TSS. Каждый *бит* БКВВ соответствует однобайтовому порту ввода/вывода. Так как *микропроцессор* может обращаться к 216 портов, полная битовая карта ввода/вывода, определяющая возможность их обслуживания, -

это строка длиной до 64 Кбит. Когда для задачи определена БКВВ, ей предоставляется дополнительная возможность выполнения команд ввода/вывода. Если обычная защита *по* привилегиям запрещает ввод/*вывод* (*уровень привилегий* задачи меньше уровня, установленного в *поле* IOPL *регистра флагов*), *процессор* обращается к БКВВ для дополнительной проверки возможности ввода/вывода на конкретных устройствах. Если соответствующие этим адресам биты карты содержат нули, то *операции* ввода/вывода разрешаются. В противном случае формируется особый случай нарушения защиты. Таким образом, БКВВ обеспечивает задаче свободный *доступ* к незащищенным портам ввода/вывода без требования понижения значения в *поле* IOPL *регистра флагов*.

За последним байтом БКВВ в TSS должен следовать заключительный *байт*, состоящий из одних единиц. *Адрес* этого байта должен соответствовать границе сегмента, определенной дескриптором TSS.

После *поле* предела должна быть указана *длина* не менее 104 *байт* (*длина* обязательной части TSS). Если указанный размер сегмента меньше 104 *байт*, то происходит *прерывание*.

Байт доступа дескриптора TSS имеет следующий вид (рис. 6.4):

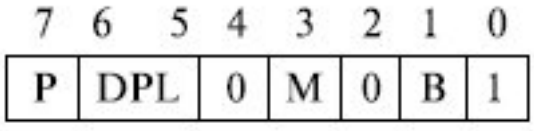

Рис. 6.4. Байт доступа дескриптора сегмента состояния задачи

У него есть ряд особенностей *по* сравнению с дескрипторами обычных сегментов.

Бит занятости B устанавливается в 1 при переключении на данную задачу. Используется для обнаружения попытки вызова задачи, выполнение которой прервано. Переключение задач производится, только если B = 0.

При B = 1 возникает *прерывание*. Установка B = 1 в дескрипторе TSS производится командами JMP и CALL, переключающими *микропроцессор* на выполнение данной задачи. При этом другие обращения в мультипрограммной системе к этой задаче будут запрещены.

В байте доступа дескриптора TSS *поле DPL* не означает *уровень привилегий* самого сегмента состояния задач, а аналогично функции поля *DPL* шлюза вызова показывает, какие программы могут обращаться к задаче, которая определяется данным дескриптором TSS.

Значение M = 0 обеспечивает совместимость с микропроцессором i286, в котором впервые появился **мультипрограммный режим работы**.

Переключение задач

Переключение задач осуществляется командами межсегментной передачи управления, при обработке прерываний и возврате из обработчиков. Если при этом управление передается **дескриптору сегмента состояния задачи** или **шлюзу задачи**, то происходит переключение задач. В соответствии с содержимым обязательной части TSS производится *загрузка* регистров МП, и он начинает выполнение поступившей задачи.

Шлюз задачи имеет следующий формат (рис. 6.5):

Рис. 6.5. Формат шлюза задачи

Он имеет статус *системного объекта*, и его тип, определенный в байте доступа, имеет *значение* 0101.

Переключение задач похоже на вызов процедуры, но при этом сохраняется больше информации. *Информация* о состоянии процессора сохраняется в TSS, а не в стеке.

Обращение к TSS осуществляется путем загрузки в *регистр* TR **селектора**, который адресует размещенный в *GDT***дескриптор** TSS соответствующей задачи. Обычно *команда LTR* загрузки TR используется только при инициализации системы для установки начального содержимого TR.

В дальнейшем этот *регистр* загружается микропроцессором при выполнении команд, переключающих задачу.

При переключении задач проверяются привилегии, установленные для доступа к данным:

DPL' <= max (CPL, RPL),

т. е. допускается переключение на задачи, чья степень защиты меньше или равна **уровню привилегий** текущей программы и запроса.

Как правило, *поле DPL* дескриптора TSS равно 0, поэтому переключение задач могут производить только привилегированные программы.

Для переключения на задачи с большим приоритетом используется механизм **шлюзов задач**, аналогичный **шлюзам вызова**. При этом необходимо сохранение стеков более привилегированных задач, что и осуществляется с помощью сохранения *SSi*, ESPiв **сегменте состояния задачи**.

Для переключения задач в командах межсегментных переходов можно указать селектор шлюза задачи (косвенное переключение задачи) или селектор дескриптора TSS (прямое переключение задачи без привлечения шлюза задачи). Принципиальной разницы между этими переключениями нет. Но обычно цель использования шлюза задачи связана с мультизадачностью. Так как для каждой задачи имеется единственный **дескриптор** TSS, который должен находиться в *GDT*, все достаточно привилегированные

задачи имеют к нему *доступ* и могут произвести переключение задачи. Но если определить для данного **дескриптора** TSS несколько шлюзов задачи и разместить их в различных *LDT*, можно разрешить переключать задачи только тем задачам, которые имеют *шлюз* задачи в своих *LDT*. При этом сохраняется *доступ* к дескриптору TSS для программ с нулевым уровнем привилегий, которые могут прямо обращаться к нему в *GDT*.

Рассмотрим теперь переключение задач более подробно на примере механизма прямого переключения (рис. 6.6).

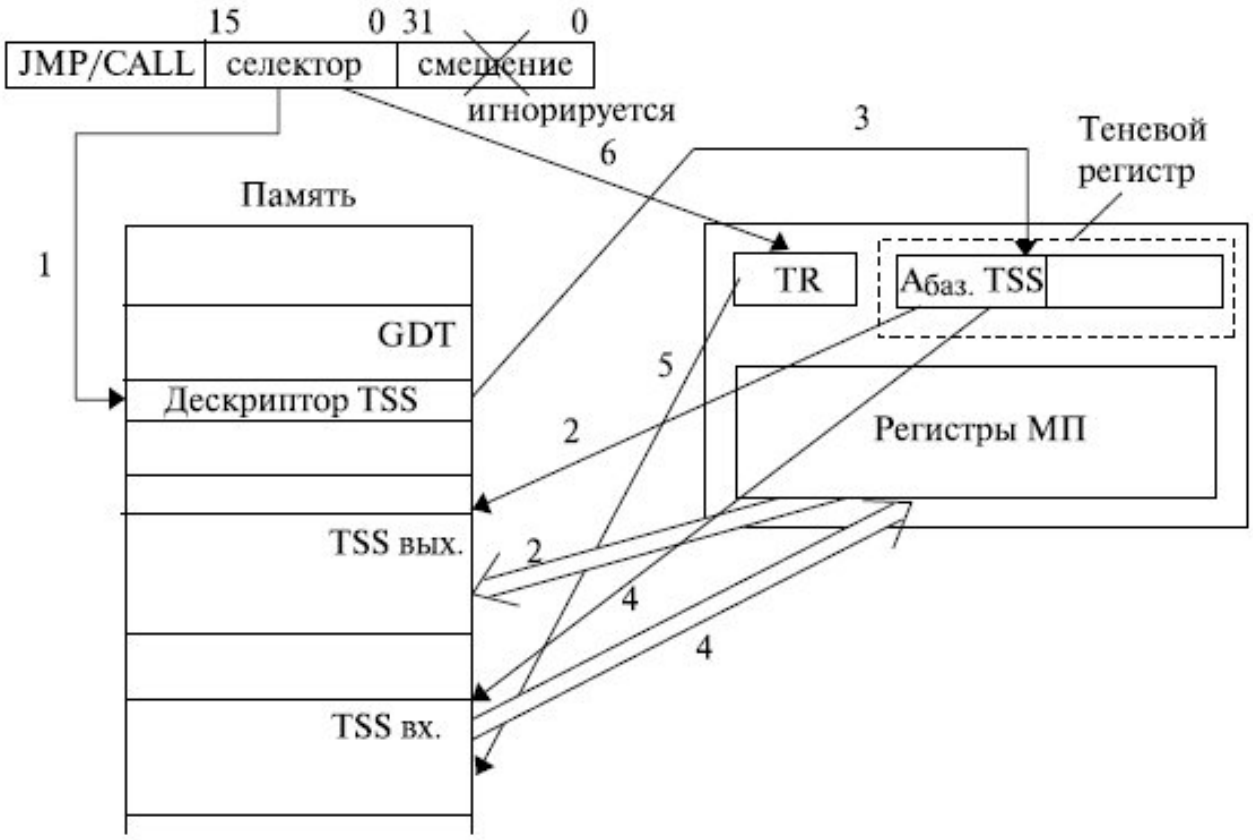

Рис. 6.6. Механизм прямого переключения задач

В общем случае *команда* межсегментного перехода содержит 3 поля:

- поле *кода операции* (JMP - *безусловный переход*, CALL - переход с возвратом),
- селектор нового сегмента команд, который заносится в *сегментный регистр* CS, и
- поле смещения, содержимое которого заносится в регистр - *указатель команд EIP*.

При переключении задач выполняется следующая последовательность действий:

1. В случае, когда тип дескриптора, определяемого вторым полем *команды перехода*, указывает, что данный дескриптор является дескриптором TSS и новая задача не занята (B = 0), запускается механизм переключения задач. В этом случае поле смещения в команде игнорируется.
2. В теневом регистре регистра задач TR микропроцессора к данному моменту содержится дескриптор TSS исполняемой задачи. Его поле адреса указывает на область памяти, где должно быть сохранено состояние текущей задачи. Микропроцессор записывает в этот сегмент необходимую информацию.
3. Дескриптор TSS новой задачи переписывается из глобальной таблицы дескрипторов в теневой регистр регистра задач микропроцессора. Этот дескриптор определяет положение в памяти сегмента состояния новой задачи.
4. Информация о новой задаче переписывается из своего TSS в регистры микропроцессора.
5. Если команда, вызвавшая переключение задач, предполагает последующий возврат к старой задаче (CALL), то в поле **селектора возврата** TSS входящей задачи заносится содержимое

регистра TR снимаемой с обработки задачи. При этом устанавливается значение бита вложенной задачи NT = 1 в регистре флагов EFLAGS.

6. В регистр TR микропроцессора из второго поля команды заносится селектор дескриптора TSS новой задачи.

Использование флага вложенной задачи NT и бита занятости B позволяет организовать корректную обработку вложенных задач. Порядок установки этих *бит* в зависимости от *типа команды*, вызвавшей переключение задач, указан в табл. 6.3.

Таблица 6.3. Модификация флагов занятости и вложенности при переключении задач

Тип команды	**Входящая задача**		**Выходящая задача**	
	Флаг NT в EFLAGS	**Бит в дескрипторе TSS**	**Флаг NT в EFLAGS**	**Бит в дескрипторе TSS**
CALL	1	1	X	1
JMP	0	1	X	0

Команда JMP при переключении на новую задачу не сохраняет в TSS **селектор возврата** и устанавливает NT = 0,а также B = 0 в дескрипторе старой задачи и B = 1 в дескрипторе новой задачи.

Команда CALL устанавливает B = 1 для новой задачи, но сохраняет B = 1 для предыдущей. Таким образом, каждая задача в цепи вызовов оказывается занятой, что запрещает применение рекурсивных процедур и реентерабельных программ.

Возврат из задачи осуществляется *по* команде IRET, которая анализирует флаг NT и при NT = 1 осуществляет переключение на задачу, задаваемую **селектором возврата** в TSS текущей задачи. При NT = 0 осуществляется обычная процедура возврата из процедуры с восстановлением из стека содержимого CS, *EIP*, EFLAGS.

Обычная *команда* RET возврата из подпрограммы не учитывает вложения задач.